PDF-Datei: SQL_WIN.PDF

Diese Datei darf unentgeltlich heruntergeladen und auch weitergereicht werden. Sie darf nicht als Bestandteil eines Buches oder sonst in irgendeiner Form weiterverarbeitet werden.

Dr. Norbert Kessel

SQLWindows

Einführung in die Programmierung von Datenbankanwendungen

Norbert Kessel

Inhaltsverzeichnis

1 ALLGEMEINER TEIL 2
1.1 Einführung in die Programmierung von Datenbankanwendungen 2
1.2 Einführung in die Sprache SQL unter Verwendung von SQLTalk 11
1.2.1 Einführung in die Sprache SQL 11
1.2.1.1 Geschichte von SQL 11
1.2.1.2 Anmerkungen zur Sprache SQL 12
1.2.1.3 SQL-Anweisungen 13
1.2.1.3.1 Datenbanken anlegen (create), reorganisieren (reorganize), löschen (drop) 13
1.2.1.3.2 Tabellen anlegen (create), ändern (alter), löschen (drop), Indexe anlegen (create Index) 14
1.2.1.3.3 Tabellen und Datensätze 15
1.2.1.3.3.1 Datensätze eingeben 15
1.2.1.3.3.2 Datensätze ändern 15
1.2.1.3.3.3 Datensätze löschen 16
1.2.1.3.3.4 Datensätze abfragen 16
1.2.1.3.3.4.1 Einfache Abfragen 16
1.2.1.3.3.4.2 Zusammenfassende Abfragen 18
1.2.1.3.3.4.3 Unterabfragen 19
1.2.2 Die Benutzung von SQLTalk zur Erzeugung einer Datenbank mit mehreren Tabellen 20
1.2.2.1 Anlegen der Datenbank FIRMA mit SQLTalk 21
1.2.2.2 Anlegen der Tabellen 22
1.2.2.3 Eingabe von Datensätzen 24
1.3 Einführung in SQLWindows 26
1.3.1 Generelles Arbeiten mit SQLWindows 28
1.3.1.1 Verzeichnisse, Dateien und die Programmgruppe SQLWindows 28
1.3.1.2 Der Startbildschirm von SQLWindows 30
1.3.1.3 Das Menü von SQLWindows 34
1.3.1.4 Ein erster Versuch: Das Programm "hello world" 37
1.3.2 Die Gliederung einer SQLWindows-Anwendung 40
1.3.2.1 Die unveränderte Vorlage 40
1.3.2.2 Verändern der Vorlage 43
1.3.3 SAL, Die Programmiersprache von SQLWindows 48
1.3.3.1 Datentypen 48
1.3.3.2 Variable 49
1.3.3.3 Konstanten 55
1.3.3.4 Statements 58
1.3.3.4.1 Set 58
1.3.3.4.2 If, Else, Else If 59
1.3.3.4.3 Select Case 60
1.3.3.4.4 Call 61
1.3.3.4.5 On 62
1.3.3.4.6 While, Loop 63

1.3.3.4.7 Return 65
1.3.3.4.8 When SqlError 67
1.3.3.5 Funktionen 69
1.3.3.6 Operatoren 70
1.3.3.7 Kommentare 71
1.3.4 Objekte in SQLWindows und deren Verwendung 72
1.3.4.1 Top-Level Objekte 74
1.3.4.1.1 Form Window 74
1.3.4.1.2 MDI Window 81
1.3.4.1.3 Dialog Box 82
1.3.4.1.4 Table Window 83
1.3.4.1.5 Quest-Objekte 89
1.3.4.2 Child Objekte 95
1.3.4.2.1 Objekte zur Anzeige von Linien, Kommentaren und Bildern 97
1.3.4.2.1.1 Background Text 98
1.3.4.2.1.2 Frame 98
1.3.4.2.1.3 Line 99
1.3.4.2.1.4 Picture 100
1.3.4.2.2 Objekte zur Ein- und Ausgabe numerischer Daten, Zeichen und Datumswerten 102
1.3.4.2.2.1 Data Field 102
1.3.4.2.2.2 Multiline Field 109
1.3.4.2.2.3 List Box 111
1.3.4.2.2.4 Combo Box 118
1.3.4.2.2.5 Table Window, Quest Window 120
1.3.4.2.3 Objekte, zur Ein- und Ausgabe von logischen Informationen 123
1.3.4.2.3.1 Radio Buttons 124
1.3.4.2.3.2 Check Box 128
1.3.4.2.3.3 Option Button 131
1.3.4.2.4 Objekte für Benutzer-Aktionen 133
1.3.4.2.4.1 Push Button 133
1.3.4.2.4.2 Scroll Bars 138
1.3.4.3 Menu-Objekte 140
1.3.4.3.1 Menu unter Verwendung der Options Bar selbst erstellen 146
1.3.4.3.2 Menü unter Verwendung des Menügenerators erstellen lassen 147
1.3.4.3.3 Menü benutzen 151

2 ENTWICKELN VON ANWENDUNGEN 155
2.1 Entwickeln einer einfachen Anwendung mit den Daten einer Tabelle 155
2.1.1 Anbinden einer Anwendung an eine Datenbank 155
2.1.2 Einzelsatzmodus 157
2.1.2.1 Aufnehmen von Data Fields in frmMain 158
2.1.2.2 Anzeigen von Datensätzen 160
2.1.2.3 Blättern 164

2.1.2.4 Editieren von Datensätzen 167
2.1.2.4.1 Ändern von Datensätzen 167
2.1.2.4.2 Löschen von Datensätzen 172
2.1.2.4.3 Einfügen von Datensätzen 173
2.1.3 Tabellenmodus 175
2.1.3.1 Tabelle in frmMain einbinden 176
2.1.3.2 Anzeigen von Datensätzen 178
2.1.3.3 Editieren von Datensätzen 180
2.1.3.3.1 Eingeben von Datensätzen 181
2.1.3.3.2 Ändern von Datensätzen 186
2.1.3.3.3 Löschen von Datensätzen 188
2.1.4 Ausgeben von Datensätzen mit einem Report 190
2.1.4.1 Daten von der Platte lesen und anzeigen 191
2.1.4.1.1 Template anlegen 191
2.1.4.1.2 Anzeige in der Seitenvorschau 194
2.1.4.1.3 Drucken 201
2.1.4.2 Daten aus einer angezeigten Tabelle drucken 203
2.1.4.2.1 Template anlegen lassen 205
2.1.4.2.2 Anzeige in der Seitenvorschau 205
2.1.4.2.3 Bericht drucken 206
2.1.5 Abfragen von Daten 207
2.2 Entwickeln von Master-Detail-Anwendungen 213
2.2.1 Beziehung zwischen zwei Tabellen 213
2.2.2 Auftragserfassung 224

3 VERTIEFUNG 232
3.1 OOP (Object Oriented Programming) 232
3.2 Arbeiten im Netz 241
3.2.1 Die Funktion SqlSetIsolationLevel() 242
3.2.2 Commit, Rollback 243
3.2.3 Beispiel 247
3.2.3.1 Stammdaten aus Netz2.app 247
3.2.3.2 Bewegungsdaten 249

4 ANHANG 250
4.1 Datenbank "FIRMA", Struktur und Datensätze der verwendeten Tabellen. 250
4.1.1 Kundentabelle 251
4.1.2 Lagertabelle 252
4.1.3 Lieferantentabelle 253
4.1.4 Auftragstabelle 254
4.1.5 Kontrolltabelle 255
4.2 Begriffe 256
4.3 SQLTalk und SQL 261
4.4 Funktionen von SQLWindows 265

4.5 Nachrichten von SQLWindows.....281
4.6 Beispiel zu Datenimport.....285

5 STICHWORTVERZEICHNIS 293

Vorwort

Dieses Handbuch ist ein einführendes Werk in die Programmierung von Datenbankanwendungen mit dem Programmpaket SQLWindows. Es richtet sich dabei an den Personenkreis, der bereits über gewisse Erfahrungen im Umgang mit der Programmierung verfügt, ohne jedoch spezielles Wissen vorauszusetzen. In gewisser Hinsicht können auch fortgeschrittene Leser von diesem Buch profitieren, da alle Objekte von SQLWindows in ihrer Funktion und Verwendung umfassend dargestellt sind.

Neben dem einführenden Kapitel 1, das grundlegende Informationen enthält, werden in Kapitel 2 einfache Anwendungen erstellt. Ziel beider Teile ist dabei das möglichst schnelle Verstehen von SQLWindows, so daß in kurzer Zeit einfache Anwendungen selbst geschrieben werden können. Um dieses Ziel in kurzer Zeit zu erreichen, können nicht alle in SQLWindows enthaltenen Funktionen vorgestellt werden. Für die in diesem Handbuch nicht beschriebenen Funktionen steht Ihnen die von SQLWindows mitgelieferte Literatur zur Verfügung, auf die Bezug genommen wird.

In Kapitel 3 werden die in den vorangegangenen Kapiteln erworbenen Kenntnisse erweitert und vertieft; in Kapitel 4 findet sich der Anhang, der allgemeine Informationen zur Sprache SQL, zu SQLTalk, zu den verwendeten Tabellen und zu Bestandteilen von SAL, der Programmiersprache in SQLWindows, enthält.

Zur Vertiefung des durch dieses Handbuchs erworbenen Wissens sollten Sie einige der im Lieferumfang von SQLWindows enthaltenen Bücher durcharbeiten:

- SQLWindows, Developer´s Reference sowie
- SQLWindows, Function Reference.

Dieses Buch kann unter folgender Adresse bestellt werden:

Dr. Norbert Kessel
Eifelweg 37
53424 Remagen-Oberwinter
Fax: 02228 - 493

1 Allgemeiner Teil

Im allgemeinen Teil werden die Grundlagen der Programmierung von Datenbankanwendungen und die Sprache SQL vorgestellt. Die Vorstellung der Sprache SQL geschieht unter Verwendung des Programms SQLTalk, das im Lieferumfang von SQLWindows enthalten ist und bei der Installation (die hier nicht beschrieben wird) im Verzeichnis \Gupta\ zu finden ist. Falls Sie weitergehende Informationen zur Sprache SQL wünschen, dann finden Sie diese in den mitgelieferten Handbüchern:

- SQLTalk / Windows User´s Guide und
- SQLBase / SQLTalk Language Reference Manual.

Schließlich wird in diesem Kapitel eine Einführung zu SQLWindows gegeben. Diese Einführung ist in Verbindung mit den in Kapitel 2 zu erzeugenden Applikationen zu sehen, da die dort erzeugten Objekte, die für eine Windows-Anwendung typisch sind (Push Buttons, List Box usw.), in die Applikationen eingebaut werden.

1.1 Einführung in die Programmierung von Datenbankanwendungen

Allgemeines

Die diesem Buche zugrunde liegenden Daten sind in einer Datenbank mit dem Namen "FIRMA" abgespeichert. Die in dieser Datenbank enthaltenen Tabellen haben die Namen

- KUNDEN,
- LAGER,
- LIEFERANT,
- AUFTRAG und
- KONTROLL.

In den Tabellen "KUNDEN", "LAGER", "LIEFERANT" werden **Stammdaten** abgespeichert, die Tabelle AUFTRAG enthält die einzelnen Positionen einer Buchung (**Bewegungsdaten**), die Tabelle KONTROLL faßt jeden Auftrag kurz zusammen. Alle diese Tabellen dienen dazu, wichtige Sachverhalte zu erfassen und grundsätzliche Überlegungen zu demonstrieren. Deshalb sind diese Tabellen nicht "vollständig". So ist es durchaus denkbar, daß eine Kundentabelle mehr als 40 Spalten enthält, während die hier verwendete lediglich aus acht Spalten besteht. Auch die Breite der Spalten ist in der Regel hier für Praxiszwecke zu kurz gewählt. Dieser Sachverhalt muß bei der Übertragung der Beispiele auf die Arbeit in der Praxis beachtet werden.

Stammdaten, Bewegungsdaten

Die Begriffe Stammdaten und Bewegungsdaten gehören zu den wichtigsten Begriffen im Zusammenhang mit der Programmierung von Datenbankanwendungen.

Als **Stammdaten** werden in der EDV Daten bezeichnet, deren Inhalt und Umfang sich nicht wesentlich ändert. Dies können z.B. Personaldaten eines Unternehmens sein, die, nachdem sie einmal angelegt wurden, nur unwesentlich modifiziert werden. Beispiele für Stammdaten sind z.B.: Lagerdaten, Lieferantendaten, Herstellerdaten, Inventardaten.

Die folgende Tabelle zeigt Stammdaten. Es handelt sich dabei um Artikel, die im Lager vorhanden sind.

Bezeichnung	Hersteller	Ist-Bestand	Einkaufspreis
Hammer	100973	12	14,80
Kelle	552120	20	11,98

Obwohl sich Teile der Information ändern können (z.B. der Ist-Bestand nach einem Verkauf, der Einkaufspreis nach einer Preiserhöhung), handelt es sich dennoch gemäß der oben stehenden Definition um Stammdaten.

Beachten Sie, daß unter Hersteller nicht der Name des Herstellers (z.B. Fa. Meier & Müller) eingetragen wurde, sondern eine Nummer. In einer zusätzlichen zweiten (hier nicht dargestellten) Tabelle wäre noch zu definieren, welcher Hersteller sich hinter welcher Nummer verbirgt (mehr dazu finden Sie in dem Kapitel "Normalisierung von Daten).

Unter **Bewegungsdaten** versteht man Daten, die dadurch entstehen, daß etwas "bewegt" wird (z.B. durch den Verkauf eines Artikels). Der Umfang dieser Daten ändert sich häufig, in der Regel bestehen diese Änderungen darin, daß neue Datensätze an die bereits vorhandenen Datensätze angehängt werden.

Wenn Sie den folgenden Ausschnitt aus einer Rechnung betrachten, dann werden Sie feststellen, daß die Bewegungsdaten Informationen zu verkauften Artikeln beinhalten. Dabei ist es unerheblich, ob es sich um die Rechnung eines Supermarktes, einer Kfz-Werkstatt oder eines Arztes handelt.

Artikel Nr.	Bezeichnung	Menge	Preis
1000	Hammer	1	9,00
1001	Kelle	1	7,80
1002	Eimer	1	5,30

Im Sinn der vorstehenden Definition handelt es sich hierbei um Bewegungsdaten. Insgesamt sind 3 Datensätze dargestellt.

Gespeichert werden aber nicht alle angezeigten Informationen, nur die Artikelnummer und die abgegebene Stückzahl (sowie die hier nicht erkennbare Kundennummer des Kunden) werden auf den Datenträger geschrieben. Auch die Gesamtsumme, üblicherweise unten, am Ende der Rechnung zu sehen, wird in der Regel nicht gespeichert, da sie sich jederzeit neu aus den vorhandenen gespeicherten Daten errechnen läßt.

Für die spätere Programmierung von Datenbankanwendungen ist das Verständnis der Unterscheidung von Stamm- und Bewegungsdaten von größter Bedeutung. Aus diesem Grund sollten Sie vor Erstellung eines Programms zunächst versuchen, Daten gemäß der vorstehenden Definition "richtig" einzuordnen.

Normalisierung von Daten

Unter der Normalisierung von Daten versteht man in erster Linie das Entfernen von sich wiederholender Information. So wird, um auf die vorstehende Bewegungstabelle Bezug zu nehmen, bei der Erfassung eines Artikels nicht der Name des Lieferanten gespeichert, sondern nur eine Nummer. Welcher Lieferant dieser Nummer zugeordnet ist, ergibt sich aus einer zweiten Tabelle, in der alle Informationen zu Lieferanten (einmalig) erfaßt sind.

Das folgende Beispiel zeigt, wie Daten *nicht* gespeichert werden sollten.

Bezeichnung	Bestand	EK-Preis	Hersteller	Ort	Telefon
Hammer	12	14,80	Meier & Meier	Karlsruhe	0721-1123
Kelle	20	11,98	Kellermann	München	089-22345

Die Informationen zum Hersteller sind zwar sehr ausführlich, verursachen in der Praxis aber einen unnötigen Aufwand: Werden von einem Hersteller mehrere Artikel bezogen, muß die gleiche, bereits vorhandene, Information nochmals eingegeben werden, so wie das die folgende Übersicht zeigt.

Bezeichnung	Bestand	EK-Preis	Hersteller	Ort	Telefon
Hammer	12	14,80	Meier & Meier	Karlsruhe	0721-1123
Kelle	20	11,98	Kellermann	München	089-22345
Axt	14	23,56	Meier & Meier	Karlsruhe	0721-1123

Hierdurch wird nicht nur unnötig Zeit verschenkt, die Daten werden auch unsicherer, da mit großer Wahrscheinlichkeit irgendwann ein Eingabefehler unterlaufen kann; bei zwei verschiedenen Telefonnummern ist dann nicht klar, welche die richtige ist. Deshalb muß bei Datenbanksystemen darauf geachtet werden, daß die Daten bei der Speicherung in normalisierter Form vorliegen, d.h. daß keine Wiederholungen (redundante Daten) auftreten.

Es ist bei relationalen Datenbanken - so wie in der vorliegenden Datenbank SQLBase - nicht möglich **alle** redundanten Daten zu vermeiden, Ziel ist vielmehr die Minimierung der Redundanz.

Wird die Minimierung der Redundanz konsequent weiter gedacht, werden Sie auf die Idee kommen, daß bei zwei verschiedenen Lieferanten, die aus dem gleichen Ort kommen, der Ortsname doppelt vorhanden ist. Durch die Benutzung einer eindeutigen Identifikationsnummer für den Ort (hierfür wäre z.B. die Postleitzahl zu gebrauchen) würde Redundanz verhindert werden. Hier wird aber häufig nicht weiter "normalisiert", d.h. ab einem gewissen Grad werden Kompromisse eingegangen (hier z.B. aus dem Grund, daß der Aufwand zur Verwaltung der Ortsnamen, der durch die zusätzliche Tabelle entsteht, zu hoch wird).

Um Daten zu normalisieren, wurden fünf Normalformen definiert, von denen üblicherweise aber nur die ersten drei realisiert werden. Der Grund hierfür ist in der bereits oben erwähnten Eigenschaft zu suchen, daß vollständig normalisierte Daten zu einer großen Zahl von Tabellen führen.

Zum Verständnis der Klassifizierung muß der Begriff **Schlüssel**, dessen Erläuterung erst später folgt, bereits hier benutzt werden: Ein Schlüssel ist ein eindeutiges Merkmal eines Datensatzes, das gewährleistet, daß der Datensatz eindeutig gekennzeichnet ist (z.B. eine Artikelnummer, eine Kundennummer, eine Lieferantennummer

Von den fünf Regeln die zur Erreichung der Normalformen bekannt sind, sollen hier nur die ersten drei Regeln genannt werden. Ausführliche Erläuterungen sind aus der Standardliteratur (z.B. CODD, DATE) zu entnehmen.

- **Erste Normalform**: Kein Attribut darf mehrfach in einer Tabelle vorhanden sein. Dies bedeutet, daß Wiederholungen (z.B. des Lieferantennamens) verhindert werden sollen.

- **Zweite Normalform**: Die von einfachen Schlüsseln abhängigen Informationen werden in einer Tabelle zusammengefaßt, die von mehreren Schlüsseln abhängigen Informationen werden ebenso in einer separaten Tabelle abgelegt. Dies bedeutet, daß bei einer Auftragsverwaltung neben den Stammdaten (Kunden- und Lagerdaten) eine zusätzliche Tabelle einzig und allein für die Details der Aufträge anzulegen ist; diese Tabelle ist im Sinne der vorhergehenden Definition eine Bewegungstabelle (s. hierzu die Tabelle Auftrag im Anhang).

- **Dritte Normalform**: Felder (Informationen), die nicht Schlüsselfeld sind, müssen direkt vom Schlüssel abhängen. Sie dürfen nicht von einer anderen Information abhängen. Das bedeutet, daß z.B. der Ortsname nur von der Postleitzahl, nicht aber von dem Namen oder der Kundennummer des Kunden abhängig sein darf. Gemäß dieser Anforderung müßte also eine zusätzliche Tabelle für die Orte angelegt werden (was aber häufig nicht getan wird).

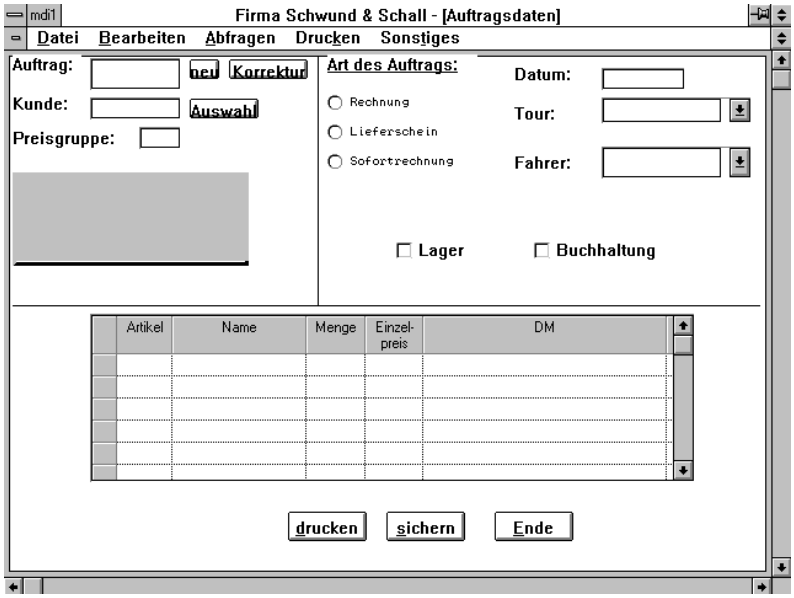
Master-Detail Beziehungen

Der Begriff Master-Detail wird im Zusammenhang mit relationalen Datenbanksystemen häufig gebraucht. Er steht dafür, daß Informationen aus verschiedenen Tabellen über sogenannte Relationen zusammengeführt werden. Dabei verbinden diese Relationen sinnvollerweise zwei Felder gleichen Inhalts miteinander. Damit kann etwa, obwohl z.B. bei einer Buchung nur die Kundennummer erfaßt wurde, der Name und die Adresse des Kunden jederzeit angezeigt werden. In dem genannten Fall wäre die Kundentabelle die **Master-Tabelle** und die Buchungstabelle die **Detail-Tabelle**.

Zwar ist es grundsätzlich möglich, auch ohne Master-Detail-Beziehungen zu arbeiten, aber ein effektiver Einsatz eines Datenbanksystems setzt die sinnvolle Berücksichtigung von Master-Detail-Beziehungen voraus.

In der folgenden Abbildung 1.1 sind oben links Informationen zu dem Kunden (aus der Master-Tabelle) und unten Informationen zu den verkauften Artikeln (aus einer Detail-Tabelle) angezeigt. Oben rechts werden zusätzlich die Daten aus einer anderen Tabelle angezeigt, die zusammenfassend für jeden Auftrag Informationen zur Art des Auftrags, dem Datum der Auftragserfassung usw. anzeigen. Diese Informationen werden weder in der Kunden- noch in der Auftragstabelle gespeichert, sondern in einer zusätzlichen Tabelle (Kontrolltabelle, s. Anhang). Diese Verbindung von drei Tabellen über gemeinsame Schlüsselfelder ist durchaus nichts ungewöhnliches.

Abb. 1.1: Master-Detail, oben links Informationen zum Kunden (Master-Tabelle), unten die im Auftrag enthaltenen Daten (Detail-Tabelle), oben rechts Daten aus der Kontrolltabelle.



Voraussetzung für eine solche Art der Darstellung ist, daß die zu speichernden Daten entsprechend aufbereitet sind (s. Begriffe Stammdaten, Bewegungsdaten, Normalisierung).

Bei der Arbeit mit Daten, die in mehreren Tabellen vorliegen, werden Sie verschiedene Arten von Beziehungen kennenlernen:

- **1: n**
  Diese Beziehung zweier Tabellen ist die am häufigsten anzutreffende. 1 bedeutet hierbei, daß einem Datensatz aus der Mastertabelle mehrere Sätze aus der Detailtabelle entsprechen. Dies tritt z.B. dann auf, wenn einem Lieferanten aus der Lieferantentabelle mehrere Datensätze aus der Artikeltabelle zugeordnet werden können (die Zuordnung findet dabei über das Feld mit der Lieferantennummer statt).

- **m : n**
  Mehreren Sätzen aus einer ersten Tabelle können mehrere Sätze aus einer anderen Tabelle zugeordnet werden. Dies stellt häufig die Ausgangssituation vor einer Normalisierung der Daten dar. Typischerweise werden die Daten normalisiert, was zur Folge hat, daß zunächst eine oder mehrere Tabellen anzulegen sind, mittels derer die Daten in die Normalform zu überführen sind.

- **1 : 1**
  Einem Datensatz aus der ersten Tabelle kann genau ein Datensatz aus einer anderen Tabelle zugeordnet werden. Diese Situation tritt relativ selten auf, da Sie üblicherweise die Information genauso

in einer einzigen Tabelle unterbringen können. Wahrscheinlicher ist es, daß Sie aus einer vorhandenen Tabelle (z.B. der Kundentabelle) eine identische Tabelle erzeugen, in der dann ein Feld aufgenommen wird, das in der ursprünglichen Tabelle nicht erscheint (z.B. der Hinweis, ob ein bestimmter Betrag für das Betriebsjubiläum des Kollegen bereits entrichtet wurde).

Techniken der Programmierung (Top-Down, Bottom-Up)

Ein Computerprogramm besteht üblicherweise aus vielen einzelnen Modulen. Dabei finden sich Module (wie z.B. das Menü des Programms), die eher allgemein zu halten sind und die sich zwischen verschiedenen Programmen auch nicht sehr unterscheiden. Daneben gibt es sehr spezielle Module, die nur zur Lösung eines speziellen Problems beitragen. Zur Entwicklung dieser häufig nicht auf andere Programme übertragbaren Module wird oft viel Zeit benötigt. Bei der Erstellung von Computerprogrammen ist zunächst zu entscheiden, ob Sie eher von allgemeinen Modulen zu den speziellen vorgehen, oder ob Sie umgekehrt vorgehen, das heißt zuerst die speziellen Module und danach die eher allgemeinen Programmteile entwickeln.

Unter einer **Top-Down** Programmierung versteht man die Entwicklung eines Programms von oben nach unten, das heißt vom allgemeinen in das besondere. Das bedeutet, daß zunächst allgemeine Programm-Module erstellt werden, die nach und nach durch immer speziellere Module ergänzt und verfeinert werden. Die zu lösenden (Programmier-) Probleme werden dabei in immer kleinere Einheiten zerlegt, die dann mit relativ geringerem Aufwand letztendlich gelöst werden. Dadurch, daß man die Lösungen der Einzelprobleme zusammensetzt, erhält man schließlich die Lösung des Gesamtproblems. Von Vorteil ist bei dieser Technik, daß Sie unmittelbar nach Beginn der Programmierung Ergebnisse sehen, außerdem entspricht diese Art der Arbeit der "normalen Sicht" der Dinge.

Die **Bottom-Up** Programmiertechnik beginnt genau am anderen Ende: Zunächst werden die Einzelmodule (z.B. die Druckausgabe eines Lieferschein-Formulars) programmiert, nach und nach werden die allgemeineren Teile um die speziellen Module herum entwickelt. Die Entwicklung beginnt also "unten". Von Nachteil ist hierbei, daß Module einer übergeordneten Ebene erst entwickelt werden können, wenn die der untergeordneten fertiggestellt sind. Von Vorteil dagegen ist, daß Fehler (z.B. fehlende Spalten einer Tabelle) im Entwurfsstadium bereits sichtbar werden, weshalb die Korrektur oft einfacher ist.

Insgesamt könnte man sagen, daß die Bottom-Up Methode Sie dazu zwingt, die genauen Konzepte zu verstehen, *bevor* Sie beginnen zu programmieren. Die Top-Down Methode dagegen verschafft Ihnen schnelle Erfolgserlebnisse führt aber gelegentlich dazu, daß Sie umfangreiche Module nachträglich an spezielle Verhältnisse anpassen müssen, was oft einen beträchtlichen Aufwand verursacht.

Natürlich ist es Ihre Sache, wie Sie Ihre Programme entwerfen und erstellen; ob Sie sich nun für Top-Down oder Bottom-Up entscheiden ist dabei letztlich unerheblich. Besonders wichtig dagegen ist, daß Sie sich zunächst die theoretische Seite des Problems verdeutlichen. Bevor Sie mit der Programmierung beginnen, sollten Sie mit dem späteren Anwender mittels Skizzen, Schaubildern und Abbildungen die grobe Richtung des Programms festlegen.

Häufig werden beide Vorgehensweisen - die Top-Down und die Bottom-Up Methode - in der Form vermischt, daß Sie sich zuerst einen Eindruck verschaffen, welche grundsätzlichen Module zu programmieren ist. Dann

beginnen Sie einige wenige Module zu entwickeln und begeben sich damit in die Tiefe. Das Ergebnis könnte man als frühen Prototypen bezeichnen, den Sie dem Kunden vorstellen. Dieser wird dann oft genug Wünsche äußern, die Ihren Entwurf mehr oder weniger stark umwerfen. Abhängig davon ändern Sie Ihre Module von unten nach oben bzw. von oben nach unten so ab, daß sich der Prototyp und die Wünsche des Kunden mehr und mehr decken.

Beachten Sie dabei, daß Ihnen eine gute Optik zwar oft einen Vertrauensvorschuß verschafft, daß aber viele Benutzer unbestechlich sind: Solange das Programm nicht genau das macht, was sie erwarten, bleiben sie unzufrieden!

Einen Aspekt sollten Sie schließlich nicht vernachlässigen: Schreiben Sie Ihre Programme so, daß Sie auch nach einigen Wochen noch verstehen. Dazu trägt erfahrungsgemäß nicht nur eine sorgfältige Kommentierung bei, sondern auch eine gewisse Modularität, d.h. die Aufteilung von umfangreichen Programmteilen zu logischen Teilen. Das heißt, schreiben Sie Ihre Module so, daß dabei stabile, kleine, überschaubare und in sich geschlossene Programmteile entstehen: Um z.B. bei einer Auftragserfassung den Preis eines Artikels zu erhalten, sollten Sie eine Funktion schreiben, die auch genau das tut. Die Funktion sollte zunächst prüfen, ob ein Sonderpreis für diesen Kunden und diesen Artikel bestimmt wurde, falls nein, sollte die Funktion abhängig von der dem Kunden zugeordneten Preisgruppe, den Preis des Artikels wählen. Jedesmal, wenn ein Preis benötigt wird, wird diese Funktion dann benutzt. Nur auf diese Weise können Sie in vertretbarer Zeit Änderungen in Ihre Programme einfügen.

Unterschiede von herkömmlichen (DOS-Programmen) zu Windows-Programmen

Grundsätzlich lassen sich die Schritte bei Abarbeitung eines Programmes durch einen Computer mit folgenden drei Begriffen beschreiben:

Eingabe - Verarbeitung - Ausgabe

In DOS-basierenden Anwendungen ist es außerdem so, daß der Programmierer dem Benutzer den Weg durch das Programm mehr oder weniger vorschreibt. In Windows-Anwendungen dagegen, kann der Programmierer dagegen nicht im voraus wissen, was der Benutzer als nächstes macht: Drückt er einen Push Button, wählt er einen Menüpunkt aus, wechselt er in ein anderes Fenster oder beendet er das Programm durch Drücken von Alt+F4.

Genaugenommen haben Sie zwar noch immer die Abfolge "Eingabe-Verarbeitung-Ausgabe", doch finden Sie diesen Mechanismus nun bei jedem Objekt, das auf dem Bildschirm erscheint: Jeder Push Button, jeder Menüpunkt, jedes Fenster, jede Check Box und jeder Radio Button sind in der Lage, auf "Eingaben" zu reagieren.

Man spricht bei Windows-Programmen von "Ereignissen". Dabei ist ein Ereignis häufig ein Mausklick, z.B. auf einem Push Button. Sinngemäß werden deshalb jedem der verwendeten Objekte (Push Buttons, Check Boxen usw.) eigene Programmteile zugeordnet, in denen die Reaktion auf ein bestimmtes Ereignis vorbestimmt wird. Einige Begriffe sind in diesem Zusammenhang so wichtig, daß sie kurz erläutert werden sollen:

- **Ereignisorientierte Programmierung (Event-Driven Programming)**: Dies bedeutet, daß in Erweiterung der klassischen Programmierung andere Regeln für den Ablauf von Programmen unter Windows gelten. Der Ablauf eines Programms gestaltet sich so, daß auf bestimmte Ereignisse reagiert wird. Diese Ereignisse, die von der Benutzung verschiedener Objekte (z.B. Push Buttons, s.u.) abhängen, werden dabei mittels Nachrichten (s.u.) verwaltet.

- **Objekte (Objects)**: Teil eines Programms, das Nachrichten (Messages) erhält. So kann z.B. ein Push Button die Nachricht erhalten, daß er gerade gedrückt wurde. Dieser Nachricht, codiert mit dem Begriff "On SAM_Click" ("SAM" ist die Abkürzung für SQLWindows Application Message) werden dann Programmschritte zugeordnet, so z.B. die Anweisung, ein Fenster zu schließen oder einen Ausdruck zu starten.

- **Nachrichten (Messages)**: Nachrichten werden innerhalb von SQLWindows-Anwendungen verschickt, um über bestimmte Programmschritte oder vorgenommene Aktionen zu informieren. So kann unter Verwendung der Nachricht "SAM_Click" auf einfache Weise ein Bezug zwischen einem Push Button und der Aktion des Benutzers, nämlich dem Klicken auf eben diesen Push Button, Bezug genommen werden. Neben den vorgefertigten Nachrichten, die mit der Zeichenfolge "SAM" beginnen (eine Auflistung findet sich im Anhang), besteht auch die Möglichkeit eigene Nachrichten zu erzeugen und zu verwalten. Die Verwendung dieser Nachrichten, die z.B. mit der Zeichenfolge "AM" beginnen könnten (das ist beliebig wählbar, sollte jedoch immer unterscheidbar sein von SAM-Nachrichten), erlaubt es flexiblere Programme zu schreiben.

Dieser Sachverhalt hat für den Programmierer große Bedeutung: Programme unter Windows können dem Benutzer nun nicht mehr starr vorgeben, was er tun oder lassen kann, so wie das mit normalen DOS-Programmen möglich und auch normal war. Jetzt muß ein Programm so erstellt werden, daß jederzeit auf jede mögliche Benutzereingabe reagiert werden kann. Der Programmierer muß im voraus dafür sorgen, daß alle möglichen Eingaben des Benutzers richtig aufgenommen und verarbeitet werden.

Während dies bei Verwendung von Programmiersprachen wie z.B. "C" oder "Pascal" nicht unbedingt einfach ist, ist SQLWindows dafür schon vorbereitet, das heißt die Fähigkeit zur Reaktion auf vielfältige Eingaben ist bereits in SQLWindows enthalten. Der Programmierer prüft z.B. nur noch, ob der Benutzer in einem zuvor aktivierten Fenster den OK Push Button gedrückt hat. Da der dann auszuführende Programmcode an diesen Push Button angebunden ist, kann die Aktion des Benutzers korrekt weiter verarbeitet werden.

Vorschlag zum Aufbau eines Menüs

Selbstverständlich können Sie das Menü eines Computerprogramms so aufbauen wie es Ihren Wünschen entspricht. Hinzu kommen dann noch die Wünsche des Benutzers: Problematisch hierbei ist allerdings, daß eine klare Benutzeroberfläche dabei aber häufig nicht entsteht. Was dem einen wichtig und ohne großen Aufwand erreichbar sein soll, ist für den anderen eine Nebensächlichkeit.

Ein anderer Ansatz wird hier vorgeschlagen: Orientieren Sie sich an der Oberfläche von Windows-Programmen und ergänzen Sie die von Windows bekannten Menüpunkte um solche, die geeignet sind, das Programm in logische Teileinheiten zu trennen. Durch die Anlehnung an Windows schaffen Sie für jeden Benutzer, der sich auch nur ein wenig in Windows auskennt, ein Gefühl der Vertrautheit. Allein das Wissen,

an welcher Stelle des Menüs man eine Anwendung beenden kann (nämlich im Menüpunkt "Datei"), ist schon sehr wichtig: Es gibt genügend Programme, die den Benutzer nur schlecht unterstützen.

Sie sollten bei der Programmierung von Menüsystemen immer mit einer Befragung des Benutzers beginnen. Sinnvollerweise zeichnen Sie dann einen Entwurf auf einem Blatt, viele Widersprüche klären sich dabei schon auf. Versuchen Sie herauszufinden, aus welchen *grundsätzlichen* Modulen das Programm bestehen soll (z.B. Eingabe-, Ausgabeteile), skizzieren Sie diese und versuchen Sie, die Wünsche des späteren Benutzers umzusetzen, ohne dabei logische Fehler zu übernehmen, die oft durch die zu schnelle Übernahme dessen Wünsche in ein Programm entstehen. Gehen Sie dabei aber Kompromisse ein: Ein Menüpunkt "Druck" ist durchaus sinnvoll, das sollte jedoch nicht dazu führen, daß überhaupt nur über diesen Menüpunkt gedruckt werden kann. Vorstellbar ist, daß die Aktion zum Druck einer solchen Liste zusätzlich in dem Tabellenfenster möglich ist, in der die Lagerdaten angezeigt werden, oder in einem anderen Formular mit dem, der Benutzer vorwiegend arbeitet (hierzu werden dann typischerweise keine zusätzlichen Menüpunkte sondern Push Buttons eingerichtet).

Ein Menü für alle Fälle existiert nicht, weshalb das folgende Menü nur als Vorschlag zu verstehen ist:

Menü-Vorschlag für SQLWindows-Programme.

Datei	Bearbeiten	Abfragen	Berichte	Sonst.	?
Kunden	Ausschneiden	Kunden	Kundenlisten	Export	Hilfe
Lager	Kopieren	Lager	Lagerlisten	Import	Info
Lieferanten	Einfügen	Lieferanten	Lieferanten		
Auftrag		Auftrag	Aufträge		
Kontroll		Kontroll	Kontrolle		
Ende					

In dem Menüpunkt **Datei** finden sich alle Namen von Dateien, die von oder in der Anwendung benutzt werden (genaugenommen handelt es sich natürlich um **Tabellen** ohne Dateinamen, da wir es hier mit einem Datenbankprogramm zu tun haben). Eine weitere Unterteilung innerhalb des Menüpunktes könnte nach Stammdaten (Kunden, Lager, Lieferanten) und Bewegungsdaten (Auftrag, Kontrolle) trennen. Eine Option zum Beenden des Programms sollte hier immer zu finden sein.

Unter dem Menüpunkt **Bearbeiten** werden die unter Windows üblichen Werkzeuge plaziert: Ausschneiden, Einfügen, Herstellen einer DDE-Verbindung (z.B. um Daten an Excel zu übergeben), Zurücknehmen des letzten Befehls usw. Hier sollten Sie sich sehr genau an die Vorlage halten, die Windows selbst vorgibt, da diese Begriffe üblicherweise in allen Programmen unter Windows gebraucht werden.

Der Menüpunkt **Abfrage** könnte dazu genutzt werden, Daten auszuwählen, um diese dann weiter auszuwerten: Offene Rechnungen, Lagerbestände die unter den Soll-Bestand abgesunken sind, aber auch eine Kundenauswahl für eine Mailing-Aktion könnten hier beispielsweise untergebracht werden.

In dem Menüpunkt **Berichte**könnten alle Module untergebracht werden, die zu einer Druckausgabe führen. Natürlich könnte dies auch mit einer Abfrage kombiniert werden, damit nicht sämtliche Daten gedruckt werden, sondern nur die gewünschten.

In dem Menüpunkt **Sonstiges** könnte all das untergebracht werden, was inhaltlich weniger gut unter die anderen Menüpunkten gefaßt werden kann. Im Beispiel wurde der Export und Import von Daten hier eingetragen. Es könnte auch der Start von Fremdanwendungen (z.B. Excel) auftauchen. Gegebenenfalls müssen Sie entscheiden, ob eine Option, die Sie hier eingebaut haben, eventuell so wichtig ist, daß ein separater Menüpunkt daraus anzufertigen wäre, bei dem vorliegenden Beispiel ist dies für den Datenimport bzw. Datenexport vorstellbar.

Der mit dem Fragezeichen **?** bezeichnete Menüpunkt könnte für den Benutzer Hilfe in Form von Instruktionen bereithalten. Schließlich findet man bei vielen Programmen häufig einen Hinweis auf den Programmierer.

Falls Sie Ihre Programme klar gliedern, wird auch die Arbeit des Benutzers klar gegliedert, was zur Folge hat, daß auch andere Mitarbeiter, z.B. eine Urlaubsvertretung, mit der Anwendung schnell zurecht kommen.

1.2 Einführung in die Sprache SQL unter Verwendung von SQLTalk

Dieses Kapitel beschäftigt sich mit zwei Dingen, nämlich der Sprache SQL sowie dem Programm SQLTalk, das im Lieferumfang von SQLWindows enthalten ist und das die Verwendung dieser leistungsfähigen Sprache erlaubt. Zunächst findet eine Einführung in die Sprache SQL statt, danach werden kommentierte Beispiele die Verwendung dieser Sprache in SQLTalk erläutern. Hierbei werden Sie eine Datenbank sowie mehrere Tabellen anlegen, die in den späteren Kapiteln dieses Buchs gebraucht werden.

1.2.1 Einführung in die Sprache SQL

1.2.1.1 Geschichte von SQL

Die Ursprünge der Sprache SQL (Structured Query Language) sind in den 70er Jahren zu sehen. Die Namen IBM und E. F. CODD, einem bei IBM beschäftigten Mathematiker, sind damit verbunden. Ausgehend von einem relationalen Datenmodell sollte eine Sprache entwickelt werden, die - mit Elementen der Umgangssprache ausgestattet - einen möglichst einfachen Zugriff auf Datenbanken ermöglichen sollte. Ursprünglich mit dem Namen SEQUEL bezeichnet, wurde seit ca. 1980 die Abkürzung SQL (von Insidern oft als "sikwel" ausgesprochen) gebräuchlich.

Die Sprache wurde von mehreren Behörden als Standard-Interface für relationale Datenbanken anerkannt und auch standardisiert.

Die Vorteile der Sprache SQL liegen in folgenden Punkten:

- Die Sprache ist standardisiert und ist auf vielen Hardware-Plattformen verfügbar.

- SQL ist leistungsfähig und kann auch für sehr komplexe Auswertungen verwendet werden.
- SQL ist einfach zu erlernen.

1.2.1.2 Anmerkungen zur Sprache SQL

Dadurch, daß der Kern der Sprache SQL aus nur ca. 30 Anweisungen besteht, ist es möglich, die Sprache in sehr kurzer Zeit zu erlernen. Neben dem kleinen Wortschatz liegt ein weiterer Vorteil der Sprache darin, daß dieser Wortschatz aus Worten besteht, die aus der Umgangssprache entlehnt sind: Begriffe wie select, update, delete etc. sind selbst Benutzern mit nur rudimentären Englischkenntnissen leicht eingängig.

Ebenfalls eine Folge des kleinen Wortschatzes von SQL ist der Umstand, daß es mit relativ wenig Aufwand möglich ist, eine Schnittstelle zwischen dem Benutzer (am Front-End) und der Datenbank (als Back-End) so zu programmieren, daß die Abfrage oder Neu-Eingabe von Daten mittels einer SQL-Anweisung ohne große Mühe möglich ist. Nicht zu vergessen ist dabei allerdings, daß die eigentliche Arbeit dabei von den Programmierern geleistet wurde, die diese Datenbank programmiert haben, denn selbstverständlich versteht eine Datenbank zunächst nichts von SQL. Erst durch ein Datenbank-Managment-System (DBMS, Database Management System), ein separates, auf dem Server installiertes Programm, wird es möglich eine solchermaßen einfach zu bedienende und trotzdem extrem leistungsfähige Datenbanksprache wie SQL zu benutzen.

Allerdings kann SQL niemals ein Ersatz für Sprachen wie beispielsweise "C" oder "Pascal" sein, da erst diese Sprache z.B. einen Zugriff auf die Hardware des Computers erlauben. Allerdings kann SQL als Teil dieser Sprache benutzt werden und somit die größere Leistungsfähigkeit einer Sprache wie C zusätzlich um die Datenbank-Fähigkeiten von SQL zu erweitern. Diese Art der Benutzung wird als "embedded SQL" bezeichnet und von vielen Programmierern angewendet.

Es gibt aber auch Nachteile der Sprache SQL, die unter anderem aus dem kleinen Sprachschatz der Sprache SQL herrühren. So ist z.B. folgendes nicht möglich:

- Strukturierte Programmierung: Die aus anderen Sprachen bekannten Möglichkeiten von Schleifen, If- und Case-/Switch-Anweisungen kennt die Sprache SQL nicht.
- Benutzung von Variablen: Innerhalb von SQL können keine Variablen definiert werden, diese müssen von einer vorhandenen Umgebung bereitgestellt werden.
- Erzeugung von Eingabemasken: Zur Dateneingabe kann nur die Insert-Anweisung mit der Liste von Spaltennamen und der Liste der einzufügenden Spalteninhalte verwendet werden; eine formatierte Eingabe in Form von Eingabemasken kennt SQL nicht, auch hierfür muß eine Umgebung, wie z.B. SQLWindows, benutzt werden.

Darüber hinaus wird die Situation insgesamt dadurch kompliziert, daß es eine Vielzahl von sog. "SQL-Dialekten" gibt, d.h. von Herstellern als wichtig betrachtete Anweisungen, die nicht im eigentlichen Sprachstandard von SQL enthalten sind. Da nun jeder Hersteller von seinen Kommandos denkt, sie seien die

wichtigsten, kann tatsächlich nur der Kern der Sprache als standardisiert gelten. Dies ist aber in Verbindung mit diesem Handbuch nicht von Bedeutung, da die SQL-Anweisungen sehr einfach gehalten sind.

Bei der Übertragung der Beispiele auf andere Datenbanken ist jedoch zu prüfen, ob die SQL-Anweisungen uneingeschränkt übertragbar sind (so ist z.B. der Schrägstrich in SQLTalk, der ein Kommando beendet, nicht unbedingt von anderen Datenbanksystemen als solcher zu erkennen, z.T. werden Semikola (";") oder andere Zeichen erwartet.

1.2.1.3 SQL-Anweisungen

In den folgenden Seiten werden die wichtigsten SQL-Anweisungen in Form von Tabellen erläutert. Dabei sind die Beispiele so aufgebaut, daß in der Tabelle links die SQL-Anweisung und rechts ein Kommentar hierzu dargestellt ist.

Sie können die Anweisungen mit SQLTalk, Quest oder innerhalb von SQLWindows (Menü "View" Option "Database Browser") durchführen lassen (Schrägstrich als Abschluß der Anweisung nicht vergessen). Beachten Sie aber, daß das Ergebnis einer Operation (update, delete usw.) häufig nicht angezeigt wird. Sie können sich die Ergebnisse z.B. nach einer delete-Anweisung aber jederzeit durch Eingabe einer neuen zusätzlichen select-Anweisung anzeigen lassen.

Die Untergliederung des Kapitels orientiert sich an dem üblichen Ablauf der Arbeit mit SQL. Das heißt, es werden zunächst die Befehle erläutert, die in Verbindung mit einer Datenbank wichtig sind, danach diejenigen, die zur Arbeit mit Tabellen notwendig sind und schließlich diejenigen, die zur Arbeit mit Datensätzen erforderlich sind.

Beachten Sie, daß eine SQL-Anweisung und die durch sie angestellten Änderungen in den Daten solange zurückgenommen werden kann, solange keine spätere SQL-Anweisung gegeben wurde. Durch die SQL-Anweisung "rollback" können die ursprünglichen Daten wieder hergestellt werden. Andererseits können Sie, nachdem Sie eine beliebige SQL-Anweisung abgeschickt haben, die durch sie vorgenommenen Änderungen fixieren, indem Sie "commit" ausführen lassen. Außerdem können Sie durch die Verwendung von Transaktionen sicherstellen, daß bei einer Gruppe von Anweisungen entweder alle oder keine der Anweisungen ausgeführt wurde. Ausführliche Hinweise hierzu finden sich im Handbuch zu SQLBase im Lieferumfang von SQLWindows.

1.2.1.3.1 Datenbanken anlegen (create), reorganisieren (reorganize), löschen (drop)

create database test	Legt die Datenbank "test" auf der Festplatte an. Dabei wird zunächst ein Verzeichnis mit dem Namen "test" angelegt, danach wird die im Verzeichnis GUPTA befindliche Datenbank start.dbs unter dem Namen test.dbs in dieses Verzeichnis kopiert.

reorganize	Reorganisiert eine Datenbank: Sorgt dafür, daß die Datenbank und die darin enthaltenen Tabellen "an einem Stück", d.h. unfragmentiert, vorliegen.

drop database test	Löscht die angegebene Datenbank mit den darin enthaltenen Tabellen.

1.2.1.3.2 Tabellen anlegen (create), ändern (alter), löschen (drop), Indexe anlegen (create Index)

create table	Erzeugt eine neue Tabelle, fügt sie der offenen Datenbank hinzu.
create table ausgaben (Datum Timestamp NOT NULL, Grund Varchar(20), Betrag real)	Tabelle "ausgaben" anlegen, die drei Spalten enthält (Datum, Grund, Betrag). Die Spalte "Datum" ist vom Datentyp "timestamp", einen Datums- und Zeitwert. In diese Spalte muß vom Benutzer ein Inhalt eingegeben werden ("NOT NULL"). Die Spalte "Grund" ist vom Typ "varchar", es können verschieden lange Texte eingegeben werden. Die Spalte "Betrag" ist vom Typ "real", in ihr können Dezimalzahlen eingeben werden. Nach Anlage einer Tabelle sind natürlich noch keine Sätze enthalten; Sätze können Sie mit der Anweisung "insert" einfügen lassen.

alter table	Ändert die Struktur einer Tabelle.
alter table personal add gehalt integer	Fügt der Tabelle Personal eine neue Spalte mit dem Namen "gehalt" und dem Datentyp "integer" hinzu.
alter table personal drop gehalt	Löscht aus der Tabelle Personal die Spalte "gehalt".

drop table	Löscht eine Tabelle.
drop table test	Löscht die Tabelle "test" aus der offenen Datenbank. Alle Daten aus der Tabelle sind danach verloren (wenn nicht dieser letzte Befehl mit einer rollback-Anweisung zurückgenommen wird).

create Index ... on...	Legt zu einer Tabelle eine Hilfstabelle an, um die Anwortzeiten bei Abfragen zu verbessern (dient nicht der Sortierung).
create index i_name on gaesteliste(name)	Indexname lautet i_name, indiziert werden soll nach dem Inhalt des Feldes "name" aus der Tabelle "Gaesteliste".
create index isort on gaesteliste(name,zimmer)	Verknüpfter oder Verketteter Index, basierend auf zwei Spalten (die Datentypen dürfen verschieden sein).
drop index isort	Löscht den Index mit dem Namen "isort".

1.2.1.3.3 Tabellen und Datensätze

1.2.1.3.3.1 Datensätze eingeben

insert	Einfügen von Sätzen in eine vorhandene Tabelle. Es müssen dabei Feldnamen und Feldinhalte angegeben werden.
insert into BELEGE (Rechnr, Kdnr, Betrag) values ('36000980', '1004267', 380.20)	Neuer Datensatz für die Tabelle "Belege": Die Inhalte für die Rechnungsnummer ('36000980'), Kundennummer ('1004267') und den Betrag (380.20) sind genannt.

1.2.1.3.3.2 Datensätze ändern

update	Ändert die Inhalte von Spalten einer Tabelle.
update personal set gehalt = 5000	ACHTUNG: **Alle** Datensätze erhalten im Feld "Gehalt" den Inhalt 5000.
update personal set gehalt = 5000 where abteilung = "ZXD"	Nur die Datensätze, die in der Spalte "abteilung" den Inhalt "ZXD" enthalten, erhalten einen neuen Inhalt der Spalte "gehalt".

update personal set gehalt = 5000 where abteilung = "ZXD" and eintritt >= '12-1-79'	S.o., aber zusätzlich muß das Datumsfeld "Eintritt" einen Datumswert beinhalten, der nach dem 1.12.1979 liegt (beachten Sie die andere Anordung von Tag und Monat im amerikanischen Datumsformat!).
update lager set vk1 = vk1 * 1.10 where lieferant = 10243	Preiserhöhung: alle Preise von Artikeln des Lieferanten mit der Nummer 10243 werden um 10 % erhöht.

1.2.1.3.3.3 Datensätze löschen

delete	Löscht Datensätze aus einer Tabelle.
delete from test	ACHTUNG: **Alle** Datensätze aus der Tabelle "test" werden gelöscht.
delete from test where name = 'Meier'	Die Datensätze werden gelöscht, die in der Spalte "name" die Zeichenfolge "Meier" beinhalten.

1.2.1.3.3.4 Datensätze abfragen

Das Abfragen von Datensätzen, die in Form von Tabellen gespeichert sind, ist der wohl wichtigste Aspekt bei der Arbeit mit Datenbanken. Betrachtet man die dabei erzeugte Ergebnismenge, so ergeben sich drei Arten, wie die Datenbank eine Abfrage beantwortet:

- **Projektion**: Die Auswahl betrifft die Tabellenspalten, d.h. es werden nicht alle *Spalten* der Tabelle angezeigt.
- **Selektion**: Es wird nur ein Teil der *Zeilen* aus der Tabelle angezeigt, aber dafür alle Spalten.
- **Join**: Daten aus verschiedenen Tabellen werden zusammengeführt.

Zwischen diesen Arten gibt es beliebig viele Kombinationen, die zusätzlich um Funktionen und andere Bestandteile der Sprache SQL erweitert werden können.

1.2.1.3.3.4.1 Einfache Abfragen

select * from lager	Alle Datensätze aus der Tabelle Lager auflisten.

select * from lager order by artnr	S.o., zusätzlich sollen die Sätze nach dem Inhalt des Feldes "artnr" sortiert werden.
select * from lager **where** lieferant = 10243	Alle Datensätze auflisten, die in der Spalte "Lieferant" den Inhalt 10243 enthalten.
select * from lager **where** lieferant = 10243 **and** ekpr < 12	Alle Datensätze auflisten, die in der Spalte "Lieferant" den Inhalt 10243 enthalten und deren Inhalt des Feldes "ekpr" kleiner als 12 ist.
select * from lager **where** lieferant = 10243 **or** lieferant = 10244	Alle Datensätze auflisten, die in der Spalte "Lieferant" den Inhalt 10243 oder 10244 enthalten.
select * from lager where lieferant is null	Alle Datensätze die im Feld "lieferant" keinen Eintrag enthalten.
select * from lager **where** name **like** 'H%'	Alle Datensätze anzeigen, deren Inhalt des Feldes "name" mit einem "H" beginnt.
select * from lager **where** name **like** '_a%'	Alle Datensätze anzeigen, deren Inhalt des Feldes Name ein beliebiges erstes Zeichen, an zweiter Stelle das Zeichen "a" und dann eine beliebige Zeichenfolge enthält.
select * from lager **where** ekpr **between** 1 and 3	Alle Datensätze anzeigen, deren Inhalt im Feld "ekpr" zwischen 1 und 3 liegt.
select rechnr, betrag, betrag*1.10 from belege	Die Spalten "rechnr" und "betrag" anzeigen lassen, eine neue Spalte errechnen und anzeigen lassen (Betrag * 1.10).
select name || ' ' || trainer from gaesteliste	Die Spalten "Name", "Trainer" mit einem Leerzeichen getrennt werden angezeigt.
select * from lager where lieferant in (10243, 55555)	Alle Datensätze aus der Lagertabelle anzeigen, deren Inhalt des Feldes "Lieferant" in der angegebenen Werteliste enthalten ist.

1.2.1.3.3.4.2 Zusammenfassende Abfragen

Abfragen mit Funktionen

select min(ekpr), max(ekpr), avg(ekpr), count(ekpr) from lager	Zeigt den kleinsten, größten und mittleren Inhalt aus der Spalte "ekpr" an und zählt die Datensätze .
select name, min(ekpr) from lager	ACHTUNG: führt zu einer Fehlermeldung. Eine zusammenfassende Funktion kann nicht mit einem Spaltennamen kombiniert werden.

Abfragen mit group by

select kdnr, count (rechnr) from belege **group** by kdnr	Anzahl der Rechnungen je Kundennummer.
select kdnr, count(rechnr), sum(betrag) from belege **group** by kdnr	Tabelle, sortiert nach Kundennummern mit Anzahl der Rechnungen und der Summe der Beträge (Zwischensummen nach Kundennummern).
select kdnr, count(rechnr), sum(betrag) from belege **group** by kdnr **having** count(rechnr)>2	Having - nur in Verbindung mit group by möglich - bezieht sich auf eine Gruppe von Datensätzen mit einer bestimmten Eigenschaft. Hier soll die Zahl der dem Kunden zugeordneten Rechnungen größer als 2 sein

Inner Joins (verbindet zwei oder mehr Tabellen; Daten aus der einen Tabelle werden nur dann angezeigt, wenn sich ein entsprechender Eintrag in der anderen Tabelle findet)

select lieferant.liefnr, lieferant.name, lieferant.telefon, lager.artnr, lager.name from lieferant, lager	Die Anweisung soll beide Tabellen kombinieren und die Datensätze so anzeigen, daß neben dem Lieferantennamen die von ihm gelieferten Artikel angezeigt werden. ACHTUNG: Zeigt das **kartesische Produkt** der beiden Tabellen "Lieferant" und "Lager". D.h., alle Datensätze aus der ersten Tabelle werden mit allen Datensätzen aus der zweiten Tabelle kombiniert und angezeigt. Aber: Das Ergebnis ist nicht zu gebrauchen.

select lieferant.liefnr, lieferant.name, lieferant.telefon, lager.artnr, lager.name from lieferant, lager **where lieferant.liefnr = lager.lieferant**	Durch die **zusätzliche where-Anweisung** werden die Daten korrekt angeordnet. Es werden aber nur die Sätze angezeigt, die tatsächlich verknüpft werden können, d.h. falls es einen Datensatz in Lager gibt, der eine Lieferantennummer enthält, die es in der Lieferantentabelle nicht gibt, dann wird dieser Satz nicht angezeigt.

Outer joins (verbindet zwei oder mehr Tabellen in der Form, daß aus *einer* der Tabellen alle Datensätze angezeigt werden)

select lieferant.liefnr, lieferant.name, lager.artnr, lager.name from lieferant, lager where lieferant.liefnr**(+)** = lager.lieferant	Das in Klammern gesetzte Pluszeichen erzwingt die Anzeige aller Datensätze (hier aus der Tabelle mit den Lieferantendaten).
select lieferant.liefnr, lieferant.name, lager.artnr, lager.name from lieferant, lager where lieferant.liefnr = lager.lieferant**(+)**	Wie eben, hier aus der Tabelle mit den Lagerdaten. Sie dürfen das Pluszeichen nur einmal verwenden.

1.2.1.3.3.4.3 Unterabfragen

Single row (liefert nur einen Datensatz zurück)

select artnr from auftrag where menge = (select max(menge) from auftrag)	Zeigt die Artikelnummer des Artikels an, von dem die größte Stückzahl abgegeben wurde.

Multiple row (liefert mehrere Datensätze zurück)

select * from lager where artnr in (select artnr from auftrag)	Die Unterabfrage (innerhalb der Klammer) stellt alle Artikelnummern aus der Auftragstabelle fest (d.h. eine Liste aller abgegebenen Artikel), die Hauptabfrage liefert zu den durch die Unterabfrage bestimmten Artikeln die restlichen Informationen aus der Tabelle mit den Lagerdaten.

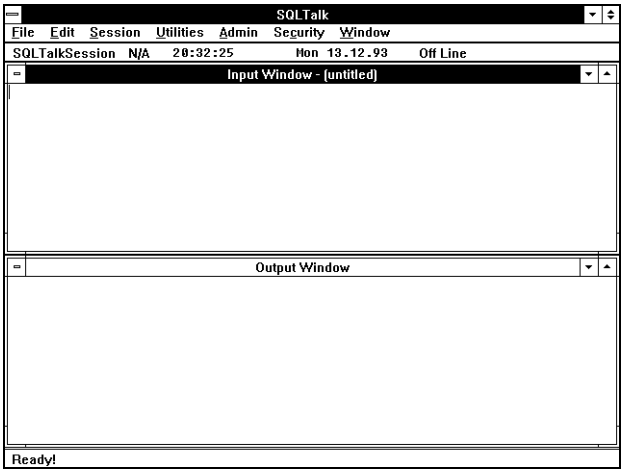
1.2.2 Die Benutzung von SQLTalk zur Erzeugung einer Datenbank mit mehreren Tabellen

SQLTalk ist ein im Lieferumfang von SQLWindows enthaltenes Programm, mit dem SQL-Datenbanken angelegt, Daten eingegeben und Daten abgefragt werden können. Es liegt in zwei Versionen vor, einer Version, die unter DOS läuft, und einer weiteren Version, die unter Windows läuft. Obwohl die Beispiele grundsätzlich mit beiden Versionen anwendbar sind, wird nur auf die Windows-Version von SQLTalk Bezug genommen.

SQLTalk bietet eine sehr einfach zu benutzende Umgebung, die es Ihnen ermöglicht, die in Datenbanken enthaltenen Daten aus der Datenbank abzufragen, ist aber kein Ersatz für komplette Front-End-Applikationen wie z.B. Quest oder mit SQLWindows zu erstellende Anwendungen.

Starten Sie SQLTalk aus Ihrer Programmgruppe SQLWindows, dann erhalten Sie folgenden Bildschirm.

Abb. 1.2: SQLTalk.



Im wesentlichen besteht die Oberfläche von SQLTalk aus zwei Hälften: In der oberen Hälfte können SQL-Anweisungen eingegeben werden (das sogenannte "Input Window"), in der unteren Hälfte werden die von SQLTalk gelieferten Antworten, häufig Ergebnismengen, angezeigt (das sogenannte "Output-Window"). Das Menü von SQLTalk dient vor allem dazu, an die Datenbank anzubinden sowie die von Ihnen eingegebenen

SQL-Anweisungen ausführen zu lassen, bietet darüber hinaus aber auch die Möglichkeit, einen Datenimport von Daten vorzunehmen, die in Fremdformaten vorliegen (CSV, DIF, Lotus 123, dBASE).

Da die Benutzung des Menüs und die Abfolge der Bedienung für Sie von einiger Bedeutung ist, soll ein kleiner Fahrplan die generelle Abfolge der Arbeit in SQLTalk zusammenfassen und erläutern.

Die Reihenfolge der Arbeit in SQLTalk ist in der Regel folgende:

- Verbindungsaufbau zur Datenbank (über Menü: "Session", Option "Connect" oder durch Drücken der F2-Taste);
- Eingabe von SQL-Anweisungen (im oberen Input-Window, dabei darf der Schrägstrich nicht vergessen werden, der eine Anweisung abschließt);
- Ausührung der SQL-Anweisungen (über Menü: "Session", Option "Current Command" oder durch Shift+F2-Taste);
- Prüfung der Ergebnismenge (Output-Window);
- Beenden der Verbindung zur Datenbank (über Menü: "Session", Option "Disconnect" oder durch F4-Taste).

Im folgenden werden einige SQL-Anweisungen vorgestellt, die eine Datenbank sowie verschiedene Tabellen anlegen. Ihre Aufgabe ist es, zunächst eine Datenbank anzulegen, danach die Tabellenstrukturen zu definieren und schließlich Daten in die Tabellen einzugeben.

Beachten Sie, daß Sie diesen Teil des Handbuchs nicht überspringen dürfen, da ansonsten keine Daten für die später folgenden Beispielanwendungen vorhanden wären.

1.2.2.1 Anlegen der Datenbank FIRMA mit SQLTalk

Drücken Sie in SQLTalk die Tastenkombination "**Strg+I** " (Install Database). Geben Sie als Namen für die Datenbank "FIRMA" ein, klicken Sie auf der "New"- Check Box um anzuzeigen, daß eine neue Datenbank von Grund auf neu angelegt und nicht nur installiert werden soll (auf die Eingabe eines Paßworts wird hier verzichtet). Alternativ könnten Sie auch die Anweisung "install database firma/" eingeben und ausführen lassen.

Bestätigen Sie nun die Eingabe durch Drücken des OK-Push Buttons; daraufhin wird die Datenbank angelegt: Falls Sie mit der Einzelplatzversion der Gupta-Datenbank SQLBase arbeiten, wird die neue Datenbank lokal auf Ihrem Rechner angelegt; falls Sie mit der Netzversion arbeiten, können Sie die Datenbank auf dem Server anlegen lassen. Da diese Datenbank nur zu Übungszwecken verwendet werden soll und in keinem Zusammenhang mit "echten" Daten steht, sollten Sie die Datenbank aber in jedem Fall lokal anlegen lassen.

Falls Sie diese Datenbank bereits angelegt haben, können Sie mit der Funktionstaste F2 sofort die Verbindung zu ihr herstellen.

1.2.2.2 Anlegen der Tabellen

Da die Strukturen der Tabellen im Anhang ausführlich dargestellt werden und die SQL-Anweisungen hierfür bereits im vorstehenden Kapitel dargestellt wurden, werden hier nur die zur Erzeugung der Tabellen nötigen SQL-Anweisungen genannt.

Beachten Sie folgendes, bei der Eingabe der Kommandos:

- Groß- und Kleinschreibung können gemischt werden.
- Leerzeichen zwischen den einzelnen Worten sind notwendig; wo ein Leerzeichen steht, dürfen auch mehrere Leerzeichen stehen.
- Zeilenumbrüche sind beliebig oft verwendbar.
- Einzelne SQL-Anweisungen müssen (in SQLTalk) mit dem Schrägstrich (/) abgeschlossen werden, dies gilt auch für die letzte Anweisung.
- Einzelne Kommandos können mit der Tastenkombination Shift+F2 abgeschlossen werden.
- Eine Gruppe von SQL-Anweisungen kann in einer sog. Script-Datei "verpackt (Extension: *.WTS), als Ganzes geladen und ausgeführt werden. Skript-Dateien können auch durch CASE-Tools wie z.B. ERWin erzeugt und hier in SQLTalk ausgeführt werden.

Die einzelnen Tabellen werden durch folgende SQL-Anweisungen angelegt:

Die Tabelle **KUNDEN**

```
Create table Kunden(
Kundennummer integer not null,
name         character(10),
vorname      character(10),
plz          character(5),
strasse      character(10),
ort          character(10),
telefon      character(10),
kundengruppe smallint,
bemerkung    longvarchar)
/
```

Vergessen Sie nicht den Schrägstrich, um die Anweisung zu beenden. Lassen Sie danach die Anweisung durch Drücken der Shift+F2-Taste ausführen. Verfahren Sie danach mit allen folgenden create-Anweisungen genauso.

Die Tabelle **LAGER**

```
Create    table lager(
artnr     integer not null,
name      character(10),
ekpr      decimal(10,2),
vk1       decimal(10,2),
vk2       decimal(10,2),
vk3       decimal(10,2),
lieferant integer)
/
```

Die Tabelle **LIEFERANTEN**

```
create table lieferanten(
liefnr  integer,
name    character(10),
telefon character(10))
/
```

Die Tabelle **Auftrag**

```
create table auftrag(
aufnr   integer,
artnr   integer,
menge   integer,
e_preis decimal(10,2))
/
```

Die Tabelle **Kontroll**

```
create table kontroll(
aufnr  integer,
kundennummer integer,
datum  date,
gesamt decimal(10,2),
art    character(1))
/
```

Hiermit sind alle Tabellen angelegt, die in den folgenden Beispielen verwendet werden. Geben Sie zur Kontrolle die Anweisung "select * from systables/" ein, lassen Sie mit Shift+F2 die Anweisung ausführen, um eine Übersicht aller eben angelegten Tabellen anzeigen zu lassen.

Nun können Datensätze eingegeben und an die Tabellen übergeben werden. Dies wird im folgenden Kapitel erläutert.

1.2.2.3 Eingabe von Datensätzen

Die Eingabe von wenigen Datensätzen je Tabelle soll genügen. Beachten Sie dabei aber, daß die Sätze genau übernommen werden, denn

- SQLTalk nimmt keine Längenbegrenzung vor. Geben Sie überlange Informationen ein, d.h. Informationen, die über die Länge der dafür vorgesehenen Spalte hinausgehen, erhalten Sie eine Fehlermeldung,

- die Inhalte der Schlüsselfelder (Kundennummern, Artikelnummern, Lieferantennummern) werden später in den Beispielen gebraucht.

Selbstverständlich können mehr Datensätze als die hier genannten eingegeben werden.

Daten für die Tabelle Kunden

```
insert into kunden
(kundennummer, name, vorname, plz,
strasse, ort, telefon, kundengruppe)
values
(10000, 'Meier', 'Bernd', '12345',
'A-Straße 2', 'A-Dorf', '012-345', 1)
/
insert into kunden
(kundennummer, name, vorname, plz,
strasse, ort, telefon, kundengruppe)
values
(10001, 'Müller', 'Willi', '12345',
'A-Straße 3', 'B-Dorf', '023-456', 1)
/
insert into kunden
(kundennummer, name, vorname, plz,
strasse, ort, telefon, kundengruppe)
values
(10002, 'Keller', 'Erwin', '12345',
'A-Straße 4', 'C-Dorf', '034-567', 2)
/
```

Daten für die Tabelle Lager

```
insert into lager
(artnr, name, ekpr, vk1, vk2, vk3, lieferant)
values
(1000, 'Hammer', 5.00, 9.00, 8.00, 7.00, 10243)
/
insert into lager
(artnr, name, ekpr, vk1, vk2, vk3, lieferant)
values
(1001, 'Kelle', 3.00, 7.00, 6.00, 5.00, 10243)
/
insert into lager
(artnr, name, ekpr, vk1, vk2, vk3, lieferant)
values
(1002, 'Eimer', 2.00, 5.00, 4.00, 3.00, 10243)
/
```

Daten für die Tabelle Lieferanten

```
insert into lieferanten
(liefnr, name, telefon)
values
(10243, 'Müller', '012-234')
/
insert into lieferanten
(liefnr, name, telefon)
values
(10244, 'Werner', '056-339')
/
```

Lassen Sie sich durch die folgenden Anweisungen die soeben eingegebenen Datensätze anzeigen:
select * from kunden
/
select * from lager
/
select * from lieferanten
/

Die einzelnen Anweisungen lassen Sie jeweils mit der Tastenkombination Shift+F2 ausführen.

Schließlich können Sie durch Drücken der Taste F4 ein Disconnect herbeiführen, um die Verbindung zur Datenbank aufzuheben. Dies ist bei der Single-User Version von SQLBase die Voraussetzung für den

Verbindungsaufbau aus einer SQLWindows-Anwendung heraus, da bei dieser Version immer nur ein Benutzer zugelassen ist. Benutzen Sie hingegen eine Multi-User-Version von SQLBase, brauchen Sie kein Disconnect herbeizuführen, da dort mehrfache Verbindungen zur Datenbank möglich sind. Allerdings ist es auch in letzterem Fall sinnvoll, SQLTalk zu beenden, da Sie die späteren Abfragen (zur Kontrolle von Spaltennamen oder Tabelleninhalten) aus SQLWindows heraus sehr elegant durchführen können (in SQLWindows: Menü "View", "Database Browser").

1.3 Einführung in SQLWindows

SQLWindows ist eine **4GL-Entwicklungsumgebung** zur Programmierung von **Client-Server** Datenbankanwendungen.

Unter **4GL** versteht man "fourth-Generation-Language", also eine Sprache der vierten Generation: Während früher zunächst in Maschinensprache (erste Generation), dann in Assembler (zweite Generation) und schließlich in einer Hochsprache wie Pascal oder C (dritte Generation) programmiert wurde, existieren seit einigen Jahren Sprachen der vierten Generation. Sie erlauben dem Entwickler z.B. eine Maske zu zeichnen anstatt den dafür notwendigen Programmcode einzutippen. Ein anderes Programm (man könnte dieses Programm als eine Art Assistenten bezeichnen) erstellt aus dem gezeichneten Entwurf den eigentlichen Programmcode.

Als **Client-Server** wird ein System dann bezeichnet, wenn es einen zentralen Datenbestand für eine Gruppe von Benutzern bereithält. Der Server (der aus mehreren Rechnern bestehen und für den Benutzer nicht sichtbar aufgestellt sein kann) vermag dabei durch die Bereithaltung von Datenbeständen einem gesamten Unternehmen zu dienen. Informationen über Kunden, die in Form einer oder mehrerer Tabellen in der Datenbank gespeichert ist, sind gemäß dieser Definition in einem (idealen) Unternehmen nur einmal vorhanden. Der Client ist in diesem System ein Benutzer, der über ein Netzwerk in Verbindung mit dem Server steht und die dort gespeicherten Daten nutzt.

SQLWindows ist zwar nicht das einzige Entwicklungssystem für Client-Server-Anwendungen, es ist aber eines, das besonders ausgereift und leistungsfähig ist, da eine frühe Orientierung an der Oberfläche Windows erfolgt ist. Zur Verdeutlichung:

- Durch zusätzliche Module (sog. Router) können die mit SQLWindows erzeugten Anwendungen auf nahezu alle verfügbaren Datenbanken wie z.B. Oracle, Informix, SQL Server OS/2, AS/400 und Netware zugreifen.

- Durch die Kombination der Sprache SQL mit der in SQLWindows enthaltenen Programmiersprache SAL (SQLWindows Application Language) wird eine äußerst hohe Produktivität erreicht. Deshalb wird der Aufwand zur Erstellung von Client-Server-Anwendungen gegenüber anderen Systemen deutlich reduziert.

- Durch das Vorhandensein eines Tools, das es ermöglicht Teams von Programmierern zu koordinieren (Team Windows), können auch größte Software-Projekte überschaubar geplant, entwickelt und veränderten Anforderungen angepaßt werden. Außerdem können mit diesem Werkzeug einmal

entwickelte Module so bereitgestellt werden, daß sie für andere Anwendungen einfach zu verwenden sind.

In den folgenden Kapiteln werden Begriffe in der gleichen Weise benutzt, wie sie in amerikanischen Handbüchern und der Entwicklungsumgebung von SQLWindows Verwendung finden. Obwohl dies zunächst einen erhöhten Lernaufwand bedeutet, profitieren Sie später davon, da Sie immer mit den gleichen Begriffen arbeiten; z.B. mit dem Begriff "Push Button" anstelle des deutschen Begriffs "Schaltfläche".

Die folgende Übersicht faßt die Namen von Objekten zusammen und gibt gleichzeitig eine Empfehlung, welcher Präfix für diese Objekte bei der späteren Entwicklung von Anwendungen verwendet werden sollte: So wird z.B. empfohlen einem Push Button den Namen "pbEnde" zu geben (also den Präfix pb zu benutzen). Obwohl dies nicht so sein muß, empfiehlt es sich dieser Empfehlung zu folgen, da andere oder man selbst nach einer gewissen Zeit die gemäß dieser Vorgabe erstellten Anwendungen einfacher lesen kann. Zu beachten ist, daß SQLWindows bei Objekten, die in eine Anwendung eingefügt werden, ebenfalls den Präfix benutzt und eine laufende Nummer anhängt. So erhält der erste eingefügte Push Button den Namen "pb1", der zweite "pb2" usw.. Das heißt, Sie sollten sich angewöhnen in jedem Fall die von SQLWindows vorgeschlagenen Namen zu verändern; daß dies in dieser Einführung nicht immer geschieht liegt allein daran, daß der Leser möglichst schnell in die Lage versetzt werden soll, die vorgestellten Objekte zu benutzen.

Die folgende Übersicht, mit der die in SQLWindows vorhandenen Objekte vorgestellt werden, gliedert sich gemäß den Vorgaben von SQLWindows, das zwei Arten von Objekten kennt in zwei Teile:

- Top-Level Objekte und
- Child Objekte.

Top-Level Objekte sind solche Objekte, die üblicherweise den Kern einer Anwendung ausmachen. Sie werden in der Regel nach dem Start einer Anwendung als erste Objekte auf dem Bildschirm angezeigt, werden aber zusätzlich noch an den verschiedensten Stellen innerhalb einer Anwendung eingebaut. Das wohl am häufigsten verwendete Top-Level Objekt ist das Form Window.

Child Objekte sind dagegen solche Objekte, die nur als eingebettete Objekte innerhalb von Top-Level Objekten verwendet werden. Typische Child Objekte sind z.B. Push Button, Data Field, List Box oder Combo Box.

Die folgenden zwei Übersichten fassen - getrennt nach Top-Level und Child Objekten - die Objektnamen und die Präfixe in SQLWindows zusammen.

Top-Level Objekte

Objektname	Präfix	übersetzter Objektname
Form Window	frm	Formular, Maske, Fenster
Table Window	tbl	Tabellenfenster
Dialog Box	dlg	Dialogfenster

Quest Table Window	tbl	Quest-Tabellenfenster
Multiple Document Interface	mdi	Multi-Form

Child Objekte

Objektname	Präfix	übersetzter Objektname
Push Button	pb	Schaltfläche
Data Field	df	Feld
Multiline Field	ml	Editierbereich
Radio Button	rb	Optionsfeld
Option Button	ob	Schaltfläche
Check Box	cb	Kontrollkästchen
List Box	lb	Listenfeld
Combo Box	cmb	Popup-Tool
Horizontal Scroll Bar	hsb	horizontale Bildlaufleiste
Vertical Scroll Bar	vsb	vertikale Bildlaufleiste
Column	col	Spalte
Picture	pic	Bild

Beachten Sie, daß die deutschsprachigen Begriffe in der Literatur nicht einheitlich verwendet werden; konzentrieren Sie sich deshalb auf die englischen Originalbegriffe.

1.3.1 Generelles Arbeiten mit SQLWindows

Das generelle Arbeiten mit SQLWindows soll vor dem Hintergrund einer Benutzung von Datenbanken beschrieben werden. Wenn es auch möglich ist, SQLWindows Anwendungen ohne Benutzung von Datenbanken zu erstellen, dürfte die Benutzung einer Datenbank und der darin enthaltenen Daten eher die Regel sein.

1.3.1.1 Verzeichnisse, Dateien und die Programmgruppe SQLWindows

Durch die Installation von SQLWindows sind auf Ihrem Datenträger zwei Verzeichnisse angelegt worden. Die Namen der Verzeichnisse lauten

- SQLBASE sowie
- GUPTA.

Im Verzeichnis **SQLBASE** befindet sich eine Datei mit dem Namen START.DBS. Dies ist eine (leere) Datenbank, ein Muster, das immer dann in ein neues Verzeichnis kopiert (und umbenannt) wird, wenn Sie eine neue Datenbank einrichten. Richten Sie z.B. eine neue Datenbank mit dem Namen "FIRMA" ein, so wird zunächst ein Verzeichnis unter dem Verzeichnis SQLBASE angelegt (C:\SQLBASE\FIRMA), danach wird die

Datei START.DBS in dieses Verzeichnis kopiert und umbenannt. Damit ist die neue Datenbank eingerichtet und für das Anlegen von Tabellen verwendbar. Achten Sie darauf, daß alle (lokalen) Datenbanken unter dem Verzeichnis SQLBASE angelegt werden.

Arbeiten Sie dagegen im Netz, so liegen die Datenbanken zwar ebenfalls unter dem Verzeichnis SQLBASE, aber nicht unbedingt auf Laufwerk "C:", sondern in einem Laufwerk des Netzes.

Üblicherweise befinden sich nach der Standardinstallation die Datenbanken "TEAMWIN", "TWSAMPLE", "QUEST", "SWDEMO" und "SPA" auf Ihrer Platte. Sie sollten sich die in diesen Datenbanken enthaltenen Tabellen einmal anzeigen lassen (mit "SQLTalk" oder "Quest"). Nachdem Sie das vorhergehende Kapitel durchgearbeitet haben, befindet sich eine zusätzliche Datenbank mit dem Namen "FIRMA" in dem Verzeichnis \SQLBASE\FIRMA, sie stellt die Daten zur Verfügung, die für die unten folgenden Beispiele benötigt werden.

Im Verzeichnis **GUPTA** befinden sich zunächst alle EXE-Dateien, mit denen Sie arbeiten werden, z.B. "SQLWIN50.EXE", "SQLTALK.EXE", "TEAMWIN.EXE" und "QUEST30.EXE". Daneben finden sich noch andere Dateien vom Typ "DLL" (Dynamic Link Libraries) im Verzeichnis. Diese Dateien werden gebraucht, um aus Ihrer entworfenen, zunächst nur innerhalb von SQLWindows lauffähigen Anwendung, eine stand-alone Anwendung zu machen, d.h. eine Anwendung, die allein lauffähig ist. Zu beachten ist allerdings, daß die von Ihnen vertriebene EXE-Datei, die eigentliche Anwendung, zusätzlich die im Verzeichnis "DEPLOY" vorhandenen Dateien benötigt; nähere Angaben hierzu finden Sie in den mitgelieferten Handbüchern von SQLWindows.

Von den anderen vorhandenen Verzeichnissen ist vor allem das Verzeichnis mit dem Namen "SAMPLES" für Sie von Bedeutung. Die dort gespeicherten Anwendungen (vom Typ "APP") können Sie für Testzwecke in SQLWindows laden und ausführen lassen. Obwohl Sie aus diesen Beispieldateien sehr viele Anregungen für Ihre eigenen Programme holen können, sollten Sie nicht unbedingt versuchen diese Anwendungen für Ihre eigenen Zwecke umzubauen. Üblicherweise erhalten Sie damit zwar schnell lauffähige Programme, doch werden Änderungen von Programmen, die von anderen entwickelt wurden sehr schnell zur Tortur, vor allem für Anfänger. Das heißt, Sie sollten diese Anwendungen in erster Linie für Studienzwecke benutzen und sich etwas "Appetit holen".

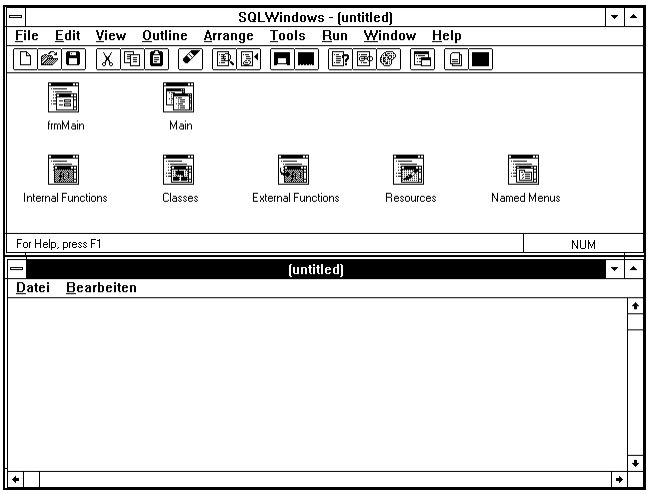
Neben den genannten Verzeichnissen wird bei der Installation auch eine neue Programmgruppe eingerichtet. Diese Programmgruppe mit dem Namen "Gupta" enthält im wesentlichen das Programm SQLWindows. Die anderen Programme, die alle im gleichnamigen Verzeichnis zu finden sind und in der folgenden Übersicht erläutert werden, können Sie sich über den Programm-Manager in das Fenster einbinden lassen.

Symbol (Icon)	Bedeutung
SQLWindows Designer	Entwicklungsumgebung für Anwendungen.
Quest	Front-End für Erzeugung von Datenbanken, Tabellen und darauf basierenden Abfragen, Reports, Views. Neben der Menüführung ist auch die direkte Eingabe von SQL-Kommandos möglich.
SQLWindows Sampler	Sehr sehenswerte Beispielanwendung, die viele Aspekte der Programmierung von SQLWindows-Anwendungen demonstriert (z.B. MDI, DDE, OLE).
Team Windows	Programm, das bei der Organisation und Verwaltung von SQLWindows-Anwendungen hilft. Damit können Sie neben Zugriffsrechten auch die verschiedenen Versionen von Programmodulen kontrollieren. Dieses Tool ist zwar besonders für Teams von Programmierern gedacht, hilft aber auch einzelnen Programmierern, den Überblick über komplexe Anwendungen zu behalten.
Report Windows	Programm, das zum Anfertigen von Reports benutzt wird. Es wird üblicherweise aus SQLWindows heraus aufgerufen, kann aber auch innerhalb des Programm-Managers gestartet werden.
Edit Windows	Mit diesem Programm können Sie das Aussehen von fertigen Anwendungen nachträglich an die Bedürfnisse des Benutzers anpassen.
Data Dictionary Manager	Wörterbuch, das Informationen zu Datenbanken, Tabellen, Benutzern usw. bereithält. Dient vor allem der Zusammenarbeit mit Team Windows.
Configure Repository	Dient der Einstellung der von Team Windows benutzten Datenbank, in der Informationen zum Arbeitsverzeichnis, Passwort, Zugriffsrechten usw. gespeichert werden.
SQLTalk	Front-End für das Anlegen von Datenbanken, Tabellen, Abfragen, Views. Die SQL-Anweisungen müssen eingegeben werden.
SQLBase Server	Die eigentliche Datenbankmaschine, die Daten aus den einzelnen Datenbanken zur Verfügung stellt.
SQLConsole	Programm zur Einrichtung von Usern, Datenbanken und zur Beobachtung des Daten-Verkehrs z.B. im Netz.
Release Notes	Letzte Anmerkungen zu SQLWindows, die nicht in der Dokumentation zu finden sind (sollte ausgedruckt werden).

1.3.1.2 Der Startbildschirm von SQLWindows

Wenn Sie SQLWindows zum ersten Mal starten, sehen Sie einen Bildschirm mit folgender Aufteilung (es ist möglich, daß unten besprochene, zusätzliche Fenster sichtbar sind, die Sie zunächst schließen können; das beim Start angezeigte Fenster mit "Quick-Objects", wurde entfernt):

Abb. 1.3: Startbildschirm.



Die **obere Hälfte** des Bildschirms wird als **Outline-Fenster** bezeichnet; Outline kann mit Gliederung oder Umriß übersetzt werden. Hier werden alle Module einer SQLWindows-Anwendung angezeigt, im Bild oben in verkleinerter Form als Symbol, die jederzeit vergrößerbar sind. Die Symbole bedeuten im einzelnen:

- **frmMain**: Programmcode, der dem Top-Level Objekt "frmMain" zugeordnet ist. Hierin sind z.B. einzelne Child Objekte, die in das Formular "frmMain" eingefügt wurden, sicht- und editierbar.

- **Main**: Programmcode der kompletten Anwendung (alle Module, die in der Anwendung enthalten sind).

- **Internal Functions**: Selbst entwickelte Funktionen, die innerhalb dieser Anwendung codiert sind.

- **Classes**: Übersicht der Klassen, die für das objektorientierte Programmieren angelegt sind.

- **External Functions**: Externe Funktionen, die aus Bibliotheken stammen. Diese Funktionen können in SAL oder in einer Sprache wie z.B. C codiert sein.

- **Resources**: Enthält Angaben zu Quelldateien (z.B. Bitmaps), die in der Anwendung Verwendung finden können.

- **Named Menus**: Enthält Namen und Angaben zu Menüpunkten, die mit ihrem Namen aufgerufen und in eine Anwendung eingebaut werden können.

Immer, wenn Sie eine Eingabe machen, z.B. eine SQL-Anweisung, die dazu dient eine Tabelle mit Daten zu füllen, tun Sie das in dem oberen Fenster. Beachten Sie auch, daß es sich bei diesem Fenster um ein sogenannten **MDI-Fenster** handelt. Diesen Fenstertyp können auch Sie erzeugen und für Ihre Anwendungen verwenden.

Die folgende Übersicht beschreibt die in dem Outline-Window enthaltenen Schaltflächen. Beachten Sie, daß Sie durch Doppelklicken zwischen den vorhandenen Schaltflächen (auf dem Hintergrund) ein Fenster aktivieren können, in dem Sie selbst bestimmen, welche der unten angezeigten Push Buttons angezeigt werden sollen. Auch die Reihenfolge der einzelnen Push Buttons kann dabei festgelegt werden.

Abb. 1.4: Push Buttons in der Tool-Bar des Outline Windows.

Abb. 1.5: Erläuterung der Push Buttons in der Tool-Bar.

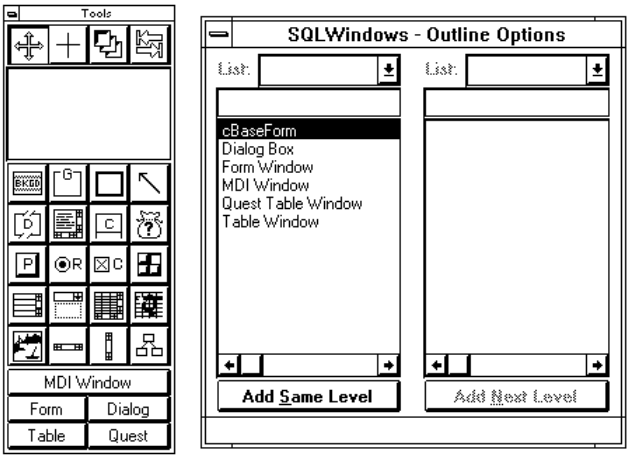
Push Button	Erläuterung
	Neue Anwendung (Application) erstellen.
	Datei öffnen.
	Aktuelle Datei speichern.
	Markierten Bereich ausschneiden.
	Markierten Bereich in die Zwischenablage kopieren.
	Inhalt der Zwischenablage einfügen.
	Letzte Änderung rückgängig machen.
	Suchen nach Text.
	Suchen und Ersetzen von Text.
	Auswahl und Anzeige eines Fensters.
	Schließen eines Fensters.
	Fenster mit Outline-Options aktivieren.
	Libraries anzeigen und auswählen.
	Tool-Palette aktivieren.
	Neue View (Blick auf die Anwendung) erzeugen.
	Report-Windows starten.
	Quest starten.

Die **untere Hälfte** des Fensters hat den Namen **Form Window**, was mit Formular-Fenster übersetzt werden kann. In diesem Fenster wird gerade ein Modul mit dem Namen *frmMain* angezeigt. Für eine noch zu entwickelnde Anwendung wäre dies das erste Fenster, aus dem heraus alle anderen Module (Tabellenfenster, weitere Formularfenster etc.) aufgerufen werden. In diesem "frmMain"-Fenster können alle in Windows bekannten Objekte (Push Button, Combo Box, List Box, Radio Buttons etc.) angezeigt werden

(selbstverständlich können Sie dieses vorgeschlagene Fenster entfernen, und - wenn Sie möchten - an dessen Stelle ein MDI-Fenster setzen).

Neben diesen beiden Fenstern, die in ihrer Größe und Anordnung völlig variabel sind, werden zwei zusätzliche, kleinere Fenster angezeigt, die das Programmieren wesentlich vereinfachen können (falls diese Fenster nicht angezeigt werden, können Sie über den Menüpunkt View anzeigen lassen, alternativ können Sie auch die Funktionstasten F2 bzw. F4 drücken). Die Fenster haben die Namen **Tool-Palette** und **Outline-Options**. Anstelle einer getippten Eingabe ermöglichen diese Fenster die Übernahme von Objekten (Tool-Palette) bzw. SQLWindows-Anweisungen oder Variablennamen (Outline Options) aus einer Menü-Auswahl heraus.

Abb. 1.6: Tool-Palette (links) und Outline-Options (rechts).



Die **Tool-Palette** (zu Deutsch etwa Werkzeug-Palette) gibt Ihnen die Möglichkeit, ohne eine einzige getippte Anweisung z.B. einen Push Button zu erzeugen. Um das zu testen, können Sie mit der linken Maustaste auf dem Symbol für einen Push Button (mit einem "P" beschrieben) einmal klicken, den Mauszeiger in das leere Form Window "frmMain" ziehen (worauf sich der Mauszeiger in einen stilisierten Push Button verwandelt) und dort erneut einmal mit der linken Maustaste klicken. Sie haben soeben einen Push Button in ein Formular-Fenster eingefügt. Sein (von SQLWindows vorgeschlagener) Name ist "pb1", seine Beschriftung lautet zunächst "untitled", diese Beschriftung kann sofort überschrieben werden (Name, Beschriftung usw. können durch Klicken mit der *rechten* Maustaste angezeigt und editiert werden).

Das Fenster mit dem Namen **Outline-Options** (rechts) kann als Ergänzung zur Tool-Palette verstanden werden. Während Sie mit der Tool-Palette im wesentlichen neue Objekte in eine Anwendung einbauen, definieren Sie mit den Outline-Options, was genau zu geschehen hat, wenn der Benutzer z.B. einmal auf den zuvor eingefügten Push Button drückt. Hierzu ein kurzes Beispiel:

Fügen Sie einen Push Button ein, indem Sie zuerst (mit der linken Maustaste) auf dem Symbol für den Push Button klicken. Ziehen Sie danach den Cursor in das (leere) Formular (dabei ändert sich die Form

des Mauszeigers). Klicken Sie einmal (linke Maustaste) um den Push Button einzufügen, danach überschreiben Sie den Titel des Push Buttons (untitled) mit dem Wort "Drücken"; drücken Sie danach die Enter Taste. Der Push Button ist damit eingefügt. Jetzt klicken Sie einmal mit der rechten Maustaste auf den Push Button. Ein kleines Fenster öffnet sich (der sog. Customizer), in dem die "Feineinstellung" des Push Buttons verändert, aber auch die eigentliche Aktion nach Drücken des Push Buttons definiert werden kann. Drücken Sie hierzu auf die Option Edit Actions (ganz unten). Drücken Sie ggf. die F2-Taste, um die Outline Options anzeigen zu lassen, klicken Sie zweimal auf dem Eintrag "On SAM_Click". Die Anweisung wird nun in das Outline Fenster kopiert. Klicken Sie jetzt in dem anderen Outline Options Fenster die Anweisung "Call" zweimal an, um eine Funktion aufzurufen: Ihre Schreibmarke steht jetzt direkt hinter der "Call"-Anweisung. Geben Sie dort folgenden Text (ohne Anführungszeichen) ein: "Call SalQuit( )", achten Sie genau auf die Schreibweise! Drücken Sie danach die Enter-Taste und lassen Sie durch Drücken der F7-Taste die Anwendung ausführen (Speichern Sie unter dem Namen "test"). Ihr Programmcode sollte folgendermaßen aussehen:

```
On SAM_Click
        Call SalQuit()
```

Nach dem Start der Mini-Anwendung klicken Sie auf den Push Button, dadurch wird die Funktion "SalQuit()" aufgerufen, die genau das tut, was zu erwarten war: Sie beendet Ihre Mini-Anwendung.

Die Abkürzung "SAM" steht für SQLWindows-Application-Message und bedeutet das Versenden von speziellen Nachrichten innerhalb von SQLWindows. So z.B. die Nachricht "Push Button pb1 wurde soeben gedrückt". Diese Nachricht wird zusammen mit einigen anderen benutzt, um auf Eingaben des Anwenders zu reagieren.

1.3.1.3 Das Menü von SQLWindows

Das Menü von SQLWindows ist Ihre Schaltzentrale zur Anfertigung und Änderung von Anwendungen. Viele Aktionen und Optionen, die SQLWindows kennt, werden über das Menü eingestellt und variiert. Oft ist es zwar geschickter, die Tool-Palette zu benutzen, z.B. um einen Push Button einzufügen, aber manche Aktionen lassen sich nur über das Menüsystem zuweisen.

Im folgenden werden einige wichtige Menüpunkte aus SQLWindows vorgestellt, die Sie sich anschauen sollten

Menüpunkt **File**

Option	Erläuterung
Open	Öffnet vorhandene Anwendung.
Save	Speichert die aktuelle (offene) Anwendung ohne SQLWindows zu verlassen.
Make Executable	Erstellt aus der offenen Anwendung (Application) mit der Extension APP eine neue Datei mit der Extension EXE. Diese Datei wird dann zusammen mit dem Inhalt des Verzeichnisses "DEPLOY" an den Benutzer weitergegeben.

Print Outline	Druckt den Inhalt der App-Datei oder markierte Teile davon. Achtung: Es werden alle Optionen gedruckt, also z.B. auch Angaben zu Größe und Position von Fenstern, weshalb die ausgedruckte Datei durchaus größeren Umfang annehmen kann. Deshalb ist es oft sinnvoller Teile die von Interesse sind, im Outline-Fenster mit der Maus zu markieren, um dann nur die markierten Teile zu drucken.
Preferences	Angaben zum Namen der Musterdatei (NEWAPP.APP), d.h. das Muster für neu anzulegende Anwendungen. Zusätzlich einzubindende Bibliotheken und Einstellungen für die Anzeige des Outline-Text-Fensters .
Exit	Verlassen von SQLWindows.

Menüpunkt **View**

Option	Erläuterung
Tool Bar	Tool Bar anzeigen ja/nein.
Status Bar	Statuszeile anzeigen ja/nein.
Outline Options (F2)	Outline Optionen anzeigen ja/nein.
Library Browser	Zeigt eine Dialog Box mit den vorhandenen Bibliotheksdateien (*.APL).
Database Browser	Zeigt Fenster mit Daten über Tabellen, Spalten; erlaubt die Eingabe von SQL-Anweisungen.
Tool Palette (F4)	Tool Palette anzeigen ja/nein. Zeigt Auswahlfenster mit Top-Level und Child Objekten.
Show Window (Strg+1)	Zeigt Auswahlfenster mit den in der Anwendung vorhandenen Fenstern, erlaubt die Anzeige eines Fensters.
Hide Window (Strg+2)	Schließt das aktuell geöffnete Fenster.

Menüpunkt **Outline**

Option	Erläuterung
Insert Line	Neue Zeile einfügen (alternativ kann die Einfg-Taste gedrückt werden).
Comment Items	Kommentieren von zuvor markierten Zeilen (dabei wird als erstes Zeichen in der Zeile das Ausrufezeichen "!" eingefügt; kann auch manuell geschehen.
Uncomment Items	Kommentarzeichen ("!") entfernen.
Expand one level +	Expandieren (Öffnen) des markierten Bereichs.
Collapse -	Schließen des markierten Bereichs.
Expand all levels *	Alle Bereiche öffnen.
Collapse Outline (Strg+O)	Alle Bereiche schließen.

Move Left (Alt+links)	Verschiebt den markierten Bereich eine Ebene nach links.
Move Right (Alt+rechts)	Verschiebt den markierten Bereich eine Ebene nach rechts.
Move Up (Alt + hoch)	Verschiebt einen markierten Bereich nach oben.
Move Down (Alt+tief)	Verschiebt einen markierten Bereich nach unten.

Menüpunkt **Arrange**

Option	Erläuterung
Bring to Front	Bringt ein "hinten" angezeigtes Objekt zur Ansicht bzw. für das Editieren nach vorne.
Send to Back	Verschiebt ein vorne sichtbares Objekt nach hinten.
Tab Order	Erlaubt die Reihenfolge z.B. des Editierens von Datenfeldern neu zu ordnen.

Menüpunkt **Tools**

Hier wurde auf eine Auswahl verzichtet, da zwar alle Optionen wichtig sind, aber überwiegend mit der Tool-Palette gearbeitet werden kann.

Menüpunkt **Run**

Option	Erläuterung
User Mode F7	Compilieren und Starten der aktuellen Anwendung (das bedeutet aber nicht, daß eine EXE-Datei angefertigt wird).
Compile F8	Kompilieren der aktuellen Anwendung (kann zur Fehlersuche dienen).
Break	Setzen und Entfernen von Break-Punkten. An diesen Stellen wird später der Programmablauf unterbrochen, man kann dann z.B. den Inhalt von Variablen anzeigen lassen.
Animate	Spezieller Modus, der in einem separaten Fenster nach Start des Programms die gerade ausgeführten Zeilen der Anwendung anzeigt.
Slow Animate	Wie vorstehend, doch pausiert das Programm bei jeder Programmzeile für eine Sekunde (Dauer ist einstellbar unter File/Preferences). Sehr nützlich bei der Fehlersuche.
No Animate	Animate-Modus (s.o.) wird hiermit abgestellt.

Menüpunkt **Window**

Option	Erläuterung
New Outline View F11	Neue Darstellung (Sicht) der Anwendung. Erlaubt die Auswahl der Symbole (Icons), die am unteren Rand des Outline Fenster darzustellen sind.

Global	Zeigt Fenster mit globalen Variablen, Konstanten, Funktionen, Klassen und Aktionen an.
Current	Zeigt von einem ausgewählten Objekt (z.B. Form Window) lokale-/static-/Klassen-Variable, lokale Aktionen und lokale Funktionen an.
Main	Fenster mit dem Namen "main" anzeigen (das ist das Fenster, in dem alle Module eingebettet sind.

Menüpunkt **Help**

Option	Erläuterung
Index	Zeigt den Hilfe-Index von SQLWindows an.
Search	Erlaubt die Suche nach Ausdrücken im Hilfesystem.
Functions	Zeigt Übersicht der Funktionen an.
Messages	Zeigt Übersicht der Nachrichten an.

1.3.1.4 Ein erster Versuch: Das Programm "hello world"

Es hat gewisse Tradition, in jedem Handbuch das die Programmierung von Computerprogrammen erläutert, ein Programm vorzustellen, das den Rest der Welt durch die Ausgabe der Zeichenfolge "hello World" grüßt. Daß dieser Gruß freilich nur auf dem eigenen PC stattfindet, muß die Freude nicht unbedingt trüben. Verstehen Sie deshalb das folgende kleine Programm als ersten Einstieg in die Programmierung von SQLWindows.

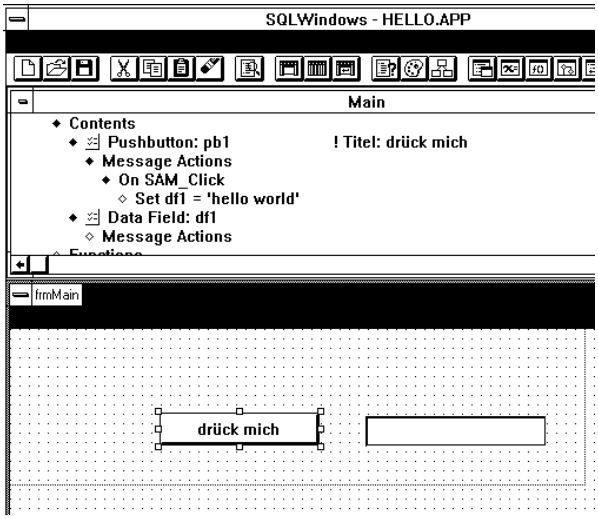
Starten Sie SQLWindows. Sorgen Sie dafür, daß Sie die Fenster mit den Application-Actions (oben) und das Form Window (unten) auf dem Bildschirm haben, so wie das in der Abbildung 1.7 unten dargestellt ist.

Im unteren Form Window soll nun ein Push Button und ein Data Field so eingefügt werden, daß später durch Drücken des Push Buttons der Gruß an den Rest der Welt angezeigt wird. Hierzu lassen Sie sich ggf. die Tool-Palette durch Drücken der Funktionstaste F4 anzeigen.

Klicken Sie einmal auf das Symbol für **Push Button**, ziehen Sie den Mauszeiger auf das Formular (der Mauszeiger wechselt zu einem stilisierten Push Button), klicken Sie einmal auf dem Formular: ein neuer Push Button mit dem Namen "pb1" (wird im oberen Outline-Fenster angezeigt) und der Beschriftung "untitled" (siehe Push Button Aufschrift) ist damit eingefügt. Da die Bezeichnung "untitled" hervorgehoben (markiert ist), überschreiben Sie dies. Geben Sie "drück mich" (ohne die Anführungszeichen) ein, bestätigen Sie durch Drücken der Enter-Taste.

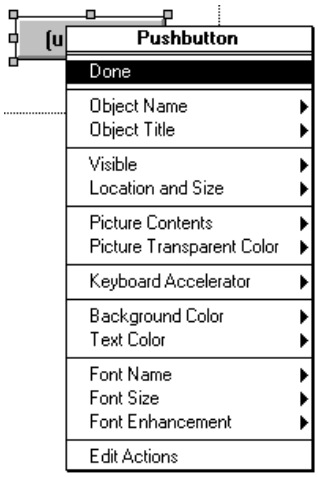
Fügen Sie dann auf gleiche Weise ein **Data Field** ein: Zuerst klicken Sie einmal auf dem Symbol für das Data Field (mit einem "D" versehen), klicken dann in irgendwo in Ihrem Formular einmal, um ein neues Data Field einzufügen. Es erhält den Namen "df1". Ihr Bildschirm sollte ungefähr so aussehen wie die folgende Abbildung (beachten Sie, daß in dem oberen Fenster unter der Sektion "Message Actions" schon der Programmcode eingetragen ist.

Abb. 1.7: Push Button und Data Field (unten), Message Actions (oben) für das Programm "hello World".



Jetzt soll vereinbart werden, daß nach einem Klicken auf dem Push Button der Text "hello world" angezeigt wird (natürlich können Sie auch "hallo Inge" oder etwas ähnliches eingeben). Klicken Sie dazu einmal mit der rechten Maustaste auf dem Push Button. Es erscheint ein Fenster, in dem alle Eigenschaften des Push Buttons eingestellt werden können, der sogenannte Customizer.

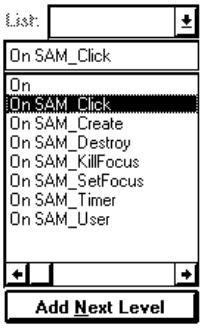
Abb. 1.8: Möglichkeiten zur Konfiguration des Push Buttons über den Customizer.



Hier ist zunächst nur die letzte Zeile des angezeigten Fensters von Bedeutung: "Edit Actions". Klicken Sie darauf (linke Maustaste) und lassen sich mit der Funktionstaste F2 die Outline-Options anzeigen. Dort

wählen Sie aus der angezeigten Liste (die über dem Push Button mit dem Titel "Add Next Level" angezeigt wird) die Anweisung "On SAM_Click" mit einem doppelten Mausklick.

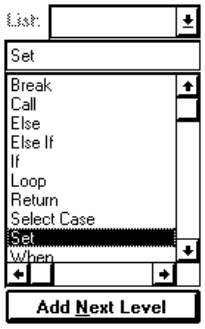
Abb. 1.9: Auswahlfenster mit SAM_Click.



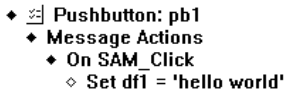
Der Text "SAM_Click" erscheint dann im oberen Fenster eingerückt unter dem Begriff "Message Actions".

Danach ändern sich die angezeigten Texte: Sie sind kontextsensitiv, d.h. es wird nur das angezeigt, was an der entsprechenden Stelle im Outline-Fenster auch sinnvollerweise verwendet werden kann. Suchen Sie den Begriff "Set" (bei "Add Next Level"), indem Sie die Anzeige rollen lassen.

Abb. 1.10: Auswahlfenster mit Set-Anweisung.



Klicken Sie auch darauf zweimal. Auch dieser Text wird in Ihr Outline-Fenster übernommen, wiederum eingerückt unter dem zuvor eingerückten "On SAM_Click"). Die Schreibmarke blinkt, Sie können jetzt den Text ergänzen, so daß die Anweisung lautet: **Set df1 = 'hello world'** (achten Sie auf das einfache Anführungszeichen!).



Jetzt können Sie Ihr erstes SQLWindows-Programm ausführen lassen: Klicken Sie auf den Menüpunkt "Run", danach "User Mode". Die Frage nach dem Speichern beantworten Sie mit "ja" und geben einen Dateinamen ein; nun wird die Anwendung gestartet. Klicken Sie auf den Push Button und Sie werden den Text angezeigt erhalten.

Sie können das Programm, wie jedes andere Windows-Programm, mit der Tastenkombination Alt+F4 beenden.

Anmerkung: Das eben erzeugte Programm hat die Extension "APP" und ist nur in Verbindung mit SQLWindows lauffähig. Falls Sie eine EXE-Datei erzeugen möchten, die auch ohne Windows auskommt, müssen Sie dies unter der Menüoption "File", "Make Executable" anweisen. Dann können Sie dieses Programm wie eine übliche Windows-Applikation behandeln.

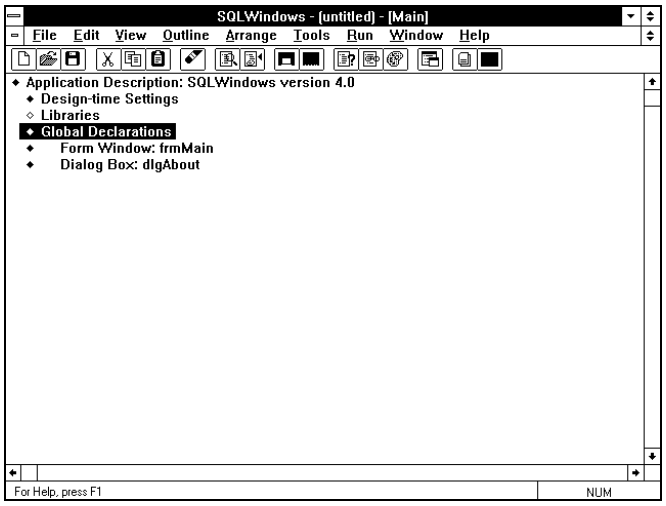
1.3.2 Die Gliederung einer SQLWindows-Anwendung

Im folgenden Kapitel wird zunächst die Gliederung einer "normalen" SQLWindows-Anwendung beschrieben. "Normal" bedeutet in diesem Zusammenhang die Gliederung und Zusammensetzung einzelner Module nach der Installation von SQLWindows die sich aus den Inhalten der Datei "Newapp.App" ergeben. Da Sie häufig eigene Vorstellungen entwickeln, wie Ihre Anwendungen auszusehen haben, soll dem Anliegen einer veränderten Gliederung in dem darauf folgenden Kapitel Rechnung getragen werden.

Es ist wichtig zu wissen, woher SQLWindows die "normale" Gliederung holt. Diese Gliederung ist nichts anderes als eine Kopie der Datei "Newapp.App". Jedesmal, wenn Sie in SQLWindows eine neue Datei anlegen, schaut SQLWindows in der Datei "Newapp.App" nach, kopiert die Inhalte und stellt Sie Ihnen zur Verfügung.

1.3.2.1 Die unveränderte Vorlage

In der nachstehenden Abbildung sehen Sie die "normale" Gliederung einer neuen SQLWindows-Anwendung. Sie erhalten diese Gliederung immer dann, wenn Sie über den Menüpunkt "File", "New" eine neue Datei anlegen lassen. Um die Anwendung so darzustellen, wie in der folgenden Abbildung, drücken Sie die Tastenkombination Strg+O. Abb. 1.11: Outline Fenster nach Drücken von Strg+O.



Die einzelnen Teile der Anwendung, die nachfolgend beschrieben werden, können Sie durch Doppelklicken auf der Raute öffnen bzw. schließen (klicken Sie zweimal auf die Raute vor der Zeile "Form Window: frmMain"). Beachten Sie, daß die Raute in zwei Versionen angezeigt wird: Eine innen ausgefüllte Raute bedeutet, daß (eingerückte) Programmteile innerhalb des angezeigten Textes bestehen, eine innen leere (hohle) Raute weist darauf hin, daß keine Programmteile enthalten sind (s. Abb. 1.11 "Libraries").

- **Application Description:** Hier findet sich eine Beschreibung der eigentlichen Anwendung (Application). Beachten Sie, daß keine Zeile innerhalb einer SQLWindows-Anwendung soweit links steht wie diese erste Zeile. Hierin kann beschrieben sein, mit welcher Version von SQLWindows gearbeitet wurde, wer einzelne Module programmiert hat, wann wichtige Änderungen vorgenommen wurden etc.. Beachten Sie, daß Sie das Editieren des hier angezeigten Textes beginnen können, nachdem Sie mit der linken Maustaste auf einem der eingegebenen Worte geklickt haben. Falls Sie einen Zeilenumbruch einfügen wollen, müssen Sie das mit der Tastenkombination Strg+Enter tun; das Drücken der Enter-Taste dagegen schließt das Eingabefeld und beendet die Editiermöglichkeit. Sie können natürlich jederzeit durch Klicken mit der Maustaste in dem eingegebenen Text das Editieren erneut beginnen.

- **Design-time Settings:** Hier werden Größe und Position von einzelnen Fenstern gespeichert. Schieben Sie z.B. ein Fenster im Entwurf nach unten rechts, erscheint dieses Fenster auch bei Ablauf des Programms und bei einem erneuten Editieren in dieser Position. Die Eintragungen bedeuten im einzelnen: "Outline Window State" speichert den zuletzt gewählten Status (Maximized, Minimized, Normal) des Outline Windows ab. "Outline Window Location and Size" speichert die letzte Position des Outline Fensters ab. "Option Box Location" speichert die letzte Position der Option Box (das Auswahlfenster mit Funktionsnamen usw.) "Class Editor Location" speichert die zuletzt gewählte Position des Class editors ab. "Tool Palette Location" speichert die letzte Position der Tool Palette ab.

  Daneben kann über die Einstellung "Fully Qualified external references" bestimmt werden, daß Sie bei Verwendung von Objekten nicht einfach deren Namen eingeben (z.B. Set **df1** = 'hello world') sondern deren Referenz ausführlich bestimmen müssen: Set **frmMain.df1** = 'hello world' (der Name des Form Windows, in dem das zu ändernde Objekt "df1" enthalten ist ("frmMain"), ist dem Objekt vorangestellt;

verbunden werden die beiden Namen durch einen Punkt). Üblicherweise erzwingen Sie damit von sich selbst einen sauberen Programmierstil.

Mit der Einstellung "Reject Multiple Window Instances" kann verhindert werden, daß von einem Objekt mehrfache Instanzen angefertigt werden. Dies kann immer dann auftreten, wenn Sie z.B. nach Öffnen eines Fenster den hierfür nötigen Menüpunkt oder Push Button mehrfach drücken, aber vergessen haben ihn zu deaktivieren (was durchaus vorkommen kann). Indem Sie dies hiermit unterbinden, erhalten Sie sofort eine Meldung, daß das mehrfache Aufrufen von Objekten nicht gestattet ist. Das Problem ist damit zwar nicht gelöst, aber Sie werden sicher daran erinnert, daß Sie sich darum kümmern müssen und verhindern, daß irgendwelche Teile Ihrer Anwendung irgendwann ein Eigenleben entwickeln (die Lösung dieses Problems findet sich in Kap. 1.3.4.1.1).

- **Libraries:** Unter Libraries versteht man Programme bzw. Teile davon, die von mehreren Anwendungen oder Entwicklern gemeinsam genutzt werden können. Falls Sie z.B. ein Info-Fenster mit Informationen verwenden und dieses Fenster mehrfach verwendet werden soll, können Sie es mehreren Anwendungen oder Entwicklern gleichzeitig zur Verfügung stellen. Das gleiche gilt auch für Funktionen, die bereits so flexibel erstellt wurden, daß sie in allen Programmen verwendet werden können. Das erlaubt Ihnen, eine Funktion (z.B. Löschen der Datenfelder einer Eingabemaske) nur einmal zu codieren, sie in einer "Library" abzuspeichern um sie jederzeit verwenden zu können. Genaugenommen sollte das Ziel jeder Programmierung sein: Schreiben Sie Ihre Module so, daß Sie sie jederzeit wiederverwenden können. Erfahrungsgemäß ist das erst nach einer gewissen Einarbeitungsphase möglich, doch sollten Sie diesen Gedanken nicht vergessen. Letztendlich werden Ihre Programme damit auch stabiler und kompakter).

- **Global Declarations:** Hier können Sie z.B. das Aussehen Ihrer Objekte generell für Ihre Anwendung definieren (Window defaults, Formats). Über "External Functions" können Libraries eingebunden werden. Unter "Constants" können Konstanten eingetragen werden; insbesondere die Verwendung eigener selbst-definierter Nachrichten (user defined Messages) setzt voraus, daß hier ein Eintrag stattfindet). Unter "Resources" können Bitmaps, Icons oder Cursor (zur Verwaltung von SQL-Anweisungen und vom Datenbanksystem gelieferten Ergebnismengen) definiert werden. Unter "Variables" werden globale Variable vereinbart: Variable, die dem gesamten System bekannt sind. Insbesondere die "SqlHandles", über die eine Verbindung zwischen Datenbank und der selbst erstellten Anwendung möglich wird, müssen hier vereinbart werden (dies wird unten ausführlich behandelt). Unter "Internal Functions" werden Funktionen, die unter Verwendung von SAL erstellt wurden, formuliert. "Named Menus" beinhaltet Menüeinträge, die unter eigenem Namen abgespeichert wurden (hier sind bereits Eintragungen enthalten, die SQLWindows vorschlägt). Unter "Class Definitions" werden Angaben zu neuen, von Ihnen selbst definierten Klassen gespeichert.

  In der Sektion "Application Actions" werden Programmteile untergebracht, die bei Beginn oder bei Ende einer Anwendung zu durchlaufen sind. Hierin finden sich die Zeilen **On SAM_AppStartup** bzw. **On SAM_AppExit**, Programmteile also, die später oft dazu verwendet werden, eine Verbindung zur Datenbank herzustellen.

Die folgenden zwei Module müssen nicht unbedingt vorhanden sein.

- **Form Window: frmMain:** Das Form Window mit dem Namen "frmMain" ist ein neues Fenster, das Sie als Grundlage Ihrer Anwendung betrachten können. Es ist somit das erste Modul für eine zu erstellende Anwendung, aus dem heraus alle anderen Module über Objekte (Menüoption, Push Button oder ähnliches) gestartet werden.

- **Dialog Box: dlgAbout:** Betrachten Sie diese Dialog Box als einen besonderen Typ von Fenster, der Ihre Arbeit an anderen Programmteilen solange verhindert, bis Sie einen Push Button gedrückt haben. Typischerweise handelt es sich bei diesem Fenster um eine Meldung, die den Benutzer auf einen wichtigen Sachverhalt hinweist. Von Windows selbst wird z.B. ein solcher Fenstertyp verwendet, um Sie darauf hinzuweisen, daß in Ihrem System ein Fehler aufgetreten ist. Hier ist das so zu verstehen, daß Sie Ihren Firmennamen, Ihr Logo usw. unterbringen können, um dem Benutzer mitzuteilen, wer das aktuelle Programm erstellt hat. Weiterarbeiten kann der Benutzer aber erst, nachdem er den Push-Buttons gedrückt hat.

Zusammenfassend die wichtigsten Anmerkungen

- Eine SQLWindows-Anwendung besteht zunächst aus 5 Modulen. Die Zahl der Module und besonders ihre Inhalte sind beliebig veränderbar.

- Module oder auch Teile davon können mit wenigen Ausnahmen durch ein vorangestelltes Ausrufezeichen ("!") als Kommentar markiert werden.

- Eine ausgefüllte Raute deutet an, daß innerhalb eines Moduls weitere Programmteile enthalten sind; eine leere Raute weist darauf hin, daß keine weiteren Teile enthalten sind.

- Die zum Anfertigen einer neuen Anwendung benutzte Vorlage ist eine Datei mit dem Namen "Newapp.App". Ihr Inhalt kann eigenen Vorstellungen angepaßt werden (s. folgendes Kapitel).

1.3.2.2 Verändern der Vorlage

Häufig entwickeln Sie bei der Arbeit mit SQLWindows eigene Vorstellungen, wie Ihre Anwendungen aussehen sollen. Damit Sie in zukünftigen Anwendungen nicht immer die gleichen Dinge ändern müssen (z.B. das Ersetzen der englischen Ausdrücke im Menü durch deutsche) können Sie sich selbst eine Datei anlegen, die als Rohling, d.h. Vorlage für neue Anwendungen, benutzt werden kann. Wird diese Datei dann unter dem Namen "Newapp.App" gespeichert, greift SQLWindows jedesmal, wenn Sie eine neue Anwendung entwickeln, auf diese Datei zu und berücksichtigt die von Ihnen gemachten Änderungen.

Um dies zu demonstrieren, wird im folgenden beschrieben, wie

- die Menüeinträge vion "frmMain"

- die Anbindung an eine Datenbank und

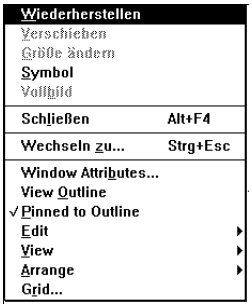
- die Inhalte der Dialog Box "dlgAbout" geändert werden können.

Falls Sie selbst noch nicht so sicher sind, ob Sie die Vorlage tatsächlich verändern wollen, können Sie die auf dem Datenträger bereits vorhandene Datei "Newapp.App" z.B. in die Datei "Newapp.Sik" kopieren.

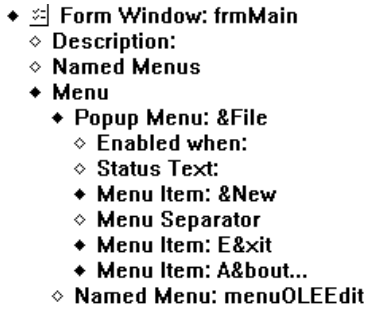
Zunächst weisen Sie SQLWindows an, eine neue Datei anzulegen (Menü: "File", "New"). In dieser Datei sollen die Änderungen vorgenommen werden.

Um das **Menü von "frmMain" zu ändern**, lassen Sie zunächst mit der Tastenkombination Strg+1 das Fenster "Show Window" anzeigen. Wählen Sie den Eintrag "frmMain" (zweimal auf "frmMain" klicken), das Fenster wird daraufhin angezeigt. Jetzt klicken Sie auf den Menüpunkt "File" (ignorieren Sie, daß sich das Menü öffnet) und klicken danach einmal in das Systemmenü des Formulars.

Abb. 1.12: Systemmenü von "frmMain" mit Option "View Outline".



Wählen Sie die Option "View Outline", um den Text aus dem Outline-Fenster anzeigen zu lassen. Dort sehen Sie jetzt den Programmcode, der zur Anzeige des eigentlichen Menüs führt.



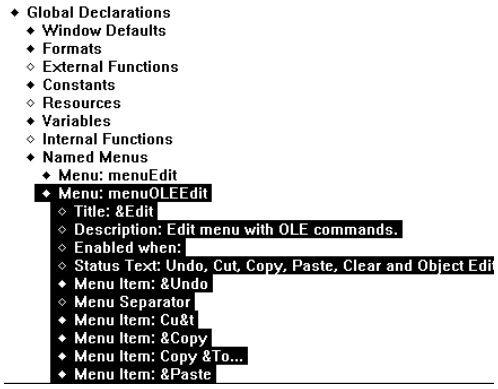
Sie sollten sich vergegenwärtigen, daß es sich bei "File" bzw. "Edit" um zwei unterschiedliche Arten von Menüpunkten handelt.

- Der Menüpunkt "**File"** ist als einfache, eingetragene Beschriftung zu verstehen, die Sie nach dem Klicken einfach durch das Wort "&Datei" ersetzen können (das Makrozeichen stellt das darauf folgende Zeichen

mit Unterstrich dar, die Auswahl mit dem Anfangsbuchstaben über die Tastatur mittels Alt+F bzw. Alt+D ist damit möglich.

- Demgegenüber handelt es sich bei dem Menüpunkt **"Edit"** um ein Menü das mit den Windows-typischen Funktionen ausgestattet ist und hier nur aufgerufen wird. Es wurde an anderer Stelle definiert, welche Optionen darin enthalten sein sollen. Würden Sie also hier den Begriff "Bearbeiten" eintragen, so hätte das keinen Sinn, vielmehr müssen Sie dort, wo das Menü mit dem angegebenen Namen **"MenuOLEEdit"** definiert wurde, Ihre Änderungen anbringen.

  Lassen Sie hierfür über den Menüpunkt "Window" das Fenster "Main" anzeigen, in dem Fenster "frmMain" dann die Sektion "Global Declarations" und dort die Sektion **"Named Menus"**. Öffnen Sie dann die Sektion Menu: "MenuOLEEDIT", Ihr Bildschirm sollte jetzt folgenden Inhalt haben:

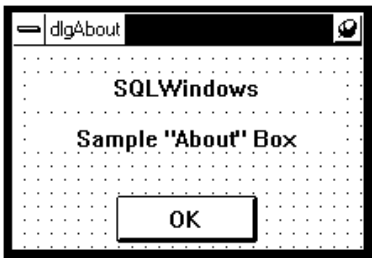


  Ersetzen Sie jetzt die Begriffe; Sie können z.B. den Eintrag "&Edit" in den Eintrag "&Bearbeiten" verändern (auf das Wort "&Edit" klicken, löschen bzw. überschreiben). Mit den anderen Menüpunkten gehen Sie gleichermaßen vor.

Sie können Ihre Anwendung jetzt bereits zwischenspeichern. Im nächsten Schritt wird erläutert, wie Sie die Dialog Box ändern.

Um die **Dialog Box mit dem Namen "dlgAbout" zu ändern**, drücken Sie zunächst die Tastenkombination Strg+F1 um das "Show Window Fenster" zu öffnen, wählen Sie daraus den Eintrag "dlgAbout". Die folgende Abbildung zeigt das noch unveränderte Fenster.

Abb. 1.13: Dialog Fenster dlgAbout.



Um den angezeigten Text zu ändern, gibt es zwei Möglichkeiten: Sie können die vorhandenen Texte mit der *linken* Maustaste anklicken und durch anschließendes Drücken der Entf-Taste auf der Tastatur löschen um anschließend ein neues Objekt vom Typ "Background Text" einzufügen, oder Sie können durch Klicken mit der *rechten* Maustaste das Feld editieren. Beides soll kurz beschrieben werden.

- **Löschen** eines vorhandenen Textes und **Neueingabe** eines anderen Textes: Klicken Sie mit der linken Maustaste auf dem angezeigten Text "SQLWindows" um ihn zu markieren, drücken Sie anschließend die "Entf-Taste", der Eintrag ist damit entfernt. Jetzt lassen Sie sich mit der Funktionstaste F4 die Tool-Palette anzeigen, klicken einmal auf das Symbol mit der Zeichenfolge "BKGD", gehen mit der Maus in das Dialogfenster, klicken einmal, halten die Maustaste gedrückt, ziehen die Maus nach rechts und lassen die Maustaste los, wenn das Feld ca. 6 cm breit ist. Jetzt erscheint die Schreibmarke, Sie können sofort losschreiben, geben Sie z.B. "Computer-Enterprises" ein, drücken Sie danach die Enter-Taste, der Text ist damit eingegeben.

- **Editieren des vorhandenen Textes**: Klicken Sie mit der rechten Maustaste einmal auf dem Text "Sample About Box", wählen Sie aus dem Fenster die Option "Object Title" und ändern Sie den angezeigten Text. Anschließend drücken Sie die Enter-Taste.

Um das Aussehen des Textes zu ändern, können Sie mit der rechten Maustaste auf den Text klicken und dann die gewünschten Eigenschaften (z.B. Farben, Schriftart, Zentrierung usw.) ändern.

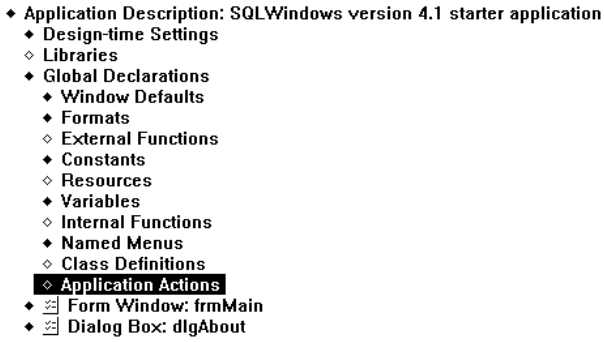
Sie können die Anwendung jetzt nochmals zwischenspeichern.

Die **Anbindung an die Datenbank** wird für SQLWindows-Anwendungen häufig nötig sein. Selbstverständlich wird nicht von allen Anwendungen die gleiche Datenbank benutzt. Deshalb wird die Anbindung an die Datenbank zwar in die Datei "Newapp.App" eingefügt, aber durch das vorangestellte Ausrufezeichen deaktiviert. Bei Bedarf müssen Sie dann nur den Namen der Datenbank angeben und das Kommentarzeichen entfernen.

Zunächst muß eine Variable erzeugt werden, über deren Namen die Verbindung zur Datenbank hergestellt wird. Wählen Sie hierzu aus dem Menü "Window" den Eintrag "Main", um sich den Programmtext der kompletten Anwendung anzeigen zu lassen. Schließen Sie durch "Strg+O" die einzelnen Sektionen. Öffnen Sie die Sektion "Variables" (einmal auf "Variables" klicken, danach Einfg-Taste drücken) und geben Sie folgendes ein (achten Sie auf Groß- und Kleinschreibung!).

Damit haben Sie einen SQL-Handle angelegt, d.h. eine Variable, die den Zugriff auf die Daten in der Datenbank erlaubt.

Klicken Sie einmal auf dem Eintrag "Application Actions", der zunächst noch leer ist.



Wählen Sie aus den Outline Options den Eintrag "On SAM_AppStartup", indem Sie zweimal darauf klicken.

Abb. 1.14: Outline Options mit dem Eintrag "SAM_AppStartup".

Der Eintrag wird dann unter "Application Actions" eingerückt dargestellt. Die folgenden Eingaben könnten Sie zwar auch aus den Menüpunkten zusammenklicken, doch ist es einfacher, die Anweisungen von Hand einzugeben. Hierzu müssen Sie zunächst die Einfg-Taste drücken, um das Editieren mit einer leeren einzufügenden Zeile zu beginnen. Drücken Sie also die Einfg-Taste, und geben Sie die folgenden Zeilen ein:

Zunächst wird die Datenbank "FIRMA" über das Schlüsselwort "SqlDatabase" bekannt gemacht (dieses Wort "SqlDatabase" ist im System vorformuliert und braucht von Ihnen nicht deklariert werden).

Über die Anweisung "Call" werden Funktionen der Sprache SAL aufgerufen, hier die Funktion "SqlConnect", um zur Datenbank anzubinden. Damit ist die Datenbank mit dem Namen "Firma" in Ihrer Datei formuliert.

Als letztes sollten Sie jetzt diesen Eintrag als Kommentar markieren, damit Sie selbst bei Bedarf diese Anbindung aktivieren können. Klicken Sie hierzu auf der Zeile "On_SAM_AppStartup" und wählen Sie danach die Menüoption "Outline" und den Eintrag "Comment Items". Die drei markierten Zeilen werden dann grau dargestellt, sie werden als Kommentar behandelt und nicht ausgeführt. An gleicher Stelle können Sie natürlich auch den Kommentar wieder zurücknehmen.

Speichern Sie die Datei unter dem Namen "Newapp.App" ab, Ihre Mustervorlage für zukünftige Anwendungen ist damit vollendet.

1.3.3 SAL, Die Programmiersprache von SQLWindows

SAL (SQLWindows Applications Language) ist die Programmiersprache, mit der SQLWindows-Applikationen codiert werden. Diese Programmiersprache kann zwar nicht direkt mit einer Sprache wie C oder Pascal verglichen werden, da sie nicht alle Möglichkeiten dieser Sprachen bietet, weist aber doch viele Ähnlichkeiten mit diesen Sprachen auf. Die Ähnlichkeiten beziehen sich auf Datentypen, Kontrollstrukturen, Funktionen, Variable, Konstanten, Operatoren, Statements, Kommentare, den von Windows-Programmiersystemen bekannten Eigenschaften der Ereignissteuerung, der Benutzung von Nachrichten sowie den Eigenschaften, die sich aus der objektorientierten Programmierung ergeben.

1.3.3.1 Datentypen

In Computerprogrammen werden Daten mit Hilfe von Variablen verarbeitet. Daten, die auf einem Datenträger vorhanden sind, müssen hierzu in den Hauptspeicher geladen und mit Hilfe von Variablen benannt werden. Allerdings gibt es Daten verschiedenster Art (z.B. einfache ganze Zahlen, gebrochene Zahlen, Zeichenketten), die alle eine besondere Behandlung verlangen. Hierfür sind in SQLWindows verschiedene Datentypen vorhanden.

SQLWindows kennt die in nahezu allen Programmiersprachen "üblichen" Datentypen. Zusätzlich kennt SQLWindows Datentypen, die sich aus der besonderen Fähigkeit von SQLWindows ergeben, mit Objekten zu arbeiten.

Die folgende Übersicht faßt die "üblichen" Datentypen zusammen und erläutert deren Verwendung.

Datentyp	Erklärung
String	Zeichenketten-Variable. Wird verwendet, um z.B. einen Namen oder eine Adresse zu speichern.
Long String	Zeichenketten mit größerer Länge, auch für Bilder zu verwenden.
Number	Numerische Daten mit einer Genauigkeit von bis zu 18 Dezimalstellen.

Date/time	Für Datums- und Zeitwerte. Die Art der Anzeige richtet sich nach der eingestellten Formatierung (z.B. in einem Data Field) und nach den Einstellungen in der Win.Ini von Windows.
Boolean	Für die Speicherung von logischen Wahrheitswerte. Nur zwei Inhalte sind möglich "TRUE" (wahr) oder "FALSE" (falsch). Die Schreibweise muß immer in Großbuchstaben erfolgen. Zu beachten ist, daß die Datenbank SQLBase diesen Datentyp nicht kennt, d.h., bei der Speicherung dieses Datentyps muß eine Umwandlung (z.B. in einen numerischen oder String-Datentyp) erfolgen.

Die folgende Übersicht faßt die etwas ungewöhnlicheren Datentypen von SQLWindows zusammen.

Datentyp	Erklärung
Window handle	Zur eindeutigen Identifikation eines Fensters (genaugenommen eines Top Level- oder Child- Objekts).
Sql handle	Zur eindeutigen Identifikation einer Verbindung zu einer SQL-Datenbank.
File handle	Zur eindeutigen Verbindung zu einer DOS-Datei, in die z.B. zu sichernde Daten geschrieben werden sollen.

Anmerkung zu dem Begriff "handle": Handle kann übersetzt werden mit "Griff", verstehen Sie diese Datentypen so, daß damit eine eindeutige Bezeichnung oder ein eindeutiger *Zugriff* auf die genannten Objekte hergestellt wird

Von den genannten Typen gibt zusätzlich jeweils eine Version, die dazu gedacht ist, die Werte von Variablen direkt manipulieren zu können, sogenannte receive-Datentypen. Sie werden innerhalb von Funktionen dazu benutzt, eine Referenz (einen Pointer) auf eine Variable zu setzten. Bei Benutzung von Funktionen, in denen solche Datentypen Verwendung finden, können Variableninhalte verändert werden, die auf einer völlig anderen Ebene zugewiesen wurde; dies ist sonst nicht möglich, da üblicherweise nur mit einer Kopie des eigentlichen Variableninhaltes gearbeitet wird. Nähere Informationen finden sich in den Technical Reference Manuals von SQLWindows unter dem Stichwort "receive data-types".

Von großer Bedeutung sind Datentypen in Verbindung mit Variablen und Konstanten, da diese erst durch die Definition eines Datentyps verwendbar sind. Hierauf wird in den folgenden Kapiteln eingegangen.

1.3.3.2 Variable

Eine Variable ist ein Platzhalter für Information. Sie können eine Variable mit einem Gefäß vergleichen in das Sie etwas (nämlich Information) hineinpacken. Abgelegt wird diese Variable dann im Hauptspeicher Ihres Computers. Dies hat zwei Aspekte für Ihre Arbeit: Zum einen sind Variable während der Laufzeit Ihres Programms immer vorhanden, zum anderen sind sie sehr schnell verfügbar, da nicht auf der Festplatte nach ihnen gesucht werden muß.

Allerdings müssen Sie sich überlegen, wie Sie eine während der Laufzeit geänderte Variable so speichern können, daß sie später in der geänderten Form wieder zur Verfügung steht (Sie könnten z.B. eine Tabelle

anlegen, in die alle Variablen die Sie im Programm benötigen, abgespeichert werden; diese Tabelle sollte dann jedesmal bei Start des Programms gelesen werden).

Üblicherweise unterscheidet man Variable nach Datentyp (dem Inhalt) und Geltungsbereich:

SQLWindows kennt in Verbindung mit Variablen die bereits im vorstehenden Kapitel genannten **Datentypen**:

- String,
- Long String,
- Number,
- Date/Time,
- Boolean,
- Window handle,
- Sql handle,
- File handle.

Der **Geltungsbereich** bestimmt, wo und wie lange eine Variable (und damit ihr Inhalt), bekannt ist. Man unterscheidet folgende Geltungsbereiche:

- **global**, wenn Variable in einer kompletten Anwendung bekannt sind,

- **lokal**, wenn Variable nur innerhalb eines Teils der Anwendung bekannt sind (z.B. innerhalb einer einzelnen Funktion) und

- **static**, wenn Variable innerhalb einer Funktion definiert wurden und nur einmal mit einem Wert gefüllt werden. Das heißt, daß bei wiederholten Funktionsaufrufen immer wieder der gleiche Wert der Variablen benutzt wird.

Wo eine Variable definiert ist bestimmt also ihren Geltungsbereich. Die allgemeine Deklaration einer Variablen findet dabei in der folgenden Form statt:

Datentyp: Variablenname

Zunächst folgt ein Ausschnitt einer Variablendefinition aus einer SQLWindows-Anwendung.

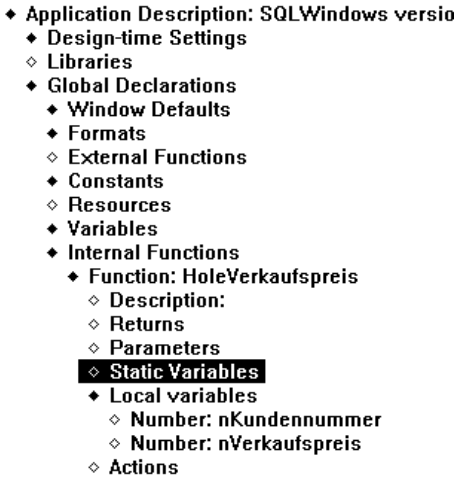
- Application Description: SQLWindows ver
  - Design-time Settings
  - Libraries
  - Global Declarations
    - Window Defaults
    - Formats
    - External Functions
    - Constants
    - Resources
    - Variables
      - String: strName
      - Long String: sLangZeichenKette
      - Number: nAnzahl
      - Date/Time: dtAnkunft
      - Boolean: bRechnung
      - Window Handle: hWndItem
      - Sql Handle: hSql
      - File Handle: hFileBuchungen
    - Internal Functions
    - Named Menus
    - Class Definitions
    - Application Actions
  - Form Window: frmMain

Diese Variablen haben, da sie zu Beginn der Anwendung unter "Global Declarations" definiert sind, einen globalen Geltungsbereich, d.h. sie sind in der gesamten Anwendung bekannt. Viele Anwender definieren ihre Variablen *nur* in dieser Sektion. Dies hat zwar den Vorteil, daß alle Variablen an einer Stelle der Anwendung zusammengefaßt sind, doch führt dies häufig dazu, daß Dutzende oder gar Hunderte von Variablen hier definiert werden. Dies ist zum einen unübersichtlich, zum anderen auch ungeschickt, da die Variablennamen besser nur dort erscheinen sollten, wo sie tatsächlich verwendet werden, z.B. am Kopf von Funktionen oder in der Sektion "Windows Variables". Neben der besseren Lesbarkeit spielt zudem die Geschwindigkeit einer Anwendung eine gewisse Rolle. Static-Variable sind besonders schonend, da sie nur wenig Aufwand durch das Programm verlangen, während globale Variable deutlich mehr Systemressourcen benötigen.

Der folgende Programmcode zeigt die Stelle, an der lokale Variable definiert werden können.

- Form Window: frmMain
  - Description:
  - Named Menus
  - Menu
  - Tool Bar
  - Contents
  - Functions
  - Window Parameters
  - Window Variables
    - String: strName

Schließlich soll dargestellt werden, wie die Definition von Variablen innerhalb einer selbst geschriebenen Funktion aussieht.



Sie sehen, daß die Art der Definition immer gleich ist, lediglich die Stelle ist entscheidend.

Variable in SQLWindows können an folgenden Stellen im Programm definiert werden:

- Sektion "Global Declarations" (globale Variable);.
- Sektion "Windows Variables" (lokale Variable) eines Formulars (z.B. "frmMain"), eines Tabellenfensters (z.B. "tbl1"), eines Dialog Fensters (z.B. "dlgAbout"), eines MDI-Windows oder eines Quest Table Windows;
- im Kopf einer selbst erstellten Funktion (Sektion "Global Declarations", "Internal Functions"), dies betrifft lokale und static Variable.

Daneben besteht noch die Möglichkeit, innerhalb von Klassen eigene Variable zu definieren, dies wird hier jedoch nicht weiter verfolgt.

Nachdem eine Variable definiert ist, wird für sie Speicherplatz reserviert. Sie enthält zunächst einen Null-Wert, bis ihr ein Wert zugewiesen wird. Diese Zuweisung geschieht mit dem Wort "Set". Die allgemeine Form einer Zuweisung lautet:

Set Variablenname = Wert

Also z.B.

- Set strName = 'Meier'
- Set nKundennummer = 12543

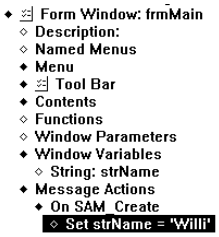
- Set dtAnkunft = '12-5-1992'
- Set bRechnung = TRUE

Natürlich müssen Sie auch hier wieder die Stelle bestimmen, an dem die Variable mit Inhalt gefüllt werden. Dieser Ort liegt sinnvollerweise *nach* der Definition der Variablen, legen Sie ihn davor, erhalten Sie eine Fehlermeldung.

Häufig werden Variable in der Sektion "Global Declaration", "Application Actions" mit einem Inhalt gefüllt. Dies gilt natürlich nur für die zuvor definierten globalen Variablen. Die folgende Abbildung zeigt diesen Sachverhalt.

Das Gesagte gilt entsprechend für die lokalen und static Variablen, die innherhalb des Moduls, in dem sie definiert wurden dann auch mit Inhalt versehen werden.

Der folgende Programmcode zeigt für ein Fenster mit dem Namen "frmMain", wie unmittelbar nach Erzeugung des *Fensters* ("On SAM_Create") eine lokale Variable "strName" mit einem Inhalt gefüllt wird.



Beachten Sie auch, daß Sie durch geschicktes Benutzen der Options Bar die Namen der Variablen nicht zu tippen brauchen: Durch Umschalten der Anzeige auf "Variables" können Sie sich dort, wo sonst vor allem Funktionen usw. angezeigt werden, auch Variablennamen anzeigen lassen und diese auch auswählen.

Neben den Variablen, die als einfache Möglichkeit betrachtet werden können, temporäre Daten zu erfassen und vorübergehend zu speichern, gibt es noch eine Variante, die man als **Array** (oder Datenfeld) bezeichnet.

Ebenso wie Variable werden auch Arrays zum Speichern von Informationen benutzt. Darüber hinaus bieten Arrays Eigenschaften, die sich nach gewisser Zeit als als deutliche Vorteile herausstellen.

Zunächst soll kurz erläutert werden, wie ein Array erzeugt wird und wozu er verwendet werden kann. Danach werden die Eigenschaften von Arrays kurz zusammengefaßt.

Arrays werden ähnlich wie Variable definiert, allerdings wird durch die Definition eines Arrays nicht eine einzige Variable angelegt sondern eine ganze Gruppe von Variablen (Elemente eines Arrays), die unter gleichem Namen angesprochen werden können. Zur Unterscheidung der einzelnen Elemente und auch für den Zugriff auf die eigentlichen Elemente eines Array wird eine laufende Nummer verwendet. Zunächst eine Definition eines Array:

String: saAnreden[10]

Mit der Datentypbezeichnung "String" wird ein Array vom Datentyp "String" angelegt. Der Name des Array lautet "saAnreden", wobei das "s" für String, das "a" für Array steht. Gedacht ist dieser Array dazu, verschiedene Anreden, also "Frau", "Herr", "Dr.", Prof." oder ähnliche Zeichenfolgen zu speichern (die maximale Anzahl an Zeichen für eine solche Bezeichnung ergibt sich aus dem verwendeten Datentyp). Die in *eckigen* Klammern eingefügte Zahl 10 bedeutet, daß das Array aus zehn Elementen bestehen soll. Dabei hat das erste Element des Arrays die Nummer 0, nicht 1, wie man das vermuten könnte, und das letzte Element die Nummer 9 (nicht 10). Falls das irritierend ist, bietet SQLWindows noch eine Möglichkeit, die es ermöglicht, die Zählung der Nummern nicht bei 0 sondern bei einer anderen Ziffer beginnen zu lassen. Die folgende Zeile erzeugt z.B. den gleichen Array wie die oben genannte, beginnt im Gegensatz dazu aber mit der Ziffer 1:

String: saAnreden[1;10]

Beachten Sie dabei das Semikolon zwischen den beiden Zahlen!

Gefüllt könnte ein solcher Array folgendermaßen aussehen:

Name, laufende Nummer	**Inhalt**
saAnreden[1]	'Frau'
saAnreden[2]	'Herr'
saAnreden[3]	'Dr.'
saAnreden[4]	'Prof.'

Nicht alle vorhandenen Elemente müssen mit Informationen gefüllt werden.

Neben der oben beschriebenen festen Vereinbarung der Anzahl von Elementen, kann auch eine variable Anzahl von Elementen vereinbart werden. Dies ist dann sinnvoll, wenn nicht bekannt ist, wieviel Variablen gebraucht werden. Die Definition verwendet dann anstelle einer Ziffer den Stern "*":

String: saKundenNamen[*]

Jetzt ist ein Array definiert, das eine unbegrenzte Anzahl von Elementen aufnehmen kann, man spricht von einem dynamischen Array. Dieser dynamische Array könnte z.B. verwendet werden, um Daten im Fremdformat zu lesen oder zu schreiben, wobei man nicht im voraus wissen kann, wieviel Spalten einer Tabelle zu lesen sind. Durch diese Flexibilität empfehlen sich Arrays besonders für das Schreiben von Funktionen in SQLWindows.

Es leuchtet ein, daß Sie durch die sorglose Verwendung von dynamischen Arrays die Speicherverwaltung Ihres Computers nicht eben erleichtern, weshalb Sie diese Art der Definition von Arrays mit der angebrachten Sorgfalt benutzen sollten.

Zu beachten ist, daß die in einem Array gespeicherten Daten nur temporär gespeichert werden. Um die Daten dauerhaft zu speichern, müssen entweder die in SQLWindows vorhandenen Funktionen zum Schreiben in DOS-Dateien verwendet werden, oder aber die Inhalte von Arrays in eine Tabelle einer Datenbank gespeichert werden (üblicherweise innerhalb einer Schleife, bei der die laufende Nummer des Array hochgezählt werden kann, um auf einzelne Elemente zuzugreifen; das ist durchaus nicht ungewöhnlich und wird von vielen Programmierern getan).

Wie werden nun die einzelnen Elemente eines Arrays mit Informationen gefüllt?

Die "klassische Methode" besteht darin, den einzelnen Elementen eines Arrays bestimmte Inhalte zuzuweisen. Analog zu dem vorher benannten Array saAnreden[10] könnten die einzelnen Elemente beispielsweise mit folgenden Anweisungen gefüllt werden:

```
Set saAnrede[0] = 'Frau'

Set saAnrede[1] = 'Herr'
```

Die anderen Elemente werden dabei analog mit Inhalten gefüllt. Beachten Sie dabei, daß die Zuweisung

```
Set saAnrede[10] = 'Mister'
```

zu der Fehlermeldung "Array contents out of bounds" führt, da das letzte Element dieses Arrays die Nummer 9 besitzt!

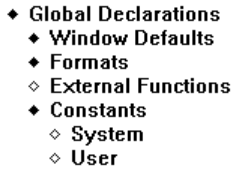
1.3.3.3 Konstanten

Konstanten enthalten, ähnlich wie Variable, einen Wert, allerdings ist dieser Wert im Gegensatz zu den Variableninhalten unveränderbar. Wird innerhalb einer Anwendung einer Konstanten der Wert 2000 zugewiesen, so kann an keiner Stelle dieser Anwendung dieser Wert geändert werden. Dies schränkt die Verwendungsfähigkeit von Konstanten scheinbar ein, doch gibt es einige Fälle, in denen gerade diese Eigenschaft der Unveränderlichkeit von Vorteil ist. Letztendlich ist für eine Anwendung die Arbeit mit Konstanten gerade durch das "Festschreiben" des Inhaltes einfacher und schneller als die Arbeit mit Variablen.

Konstanten können folgende Datentypen haben:

- String,
- Number,
- Date/Time,
- Boolean.

Auch die Stellen, an denen Konstanten definiert werden können, sind gegenüber Variablen eingeschränkt: Nur in der Sektion "Global Declarations", "Constants" können Konstanten angelegt werden. Der folgende Programmcode zeigt den betreffenden Ausschnitt:



Beachten Sie, daß es zwei Sektionen gibt, in denen eine Konstante definiert werden kann:

- Unter der Sektion **System** werden solche Konstanten vereinbart, die bei der Programmierung von Anwendungen für andere Systeme, wie z.B. OS/2 Verwendung finden. Sie werden hier nicht benötigt und deshalb nicht erläutert.

- In der Sektion **User** werden Konstanten vereinbart, die für "normale" Anwendungen benötigt werden, also alle, die in Anwendungen unter Windows verwendet werden.

Die Syntax dabei lautet:

Datentyp: Konstantenname = Ausdruck

Auffallend ist, daß hier der Datentyp und der Inhalt der Konstanten in nur einer Zeile festgelegt werden (dies ist bei Variablen nicht möglich).

So könnte es z.B. bei einem Modul, das Fremdformate erzeugt, sinnvoll sein, anstelle der Zeichen für Zeilenvorschub (LF) und des Carriage Return Zeichens (CR), eine Konstante einzuführen. Diese Konstante wäre dann in der gesamten Anwendung bekannt und wäre leichter zu lesen als der Aufruf einer Funktion mit der Nummer des ASCII-Zeichens. Die folgende Anweisung erzeugt Konstante, die Zeichen für die Enter-Taste (13) und für einen Zeilenvorschub (10) aufnehmen.

```
◆ Global Declarations
  ◆ Window Defaults
  ◆ Formats
  ◇ External Functions
  ◆ Constants
    ◇ System
    ◆ User
      ◇ Number: nEnter = 13
      ◇ Number: nVorschub = 10
```

Daneben können Konstanten erzeugt werden, die zur Erzeugung **eigener Message-Objekte** verwendet werden. Solche Nachrichten könnten Sie verwenden, um zusätzlich zu den vorhandenen SQLWindows Application Messages eigene Nachrichten zu erzeugen. Dies ist zwar für den Beginner noch etwas schwer vorstellbar, aber Sie werden beispielsweise recht bald auf das Problem stoßen, daß z.B. das Füllen eines Tabellenfensters mit den Daten einer Tabelle sehr oft codiert werden muß. Daher ist es oft sinnvoll, eine Funktion zu erstellen, die nach Aufruf immer genau diese Aktion ausführt. Hierzu brauchen Sie eine spezielle Botschaft, die genau von dieser Funktion empfangen werden kann, und somit benötigen Sie dafür eine Konstante. Der folgende Programmausschnitt zeigt Ihnen, wie eine solche Konstante erzeugt wird.

```
◆ Global Declarations
  ◆ Window Defaults
  ◆ Formats
  ◇ External Functions
  ◆ Constants
    ◇ System
    ◆ User
      ◇ Number: nEnter = 13
      ◇ Number: nVorschub = 10
      ◇ Number: AM_FILLTABLE = SAM_User + 1
```

Die Zeile "Number: AM_FILLTABLE = SAM_User + 1" erzeugt eine Konstante. Ungewöhnlich ist eigentlich nur die Verwendung des Ausdrucks "SAM_User + 1". die Erklärung hierfür ist, daß SQLWindows selbst eine Zahl benutzt, um eine eindeutige Nummer für Ihre neue Konstante zu erzeugen. "SAM_User" beinhaltet die Nummer der letzten *vorhandenen* Konstanten von SQLWindows. In der vorliegenden Version von SQLWindows ist das die Nummer 16384; zukünftige Versionen werden sicher noch mehr (interne) Konstanten haben. Erhöhen Sie den Wert dieser Konstanten um die Zahl 1, haben Sie, unabhängig von der Zahl der von SQLWindows verwendeten Konstanten, immer eine neue Konstante und kommen nicht in Konflikt mit SQLWindows.

> **Anmerkung**: Um herauszufinden, welche Zahl sich hinter SAM_User verbirgt, können Sie ein Data Field vom Datentyp Numerisch erzeugen, und anweisen, daß der Inhalt von SAM_User anzuzeigen ist. Der Programmcode hierfür lautet:

```
◆ Data Field: df1
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Set df1 = SAM_User
```

Üblicherweise werden Konstanten in Großbuchstaben geschrieben, aber dies ist nicht zwingend vorgeschrieben.

Falls Sie eine Konstante erzeugen wollen, in der ein Pfad z.B. für die Speicherung von Daten abgelegt werden soll, müssen Sie beachten, daß der Backslash als Escape-Sequenz verwendet wird. Um nun den Backslash als Teil eines Pfades zu codieren, muß er doppelt genannt werden, damit SQLWindows (genauso wie die Sprache C) ihn auch als (einzelnes) Zeichen einfügt. Die Codierung eines Pfades als Konstante hat dann folgendes Form:

String: ARCHIVIERUNG = 'D:\\ FIRMA\\SICHER.TXT'

Beachten Sie, daß der Backslash bei jeder Nennung und nicht nur bei der ersten Nennung doppelt einzugeben ist.

1.3.3.4 Statements

Statements (oder Kontrollstrukturen) dienen der Steuerung von Programmen. Sie existieren in allen Programmiersystemen. Wie bereits eingangs erwähnt, handelt es sich bei SQLWindows um eine ereignisgesteuerte Art der Programmierung, die mit Nachrichten arbeitet wie z.B. SAM_Create (bei Erzeugung eines Objekts) oder SAM_Click (bei Klicken auf einem Push Button). Deshalb treten die üblicherweise vorhandenen Kontrollstrukturen (z.B. Schleifen oder Fallunterscheidungen) etwas in den Hintergrund. Dies hat mit deren Bedeutung nichts zu tun, da Schleifen z.B. unverzichtbar sind, doch werden sehr viel häufiger Konstruktionen benutzt, die mit "SAM*", also unter Verwendung von SQLWindows Application Messages codiert werden (s. dazu auch den Anhang).

Die folgenden Abschnitte beschreiben die in SQLWindows vorhandenen Statements. Zur Erläuterung werden kurze Programmteile genannt.

1.3.3.4.1 Set

Set ist bereits aus den vorhergehenden Kapiteln bekannt. Set dient vor allem dazu, einer Variablen einen Inhalt zuzuweisen; der Befehl kann aber auch verwendet werden, um einem Handle einen Inhalt zuzuweisen.

Die allgemeine Syntax lautet:

Set Variable = Ausdruck

Um der Variablen nMehrwertsteuer einen Inhalt zuzuweisen, ist folgender Programmteil nötig:

Set nMehrwertsteuer = 15.0

Um einem Handle einen Inhalt zuzuweisen (das passiert typischerweise bei Aufruf eines Fensters über eine Menüoption), kann folgende Konstruktion verwendet werden:

Set hWndKunden = SalCreateWindow(frmKunden, hWndItem)

Dies ist immer dann nötig, wenn ein Fenster (hier: frmKunden) über einen Menüpunkt aufgerufen werden soll. Sie haben dann das Problem, daß Sie den Menüpunkt nicht auf die gleiche Weise deaktivieren können wie dies bei einem Push Button möglich ist. Sie müssen hierfür einen Window Handle benutzen.

1.3.3.4.2 If, Else, Else If

Die in allen Programmiersprachen vorhandene If-Anweisung dient dazu, einfache Fallunterscheidungen durchzuführen. Das Wort "if" kann dabei übersetzt werden mit dem Wort "falls". Im Programmablauf dient die Verwendung dieses Ausdrucks der Verzweigung in zwei (oder mehr) verschiedene Varianten. So könnte abhängig von einer logischen Variablen entschieden werden, ob es sich bei einem Auftrag um einen Lieferschein oder eine Rechnung handelt. Abhängig von einer solchen logischen Variablen wären dann bei einem Lieferschein nur die Mengen, bei einer Rechnung zusätzlich die Preise anzuzeigen. Häufig werden als Quelle der Information für solche Fallunterscheidungen logische Felder benutzt, also Check Boxes, Radio Buttons oder ähnliches; es können aber auch berechnete Ausdrücke dafür verwendet werden. Wichtig ist nur, daß es sich bei dem in der if-Anweisung angegebenen Ausdruck um einen logischen Wert handelt.

Zur Erläuterung wird ein kleines Programm vorgestellt, das aus drei Objekten besteht: einem Push Button ("pb1") und zwei Data Fields ("df1" und "df2"). Nach dem Drücken des Push Buttons soll der Inhalt des ersten Data Fields ausgewertet werden, falls dort das Wort 'Hans' enthalten ist, soll im zweiten Data Field die Zeichenfolge 'hallo Hans' angezeigt werden. Dabei wird der in SQL bekannte Operator zur Verkettung von Zeichenfolgen verwendet: "||".

Der folgende Ausschnitt zeigt die Verwendung der if-Anweisung:

```
◆ Form Window: frmMain
  ◇ Description:
  ◇ Named Menus
  ◆ Menu
  ◆ Tool Bar
  ◆ Contents
    ◆ Data Field: df1
      ◇ Message Actions
    ◆ Data Field: df2
      ◇ Message Actions
    ◆ Pushbutton: pb1
      ◆ Message Actions
        ◆ On SAM_Click
          ◆ If df1 = 'Hans'
            ◇ Set df2 = 'hallo '|| df1
```

Erst durch Drücken des Push Buttons ("On SAM_Click") wird die Auswertung mittels der if-Anweisung gestartet.

Die vorstehende if-Anweisung kann sehr einfach durch weitere if- Zweige erweitert werden:

```
◆ On SAM_Click
  ◆ If df1 = 'Hans'
    ◇ Set df2 = 'hallo '|| df1
  ◆ If df1 = 'Otto'
    ◇ Set df2 = 'guten Morgen '|| df1
```

oder durch Einführen einer zusätzlichen Else If-Anweisung die Variante "andernfalls" erweitert werden. Dadurch wird in dem Fall, daß weder Hans noch Otto im ersten Feld eingegeben wird, die Zeichenfolge "nicht autorisiert" im zweiten Data Field angezeigt.

```
◆ On SAM_Click
  ◆ If df1 = 'Hans'
    ◇ Set df2 = 'hallo '|| df1
  ◆ Else If df1 = 'Otto'
    ◇ Set df2 = 'guten Morgen '|| df1
  ◆ Else
    ◇ Set df2 ='nicht autorisiert'
```

Es gibt beliebig viele Möglichkeiten, If -, Else - und Else If - Anweisungen zu kombinieren. Sie sollten diese Konstruktionen stets genau kommentieren, da man dabei leicht den Überblick verlieren kann.

1.3.3.4.3 Select Case

Neben Konstruktionen, die unter Verwendung von If-Anweisungen aufgebaut sind, besteht auch die Möglichkeit, mehrere Fälle mit einer "Select Case"-Konstruktion auseinanderzuhalten. Im folgenden Beispiel soll eine Kundengruppe ausgewertet werden. Abhängig von der einem Kunden zugeordneten Kundengruppe könnten verschiedene Rabattsätze zugewiesen werden, hier angedeutet durch "...". Falls Sie das Beispiel nachvollziehen möchten, vergessen Sie nicht die Data Fields "df1" und "df2" auf den Datentyp "numeric" umzustellen.

Die Syntax lautet:

```
Select Case df1
        Case 1
        ...
        Case 2
        ...
        Case 3
        ...
        Default
        ...
```

Der folgende Programmausschnitt zeigt die Verwendung der "Select Case" - Konstruktion. Dabei wurden ein Push Button und zwei Data Fields verwendet.

Der Benutzer gibt in das erste Data Field ("df1") eine Ziffer ein, nach Drücken des Push Buttons ("pb1") wird in dem zweiten Data Field ("df2") der Rabattsatz angezeigt.

- Contents
  - Data Field: df1
    - Message Actions
  - Data Field: df2
    - Message Actions
  - Background Text: Rabattsatz
  - Background Text: Kundengruppe
  - Pushbutton: pb1
    - Message Actions
      - On SAM_Click
        - Select Case df1
          - Case 1
            - Set df2 = 10
            - Break
          - Case 2
            - Set df2 = 20
            - Break
          - Case 3
            - Set df2 = 30
            - Break
          - Default
            - Set df2 = 0

Beachten Sie, daß nach jeder Verzweigung das Wort "break" angeordnet ist. Nur durch Verwendung von "break" wird die "Select Case" - Anweisung an dem entsprechenden Zweig auch verlassen. Lassen Sie "break" weg, werden alle Zweige bis zum Ende der "Select Case" - Anweisung durchlaufen. Dies kann z.B. zur Zusammenfassung mehrerer Zweige verwendet werden. Im vorliegenden Beispiel wäre dies folgendermaßen möglich:

```
Select Case df1
        Case 1
        Case 2
        Case 3
                Set df2 = 30
                Break
        Case 4
                Set df2 = 40
                Break
        Default
                Set df2 = 0
```

Bei dieser Anordnung wird der Kundengruppe 1-3 der gleiche Rabattsatz zugewiesen (30), der Kundengruppe 4 dagegen ein Rabattsatz von 40. Falls keine Kundengruppe angegeben wurde, wird 0 zugewiesen.

Das Wort "Default" am Ende der "Select Case" - Anweisung bestimmt eine Anweisung, die nur dann ausgeführt wird, wenn keine der vorstehenden Bedingungen erfüllt worden ist. Die Default-Anweisung kann auch entfallen.

Beachten Sie, daß "Select Case" nur numerische Ausdrücke auswerten kann!

1.3.3.4.4 Call

Call wird in SQLWindows immer dann verwendet, wenn eine Funktion aufgerufen werden soll. Die allgemeine Syntax dabei lautet:

Call Funktionsname

Im folgenden Beispiel soll die Funktion "SalCreateWindow()" dazu benutzt werden, das Fenster mit dem Namen "frm2" nach einem Klick auf einem Push Button zu öffnen:

```
On SAM_Click
        Call SalCreateWindow(frm2, hWndNULL)
```

Beachten Sie, daß innerhalb der Klammern der Funktion "SalCreateWindow()" nicht nur der Name des Form Windows ("frm2") sondern auch der "Eigentümer" zu nennen ist. Hier wurde als Eigentümer **hWndNULL** angegeben. Das ist ein in SQLWindows bereits bekannter Name, der für *keinen* Eigentümer steht. Das heißt, daß dieses Fenster ein "Eigenleben" hat. Schließen Sie eine Anwendung, in der solchermaßen Fenster geöffnet wurden, dann werden diese nicht automatisch mitgeschlossen (das passiert häufig aus Versehen), sondern tauchen in der Task-Liste von Windows auf. Sie müssen daher separat mit Alt+F4 geschlossen werden.

Daneben kann Call benutzt werden, eine SQL-Anweisung ausführen zu lassen: so wird in dem folgenden Programmteil das SQL-Kommando 'commit' ausgeführt, das dazu dient, die letzten Änderungen unwiderruflich zu sichern. Dafür wird die Funktion "SqlImmediate()" verwendet, mit der eine Sql-Anweisung unmittelbar ausgeführt werden kann.

```
On SAM_Click
        Call SqlImmediate('commit')
```

In ähnlicher Form kann eine Konstruktion verwendet werden, um die letzten Änderungen rückgängig zu machen:

```
On SAM_Click
        Call SqlImmediate('rollback')
```

Die Schreibweise aller Worte (abgesehen der eigentlichen Sql-Anweisungen) muß mit der hier angezeigten übereinstimmen.

1.3.3.4.5 On

On dient dazu, eine Aktion nach einem bestimmten Ereignis eine Aktion ausführen zu lassen. Durch ein Ereignis (z.B. das Drücken eines Push Buttons) muß eine Nachricht verschickt worden sein, die innerhalb der Sektion "Message Actions" von genau diesem Objekt (z.B. des Push Buttons) empfangen und behandelt wird.

Die allgemeine Syntax lautet:

On message
        machirgendwas

Beachten Sie, daß die zweite Zeile eingerückt sein muß. Erst durch das Einrücken, wird angewiesen, daß die dort stehenden Anweisungen auszuführen sind. Natürlich können dort beliebig viele Aktionen angeordnet sein.

Das folgende Programmteil zeigt die Sektion "Message Actions" eines Push Buttons. Nach dem Klicken des Push Buttons ("On SAM_Click") soll eine Dialog Box mit dem Namen "dlgAbout" geöffnet werden (Call SalCreateWindow(dlgAbout)). Außerdem soll der Push Button nach dem Öffnen des Fensters deaktiviert werden, damit das Fenster nicht mehrfach geöffnet wird (falls Sie das Beispiel nachvollziehen wollen, sollten Sie die Eigenschaft "Automatically create" des Fensters auf "No" setzen, damit das Fenster nicht automatisch geöffnet wird, sondern erst dann, wenn Sie den Push Button anklicken.

```
◆ Pushbutton: pb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalCreateWindow( frm2, hWndItem )
      ◇ Call SalDisableWindow( pb1 )
```

Eingerückt unter dem Statement "On" befinden sich die zwei Zeilen, die nach Drücken des Push Buttons auszuführen sind. Innerhalb der Funktion "SalCreateWindow()" ist neben dem Namen des zu öffnenden Fensters auch der Name des "Eigentümers" anzugeben. Hier wurde "hWndItem" gewählt, das heißt, das aktuell geöffnete Fenster (z.B. "frmMain") soll der Eigentümer sein. Der Vorteil hierbei ist, daß der Name des Fensters nicht einzugeben ist. Wird nun das Fenster ("frmMain", der Eigentümer) geschlossen, werden alle Fenster, die diesem Fenster zugeordnet sind automatisch mitgeschlossen. Die zweite Zeile "Call SalDisableWindow(pb1)" deaktiviert den Push Button. In Klammer angegeben ist dabei der Name des Objekts, das zu deaktivieren ist. Anstelle von "pb1" hätte auch "hWndItem" stehen können, da der Push Button im Moment des Klickens zum aktuellen Objekt wurde.

1.3.3.4.6 While, Loop

Schleifen sind Programmelemente, die benötigt werden, um einen Programmteil wiederholt ausführen zu lassen. Sie werden in SQLWindows benutzt, z.B. um eine List Box mit den Elementen eines Array zu füllen.

Schleifen bestehen aus einem Schleifenkörper (der eigentlichen Schleife) und einem Schleifeninhalt, der die eigentlichen Anweisungen enthält. Bei diesen Anweisungen kann es sich um beliebige Sprachelemente der Sprachen SAL, SQL (oder eingebetteter Funktionen der Sprache C) handeln.

SQLWindows kennt zwei verschiedene Schleifen: While und Loop. Welche der beiden Schleifen Sie benutzen ist in vielen Fällen unerheblich, es gibt jedoch einen grundsätzlichen Unterschied, der beachtet werden muß: Bei einer While-Schleife wird die Bedingung, die zur Steuerung der Schleife formuliert ist, ausgewertet, *bevor* die in der Schleife enthaltenen Anweisungen ausgeführt werden. Bei einer Loop-Schleife dagegen *kann* die Bedingung so angeordnet werden, daß die Anweisungen innerhalb der Schleife wenigstens einmal durchlaufen werden.

Beide Schleifen werden kurz vorgestellt. Dabei soll nach dem Drücken eines Push Buttons der Signalton (das Beep) dreimal ausgeführt werden. Zu diesem Zweck wird in beiden Schleifen eine Variable mit dem Namen

nAnzahl vom Typ Numerisch benutzt, die unter "Global Actions", "Variables" definiert wurde (diese Definition wird in der folgenden Abbildung nicht angezeigt).

Der folgende Ausschnitt zeigt die Lösung unter Verwendung einer While-Schleife:

```
◆ Pushbutton: pb2
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Set nAnzahl = 0
      ◆ While nAnzahl < 3
        ◇ Call SalMessageBeep( 0 )
        ◇ Set nAnzahl = nAnzahl +1
```

Innerhalb der Sektion "Message Actions" des Push Buttons "pb2" wird das Drücken des Push Buttons (On SAM_Click) ausgewertet. Zunächst wird die Variable nAnzahl auf 0 gesetzt, danach folgt der Schleifenkörper "While nAnzahl < 3", der anweist, daß der Schleifeninhalt solange zu wiederholen ist, wie die Variable nAnzahl kleiner als 3 ist. Der Funktionsaufruf Call SalMessageBeep(0) ruft die für das Beep nötige SAL-Funktion auf und führt sie aus. Danach wird der Inhalt der Variablen nAnzahl um 1 erhöht. Hierdurch kehrt die Ausführung der Schleife auf die erste Zeile der Schleife zurück, prüft die Bedingung (nAnzahl < 3) und führt die Anweisungen innerhalb der Schleife erneut aus.

Nachdem die Variable nAnzahl auf den Wert 3 erhöht wurde, liefert die Bedingung "nAnzahl < 3" den logischen Wert "FALSE", dies führt zum Verlassen der Schleife, eventuelle Programmteile nach While (auf der gleichen Ebene) können dann ausgeführt werden.

Die gleiche Aufgabe, mit der Loop-Schleife gelöst, könnte folgendermaßen aussehen:

```
◆ Pushbutton: pb2
  ◆ Message Actions
    ◆ On SAM_Click
      ◆ Loop
        ◆ If nAnzahl = 3
          ◇ Break
        ◇ Call SalMessageBeep( 0 )
        ◇ Set nAnzahl = nAnzahl + 1
```

Insgesamt müssen mehr Zeilen geschrieben werden, um das Gleiche zu erreichen (5 gegenüber 3 Zeilen). Auffallend ist, daß die Bedingung, die zum Abbruch der Schleife führt (If nAnzahl =3), innerhalb der Schleife steht. Die If-Anweisung mit eingerücktem "Break" führt zum Abbruch der Schleife (das Wort Break ist hier nötig, dagegen braucht es bei der While-Schleife nicht benutzt werden). Würde man diese Bedingung nach unten, d.h. hinter die Call bzw. Set Anweisung verlegen, würden die Anweisungen auf jeden Fall einmal durchlaufen werden, bevor es zum Abbruch käme.

Zusammenfassend kann festgestellt werden, daß es keinen grundlegenden Unterschied macht, welche der beiden Schleifen Sie benutzen. Die While-Schleife bietet jedoch dadurch, daß die Bedingung bereits in der ersten Zeile steht und daß Sie eher unserer Umgangssprache ähnelt, leichte Vorteile gegenüber der Loop-Schleife.

1.3.3.4.7 Return

Return wird dazu benutzt, einen Wert zurückzuliefern. Return kann bei folgenden Modulen benutzt werden:

- Ende einer Funktion,
- innerhalb einer "Message Action"-Sektion und
- in einer Menü-Aktion.

Im folgenden Beispiel soll eine Funktion vorgestellt werden, die einen Wert zurückliefert. Zunächst soll der Text einer internen, d.h. durch den Benutzer zu erstellenden, Funktion vorgestellt werden, die das Quadrat einer übergebenen Zahl berechnet und diesen Wert zurückliefert.

```
◆ Function: Quadrat
  ◇ Description: Berechnet das Quadrat einer Zahl
  ◇ Returns
  ◆ Parameters
    ◇ Number: nEingabe
  ◇ Static Variables
  ◆ Local variables
    ◇ Number: nErgebnis
  ◆ Actions
    ◇ Set nErgebnis = (nEingabe*nEingabe)
    ◇ Return nErgebnis
```

Die Funktion mit dem Namen "Quadrat" hat einen Parameter (nEingabe), in dem bei Aufruf der Funktion eine Zahl abgespeichert wird. Daneben wird eine lokale Variable innerhalb der Funktion benutzt (nErgebnis), in die das Ergebnis der Berechnung gespeichert wird. Der Inhalt dieser Variablen wird als Rückgabewert dorthin zurückgeliefert, wo die Funktion aufgerufen wurde.

In dem Fenster "frmMain" wurden zwei Data Fields (df1, df2) und ein Push Button (pb1) untergebracht. In das Data Field "df1" muß nun eine Zahl eingegeben werden. Durch das Drücken des Push Buttons wird die Funktion aufgerufen. Dieser Aufruf ist so zu verstehen, daß an die Funktion der Inhalt des Data Field "df1" übergeben wird. Der Rückgabewert der Funktion wird sofort an das zweite Data Field (df2) übergeben. Der dafür notwendige Programmtext sieht folgendermaßen aus:

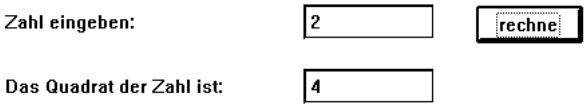
Aufruf einer internen Funktion, die über Return einen Wert zurückliefert.

```
◆ Pushbutton: pb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Set df2 =  Quadrat( df1 )
```

Die einzige Anweisung, die nötig ist, das in der Funktion berechnete Ergebnis an das Data Field "df3" zuzuweisen, lautet: "Set df2 = Quadrat (df1)".

Die Oberfläche zu diesem Programm könnte folgendermaßen gestaltet werden:

Abb. 1.16: Oberfläche des Programms zur Berechnung einer Quadratzahl:



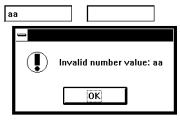
Wenn auch für diese Berechnung nicht unbedingt eine Funktion mit einem Rückgabewert nötig ist, führt doch diese Art der "Auslagerung von Programmcode" zu deutlich kürzeren Programmen, die leichter lesbar sind. Hinzu kommt, daß solche Funktionen immer wieder benutzbar sind und daß eventuelle Änderungen nur an einer Stelle im Programm vorzunehmen sind.

In der Praxis könnte diese Art der Codierung z.B. dort Verwendung finden, wo ein Preis für einen Artikel gebraucht wird, die Preise aber abhängig von einer Kundengruppe und zusätzlichen Sonderpreisen zu wählen sind (so wie das oben schon angedeutet wurde). Das hätte zur Folge, daß in einer solchen Funktion die hierfür nötigen Sql-Anweisungen untergebracht werden und über "Return" der zu einem Artikel gefundene Preis (oder 0, falls kein Preis zu finden war) zurückgegeben bzw. in das entsprechende Feld eingetragen wird.

Neben der Möglichkeit, eine Funktion mit Rückgabewert zu benutzen, soll noch eine weitere Variante vorgestellt werden, in der mit Return gearbeitet wird, nämlich bei der Behandlung von Fehlern, die zur Laufzeit des Programms auftreten.

Dazu werden zwei Data Fields benötigt (df1, df2), die vom Datentyp Numerisch sein sollen (den Datentyp umstellen, indem mit der rechten Maustaste auf das Data Field geklickt wird, Data Type "Number" auswählen). Starten Sie das Programm und geben in das erste Data Field die Zeichenfolge "hallo" ein, drücken Sie dann die Tabulatortaste, um in das zweite Data Field zu gelangen. Sie erhalten nun eine Fehlermeldung von SQLWindows, da die Eingabe ("hallo") nicht mit dem zuvor vereinbarten Datentyp (Number) übereinstimmt.

Abb. 1.17: Fehlermeldung von SQLWindows.



Diese Fehlermeldung ist von SQLWindows selbst. Sie ist vorformuliert und sollte nicht an den Benutzer weitergegeben werden. Deshalb wird eine Konstruktion erzeugt, die eine passablere Fehlermeldung anzeigt und den Benutzer gleichermaßen davon abhält, ein Data Field mit einer falschen Eingabe zu verlassen. Dafür

wird der Begriff **SAM_Validate** benutzt, eine SQLWindows Application Message, die dann verschickt wird, wenn der Benutzer das Feld verlassen möchte. Außerdem wird eine Funktion aufgerufen, die prüft, ob es sich bei der Eingabe um eine Zahl handelt; diese Funktion hat den Namen **SalIsValidNumber()** (gleichartige Funktionen gibt es auch für die Prüfung, ob eine Zeichenkette, ein Datum usw. eingegeben wurde, Sie können nach diesen Funktionen, die alle mit "SalIsValid" beginnen, suchen lassen).

Schließlich wird eine Funktion mit dem Namen **SalMessageBox()** aufgerufen, die ein Fenster mit der vom Benutzer festgelegten Fehlermeldung anzeigt.

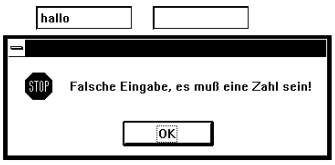
Der folgende Programmteil zeigt die nötige Konstruktion.

```
◆ Data Field: df1
  ◆ Message Actions
    ◆ On SAM_Validate
      ◆ If NOT SalIsValidNumber(df1)
        ◇ Call SalMessageBox('Falsche Eingabe, es muß eine Zahl sein!',
                             'Fehlermeldung',
                             MB_Ok | MB_IconStop )
        ◇ Return VALIDATE_Cancel
```

Innerhalb der Message Actions Sektion von "df1" wird mittels "On SAM_Validate" die Eingabe geprüft. Falls es sich nicht um eine Zahl handelt, wird eine Message Box angezeigt. Der Funktion "SalMessageBox()" werden drei Argumente übergeben (der anzuzeigende Text, die Titelzeile des Fensters und die in SQLWindows bekannten Message Buttons, die zur Anzeige des Stop-Zeichens bzw. des OK Push Buttons führen).

Zurückgegeben wird die in SQLWindows bekannte Konstante VALIDATE_Cancel. Sie hat den Wert 0 (was einem logischen "FALSE" entspricht) und verhindert deshalb das Weitergehen in das zweite Feld. Die Fehlermeldung selbst hat folgendes Aussehen:

Abb. 1.18: Selbst erzeugte Fehlermeldung.

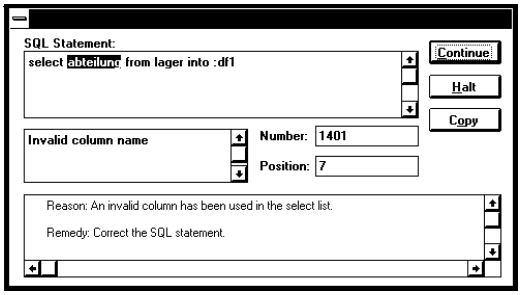


Diese Fehlermeldung entspricht den üblichen Vorstellungen von Benutzern.

1.3.3.4.8 When SqlError

Falls in Ihrer Anwendung Fehler infolge fehlerhafter Sql-Anweisungen auftreten, reagiert SQLWindows darauf mit vorgefertigten Fehlermeldungen. Eine solche Fehlermeldung könnte z.B. folgendermaßen aussehen:

Abb. 1.19: Fehlermeldung in SQLWindows.



Diese Art der Fehlermeldung wird jedoch häufig nicht gewünscht, da eigene Vorstellungen bzw. Wünsche des Benutzers zu berücksichtigen sind. Deshalb sollte der Entwickler selbst die Behandlung und Darstellung von Fehlermeldungen übernehmen.

Grundsätzlich gibt es hierfür zwei Möglichkeiten: Sie können Fehler global, das heißt an zentraler Stelle Ihrer Anwendung (in der Sektion Application Actions), oder lokal, das heißt, genau an der Stelle, an der ein Fehler aufgetreten ist (z.B. innerhalb der Sektion "Message Actions" von einem Push Button), behandeln und darauf reagieren.

Ein kurzes Beispiel soll demonstrieren, wie unter Einsatz des Ausdrucks When SqlError ein Sql-Fehler in der Anwendung aufgefangen werden kann. Dabei soll ein Fehler dadurch erzeugt werden, daß ein Data Field mit dem Inhalt einer nicht vorhandenen Tabellenspalte gefüllt werden soll.

Zunächst müssen ein Data Field (df1) und ein Push Button (pb1) in ein leeres Formular eingefügt werden. Nachdem der Push Button gedrückt wird soll die Funktion SqlImmediate() aufgerufen werden, die aus einer (nicht vorhandenen) Tabelle einen Inhalt aus einer (ebenfalls nicht vorhandenen) Spalte anzeigen soll.

Hierzu fügen Sie in der Sektion "Message Actions" des Push Buttons folgende Zeilen ein:

```
◆ Pushbutton: pb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◆ When SqlError
        ◇ Call SalMessageBox('Es ist ein Fehler aufgetreten',
          'Fehlermeldung',
          MB_Ok | MB_IconStop)
      ◇ Call SqlImmediate('select spaltenichtda from tabellenichtda into :df1')
```

Die Funktion SqlImmediate() wird mit einer SQL-Anweisung aufgerufen, die mit falschen Namen für eine Tabelle bzw. für eine Spalte zu einer Fehlermeldung führt.

Hier soll dieser kurze Hinweis genügen. Ausführliche Hinweise zur Fehlerbehandlung finden Sie im Handbuch "Technical Reference Manual" unter dem Stichwort "Error Handling".

1.3.3.5 Funktionen

Wie jede Programmiersprache enthält auch SAL (SQLWindows Application Language), die Programmiersprache von SQLWindows, einen Vorrat von bereits definierten Funktionen. SQLWindows unterscheidet dabei die folgenden 5 Kategorien:

- **SQLWindows System Funktionen**: Funktionen, die im Lieferumfang von SQLWindows enthalten sind. Sie dienen entweder dem Zugriff auf Datenbanken (dann beginnen Sie immer mit "Sql"), oder allgemeinen Zwecken in SQLWindows, wie z.B. dem Öffnen oder Schließen von Fenstern (diese Funktionen beginnen mit "Sal").

- **Interne Funktionen**: Funktionen, die durch den Entwickler unter Verwendung der vorhandenen Sprachelemente von SQLWindows erstellt werden.

- **Window Funktionen**: Funktionen, die zwar ähnlich wie interne Funktionen unter Verwendung der Sprachelemente von SQLWindows erstellt werden, die aber innerhalb von Windows (z.B. Form Windows) codiert sind. Ihr Charakter ist im Gegensatz zu internen Funktionen, als lokal zu bezeichnen, worin sie den in einem Fenster definierten lokalen Variablen ähneln.

- **Klassen Funktionen**: Klassen-Funktionen sind innerhalb einer Klasse definiert und haben in Verbindung mit Objektorientiertem Programmieren (OOP) große Bedeutung.

- **Externe Funktionen**: Externe Funktionen sind häufig in C (oder anderen Sprachen wie Pascal) erstellte Funktionen. Sie liegen eingebettet in DLL-Dateien (Dynamic Link Libraries) vor und müssen in der Sektion "External Functions" deklariert werden. Über solche DLL-Dateien kann auch auf die Funktionen des Windows SDK (System Development Kit) zugegriffen werden, z.B. um eigene Hilfe-Module für SQLWindows-Anwendungen zu erstellen.

Im folgenden soll kurz auf System- und interne Funktionen eingegangen werden. Window-Funktionen unterscheiden sich nur durch die Art ihrer Gültigkeit von ersteren und werden deshalb nicht gesondert vorgestellt. Klassen- und externe Funktionen werden ebenfalls nicht vorgestellt, da sie fortgeschrittene Kenntnisse voraussetzen.

Eine Übersicht der Funktionen - nach Themen gegliedert - finden Sie im Anhang dieses Buches sowie in den Handbüchern *Function Reference* sowie in den im Lieferumfang von SQLWindows enthaltenen Handbüchern *Technical Reference Manual (Band 1, 2)*,diese Handbücher sind alle im Lieferumfang von SQLWindows enthalten.

Einige Funktionen werden hier kurz erläutert, damit ein Gefühl für die Sprache entwickelt werden kann. Die Funktionen werden in den folgenden Kapiteln im Zusammenhang mit der Vorstellung von Objekten in SQLWindows und dem Entwickeln von Anwendungen erläutert.

Folgende Funktionen dienen in erster Linie dem Erzeugen und Aktivieren von Objekten, sowie dem Füllen von Tabellenobjekten und dem Drucken von Reports.

Funktionsname	Verwendung
SalCreateWindow()	Öffnet ein Fenster.
SalDestroyWindow()	Schließt ein Fenster.
SalTblPopulate()	Füllt ein Tabellenobjekt mit Daten, z.B. aus einer Tabelle.
SalTblDefineSplitWindow()	Teilt ein Tabellenobjekt waagerecht, damit im unteren Teil neue Datensätze eingegeben werden können.
SalTblInsertRow()	Fügt in eine Tabelle einen neuen (leeren) Datensatz ein.
SalTblSetFocusCell()	Setzt die Schreibmarke in eine anzugebende Zelle eines Tabellenobjekts.
SalReportCreate()	Legt eine Template Datei an, über die z.B. Datensätze aus einer Tabelle gedruckt werden können.
SalReportView()	Zeigt einen Report in der Seitenvorschau.
SalReportPrint()	Druckt einen Report.
SalDisableWindow()	Deaktiviert ein Objekt (z.B. einen Push Button).
SalEnableWindow	Aktiviert ein Objekt.
SalListPopulate()	Fügt Elemente in eine List Box hinzu.

Folgenden Funktionen betreffen den Zugriff auf Datenbanken.

Funktionsname	Verwendung
SqlConnect()	Stellt eine Verbindung zur Datenbank her.
SqlDisconnect()	Trennt die Verbindung zur Datenbank.
SqlImmediate()	Läßt eine Sql-Anweisung sofort ausführen; beinhaltet SqlPrepare() und SqlExecute().
SqlPrepare()	Bereitet eine Sql-Anweisung für die Ausführung vor.
SqlExecute()	Führt eine vorbereitete Sql-Anweisung aus.

Die hier genannten Funktionen werden in den folgenden Kapiteln näher beschrieben und im Zusammenhang erläutert (s. auch Anhang mit der Übersicht *aller* Funktionen, nach dem Zusammenhang geordnet).

1.3.3.6 Operatoren

Operatoren dienen dazu, zwei (oder mehr) Operanden miteinander zu verknüpfen, so daß ein neuer Ausdruck oder Wert entsteht. Neben den üblicherweise zu unterscheidenden arithmetischen Operatoren, logischen Operatoren und Vergleichsoperatoren treten in SQLWindows noch einige weniger bekannte Operatoren hinzu.

Folgende Übersicht faßt die in SQLWindows bekannten Operatoren zusammen.

Operator	Zeichen	Erklärung	Beispiel
Numerisch	+ - * /	Addition Subtraktion Multiplikation Division	dfGesamt = dfWert1 + dfWert2 dfNetto = dfBrutto -dfMwst dfGesamt = dfAnzahl * dfBetrag dfEinzel = dfGesamt / dfAnzahl
Unär	-	Negation	- dfBetrag

Relational	= != > < >= <=	ist gleich nicht gleich größer als kleiner als größer oder gleich kleiner oder gleich	If nAnzahl = 3 " " " " "
Boole	AND OR NOT	logisches und logisches oder logisches nicht	If df1 > 2 AND df1 < 8 If df1 = 100 OR df1 = 200 "
Bitweise	& |	Bitweise AND Bitweise OR	Vergleich der Inhalte von zwei Bits "
String	||	Verkettung	dfName = dfVorname || dfNachname

Die Reihenfolge der Auswertung von Operatoren in Anweisungen erfolgt immer von links nach rechts. Um eine andere Reihenfolge bei der Auswertung zu erzwingen, können Sie Klammern verwenden.

1.3.3.7 Kommentare

In SQLWindows gibt es nur ein Kommentarzeichen, nämlich das Ausrufezeichen ("!"). Sie können dieses Zeichen an den Beginn einer Zeile setzten, um damit die ganze Zeile zu einem Kommentar zu machen:

! Dies ist eine Kommentarzeile

oder Sie plazieren das Ausrufezeichen nach einer ausführbaren Anweisung, um wichtige Programmteile zu kommentieren:

Call SalCreateWindow(frmKunden) **! hier wird das Kundenfenster geöffnet**

Dabei können zwischen dem linken (ausführbaren) Teil und dem rechten (kommentierenden) Teil ein oder mehrere Leerzeichen (Blanks) bzw. Tabulatorzeichen stehen.

Es ist auch möglich, mit dem Kommantarzeichen eine bereits kommentierte Zeile zu inaktivieren:

! Call SalCreateWindow(frmKunden) ! Kommentar...

Das Ausrufezeichen kann "von Hand" eingefügt werden (was häufig die einfachere Methode ist) oder durch SQLWindows selbst eingefügt werden. Dazu wählen Sie, nachdem Sie eine Zeile markiert haben (einmal auf die Raute am Beginn der Zeile klicken), den Menüpunkt "Outline", "Comment Items".

Um das Kommentarzeichen "von Hand" zu entfernen, klicken Sie einmal in der betreffenden Zeile, bewegen die Schreibmarke auf das Kommentarzeichen und entfernen es.

Um von einer zuvor kommentierten und markierten Zeile das Kommentarzeichen zu entfernen, können Sie den Menüpunkt "Outline", "Uncomment Items" wählen.

Kommentierte Zeilen werden grau dargestellt. Diese Darstellungsart läßt sich über den Menüpunkt "File", "Preferences", "Outline Text" abändern.

1.3.4 Objekte in SQLWindows und deren Verwendung

In diesem Kapitel werden die in SQLWindows verwendbaren Objekte vorgestellt, die zur Programmierung von Anwendungen eingesetzt werden können. Bei der Darstellung wurde Wert darauf gelegt, daß die Beispiele ohne jeglichen "Ballast" und vollständig erläutert werden, damit Sie diese jederzeit nachvollziehen können.

Folgende Bedingungen müssen erfüllt sein, damit Sie die folgenden Beispiele nachvollziehen können:

- Die im Einführungskapitel vorgestellte Datenbank ("FIRMA") sowie die darin enthaltenen Tabellen und Datensätze müssen vorhanden sein. Wenn Sie das Einführungkapitel durchgearbeitet haben ist dies der Fall, falls nicht, dann müssen Sie die dort vorgestellte Datenbank und die darin enthaltenen Tabellen jetzt anlegen (mit SQLTalk oder Quest).

- Sie müssen in SQLWindows eine Variable definieren (s.u.), die für den Zugriff auf die Datenbank verwendet wird.

- Schließlich müssen Sie mit der Anweisung "SqlConnect" eine Anbindung an die Datenbank mit dem Namen "FIRMA" herstellen.

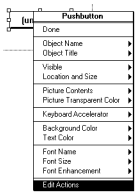
Der folgende Programmcode zeigt das Definieren der Variablen und die Anbindung an die Datenbank.

```
◆ Application Description: SQLWindows version 4.1
  ◆ Design-time Settings
  ◇ Libraries
  ◆ Global Declarations
    ◆ Window Defaults
    ◆ Formats
    ◇ External Functions
    ◆ Constants
    ◇ Resources
    ◆ Variables
      ◇ Sql Handle: hSql
    ◇ Internal Functions
    ◆ Named Menus
    ◇ Class Definitions
    ◆ Application Actions
      ◆ On SAM_AppStartup
        ◇ Set SqlDatabase = 'FIRMA'
        ◇ Call SqlConnect (hSql)
  ◆ Form Window: frmMain
```

Führen Sie die genannten Schritte durch und speichern Sie diese ab, eventuell unter dem Namen "Test". Benutzen Sie diese Anwendung (oder eine Kopie davon) für die folgenden Abschnitte).

Um eine Anwendung zu erstellen, benutzen Sie üblicherweise Objekte, die häufig aus einem angezeigten Fenster (Tool-Palette) eingefügt werden. Da der Vorgang, ein Objekt aus der Tool-Palette in die eigene Anwendung zu übernehmen stets gleich ist, wird er hier nur einmal an zentraler Stelle beschrieben.

Übernahme eines Objekts aus der Tool-Palette in das Form Window "frmMain".

1. Tool-Palette mit F4 aktivieren	
2. Einfaches Klicken auf das einzufügende Objekt (z.B. Push Button)	
3. Bewegen des Mauszeigers in das Form Window (z.B. "frmMain")	
4. Einfaches Klicken (evtl. halten und ziehen, um Größe zu ändern)	
5. "Edit Actions" definieren (einmal mit rechter Maustaste auf das Objekt klicken, letzte Option auswählen)	

Sie sollten den Bildschirm von SQLWindows für die folgenden Beispiele so aufbauen, daß zwei Fenster sichtbar sind: Oben das "Outline" - Fenster, unten das "frmMain" - Fenster. Diese Aufteilung erleichtert die Arbeit und fördert den Überblick.

Achten Sie bei der Eingabe von selbst getippten Anweisungen genauestens auf Groß- und Kleinschreibung, SQLWindows tut dies ebenfalls!

Die Gliederung der folgenden Abschnitte orientiert sich an der bereits früher angesprochenen Trennung von Objekten in SQLWindows in Top-Level- und Child Objekten.

Sie schaffen sich am besten eine "Spielwiese" für die folgenden Beispiele, indem Sie eine neue Anwendung anfertigen, in der Sie nur die Anbindung an die Datenbank herstellen.

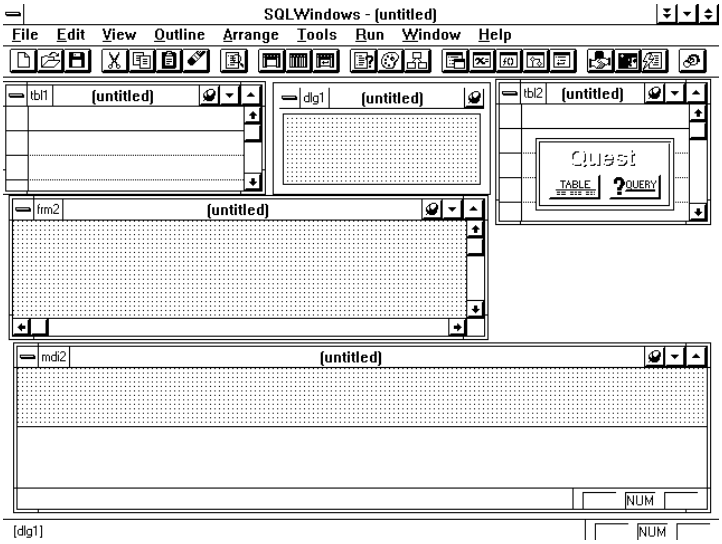
Um die Beispiele einfach zu halten, sollten Sie alle Objekte in das Fenster frmMain einbauen, hier geschieht das unter Verwendung der Tool-Palette, die mit F4 aktiviert werden kann.Die Reihenfolge der Erläuterungen innerhalb der folgenden Abschnitte orientiert sich an allgemeinen Anmerkungen, Einbau des Objekts in eine Anwendung und Benutzen des Objekts, die jeweils durch Zwischenüberschriften eingeleitet werden.

Beachten Sie, daß nicht alle Aspekte von Objekten dargestellt werden (so z.B. die Farbe eines Push-Buttons oder die Schriftart für dessen Beschriftung), dies würde den Rahmen der folgenden Abschnitte sprengen.

1.3.4.1 Top-Level Objekte

Die nachfolgende Abbildung faßt alle Top-Level Objekte zusammen.

Abb. 1.20: Top-Level Objekte in SQLWindows (Table Window, Dialog Box, Quest Activity, Form Window, MDI-Window).



1.3.4.1.1 Form Window

Form Windows werden sehr oft in Anwendungen gebraucht. Üblicherweise besteht der Startbildschirm jeder Windows-Anwendung aus einem solchen Fenster (alternativ ist auch ein MDI-Fenster möglich, das in Aufbau und Verwendung dem Form Window sehr ähnlich ist).

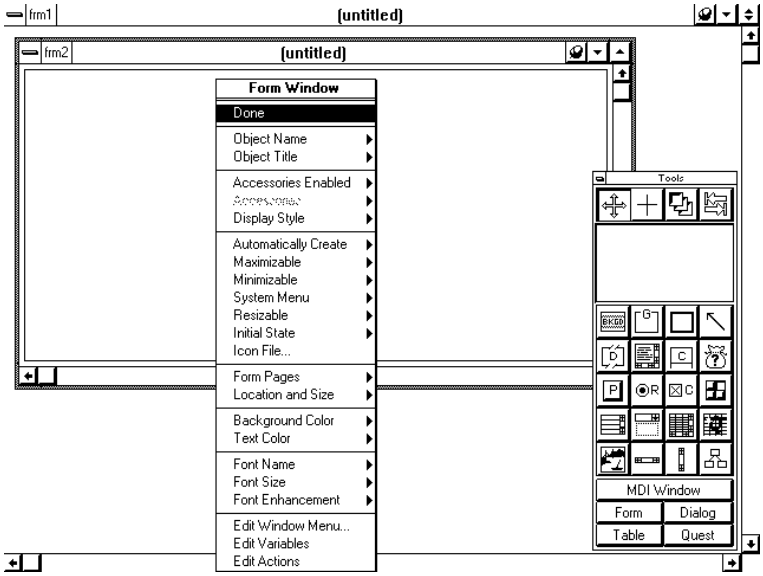
Einfügen eines Form Windows in eine Anwendung

Das Form Window in eine Anwendung einzufügen ist sehr einfach, etwas schwieriger ist es, dieses Form Window unter Kontrolle zu halten: Sie müssen sich immer darum kümmern, daß ein Form Window nur einmal eingefügt werden kann, mit anderen Worten, die Möglichkeit, ein solches Fenster zu aktivieren, muß nach dem ersten Aktivieren "abgeschaltet" werden. Nachdem ein Form Window wieder verschwunden ist, muß natürlich wieder gestattet werden, es erneut zu aktivieren.

Anmerkung: In "Design time settings" am Anfang Ihrer Anwendung Ihrer Anwendung kann durch Umstellen der Option auf "No" verhindert werden, daß ein Fenster mehrfach aktiviert werden kann, allerdings führt dies nur zu einer Fehlermeldung, verhindert das erneute Aktivieren also eher indirekt.

Um in eine Anwendung ein Form Window aufzunehmen, genügt es, im Entwurfsmodus mit der Maus auf die Schaltfläche "Form" der Tool-Palette zu klicken. Es erscheint augenblicklich ein neues, leeres Form Window auf dem Bildschirm, dessen Eigenschaften Sie nach Klicken mit der rechten Maustaste definieren können.

Abb. 1.21: Neu eingefügtes Form Window (mit geöffnetem Customizer).



Achten Sie besonders auf den Namen des soeben eingefügten Fensters (frm1, jedes neu eingefügte Fenster erhält eine um eins erhöhte, laufende Zahl) sowie auf den Status "automatically create", der auf "yes" steht: Dies bedeutet, daß dieses Fenster automatisch bei Start der Anwendung angezeigt wird (Sie können das testen, indem Sie mit F7 oder dem Menü Run/User Mode die Anwendung starten; beachten Sie, daß Sie zweimal die Tastenkombination "Alt+F4" betätigen müssen, zum einen um das Fenster zu schließen, zum anderen um die Anwendung zu beenden; dies wird später so zu ändern sein, daß automatisch bei Beenden der eigentlichen Anwendung die noch geöffneten Form Windows automatisch geschlossen werden.)

Anpassen des Form Windows

Zunächst soll die Eigenschaft, daß das Fenster automatisch erzeugt wird (**automatically create**) umgestellt werden auf "No"; das Fenster soll erst durch eine Benutzeraktion geöffnet werden. Hierzu aktivieren Sie den Customizer, indem Sie mit der rechten Maustaste auf den Hintergrund des Form Windows klicken. Wählen Sie die Eigenschaft "automatically create" und stellen Sie um auf "No".

Danach soll das Fenster einen Namen erhalten (vorgeschlagen ist "untitled": Klicken Sie auf der Option "**Object Title**" und tragen Sie dort "erstes Window" ein. Den Namen des Fensters (frm1) lassen Sie unverändert, obwohl in späteren Anwendungen diese Fenster oft aussagekräftigere Namen erhalten, wie z.B. "frmKunden" oder "frmLager".

In der folgenden Übersicht sind alle Optionen des Customizers zusammengefaßt und erläutert.

Done	Schließen des Optionsfensters.
Object Name	Name des Objekts. Vorgeschlagen frm1, frm2 usw., oft abzuändern in frmKundendaten, frmLagerdaten.
Object Title	Titel des Form Windows (erscheint dann unter anderem im Task-Manager von Windows!).
Accessories Enabled	Sollen die zusätzlichen Möglichkeiten des Form Windows zugänglich sein: ja/nein. Dazu zählen die im folgenden aufgeführten Accessories:
Accessories	• Soll die Tool-Bar (das ist ein Bereich zwischen Oberkante des eigentlichen Form Window und dem Fensterrahmen, in die z.B. Push Buttons eingebaut werden können) geöffnet sein. • Soll die Status-Bar (das ist eine Zeile am unteren Rand des Form Window, in der z.B. Meldungen an den Benutzer angezeigt werden können) sichtbar sein: ja/nein.
Display Style	Art der Anzeige im Form Window.
Automatically Create	Soll das Form Window automatisch erzeugt werden? Ja ist die Voreinstellung; hier wird häufig umgestellt auf nein, damit der Benutzer selbst die Möglichkeit erhält, ein Fenster anzeigen zu lassen.
Maximizable	Soll das Form Window auf volle Bildschirmgröße zu vergrößern sein?
Minimizable	Soll das Form Window auf minimale Größe (Icon, also Symbolgröße) zu verkleinern sein?
System Menu	Soll das Systemmenü (in der linken oberen Ecke des Form Window) vorhanden sein?
Resizable	Soll der Benutzer die Möglichkeit haben, die Größe des Fensters zu verändern?
Initial State	Zustand des Form Window bei Start der Anwendung (Maximal, max. Bildschirmgröße; Minimal, als Symbolform; Normal, so wie im Entwurf eingestellt).
Icon File	Name der Icon-Datei, die das Form Window bei Minimierung symbolisiert.
Form Pages	Zeigt ein Menü, in dem die Seitenabmessungen bei dem Druck des Form Windows eingestellt werden können, bei Verwendung der Funktion SalPrtPrintForm().
Location and Size	Ort und Größe des Form Window.
Background Color	Hintergrundfarbe.
Text Color	Textfarbe.
Font Name	Schriftart.
Font Size	Schriftgröße.

Font Enhancement	Erweiterung der Schriftart (unterstrichen, fett...).
Edit Window Menu	Wechsel zum Menü des Form Window, um es anzulegen oder zu verändern.
Edit Variables	Wechsel zur Sektion "Variables" des Form Windows, wo eigene lokale Variable des Form Windows angelegt werden können (die dort angelegten Variablen stehen nur innerhalb des Form Windows zur Verfügung und unterscheiden sich dadurch von den globalen Variablen, die in der Sektion Global Declarations/Variables angelegt werden).
Edit Action	Wechsel zum Outline-Fenster.

Aktivieren eines Form Windows

Für das Aktivieren des Form Windows werden zwei Varianten vorgestellt: zum einen das Aktivieren durch Drücken eines Push-Buttons, zum anderen das Aktivieren über einen Menüpunkt.

Aktivieren durch einen Push Button:

Hierfür brauchen Sie zunächst einen Push Button, der in das vorhandene Fenster "frmMain" plaziert wird: Klicken Sie, um das Fenster in den Vordergrund zu holen, mit der Maus auf das Fenster frmMain (falls es nicht sichtbar ist, können Sie mit der Tastenkombination "Strg+1" ein Auswahlfenster öffnen und dort frmMain anklicken). Aktivieren Sie danach die Tool-Palette (F4) und klicken Sie zuerst auf das Symbol für den Push Button; danach klicken Sie in dem Fenster, um einen neuen Push Button einzufügen (klicken Sie einmal, halten Sie die Maustaste gedrückt und ziehen Sie die Maus nach rechts, um den Push Button deutlich zu vergrößern, da ein längerer Titel einzutragen ist). Der Titel "untitled" ist nun aktiviert, überschreiben Sie ihn mit dem Text "Child Fenster öffnen"; schließen Sie mit der Enter-Taste ab.

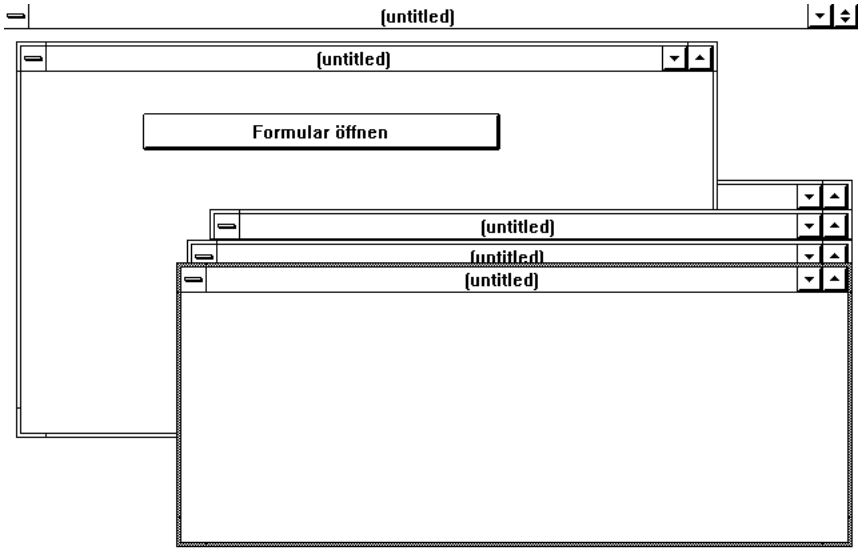
Diesem eben eingefügten Push Button (sein Name, der wieder unverändert bleibt, ist "pb1"), wird folgende Aktion zugewiesen.

Inhalt von Outline	Kommentar
Pushbutton: pb1 Message Actions On SAM_Click Call SalCreateWindow(frm1, frmMain)	Name des Push-Buttons. Zugewiesene Aktion. Bei Klicken erzeuge das Fenster frm1, Eigentümer des Fensters frm1 soll das Fenster frmMain sein.

On SAM_Click ist eine Nachricht, die dann zur Sektion Message Actions des Push Buttons gesendet wird, wenn der Benutzer genau den Push Button gedrückt hat. Diese Nachricht veranlaßt, daß die folgenden, eingerückt stehenden Anweisungen ausgeführt werden; in diesem Fall wird die Funktion SalCreateWindow() aufgerufen. Diese Funktion aktiviert ein Fenster (kann aber, wie vielleicht ihr Name vermuten läßt dieses Fenster nicht erzeugen), ihr muß dazu der Name des Fensters (der Handle) und der Eigentümer des Fenster mitgeteilt werden (frmMain).

Nachdem Sie diesen Text eingegeben haben, können Sie die Anwendung starten. Sie werden dann feststellen, daß sich das Fenster beliebig oft durch Klicken auf dem Push Button öffnen läßt.

Abb. 1.22: Mehrfaches Aktivieren eines Fensters nach wiederholtem Drücken eines Push-Buttons.



Diese Eigenschaft der mehrfachen Aktivierung wird üblicherweise unterbunden. Das geschieht dadurch, daß Sie die Einstellung "Enable" des Push Buttons verändern. Ergänzen Sie den Programmtext von oben durch die im folgenden hervorgehobene Zeile, lassen Sie dazu eine neue Zeile einfügen (klicken Sie dazu im Outline Fenster auf der Zeile Call SalCreateWindow() einmal und drücken Sie danach die Einfg-Taste):

Tab.: Die Funktion SalDisableWindow() zur Deaktivierung eines Push Buttons.

Inhalt von Outline	Kommentar
Pushbutton: pb1 Message Actions On SAM_Click Call SalCreateWindow(frm1, frmMain) **Call SalDisableWindow(pb1)**	 Pb1 deaktivieren

Starten Sie nun die Anwendung und lassen Sie das Fenster erzeugen. Nun sehen Sie, daß die vormals schwarze Beschriftung des Push-Buttons nach dem ersten Drücken grau wird, er ist deaktiviert, die zugeordnete Aktion kann nicht mehr ausgeführt werden.

Allerdings haben Sie nun ein anderes Problem: Nach dem Schließen des Fensters frm1 (z.B. über das Systemmenüs des Fensters oder nach Drücken von Strg+F4) ändert der Push Button seinen Status nicht und kann daher nicht benutzt werden, um das Fenster erneut zu öffnen. Das ist auch logisch, denn an keiner Stelle des Programms wurde bisher definiert, daß der Push Button wieder aktiviert werden soll. Um dies zu erreichen müssen Sie angeben, daß bei Schließen des Fensters (frm1) der Push Button wieder aktiviert werden soll.

Beenden Sie zunächst die laufende Anwendung (ALT+F4), lassen Sie danach das Child-Fenster frm1 anzeigen (Strg+1), klicken Sie mit der rechten Maustaste auf das Child-Fenster, wählen Sie die "Edit Actions" Option und fügen Sie dort die folgenden, hervorgehobenen Zeilen ein (Sie können das wiederum mit den Outline Options tun):

Erneutes aktivieren eines Push-Buttons nach Schließen eines durch ihn erzeugten Form Windows.

Inhalt von Outline	Kommentar
Form Window: frm1 Message Actions **On SAM_Destroy** **Call SalEnableWindow(pb1)**	 Bei Schließen des Fensters soll das Objekt mit dem Namen "pb1" aktiviert werden.

Jetzt können Sie Ihre Anwendung starten, Sie werden sehen, daß durch diese wenigen Zeilen Programmtext die Aktionen einwandfrei definiert sind und funktionieren.

Aktivieren eines Form Windows durch einen Menüpunkt

Diese Methode des Aktivierens verlangt etwas mehr Codierung als die oben geschilderte. Sie müssen zunächst einen Window-Handle für das zu öffnende Fenster definieren, danach muß aus dem Menüpunkt das Fenster aktiviert werden und der Menüpunkt selbst nunmehr "stillgelegt" werden, um ein mehrfaches Öffnen des Fensters zu verhindern. Zusammengefaßt kann man die Schritte folgendermaßen definieren:

(1) Window-Handle definieren:

Inhalt von Outline	Kommentar
Variables .. Window Handle: hWndfrm1 ..	In der Variables Section der Anwendung wird der Handle für das Fenster vereinbart.

(2) In Menu Actions die Funktion SalCreateWindow einbauen und dem Handle einen Wert geben, der sich von NULL unterscheidet (für späteren Vergleich mit hWndNULL, s.u.):

Inhalt von Outline (Menüpunkt)	Kommentar
Menu Actions Set hWndfrm1 = SalCreateWindow(frm1, frmMain)	Fülle den Inhalt der Variablen hWndForm1 mit einem Wert

(3) In "Enable When" des Menüpunktes den mehrfachen Aufruf des Fensters unterbinden:

Inhalt von Outline (Menüpunkt)	Kommentar
Enable when: hWndfrm1 = hWndNULL	Menüaktion ist nur wählbar, wenn die Variable hWndfrm1 den Inhalt NULL hat.

(4) Bei Schließen des Fensters die Benutzung des Menüpunktes wieder gestatten:

Inhalt von Outline (Fenster frm1)	Kommentar
On SAM_Destroy Set hWndfrm1 = hWndNULL	Bei Schließen des Fensters wird Inhalt der Variablen (der Handle) wieder auf NULL gesetzt.

Die fertig codierten Sektionen sehen folgendermaßen aus:

- Variables
  - Window Handle: hWndfrm1
- Internal Functions
- Named Menus
- Class Definitions
- Application Actions
- Form Window: frmMain
  - Description:
  - Named Menus
  - Menu
    - Popup Menu: &Datei
      - Enabled when:
      - Status Text:
      - Menu Item: &Fenster öffnen
        - Status Text:
        - Keyboard Accelerator: (none)
        - Menu Settings
          - Enabled when: hWndfrm1 = hWndNULL
          - Checked when:
        - Menu Actions
          - Set hWndfrm1 = SalCreateWindow(frm1, frmMain)

Folgende Abbildung zeigt den Inhalt der Sektion Message Actions des Form Windows "frm1":

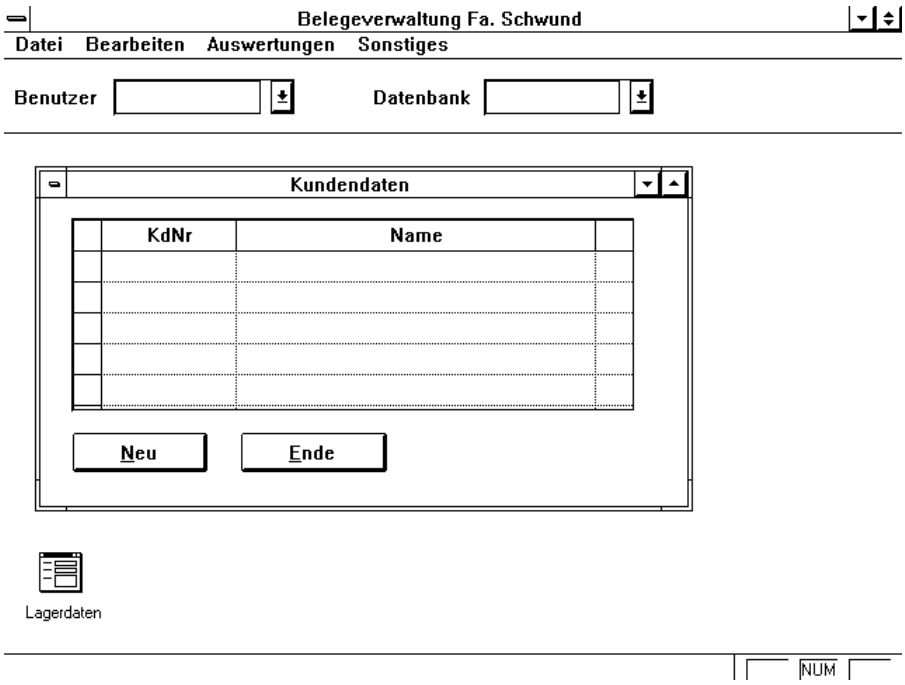
- Form Window: frm1
  - Description:
  - Named Menus
  - Menu
  - Tool Bar
  - Contents
  - Functions
  - Window Parameters
  - Window Variables
  - Message Actions
    - On SAM_Destroy
      - Set hWndfrm1 = hWndNULL

Zur Vertiefung des Stoffs könnten Sie die unten folgenden Kapitel zu den Child Objekten lesen, die üblicherweise innerhalb eines Form Windows benutzt werden.

1.3.4.1.2 MDI Window

MDI (Multiple Document Interface) Windows sind in gewisser Weise Form Windows, die um einige Besonderheiten erweitert wurden. Dieser Fenstertyp ermöglicht Ihnen, Windows Anwendungen so zu erstellen, daß mehrere Top-Level Objekte am unteren Bildrand so angeordnet werden, wie andere Programme, z.B. Textverarbeitungen dies mit Textdateien tun. Dabei können diese Objekte nicht über den "Rand" des MDI-Windows hinausbewegt werden und bleiben somit ein Bestandteil dieses Fensters. Die folgende Abbildung zeigt ein MDI Window mit darin enthaltenen Objekten, von denen eines geöffnet und das andere am unteren Bildschirmrand angeordnet ist.

Abb. 1.23: MDI-Window.



Einige der bei Einstellung von Form Windows vorhandenen Möglichkeiten sind bei MDI Windows nicht vorhanden, da sie keinen Sinn machen würden. Ansonsten sind die Möglichkeiten das MDI Window an die eigenen Vorstellungen anzupassen, die gleichen wie die Möglichkeiten die bei den Form Windows vorhanden sind.

Es ist auch möglich in ein vorhandenes MDI Window ein anderen MDI Window (als Child Window) einzubauen, allerdings sollte überlegt werden, ob der Benutzer dadurch nicht eher verwirrt wird.

Zur Bearbeitung eines MDI Windows im Programmablauf stehen folgende Funktionen zur Verfügung:

- **SalMDITile()**. Verteilt die Fenster auf dem Bildschirm, so daß der Bildschirm gefüllt ist.
- **SalMDICascade()**. Ordnet die Fenster in überlappender Form an.
- **SalMDIArrangeIcons()**. Bei Aufruf dieser Funktion werden die im MDI Window enthaltenen Icons am unteren Bildschirmrand angeordnet.

1.3.4.1.3 Dialog Box

Eine Dialog Box ist - ähnlich einem Form Window - ein Fenster, in dem Informationen angezeigt oder eingegeben werden können. Dialog Boxen werden häufig verwendet, um dem Benutzer Informationen über das Programm zu geben oder Warnungen anzuzeigen.

Grundlegende **Unterschiede zu Form Windows** sind:

- Dialog Boxen können nicht automatisch bei Start der Anwendung angezeigt werden. Die Anzeige ist nur über die Funktionen SalCreateWindow() oder SalModalDialog() möglich.

- Die Größe einer Dialog Box kann nach dem Start der Anwendung nicht verändert werden, da die dafür nötigen Schaltflächen fehlen.

- Dialog Boxen enthalten kein Menü.

Man unterscheidet **drei Typen** von Dialog Boxen:

- **Modeless**: Eine solche Dialog Box unterbricht den Programmablauf nicht, sondern bleibt auf dem Bildschirm stehen, bis der Benutzer sie wieder schließt. Dialog Boxen diesen Typs werden mit den Funktionen SalCreateWindow() geöffnet bzw. mit der Funktion SalDestroyWindow() geschlossen.

- **Modal**: Diese Dialog Box unterbricht den Programmablauf; ein Wechsel zu anderen Fenstern ist nicht möglich. Der Benutzer wird mit einer solchen Dialog Box zu einer Reaktion gezwungen; oft muß er, um die Dialog Box zu schließen, einen Push Button drücken. Um eine solche Dialog Box anzuzeigen bzw. zu schließen, müssen die Funktionen SalModalDialog() bzw. SalEndDialog() benutzt werden.

- **System modal**: Diese Dialog Box unterbricht das komplette System solange sie angezeigt wird. Nicht nur die Fenster der SQLWindows-Anwendung, sondern auch alle anderen Fenster sind während der Anzeige dieser Box nicht erreichbar. Dieser Typ wird mit den Funktionen SalModalDialog() bzw. SalEndDialog() angelegt bzw. geschlossen.

Die Inhalte des Customizers von Dialog Boxen unterscheiden sich nur in einem Eintrag von Form Windows:

Type of Dialog	Typ der Dialog Box (modeless, modal, system modal)

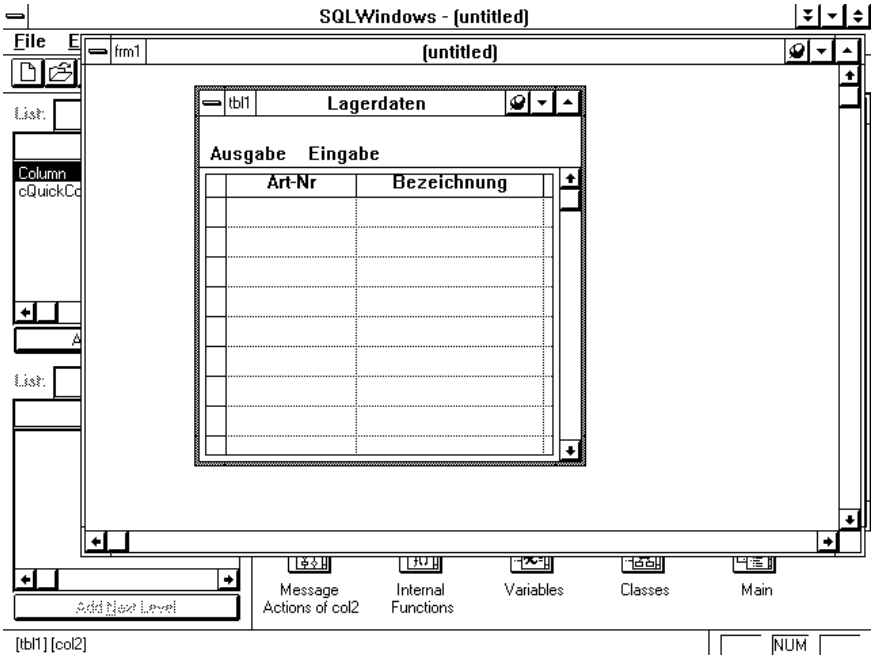
Die Benutzung der Dialog Box entspricht der Benutzung der für Form Windows; die entsprechenden Passagen können dort nachgelesen und übertragen werden.

1.3.4.1.4 Table Window

Ein Table Window ist ein Fenster, das es ermöglicht, Informationen in tabellarischer Form (in der Regel aus einer Datenbank oder Variableninhalte) darzustellen. Zwei verschiedene Arten von Table Windows sind in SQLWindows bekannt: Top-Level Table Windows und Child Table Windows (die häufig in Form Windows eingebaut sind). Sie sehen beide fast gleich aus, der grundsätzliche Unterschied besteht darin, daß in einem Top-Level Table Window ein Menü eingebaut werden kann. Dieses Top-Level Window hat dann ein

ähnliches Aussehen wie ein Form Window, kann aber im Gegensatz zu diesem keine weiteren Objekte, wie z.B. Push Buttons, darstellen.

Abb. 1.24: Top-Level Table Window mit zwei Spalten und zwei Menüpunkten.



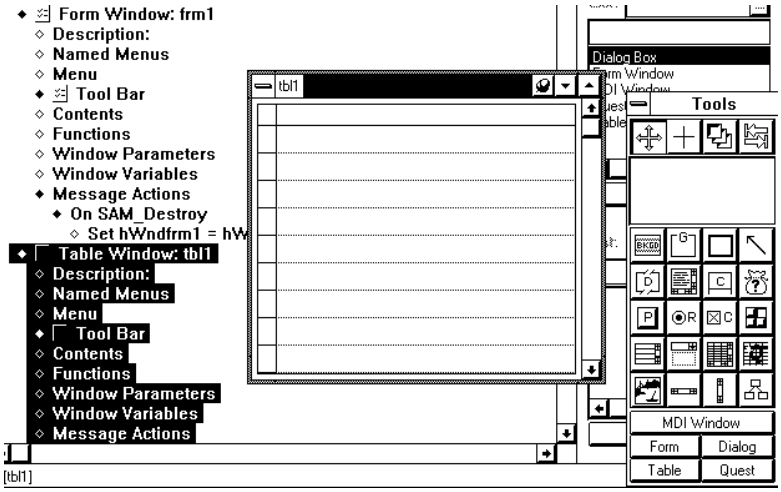
Eine allgemeine Feststellung, ob nun ein Form Window mit eingebautem Child Table Window besser ist, als ein Top-Level Table Window kann nicht getroffen werden, das muß im Einzelfall entschieden werden. Sie sollten aber berücksichtigen, daß es für den Benutzer oft einfacher ist, wenn er die Möglichkeiten zur Bedienung in Form von Push Buttons vor sich sieht und nicht erst in einem Menü suchen muß.

Einfügen eines Table Windows in die Anwendung

Um ein Table Window in eine Anwendung einzufügen gehen Sie folgendermaßen vor:

- Aktiverien Sie die Tool Palette (F4 drücken).
- Klicken Sie auf die Schaltfläche "Table" (in der Tool Palette unten links), das Table Window wird angezeigt.

Abb. 1.25: Eingefügtes Tabellenfenster "tbl1" (noch ohne Spalten).



Sie können die Eigenschaften des Table Windows über den Customizer einstellen. Die folgende Übersicht zeigt die dort vorhandenen Möglichkeiten:

Done	Schließen des Customizers.
Object Name	Name des Objekts. Vorgeschlagen werden "tbl1", "tbl2" usw., oft abzuändern in "tblKundendaten", "tblLagerdaten".
Object Title	Titel des Tabellenfensters (erscheint dann u.a. im Task-Manager!).
Accessories Enabled	Sollen die zusätzlichen Möglichkeiten des Tabellenfensters zugänglich sein: ja/nein. Dazu zählen:
Accessories	• Soll die Tool-Bar (Bereich zwischen Oberkante der eigentlichen Tabelle und dem Fensterrahmen, in den z.B. Push Buttons eingebaut werden können) geöffnet sein: ja/nein. • Soll die Status-Bar (Zeile am unteren Rand des Tabellenfensters, in der z.B. Meldungen an den Benutzer angezeigt werden können) sichtbar sein: ja/nein.
Automatically Create	Soll das Tabellenfenster automatisch erzeugt werden? Ja ist voreingestellt; hier wird häufig umgestellt auf nein, damit der Benutzer selbst die Möglichkeit erhält ein, Tabellenfenster anzeigen zu lassen.
Maximizable	Soll das Tabellenfenster auf volle Bildschirmgröße zu vergrößern sein?
Minimizable	Soll das Tabellenfenster auf minimale Größe (Icon, also Symbolgröße) zu verkleinern sein?
System Menu	Soll das Systemmenü (in der linken oberen Ecke des Tabellenfensters) vorhanden sein: ja/nein.
Resizable	Soll der Benutzer die Möglichkeit haben, die Größe des Fensters verändern zu können: ja/nein.
Initial State	Zustand des Tabellenfensters bei Start der Anwendung (Maximal, Minimal, Normal).
Icon File	Name der Icon-Datei, die das Tabellenfenster bei Minimierung symbolisiert.
Location and Size	Ort und Größe des Table Windows.
Background Color	Hintergrundfarbe.

Text Color	Textfarbe.
Font Name	Schriftart.
Font Size	Schriftgröße.
Font Enhancement	Erweiterung der Schriftart (fett, kursiv...)
Discardable	Dürfen Datensätze, die im Cache-Zwischenspeicher stehen überschreiben werden (ja ist voreingestellt)?
Max Rows in Memory	Maximale Zahl der Datensätze, die im Cache-Zwischenspeicher stehen dürfen (voreingestellt ist 100, maximal möglich: 32753 Datensätze).
Edit Window Menu	Wechsel zum Menü des Tabellenfensters, um es zu editieren.
Edit Variables	Wechsel zu der Sektion Variables des Table Windows, wo eigene lokale Variable des Tabellenfensters angelegt werden können. Die dort angelegten Variablen stehen nur innerhalb des Table Windows zur Verfügung und unterscheiden sich dadurch von den globalen Variablen, die in der Sektion Global Declarations/Variables angelegt werden!
Edit Action	Wechsel zum Outline-Fenster.

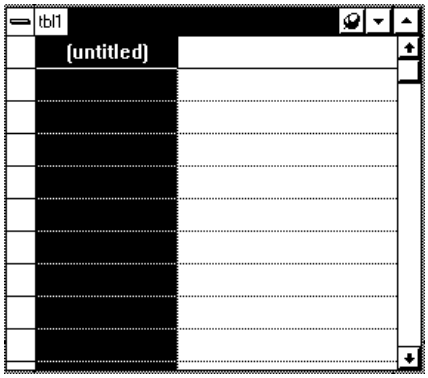
Einfügen der Spalten in das Table Window

Um in ein Table Window Spalten einzufügen, muß in der Tool Palette zunächst auf das dafür vorgesehene Symbol (mit einem "c" für column beschriftet) geklickt werden.

Abb. 1.26: Einfügen von Spalten in ein Table Window.

Ziehen Sie dann den Mauszeiger an den linken Rand der Tabelle (der Zeiger verwandelt sich dabei in das gewählte Symbol mit dem Inhalt "col"), klicken Sie einmal, damit wird die erste Spalte eingefügt (falls nichts geschieht, müssen Sie den Mauszeiger etwas seitlich bewegen und erneut klicken).

Abb. 1.27: Eingefügte Spalte.



Wiederholen Sie den Vorgang, um eine zweite Spalte einfügen zu lassen.

Die eben eingefügten Spalten haben die Namen "col1" bzw. "col2". Sie können sich diese Namen anzeigen lassen, indem Sie mit der rechten Maustaste auf den Kopf einer Spalte (untitled) klicken, um den Customizer für die Spalte zu öffnen. Wählen Sie "Object Name", um sich den Namen anzeigen zu lassen (ohne ihn zu ändern), wählen Sie danach "Object Title", um den Titel (untitled) anzeigen zu lassen; falls Sie möchten, können Sie eine eigene Überschrift eingeben.

Sie können Ihre Anwendung bereits starten (mit der F7-Taste). Sinnvollerweise sollten aber einige Datensätze angezeigt werden, weshalb das hier kurz erläutert wird.

Lesen und Anzeigen von Datensätzen aus einer Tabelle in dem eben erstellten Table Window mit der Funktion **SalTblPopulate()**:

- Klicken Sie mit der rechten Maustaste auf die Zeilen des Table Windows, um den Customizer anzeigen zu lassen.
- Wählen Sie Edit Actions, um die Anweisungen einzugeben.
- Tragen Sie folgendes in die Message Actions Sektion der Tabelle ein:

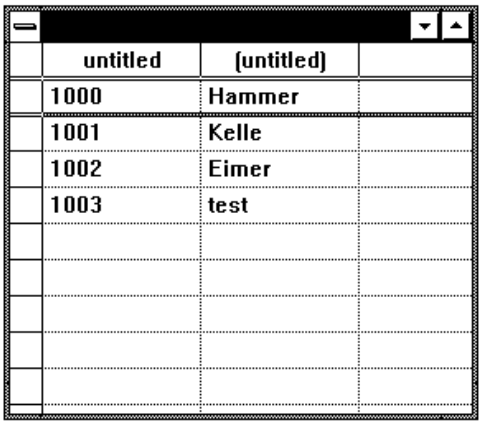
```
◆ Table Window: tbl1
  ◇ Description:
  ◇ Named Menus
  ◇ Menu
  ◆ Tool Bar
  ◆ Contents
    ◆ Column: col1
      ◇ Message Actions
    ◆ Column: col2
      ◇ Message Actions
  ◇ Functions
  ◇ Window Parameters
  ◇ Window Variables
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Call SalTblPopulate( tbl1,
        hSql,
        'select artnr, name from lager into :col1, :col2',
        TBL_FillNormal )
```

Zur Erläuterung der Funktion SalTblPopulate() :

Inhalt von Outline	Kommentar
On SAM_Create Call SalTblPopulate(tbl1, hSql, 'select artnr, name from lager into :col1, :col2', TBL_FillNormal)	Bei Erzeugen des Table Window (tbl1) Aufruf der Funktion SalTblPopulate(), mit dem Namen der Tabelle (tbl1) und dem Handle (hSql) Select-Anweisung mit Spaltennamen (name, vorname), den Zielspalten der Tabelle (:col1, :col2) und dem Modus des Füllens (TBL_FillNormal).

Nachdem Sie die Anwendung gestartet haben (F7), sieht das Table Window wie folgt aus:

Abb. 1.28: Table Window mit zwei Spalten und angezeigten Datensätzen.



untitled	(untitled)	
1000	Hammer	
1001	Kelle	
1002	Eimer	
1003	test	

Insgesamt ist der Einbau eines Tabellenfensters und die Anzeige von Datensätzen nicht schwierig. Hinzukommt, daß es neben der geschilderten Methode eine dynamische Art gibt, wie Sie Datensätze aus einer Tabelle anzeigen lassen können. Hierzu fügen Sie *keine* Spalten in das Table Window ein und wählen als Sql-Anweisung "select * from lager". Damit werden alle Datensätze aus allen Spalten in der Tabelle angezeigt.

1.3.4.1.5 Quest-Objekte

Quest Objekte (sie stehen nur in der Corporate Version von SQLWindows zur Verfügung) unterscheiden sich von den bisher dargestellten Top-Level Objekten. Der wichtigste Unterschied liegt darin, daß ein anderes Programm, nämlich Quest, zur Erstellung dieser Objekte benutzt wird. Sie starten innerhalb von SQLWindows dieses Programm (in der aktuellen Version handelt es sich um eine Datei mit dem Namen "Quest30.exe"), fertigen dort Ihr Objekt an, speichern das Objekt ab, verlassen Quest und kehren zu SQLWindows zurück.

Da die große Funktionsvielfalt von Quest hier nicht ausführlich dargestellt werden kann, sollen nur zwei grundsätzliche Möglichkeiten beschrieben werden, wie Sie aus SQLWindows heraus mit Quest Verbindung aufnehmen können.

Sie haben zwei Möglichkeiten, eingebettete Quest Objekte aus SQLWindows heraus zu benutzen:

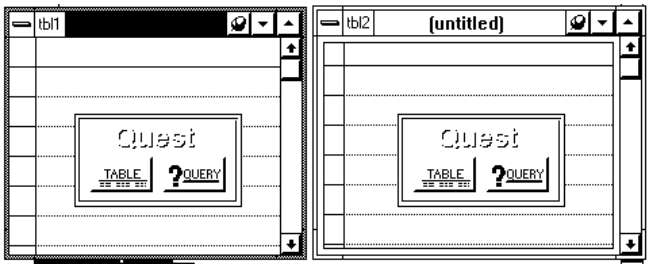
- Sie benutzen Quest, um damit Daten aus **Tabellen** darzustellen oder
- Sie verwenden Quest, um über eine **Abfrage** (Query) eine Teilmenge von Daten auszuwählen.

Der Unterschied liegt darin, daß bei Verwendung der Tabellen die Daten (zunächst) ohne Filterbedingung angezeigt werden, während Sie bei einer Abfrage eine oder mehrere Tabellen auswählen können, um mittels

Bedingungen eine Abfrage zu erzeugen. Diese Abfrage kann dabei sehr komplex sein, bleibt aber durch die Menüführung innerhalb von Quest immer einfach zu bedienen.

Sie müssen zunächst zwei Quest Objekte in Ihre Anwendung einfügen (eines für die Tabelle, eines für die Abfrage). Klicken Sie hierzu in der Tool Palette auf die Schaltfläche mit der Beschriftung "Quest" (unten rechts), um das erste Quest Fenster einzufügen, klicken Sie nochmals, um das zweite Quest Fenster einzufügen. Ordnen Sie beide Fenster so an, wie es die folgende Abbildung zeigt.

Abb. 1.29: Quest Objekte.

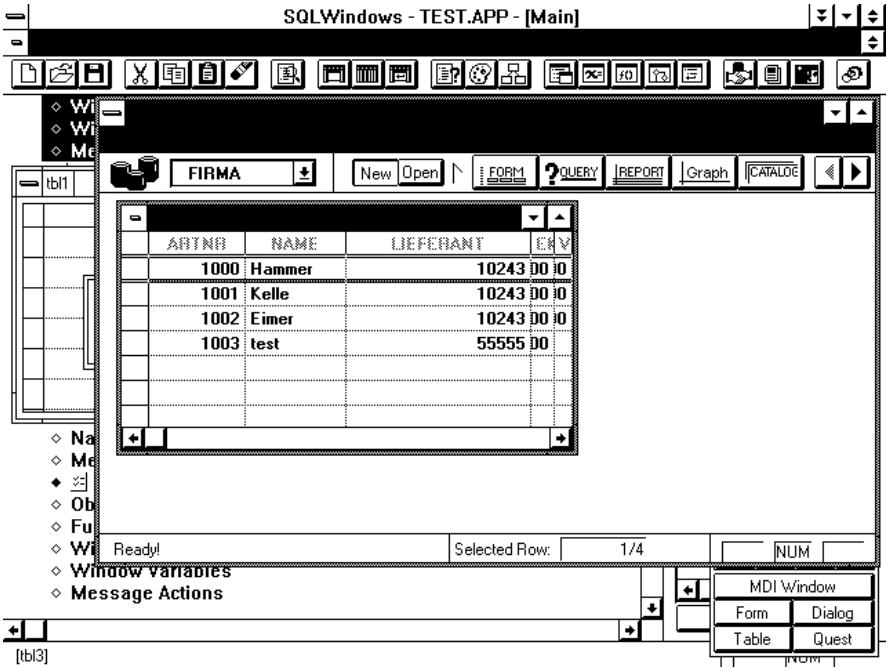


Das linke Quest Objekt (tbl1) soll als Tabellen (Table-) Objekt, das rechte (tbl2) als Abfrage (Query-) Objekt eingerichtet werden.

Einrichten eines Tabellen Objekts

Klicken Sie in dem linken Fenster auf die Schaltfläche "Table", Quest wird jetzt gestartet. Wählen Sie in dem ersten Auswahlfenster die Datenbank "FIRMA", in dem darauf angezeigten Auswahlfenster die Tabelle "LAGER" aus. Die Datenbank "Firma" ist geöffnet (s. Combo Box in der folgenden Abb.), die Daten der Tabelle "Lager" sind angezeigt.

Abb. 1.30: Quest Objekt mit geöffneter Datenbank "FIRMA" und den Daten aus der Tabelle "Lager".



Wählen Sie jetzt in Quest aus dem Menüpunkt "File" die Option "Update", um SQLWindows mitzuteilen, daß das Tabellenobjekt fertig ist. Sie erhalten in der Statuszeile die angezeigte Meldung "Ready". Verlassen Sie nun Quest, indem Sie Alt+F4 drücken (bestätigen Sie die Meldung, damit der Arbeitsbereich gespeichert wird). Sie kehren danach zu SQLWindows zurück.

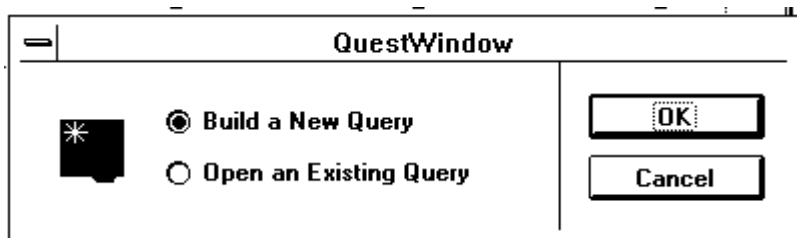
Beachten Sie, daß in der zuvor leeren Tabelle jetzt Zeilen und Spalten (aber noch keine Datensätze) angezeigt sind. Sie könnten Ihre Anwendung bereits starten, doch soll zunächst das zweite Quest Objekt eingerichtet werden.

Einrichten eines Abfrage Objekts

Klicken Sie in dem rechten Quest Fenster auf die Schaltfläche Query (Quest wird gestartet).

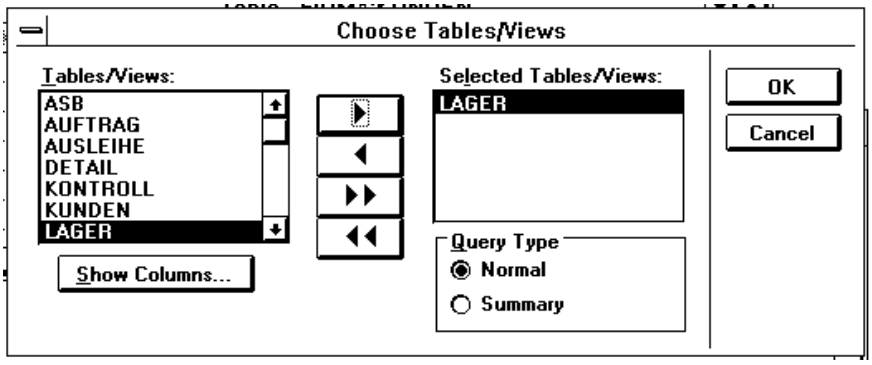
Sie werden zunächst gefragt, ob Sie einen neuen Query anfertigen oder einen vorhandenen Query öffnen wollen.

Abb. 1.31: Fenster mit der Frage, ob eine neue Abfrage (Query) angelegt, oder ob eine vorhandene Abfrage verwendet werden soll.



Wählen Sie die Option für einen neuen Query aus, klicken Sie auf "OK". Auch jetzt werden Sie zunächst nach der Datenbank gefragt ("FIRMA"), Sie erhalten dann ein Fenster mit den in dieser Datenbank vorhandenen Tabellen.

Abb. 1.32: Auswahl der Tabelle aus der Datenbank.

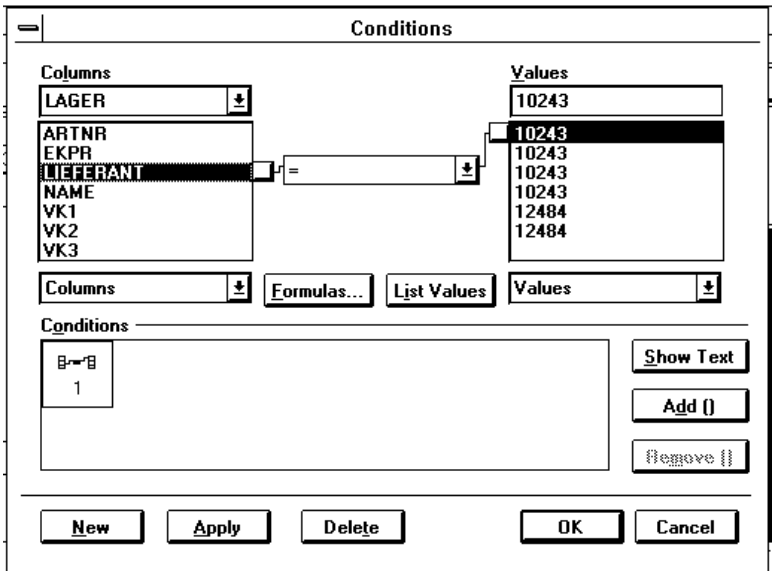


Klicken Sie zunächst links auf dem Namen der Tabelle "Lager", danach in der Mitte auf die Schaltfläche mit dem nach rechts zeigenden Dreieck (es werden keine weiteren Tabellen übernommen). Unten rechts kann der Query-Typ bestimmt werden, hier ist "Normal" zu wählen ("Summary" würde bei zusammenfassenden Queries benutzt werden, in denen z.B. gezählt oder berechnet werden soll). Klicken Sie auf "OK".

Jetzt wählen Sie aus dem Menüpunkt Query die Option Conditions. Sie erhalten dann ein Fenster, in dem Bedingungen für den Query ausgewählt werden können (beachten Sie auch die darüber stehende Möglichkeit, über "Choose Columns" einzelne Spalten für die Anzeige auszuwählen).

In dem jetzt angezeigten Fenster können Sie sich Bedingungen anfertigen, nach denen die anzuzeigenden Datensätze auszuwählen sind. Hier sollen alle Artikel des Lieferanten mit der Nummer 10243 angezeigt werden. Die dazu notwendige Bedingung sieht folgendermaßen aus:

Abb. 1.33: Bedingung für den Query.



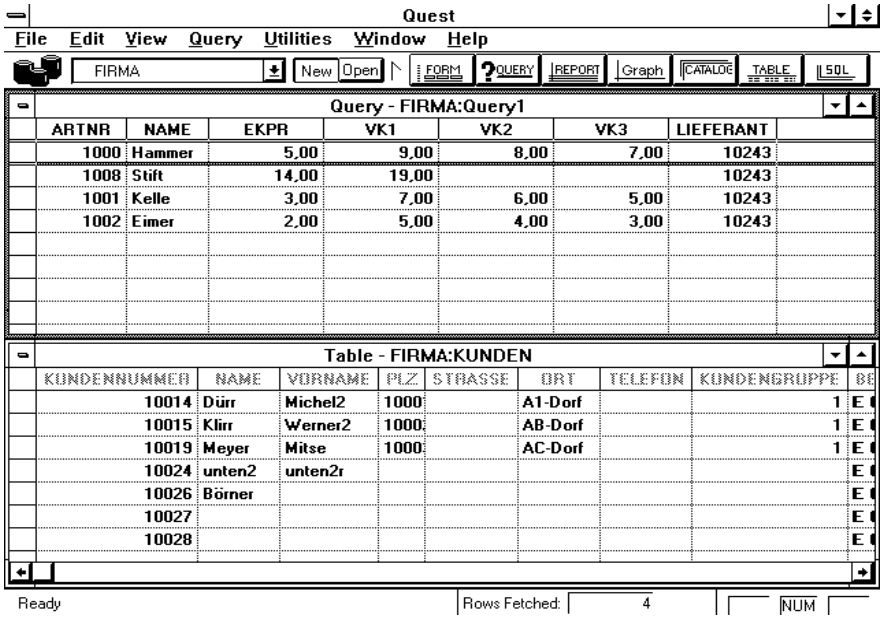
Klicken Sie auf den OK Push Button, um das Bedingungsfenster zu verlassen. Die Datensätze werden bereits auf dem Bildschirm angezeigt (falls Sie zusätzliche Bedingungen angeben wollen, können Sie das jetzt über den Menüpunkt "Query", "Conditions" tun).

Wählen Sie jetzt aus dem Menüpunkt "File" die Option "Update", um die Informationen nach SQLWindows zu schicken. Verlassen Sie danach Quest (Alt+F4, Speichern des Arbeitsbereichs bestätigen).

Sie befinden sich jetzt wieder in SQLWindows und können Ihre Anwendung starten (F7).

Beide Quest Objekte werden nach dem Start angezeigt. Die folgende Abbildung zeigt oben den Inhalt des Tabellen-Objekts und unten den Inhalt des Query-Objekts.

Abb. 1.34: Quest Table- und Query-Objekte in SQLWindows.



Jedesmal, wenn Sie diese Anwendung starten, werden die Daten frisch gelesen und angezeigt.

Um die Quest Objekte Ihren Vorstellungen anzupassen, stehen über den Customizer noch folgende Möglichkeiten zur Verfügung:

Done	Schließen des Quest-Fensters.
Object Name	Name des Objekts. Vorgeschlagen werden tbl1, tbl2 usw. (oft abzuändern in tblKundendaten, tblLagerdaten).
Object Title	Titel des Quest Windows (erscheint im Task-Manager!).
Accessories Enabled	Sollen die zusätzlichen Möglichkeiten des Quest Windows zugänglich sein: ja/nein. Dazu zählen:
Accessories	• Soll die Tool-Bar (Bereich zwischen Oberkante des eigentlichen Quest Windows und dem Fensterrahmen, in die z.B. Push-Buttons eingebaut werden können) geöffnet sein: ja/nein. • Soll die Status-Bar (Zeile am unteren Rand des Quest Window, in der z.B. Meldungen an den Benutzer angezeigt werden können) sichtbar sein: ja/nein.
Automatically Create	Soll das Quest Window automatisch erzeugt werden (ja ist voreingestellt) Häufig umgestellt auf nein, damit der Benutzer selbst die Möglichkeit erhält, das Quest Window anzeigen zu lassen.
Maximizable	Soll das Quest Window auf volle Bildschirmgröße zu vergrößern sein?
Minimizable	Soll das Quest Window auf minimale Größe (Icon, also Symbolgröße) zu verkleinern sein?

System Menu	Soll das Systemmenü (in der linken oberen Ecke des Form Window) vorhanden sein: ja/nein.
Resizable	Soll der Benutzer die Möglichkeit haben, die Größe des Fensters zu verändern?
Initial State	Zustand des Quest Window bei Start der Anwendung (Maximal, Minimal, Normal).
Icon File	Name der Icon-Datei, die das Quest Window bei Minimierung symbolisiert.
Location and Size	Ort und Größe des Quest Window.
Auto Execute	Soll der Query automatisch ausgeführt werden (ja ist voreingestellt, bei Umstellung auf nein müßte diese Aktion dann über einen Menüpunkt oder einen Push Button ausgeführt werden).
Discardable	Dürfen Datensätze, die im Cache-Zwischenspeicher stehen, überschrieben werden (ja ist voreingestellt).
Max Rows in Memory	Maximale Zahl der Datensätze im Cache-Zwischenspeicher.
Edit Window Menu	Wechsel zum Menü des Quest Windows, um es zu editieren.
Edit Variables	Wechsel zu der Sektion "Variables" des Quest Windows, wo eigene lokale Variable des Quest Windows angelegt werden können (die dort angelegten Variablen stehen nur innerhalb des Quest Windows zur Verfügung und unterscheiden sich dadurch von den globalen Variablen, die in der Sektion Global Declarations/Variables angelegt werden)
Edit Action	Wechsel zum Outline-Fenster.

1.3.4.2 Child Objekte

Child Objekte sind Objekte, die nicht alleine, sondern immer eingebettet in Top-Level Objekten (oft in Form Windows) verwendet werden. Hierzu zählen

- Objekte, die nur zur Anzeige von Linien oder Kommentaren dienen;
- Objekte, die zur Ein- und Ausgabe von Daten (numerischen Daten oder Zeichen) verwendet werden;
- Objekte, die zur Ein- und Ausgabe von logischen Informationen (und deren Gruppierung) dienen;
- Objekte, die für vom Benutzer ausgelöste Aktionen verwendet werden und
- sonstige Objekte (s. dazu die nachfolgende Übersicht).

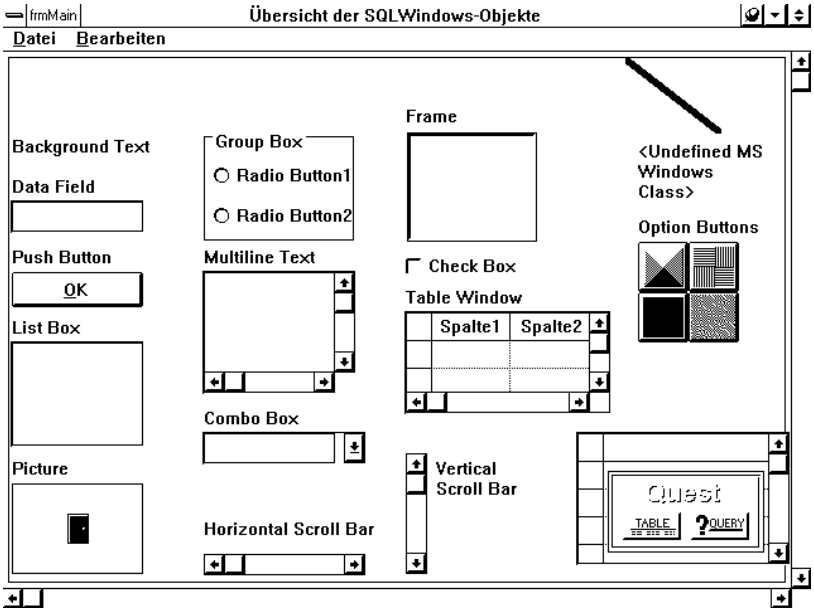
Die folgende Übersicht stellt die in SQLWindows vorhandenen Objekte, geordnet nach ihrer Verwendung vor.

Verwendung der Objekte	Name	Erklärung
nur zur Anzeige	Background Text Frame Line Picture	Hintergrundtext Rahmen Linie Bild
Ein- und Ausgabe von Daten (Datentyp numerisch, Datum oder Zeichen)	Data Field Multiline Field List Box Combo Box Table Window Quest Window Column	Datenfeld längere Texte Inhalte aus Array oder Tabelle wie List Box Datensätze Datensätze Spalte in Table Window
Ein- und Ausgabe von logischen Daten	Radio Button Check Box Option Button	Alternativauswahl von Optionen " "
Eingaben des Benutzers, die eine Aktion auslösen	Push Button Scroll Bar	Öffnen von Fenstern, Druckausgabe Bewegen von Datensätzen (blättern)
Sonstige.	Custom control Group Box	Klassenbibliotheken Gruppieren von Radio Buttons etc.

Beachten Sie, daß einige Child-Objekte durchaus als Top Level Objekte verwendet werden können (z.B. Table Window, Quest Window) und daß das "Column" Child Objekt nur als Teil eines Table Windows verwendet werden kann.

Die folgende Abbildung zeigt eine Zusammenstellung aller Child Objekte von SQLWindows.

Abb. 1.35: Zusammenstellung aller Child Objekte von SQLWindows.



Die einzelnen Child Objekte werden im folgenden so besprochen, wie sie in der Abbildung 1.35 dargestellt sind. Jeweils kurze Programmteile erläutern die Verwendung der einzelnen Child Objekte.

Beachten Sie dabei folgende Punkte:

- für das Einfügen der Objekte wird jeweils die Tool Palette benutzt (mit F4 zu aktivieren);
- unmittelbar nach dem Einfügen können Sie durch Drücken der Entf-Taste das zuletzt eingefügte Objekt wieder entfernen;
- durch einen Klick mit der rechten Maustaste können Sie den Customizer öffnen, um die Eigenschaften eines Objekts zu verändern.

Um die Beispiele nachvollziehen zu können genügt es, wenn Sie ein leeres Formular von SQLWindows vor sich haben; für die Zugriffe auf die Datenbank muß die bereits oben erwähnte Anbindung an die Datenbank vorhanden sein.

1.3.4.2.1 Objekte zur Anzeige von Linien, Kommentaren und Bildern

Objekten, die ausschließlich zur Anzeige von Informationen oder Hinweisen dienen (Linien, Kommentare), kann keine Aktion zugewiesen werden, d.h., daß es nicht möglich ist, einen Mausklick auf einem Background Text in einer Anwendung zu verarbeiten. Bei Bildern dagegen ist dies möglich.

1.3.4.2.1.1 Background Text

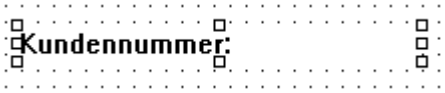
Background texte ist Text, der in einem Formular plaziert werden und zur Anzeige von Hinweisen verwendet werden kann. Häufig werden Kommentare ("Klicken Sie hier um zu drucken") oder Erläuterungen zu den angezeigten Daten ("Kundennummer:") angezeigt.

Um einen Background Text einzufügen, lassen Sie zunächst die Tool Palette aktivieren (F4) und klicken dann auf das Symbol für Background Text (in der Tool Palette links oben):

Abb. 1.36: Symbol für Background Text in der Tool Palette.

Ziehen Sie dann die Maus auf das Form Window, klicken Sie einmal (halten Sie die Maustaste gedrückt), ziehen Sie die Maus nach rechts (eventuell auch nach unten) und lassen Sie anschließen die Maustaste wieder los. Nun erscheint die Schreibmarke; Sie können den Text sofort eingeben; drücken Sie nach der Eingabe die Enter Taste.

Abb. 1.37: Background Text.



Der Background Text ist markiert, somit in Inhalt und Größe veränderbar. Häufig wird man die rechte Grenze des Feldes an die Länge des darin enthaltenen Textes anpassen. Da sich die im Customizer veränderbaren Eigenschaften dieses Objekts aus dem Inhalt ergeben, wird hier auf die Besprechung der dort genannten Eigenschaften (Ausrichtung, Schriftart, Schriftgröße usw.) verzichtet.

Falls Sie einen Text mit weiteren optischen "Tricks" hervorheben wollen (z.B. vertiefte Darstellung), müssen Sie hinter den Background Text einen Rahmen stellen.

Es ist nicht möglich, einen Background Text während der Laufzeit eines Programms zu ändern. Um dies zu erreichen müssen Sie an Stelle eines Background Textes ein Data Field benutzen.

1.3.4.2.1.2 Frame

Frames oder Rahmen werden benutzt, um andere Objekte hervorzuheben oder eine Gruppe von anderen Objekten so darzustellen, daß ein inhaltlicher Zusammenhang deutlich wird. Allerdings muß dabei deutlich getrennt werden von der Group Box: Nur die Group Box kann Objekte (z.B. mehrere Radio Buttons) so zusammenfassen, daß eine Entweder/oder- Auswahl möglich wird.

Um einen Frame einzufügen, klicken Sie in der Tool Palette auf das dafür vorgesehene Symbol:

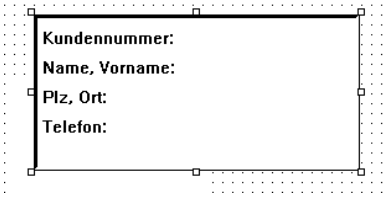
Abb. 1.38: Symbol für Rahmen in der Tool Palette.

Ziehen Sie dann den Mauszeiger in das Formularfenster, klicken Sie ein erstes Mal, halten Sie die Maustaste gedrückt, öffnen Sie einen Rahmen indem Sie die Maus nach der Seite und oben bewegen, lassen Sie nun die Maustaste los. Der Rahmen ist damit eingefügt. Klicken Sie mit der rechten Maustaste auf den Rahmen, um den Customizer zu öffnen.

Von den im Customizer angebotenen Möglichkeiten den Rahmen zu ändern sind folgende Eigenschaften interessant:

- corners (runde oder eckige Kanten);
- border style (Art des Rahmens: z.B. drop shadow für einen vertieften Rahmen, s.u.);
- border thickness (Stärke des Rahmens).

Abb. 1.39: Frame mit Background Texten.



Auch hier ergeben sich die anderen Möglichkeiten zur Einstellung des Aussehens von Rahmen aus der Übersicht im Customizer.

1.3.4.2.1.3 Line

Ein Line-Objekt ist ein Objekt, das eine Linie oder einen Strich auf dem Bildschirm darstellt. Dieser Strich kann benutzt werden, um eine Anzeige nach inhaltlichen Gesichtspunkten zu trennen oder einen Teil der Information hervorzuheben.

Um ein Line Objekt einzufügen, klicken Sie auf das dafür vorgesehene Symbol der Tool Palette.

Abb. 1.40: Line-Symbol der Tool Palette.

Ziehen Sie den Mauszeiger auf das Form Window, klicken Sie einmal, ziehen Sie die Maus zur Seite und klicken Sie nochmals. Die Linie ist damit eingefügt. Die im Customizer angebotenen Möglichkeiten zur Änderung der Linie ergeben sich von selbst, so daß hier auf weitere Kommentare verzichtet wird.

1.3.4.2.1.4 Picture

Unter einem Picture-Objekt wird ein Bild verstanden, das als Child Objekt in einem Form Window, einem MDI Window oder einer Dialog Box dargestellt ist (nicht aber in einem Table Window). Denkbar sind z.B. Firmenlogos oder Zeichnungen, weiterhin die Verwaltung großer Mengen an Zeichnungen oder die mit einer Kamera (Camcorder) aufgenommenen und digital gespeicherten Bilder.

Das Bild kann entweder als DOS-Datei vorliegen oder es kann als OLE-Objekt zur Laufzeit eingebunden werden. Dabei werden folgende Formate unterstützt:

Extension	Art
*.TIF	Tagged Image Files
*.PCX	Paintbrush
*.GIF	Graphics Interchange Format
*.DIB	Device Independant Bitmap
*.BMP	Device Dependant Bitmap
*.WMF	Windows Meta File
*.ICO	Icon File

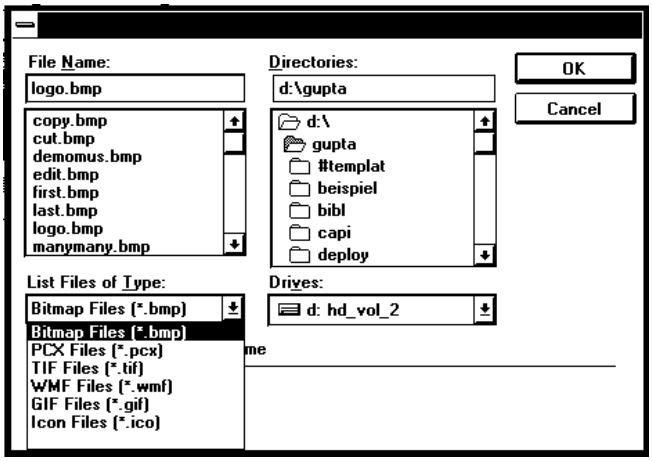
Um ein Bild in ein Form Window einzufügen klicken Sie zunächst auf das dafür vorgesehene Symbol in der Tool Palette:

Abb. 1.41: Symbol für Picture in der Tool Palette.

Ziehen Sie den Mauszeiger auf das Form Window, klicken Sie ein erstes Mal, halten Sie die Maustaste gedrückt, vergrößern Sie den angedeuteten Rahmen und klicken Sie ein zweites Mal.

Jetzt muß definiert werden, welches Bild zur Anzeige verwendet werden soll. Hierzu öffnen Sie den Customizer (Klick mit der rechten Maustaste auf das Bild), wählen Sie die Einstellung von "Picture Contents", "File Name". Sie erhalten jetzt ein Auswahlfenster, in dem Sie die Datei auswählen können, die das Bild enthält.

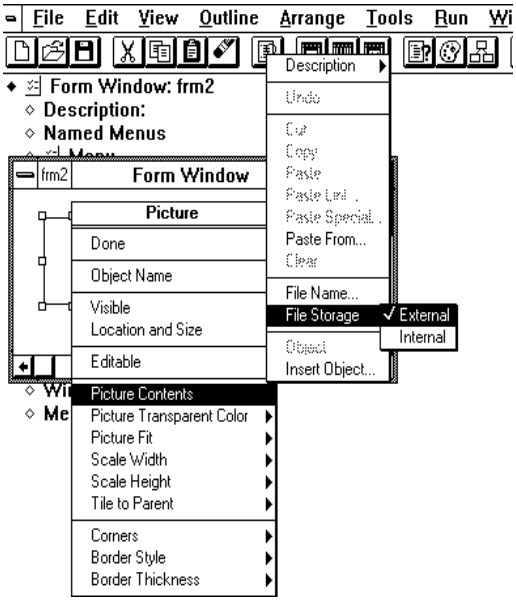
Abb. 1.42: Auswahlfenster für Datei mit Bild.



Zur Darstellung eines Bildes können Sie z.B. die Datei "Logo.Bmp" auswählen und einfügen lassen. Damit ist ein Bild fest eingefügt.

Neben der eben beschriebenen Möglichkeit, ein Bild fest einzufügen, kann ein Bild auch aus der Zwischenablage oder über eine OLE-Verbindung eingefügt werden. Diese Möglichkeit findet sich ebenfalls im Customizer (vgl. die folgende Abbildung).

Abb. 1.43: Customizer mit den Möglichkeiten ein Bild einzufügen.



"File Storage External" bedeutet, daß das Bild extern (außerhalb der Anwendung) auf der Platte gespeichert wird; "internal" bedeutet, daß das Bild in die eigene Anwendung eingebaut wird. Das vergrößert zwar den Speicherplatzbedarf der Anwendung, macht Sie aber von den Bilddateien auf einem Zielrechner unabhängig.

Schließlich können Sie über die Option "Insert Object" (im mittleren Fenster unten) ein beliebiges Objekt einfügen lassen. Hierzu zählen alle Anwendungen, die in Ihrem System als "Lieferanten" von Objekten in Frage kommen, also grundsätzlich alle neueren Windows-Programme. Auch Tonsequenzen können hiermit in eine Anwendung eingebaut werden (die sehr sehenswerte Beispielanwendung "Sample.App", die im Lieferumfang von SQLWindows enthalten ist, zeigt dies).

Interessant ist die Eigenschaft "Tile to Parent", über die Sie anweisen können, daß ein eingefügtes Bild auf den gesamten Hintergrund des Parent Windows (z.B. ein Form Window) vergrößert werden soll. Sie können damit z.B. ein Muster als Hintergrund einer Eingabemaske verwenden.

Bilder können - im Gegensatz zu den vorstehend genannten Child Objekten - Nachrichten empfangen (ein Klick oder Doppelklick z.B. auf einer Linie bleibt dagegen ohne Wirkung). Insbesondere die Nachrichten SAM_Click bzw. SAM_DoubleClick werden benötigt, um z.B. ein eingebettetes Objekt, z.B. eine Ton-Sequenz, nach einem Klick bzw. Doppelklick zu starten (die dazu nötigen Schritte sind ausführlich in den Originalhandbüchern erläutert).

1.3.4.2.2 Objekte zur Ein- und Ausgabe numerischer Daten, Zeichen und Datumswerten

Die in diesem Kapitel vorgestellten Child Objekte unterscheiden sich von den zuvor dargestellten dadurch, daß Sie damit Daten (häufig aus Tabellen oder Variablen) ein- und ausgeben können. Ihrer Natur gemäß ist die Verwendung dieser Objekte komplexer als z.B. die Verwendung eines Rahmens, weshalb Sie die hier vorgestellten Objekte und die dabei verwendeten Module sorgfältig nachvollziehen sollten.

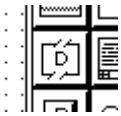
1.3.4.2.2.1 Data Field

Unter einem Data Field versteht man ein Objekt, das dazu benutzt wird, Daten ein- oder auszugeben bzw. zu editieren. Die dabei verwendeten Daten können aus einer Tabelle oder Variablen stammen oder vom Benutzer eingegeben worden sein.

Einfügen eines Data Fields in ein Formular

Klicken Sie in der Tool-Palette einmal auf das Symbol für das Data Field,

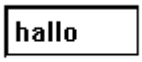
Abb. 1.44: Symbol für ein Data Field.



ziehen Sie den Mauszeiger auf Ihr Fenster frmMain, klicken Sie dort einmal mit der linken Maustaste.

Das Data Field ist damit eingefügt und kann benutzt werden. Bereits jetzt können Sie nach dem Start der Anwendung (F7-Taste) Eingaben in das Data Field vornehmen, jedoch werden diese Daten nicht weiterverarbeitet.

Abb. 1.45: Data Field mit eingegebenem Text.



Das eben eingefügte Objekt hat den Namen "df1" (oder "df2", falls sie bereits ein zweites Data Field eingefügt haben). Um das Data Field anzupassen, öffnen Sie mit einem Klick der rechten Maustaste den Customizer. Die folgende Übersicht zeigt und kommentiert die dort möglichen Einstellungen.

Done	Schließen des Optionsfensters
Object Name	Name des Objekts, vorgeschlagen z.B. df1, df2 usw. Häufig wird der Name abzuändern sein in dfKundenNummer, dfKundenName usw.; damit erhöht sich die Lesbarkeit der Programme beträchtlich.
Visible	Anzeige des Objekts: ja/nein.
Location and Size	Ort und Größe des Objekts.
Data Type	Mögliche Datentypen, die in dem Data Field dargestellt werden können (Date/Time, Number, String, Long String). Sie sollten sich immer vergewissern, daß hier der richtige Datentyp eingestellt ist. SQLWindows schlägt bei neuen Datenfeldern immer den Datentyp "String" vor.
Max Data length	Maximale Anzahl der Zeichen, die eingegeben werden können. Auch hier sollten Sie immer eine Abstimmung mit der Spaltenbreite Ihrer Tabelle vornehmen, da es bei überlangen Einträgen, d.h. solchen die über die Breite der Spalte einer Tabelle hinausgehen, beim Speichern der Daten zu Laufzeitfehlern kommt.
Editable	Ist das Datenfeld editierbar oder nicht: Sollen Werte eingegeben und verändert werden können bzw. sollen Werte nur angezeigt werden?
Border	Anzeige eines Rahmens: ja/nein.
Justify	Ausrichtung der Anzeige: links-/ rechtsbündig oder zentriert (SQLWindows schlägt immer linksbündig vor).
Format	Format der Anzeige. Abhängig vom Datentyp (Date/Time, Number, String, Long String) werden hier verschiedene Formate für die Anzeige vorgeschlagen. Diese Formate lassen sich nach Ihren Vorstellungen verändern, Sie sollten davon Gebrauch machen.

Input Mask	Eingabemaske, die sämtliche neu eingegebenen Zeichen formatiert und z.B. aus kleinen Buchstaben große macht oder Bindestriche einfügt. Folgende Symbole stehen zur Verfügung: ● **X** Alle Zeichen, wie eingegeben ● **!** Alle Zeichen, aber in Großbuchstaben angezeigt ● **a** nur Zeichen, keine Ziffern ● **A** nur Zeichen, keine Ziffern, in Großbuchstaben ● **9** nur Ziffern (0-9) ● **n** Zeichen und Ziffern wie eingegeben ● **N** Zeichen und Ziffern, Zeichen in Großbuchstaben
Country	Abweichend von der Grundeinstellung in Windows können andere Ländereinstellungen gewählt werden.
Background Color	Hintergrundfarbe des Feldes.
Text Color	Textfarbe des angezeigten/eingegebenen Textes.
Font Name	Name der Schriftart, die zur Anzeige verwendet wird.
Font Size	Größe der Schriftart.
Font Enhancement	Zusatzeinstellungen zur Schriftart (z.B. Unterstreichung, Fettdruck)
Edit Actions	Wechsel zum Outline-Fenster, um die Aktionen zuzuweisen.

Beachten Sie, daß Sie Einstellungen wie z.B. Hintergrundfarbe, Textfarbe, Schriftart usw. erst dann ändern, wenn Ihre Anwendung funktioniert. Außerdem müssen Sie prüfen, ob der Rechner, auf dem das fertige Programm später eingesetzt wird, bzw. die dort eingebaute Grafikkarte eine spezielle Schriftart und Größe überhaupt zuläßt, hier kann es durchaus zu Überraschungen kommen.

Verwenden eines Data Field

Zwei Varianten sind bei der Arbeit mit Data Fields möglich:

- Anzeigen von Daten (Inhalt aus Tabellenspalte oder Inhalt einer Variablen);
- Eingabe von Information durch den Benutzer und anschließendes Abspeichern.

Dabei muß noch berücksichtigt werden, ob das Data Field editierbar sein soll oder ob der angezeigte Inhalt unveränderbar sein soll (im Customizer über Editable einzustellen).

Bei der **Anzeige von Daten aus einer Tabelle** soll in dem folgenden Beispiel nur der Inhalt eines einzigen Feldes angezeigt werden (also z.B. das Wort "Hammer"), das aus der Tabelle Lager zu lesen sein wird. Um das Beispiel zu vereinfachen, soll die Anzeige so eingerichtet werden, daß sie ohne weiteres Zutun stattfinden soll. Hierbei gilt es zu bedenken, daß die in SQLWindows verwendeten Nachrichten neben diesem Fall, daß unmittelbar nach der Erzeugung eines Objekts eine Anzeige stattfinden soll, auch andere Möglichkeiten vorsehen, z.B. die Anzeige von Daten nach dem Drücken eines Push Buttons (vgl. Kap. 1.3.6.3.3.1).

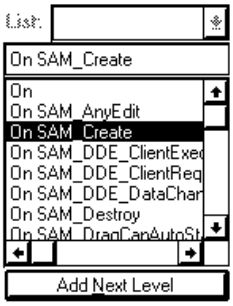
Fügen Sie hierzu ein neues Data Field in Ihr Formular frmMain ein oder benutzen Sie das oben eingefügte (der Name des Data Fields lautet df1).

In der Sektion Message Actions des Data Fields muß nun angeordnet werden, daß unmittelbar nach Erzeugung des Data Fields ein Wert aus der Tabelle Lager zu lesen ist. Die Anweisung hierzu lautet **On SAM_Create**, Sie brauchen das aber nicht zu tippen, sondern können es aus den Outline Options auswählen. Lassen Sie sich zunächst den Customizer anzeigen (mit rechter Maustaste einmal klicken), dann wählen Sie "Edit Actions". Die jetzt angezeigte Sektion der "Message Actions" ist zunächst noch leer:

Abb. 1.46: Message Actions des Data Fields.

Lassen Sie die Outline Options anzeigen (F2-Taste drücken), suchen Sie in dem unteren Fenster (Add Next Level) den Eintrag "On SAM_Create".

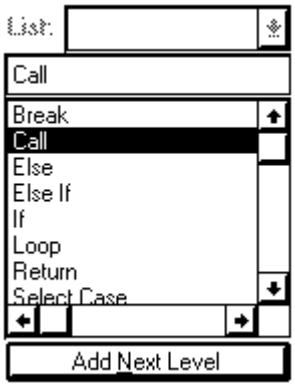
Abb. 1.47: Inhalt der Outline Options.



Klicken Sie zweimal auf dem Eintrag "On SAM_Create", er wird dann in Ihre Message Actions eingefügt, so wie das der folgende Programmausschnitt zeigt.

Jetzt wählen Sie aus dem eben benutzten Auswahlfenster (mit der Beschriftung "Add Next Level"), die Anzeige hat sich inzwischen geändert, die Anweisung "Call" mit einem Doppelklick aus.

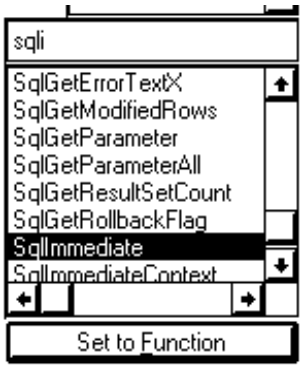
Abb. 1.48: Veränderter Inhalt der Outline Options.



Auch diese Anweisung wird in Ihre Message Actions übernommen.

In dem Fenster, in dem "Sal+User Functions" angezeigt werden, geben Sie jetzt die Zeichen "sqli" in das Data Field ein; SQLWindows sucht daraufhin Einträge, die mit dieser Zeichenfolge beginnen.

Abb. 1.49: Eingabe von "sqli" (oben), Anzeige von "SqlImmediate" (unten).



Klicken Sie jetzt zweimal auf dem angezeigten Eintrag "SqlImmediate", um diese Funktion zu übernehmen. Ihre Message Actions hat jetzt folgenden Inhalt:

- **Message Actions**
  - **On SAM_Create**
    - **Call SqlImmediate(String)**

Das Wort "String" zeigt an, daß an dieser Stelle ein String, d.h. eine Zeichenfolge, einzugeben ist. Hier muß jetzt die auszuführende Sql-Anweisung eingegeben werden. Sie brauchen das Wort String nicht zu löschen,

es verschwindet, bei Beginn der Eingabe, da es markiert ist, automatisch. Geben Sie folgende Sql-Anweisung ein (achten Sie auf jedes Zeichen - Anführungszeichen, Doppelpunkt vor df1! -):

'select name from lager into :df1'

Drücken Sie danach die Enter-Taste, Ihr Bildschirm sollte folgende Zeilen enthalten:

```
◆ Data Field: df1
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Call SqlImmediate( 'select name from lager into :df1' )
```

Die Funktion SqlImmediate(), übersetzbar mit "führe die angegebene SQL-Anweisung sofort aus", wird immer dann benutzt, wenn nur ein einzelner Datensatz oder ein Teil eines Datensatzes (ein einzelnes Feld) aus einer Tabelle zu lesen ist. Diese Funktion ist sehr einfach zu codieren, ist aber dahingehend beschränkt, daß eben immer nur *ein* Datensatz (oder Teile davon) gelesen werden können. Häufig wird sie benutzt, wenn nach der Eingabe einer Kundennummer (bzw. einer Artikelnummer) die dazu gehörende Bezeichnung aus der Tabelle zu lesen und in einem Data Field anzuzeigen ist. Sie können Sie auch benutzen, um in einer Netzumgebung eine neue Kundennummer *sofort* auf den Server zu schreiben, so daß bei gleichzeitiger Erfassung neuer Kunden - auf mehreren Rechnern - tatsächlich jeder neue Kunde eine eindeutige Kundennummer erhält..

Im vorliegenden Beispiel wird der Name des *ersten* Artikels aus der Tabelle LAGER gelesen und in das Data Field df1 kopiert (an dem vorangestellten Doppelpunkt erkennt SQLWindows, daß es sich um ein Data Field und nicht um eine Variable handelt). Genaugenommen wissen wir zwar nicht, welcher Name aus der Tabelle mit Lagerdaten daraufhin angezeigt wird, aber in diesem Fall ist das unerheblich. Falls kein Name angezeigt wurde, ist es denkbar, daß in Ihrer Tabelle ein leerer Datensatz an erster Stelle steht. Falls dem so ist, könnten Sie die oben stehende Sql-Anweisung erweitern:

'select name from lager into :df1 where artnr = 1000'

Nun sollte es funktionieren, vorausgesetzt, Sie haben einen entsprechenden Eintrag in Ihrer Tabelle (Sie können in SQLWindows über den Menüpunkt "Edit", "Database Browser" mit der Anweisung "select * from lager" alle Datensätze anzeigen lassen, ohne SQLWindows zu verlassen und dann die oben stehende Anweisung an Ihre Daten anpassen).

Eingabe von Daten aus einem Data Field in eine Tabelle

Um Information aus einem Data Field in einer Tabelle zu speichern, müssen Sie eine Sql-Anweisung so aufbauen, daß die Daten aus dem Data Field mittels "insert into" in die Tabelle eingefügt werden. Dies ist aber erst dann sinnvoll, wenn Sie einen Push Button haben, der das gewünschte nach einem Mausklick durchführt. Das folgende Beispiel beschreibt den Vorgang, muß aber dabei auf einen Push Button zurückgreifen, der erst später beschrieben wird (vgl. Kap. 1.3.6.3.3.1).

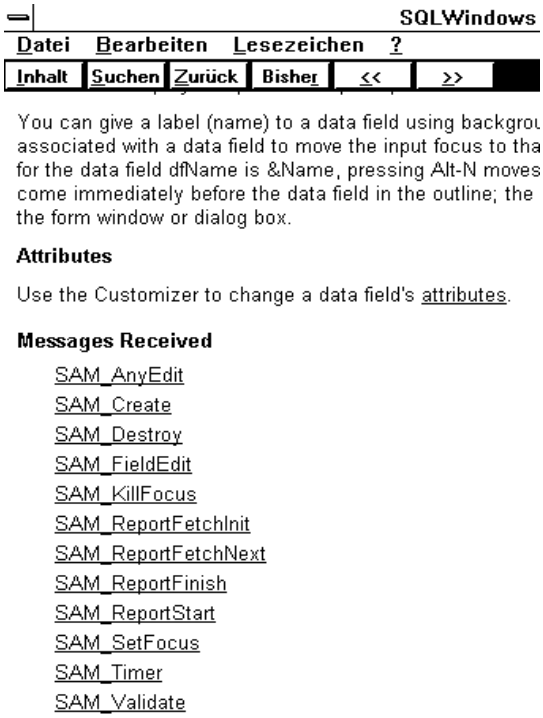
Fügen Sie zwei Data Fields in das leere Formular "frmMain" ein (eines soll für die Kundennummer, eines für den Namen des neu aufzunehmenden Kunden sein). Ein Push Button soll die Aktion der Datenübergabe zur Tabelle auslösen, fügen Sie auch diesen ein. Beschriften Sie den Push Button mit **sichern**. Klicken mit der rechten Maustaste auf den Push Button, wählen Sie die Option "Edit Action" und tragen Sie dort folgenden Text ein:

```
On SAM_Click
    Call SqlImmediate( 'insert into kunden (kundennummer, name)
    values (:df1, :df2)').
```

Im ersten Klammerpaar innerhalb der insert-Anweisung, werden die Spaltennamen genannt, in die Informationen einzufügen sind (hier die Spalten kundennummer und name). Nach dem Schlüsselwort "values", hinter dem die Werte zu nennen sind, folgt in der gleichen Reihenfolge wie bei den Spaltennamen die Liste der zu übergebenden Werte. Starten Sie die Anwendung (F7). Um zu prüfen, ob die Übergabe der Daten aus den Data Fields in die Tabelle funktioniert hat, können Sie über die Option "Edit", "Database Browser" die Daten mit einer Sql-Select-Anweisung anzeigen lassen.

Neben der genannten Möglichkeit, SQLWindows Application Messages zu verarbeiten (im vorliegenden Beispiel wurde "SAM_Create" benutzt), kann ein Data Field auch andere Messages verarbeiten. Um die Übersicht aller möglichen Nachrichten zu erhalten, drücken Sie in SQLWindows die F1-Taste für die Hilfe. Wählen Sie aus dem Hilfe-Index den Eintrag **SqlWindows Objects**. in dem dann erscheinenden Fenster mit einer Übersicht aller Objekte klicken Sie auf Data Field (rechts), Sie erhalten dann folgende Übersicht.

Abb. 1.50: Hilfesystem in SQLWindows: Messages die von Data Fields verarbeitet werden können.



Die Messages "SAM_AnyEdit", "SAM_SetFocus", "SAM_Destroy" und "SAM_Report*" werden in späteren Kapiteln noch eingehend besprochen.

1.3.4.2.2.2 Multiline Field

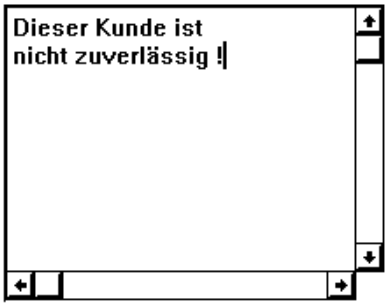
Ein Multiline Field bietet, ähnlich wie ein Data Field, die Möglichkeit, Daten ein- und auszugeben. Abweichend von einem Data Field dürfen die einzugebenden Informationen (nur String oder Long String Datentypen sind möglich) deutlich größer sein: Bis zu 32 KB große Daten können über ein solches Objekt ein- oder ausgegeben und durch den Benutzer editiert werden.

Um ein Multiline Field in ein Formular einzufügen, klicken Sie zunächst in der Tool Palette auf das dafür vorgesehenen Symbol (verwechseln Sie es aber nicht mit dem Symbol für List Box oder Tabelle):

Abb. 1.51: Symbol für Multiline Field.

Ziehen Sie den Mauszeiger in Ihr Formular und plazieren Sie das Objekt darauf. Nachdem Sie die Anwendung gestartet haben, können Sie sofort Daten eingeben:

Abb. 1.52: Multiline Field mit eingegebenen Daten.



Diese Art, Information zu erfassen, ist insbesondere dann sinnvoll, wenn Sie Kommentare, wie z.B. Angaben zum Lebenslauf oder ähnliches, in einer Tabelle abspeichern möchten.

Natürlich ist noch nicht codiert, wie oder wo der oben sichtbare Text zu speichern ist. Das Verfahren ist aber mit zuvor oben geschilderten Verfahren bei Data Fields identisch und wird hier deshalb nicht wiederholt.

Die folgende Übersicht nennt die Inhalte des Customizers.

Done	Schließen des Optionsfensters.
Object Name	Name des Objekts, vorgeschlagen z.B. ml1, ml2 usw.
Visible	Anzeige des Objekts: ja/nein.
Location and Size	Ort und Größe des Objekts.
String Type	Zwei String Datentypen sind möglich: • String (Voreinstellung) und • Long String
Max Data length	Maximale Anzahl der Zeichen die eingegeben werden können, voreingestellt ist 1000, maximal sind 32 KB (Long String) bzw. 1000 (String) möglich .
Editable	Ist das Multiline Field editierbar oder nicht: Sollen Werte eingegeben / verändert werden können bzw. sollen Werte nur angezeigt werden.
Border	Anzeige eines Rahmens:ja/nein.
Word Wrap	Soll ein automatischer Zeilenumbruch durchgeführt werden (voreingestellt: nein).
Vertical Scroll	Soll eine vertikale Scroll Bar angezeigt werden (voreingestellt: ja)
Background Color	Hintergrundfarbe.
Text Color	Textfarbe des angezeigten/eingegebenen Textes.
Font Name	Name der Schriftart, die zur Anzeige verwendet wird.
Font Size	Größe der Schriftart.
Font Enhancement	Zusatzeinstellungen zur Schriftart (z.B. Unterstreichung, Fettdruck)
Edit Actions	Wechsel zum Outline-Fenster, um die Aktionen zuzuweisen.

Ein ausführliches Beispiel zur Verwendung von Multiline Fields findet sich im Anhang dieses Handbuchs in der Beispielanwendung dBASE.APP (Kap. 4.6).

1.3.4.2.2.3 List Box

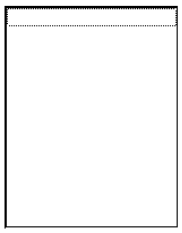
Eine List Box ist ein Objekt, das (in der Regel gleichartige) Informationen in einer Spalte untereinander darstellt. Die List Box stellt die Informationen zur Verfügung, damit der Benutzer eine Auswahl treffen kann; eine Eingabe neuer Informationen ist dagegen nicht möglich. Es kann definiert werden, ob eine einfache (single) oder eine mehrfache Auswahl (multiple selection) von Eintragungen durch den Benutzer möglich sein soll.

Um eine List Box in ein Formular einzufügen, klicken Sie zunächst auf das List Box Symbol in der Tool Palette.

Abb. 1.53: List Box Symbol in der Tool Palette.

Fügen Sie das Objekt in das Formular ein (im Formular klicken, Rahmen vergrößern, Maustaste loslassen). Wenn Sie die Anwendung starten, wird eine leere leere List Box angezeigt.

Abb. 1.54: Leere List Box.



Um nun Informationen in eine List Box zu bringen, stehen zwei Funktionen zur Verfügung, die für jeweils einen Zweck vorgesehen sind:

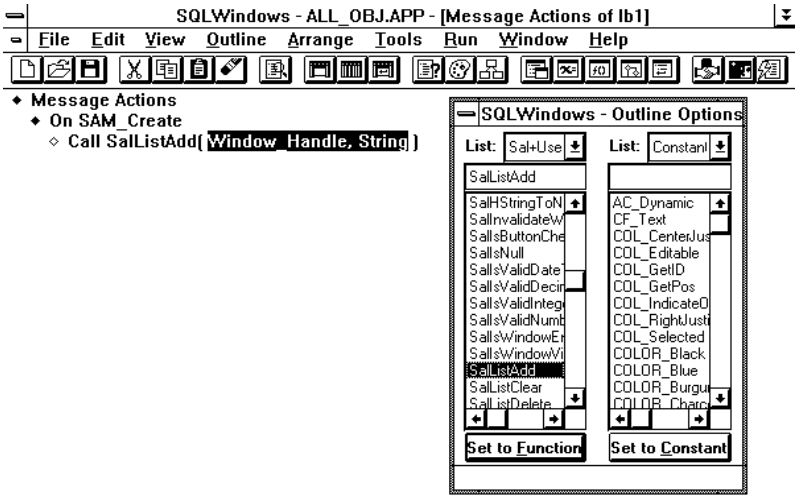
- **SalListAdd()**, die genau *einen* Eintrag in die List Box einfügt, sowie
- **SalListPopulate()**, die in Verbindung mit Sql-Anweisungen benutzt wird, um eine beliebige Anzahl von Datensätzen aus einer Tabelle anzuzeigen.

Beide Funktionen sollen mit der oben eingefügten List Box getestet werden. Dazu muß natürlich festgelegt werden, wo diese diese Funktionen aufgerufen werden. Es handelt sich dabei wieder um die Sektion "Message Actions" der List Box. Öffnen Sie den Customizer (mit der rechten Maustaste auf der List Box klicken) und wählen Sie "Edit Actions".

Füllen einer List Box mit einzelnen Einträgen

Um einzelne Einträge aufzunehmen, benutzen Sie die Funktion SalListAdd(). Nachdem Sie diese Funktion aus den Outline Options in die Message Actions Sektion eingefügt haben müssen zwei Angaben vorgenommen werden:

Abb. 1.55: Funktion **SalListAdd()** in der Sektion Message Actions der List Box



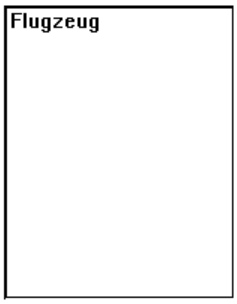
Bei dem Aufruf der Funktion müssen Sie einen Window Handle und einen String angeben. Der **Window Handle**, d.h. die eindeutige Bezeichnung eines Objekts, lautet hier "lb1". Als String können Sie den Namen einer Variablen oder ein Wort (begrenzt durch einfache Anführungszeichen) eingeben.

Um die Funktion zu testen, ergänzen Sie, so daß der Funktionsaufruf folgendermaßen aussieht:

```
◆ Message Actions
  ◆ On SAM_Create
    ◇ Call SalListAdd( lb1, 'Flugzeug')
```

Nach dem Start der Anwendung erhalten Sie folgenden Inhalt in der List Box angezeigt.

Abb. 1.56: List Box mit einem Eintrag.



Selbstverständlich können Sie weitere Einträge anfügen, um mehr Zeilen in der List Box anzeigen zu lassen.

```
◆ Message Actions
  ◆ On SAM_Create
    ◇ Call SalListAdd( lb1, 'Flugzeug')
    ◇ Call SalListAdd( lb1, 'Schiff')
    ◇ Call SalListAdd( lb1, 'Fahrrad')
```

Bei größeren Anwendungen ist dies etwas mühsam, da Sie immer zuerst zur der List Box gehen müssen, um weitere Einträge vorzunehmen. Deshalb wählt man üblicherweise einen anderen Weg:

- Definieren Sie einen Array mit dem Namen **asListen[5]** in der Sektion Global Declarations Ihrer Anwendung (Sie könnten diesen Array auch als lokale Variable definieren). Zusätzlich definieren Sie die Variablen nAnzahl, nIndex, nMin, nMax (alle vom Datentyp Number).

```
◇ String: asListen[5]
◇ Number: nAnzahl
◇ Number: nIndex
◇ Number: nMin
◇ Number: nMax
```

- Füllen Sie den Array mit Information, hier geschieht das bei "Application Actions" am Anfang der Anwendung.

```
◆ Application Actions
  ◆ On SAM_AppStartup
    ◇ Set SqlDatabase = 'FIRMA'
    ◇ Call SqlConnect (hSql)
    ◇ Set asListen[0] = 'Apfel'
    ◇ Set asListen[1] = 'Birne'
    ◇ Set asListen[2] = 'Mirabelle'
    ◇ Set asListen[3] = 'Erdbeere'
    ◇ Set asListen[4] = 'Kirsche'
```

- Fertigen Sie in der Sektion Message Actions der List Box eine Schleife an, die Inhalte des Array in die List Box überträgt (beachten Sie, daß die oben erprobten Zeilen jetzt inaktiv sind).

```
◆ List Box: lb1
  ◇ List Initialization
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ ! Call SalListAdd( lb1, 'Flugzeug')
      ◇ ! Call SalListAdd( lb1, 'Schiff')
      ◇ ! Call SalListAdd( lb1, 'Fahrrad')
      ◇ Set nIndex = 0
      ◇ Set nAnzahl =SalQueryArrayBounds(asListen, nMin, nMax)
      ◆ While nIndex < nAnzahl
        ◇ Call SalListAdd(lb1, asListen[nIndex])
        ◇ Set nIndex = nIndex +1
```

Zunächst setzen Sie den Inhalt der Variablen nIndex auf den Wert 0. Diese Variable wird gebraucht, um innerhalb des Array die einzelnen Elemente durchzuzählen; dabei hat das erste Element den Wert 0, das

letzte Element den Wert 4. Dann wird anhand der Funktion SalQueryArrayBounds() festgestellt, wieviel Elemente der Array enthält, die Werte werden in die zwei Variablen nMin und nMax gespeichert. Innerhalb der Schleife wird die bereits oben benutzte Funktion SalListAdd() benutzt, um den Inhalt des Array "asListen" in die List Box zu schreiben. Hier wird die Funktion fünfmal aufgerufen, da der Array 5 Elemente enthält (dieser Wert ist in der Variablen nAnzahl gespeichert).

Nach dem Start bietet sich folgendes Bild:

Abb. 1.57: Mit der Funktion SalListPopulate() - unter Verwendung eines Arrays - gefüllte List Box.

Dieses Verfahren ist zwar etwas aufwendig, doch hat es den Vorteil, daß Sie von nun an unabhängig von der Sektion "Message Action" der List Box sind. Um weitere Einträge aufzunehmen und darzustellen, müssen Sie nur noch die Zahl der Array-Elemente erhöhen und die Elemente mit Worten füllen.

Füllen einer List Box mit der Funktion SalListPopulate()

Die Funktion SalListPopulate() bietet eine außerordentlich einfache Möglichkeit, Datensätze aus Tabellen in einer List Box darzustellen. Sie müssen bei dem Aufruf der Funktion SalListPopulate() drei Angaben machen:

- Handle des Objekts (hier lb2);
- Name des Sql-Handle (hier: hSql) über den die Verbindung zur Datenbank hergestellt wurde (s. Sektion Global Actions) und
- die eigentliche Sql-Anweisung (hier: "select name from lager").

Der folgende Programmtext zeigt die nötige Konstruktion, um Datensätze aus einer Tabelle in einer List Box darzustellen (die Zeilenumbrüche dienen nur der Lesbarkeit). Die Anweisungen müssen dabei in der Sektion Message Actions der List Box (hier eine zusätzliche List Box mit dem Namen lb2) eingefügt werden.

```
◆ Message Actions
  ◆ On SAM_Create
    ◇ Call SalListPopulate( lb2,
                            hSql,
                            'select name from lager' )
```

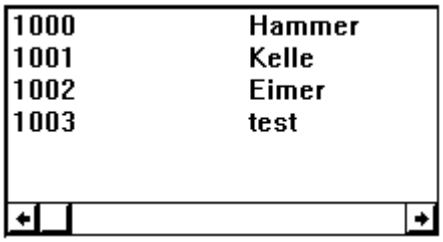
Lassen Sie das Programm starten, dann erhalten Sie in der List Box die angezeigten Daten aus der Tabelle "Lager".

Sie können sich auch mehr als nur die Daten einer einzigen Spalte anzeigen lassen. In diesem Fall müssen Sie die Sql-Anweisung folgendermaßen modifizieren:

'select artnr, name from lager'

Die vorstehende Anweisung führt zu dem folgenden Inhalt der List Box.

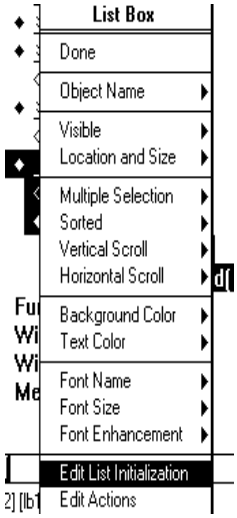
Abb. 1.58: List Box, gefüllt mit den Daten aus zwei Spalten.



List Box Initialisierung

Es existiert noch eine weitere Möglichkeit, Daten in einer List Box anzeigen zu lassen. Öffnen Sie den Customizer der List Box, beachten Sie die vorletzte Zeile "Edit List Initialization".

Abb. 1.59: Edit List Initialization.



Wählen Sie diese Option aus; Sie befinden sich dann im Outline Fenster, aber nicht bei der Message Actions Sektion der List Box, sondern im Kopf der Definition. Dort können Eintragungen vorgenommen werden. Der folgende Programmcode zeigt dies.

```
◆ List Box: lb3
  ◆ List Initialization
    ◇ Text: Ernst
    ◇ Text: Heinrich
    ◇ Text: Gerhard
```

Diese Texte werden bereits im Entwurfsfenster der List Box angezeigt. Sie können diese Art, eine List Box zu füllen, auch kombinieren mit der While-Schleife, die einzelne Elemente in die List Box einfügt.

Soviel zum Füllen der List Box. Im folgenden Abschnitt wird erläutert, wie Sie die Inhalte einer List Box verwenden können.

Markieren und Kopieren eines Elements aus in einer List Box in ein Data Field

Üblicherweise wählen Sie die in einer List Box angezeigten Informationen aus, um sie in einem anderen Objekt weiter zu verarbeiten. Häufig dient die List Box dabei als Auswahlfenster, das Sie benutzen, um Ihre Information "zusammenzuklicken", z.B. beim Verkauf von Artikeln um die Artikelnummern aus der List Box in einen Auftrag zu übernehmen (so daß diese nicht getippt werden muß).

Um die Übergabe von Daten aus einer List Box heraus zu demonstrieren, wird das folgende Beispiel vorgestellt. Ziel soll sein, daß der Benutzer durch einen Doppelklick auf einem der angezeigten Elemente das angeklickte Element in ein Data Field (df1) kopiert (der Mechanismus ist bei Verwendung eines Table Windows an Stelle eines Data Fields der gleiche).

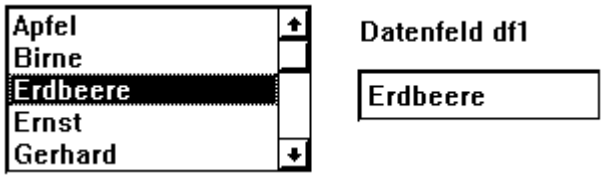
Sie benötigen die oben erzeugte List Box "lb1" und ein Data Field (df1). Öffnen Sie den Customizer der List Box (rechte Maustaste), wählen Sie "Edit Actions" und formulieren Sie folgende Anweisungen:

```
◆ On SAM_DoubleClick
  ◇ Set nIndex = SalListQuerySelection(lb1)
  ◇ Call SalListQueryText(lb1, nIndex, df1)
```

Die Variable nIndex wurde zwar für eine andere Aufgabe eingeführt, wird hier aber verwendet, um die Erklärung kurz zu halten (in späteren Anwendungen sollten Sie hierfür eine eigene Variable erzeugen).

Zunächst wird die von SQLWindows intern verwaltete Nummer des Eintrags festgestellt, auf dem doppelt geklickt wurde. Hierzu wird die Funktion SalListQuerySelection() benutzt. Danach wird die Funktion SalListQueryText() benutzt, um den Text zu holen, der hinter der internen Nummer des Eintrags steckt. Dieser Text wird dann an das Data Field "df1" übergeben, welches nun folgendermaßen aussieht:

Abb. 1.60: Kopieren eines List Box Elementes in ein Data Field



1.3.4.2.2.4 Combo Box

Eine Combo Box entspricht im Verhalten grundsätzlich einer List Box, bietet gegenüber dieser aber noch weitere Eigenschaften, so daß man sagen könnte, eine Combo Box ist eine Kombination aus List Box und darüber stehendem Data Field).

- Sie können, im Gegensatz zu einer List Box, Daten eingeben (dies kann beim Entwurf der Combo Box auch unterbunden werden s. Customizer: Editable).
- Die Combo Box hat einen nach unten aufklappbaren Teil, den Sie mit einer Schaltfläche anzeigen können; Sie können auch wählen, daß dieser Teil immer nach unten aufgeklappt bleibt (Customizer: Always show list).
- Eine Mehrfachauswahl von Elementen ist in der Combo Box *nicht* möglich.
- In der Combo Box können auch Daten vom Typ Long String dargestellt werden (Customizer: String Type).

- Die Übergabe eines selektierten (angeklickten) Elementes aus der Combo Box in ein anderes Objekt ist einfacher als es bei einer List Box ist (s. unten).

Da der Inhalt des Customizers nahezu identisch wie bei der List Box ist, wird auf eine Wiedergabe verzichtet.

Zum Einfügen einer Combo Box, klicken Sie zunächst auf dem Symbol für Combo Box in der Tool Palette.

Abb. 1.61: Combo Box Symbol der Tool Palette.

Fügen Sie die Combo Box in das Fenster frmMain ein. Beachten Sie dabei, daß der angedeutete Rahmen die Größe der *geöffneten* Combo Box zeigt.

Zum Füllen einer Combo Box mit Informationen stehen drei Möglichkeiten zur Verfügung:

- Füllen mit den Elementen eines Array (unter Verwendung der Funktion **SalListAdd()** mit einer Schleife);
- Füllen mit den Datensätzen einer Tabelle (unter Verwendung der Funktion **SalListPopulate()** mit einer Select-Anweisung);
- Füllen mit Texten, die in der List Initialisation angegeben werden (über Customizer, **Edit List Initialization**).

Die drei Möglichkeiten sind in vorstehendem Kapitel zur List Box ausführlich beschrieben und werden deshalb nicht wiederholt. Um die Varianten zu testen, können Sie die Zeilen aus der Sektion Message Actions (der List Box) mittels Zwischenablage kopieren und in die Message Actions Sektion (der Combo Box) einfügen.

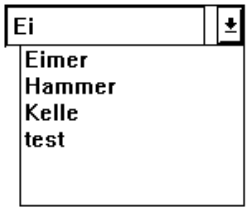
Die etwas veränderte Sektion der Combo Box hat dann folgendes Aussehen:

```
◆ Combo Box: cmb1
  ◇ List Initialization
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Call SalListPopulate( cmb1,
                              hSql,
                              'select  name from lager' )
```

Beachten Sie, daß der Window handle verändert wurde (aus "lb1" wurde "cmb1"), außerdem wurde die Sql-Anweisung auf den Namen einer Spalte reduziert.

Nach dem Start der Anwendung sehen Sie folgenden Inhalt der Combo Box:

Abb. 1.62: Combo Box mit Inhalt (unter Verwendung von SalListPopulate()).



Wie bereits oben erwähnt, ist die Übergabe eines Elements aus einer Combo Box, z.B. in ein Data Field, einfacher als die Übergabe eines Elements aus der List Box. Bei der Combo Box müssen Sie lediglich eine Zuweisung mittels "Set" vornehmen, um das gerade angeklickte Element in das andere Objekt zu übergeben. Der Programmcode (unten) zeigt dies:

```
◆ Message Actions
  ◆ On SAM_Create
    ◇ Call SalListPopulate( cmb1,
                            hSql,
                            'select  name from lager' )
  ◆ On SAM_Click
    ◇ Set df1 = cmb1
```

Die Anweisung "Set df1 = cmb1" genügt, um das angeklickte Element in dasData Field (df1) zu kopieren.

1.3.4.2.2.5 Table Window, Quest Window

Diese beiden Child Objekte existieren auch in Form von Top-Level Objekten. Da sie dort bereits beschrieben sind, sollen hier nur einige Eigenschaften des häufiger verwendeten Objekttyps "Child Table Windows" genannt werden.

Zunächst soll eine Übersicht die beiden Typen von Table Windows (Top-Level bzw. Child Level) gegenüberstellen.

Vergleich von **Top-Level** mit **Child-Level Table Window**:

	Top-Level	Child-Level
Art	wie ein Form Window	Child Objekt von Form Window oder Dialogbox.
Inhalt	columns	columns
Menü	möglich	nicht möglich
Größe	veränderbar	nicht veränderbar
Aufruf	mit Funktion SalCreateWindow()	mit übergeordneten Top-Level Window

Da ein Top-Level table Window nur über ein Menü zu steuern ist, ist es auf den ersten Blick häufig unübersichtlicher als ein normales Form Window, das neben dem Child Table Window noch zusätzlich Push Buttons zur Steuerung von Aktionen enthalten kann. Allerdings kann dies abgemildert werden, indem z.B. ein MDI Window verwendet wird, das - immer sichtbare - Option - oder Push Buttons anzeigt (als Accessories, vgl. dazu Kapitel zu MDI Windows 1.3.4.2).

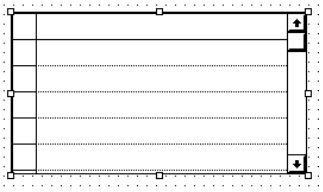
Table Windows sind wohl die am häufigsten verwendeten Objekttypen in Datenbankanwendungen. Sie bieten enorme Flexibilität und sind ausgesprochen leistungsfähig. Während es äußerst einfach ist, vorhandene Datensätze in einem solchen Tabellenobjekt anzuzeigen, ist die Dateneingabe etwas komplexer, aber nicht eigentlich schwierig.

Um ein Table Window als Child object in ein Form Window einzufügen, klicken Sie zunächst auf das dafür vorhandene Symbol (*nicht* auf der Schaltfläche mit der Aufschrift "Table"!) in der Tool Palette.

Abb. 1.63: Symbol für Table Window in der Tool Palette.

Klicken Sie dann einmal auf das Form Window, ziehen Sie den angedeuteten Rahmen etwas größer und lassen Sie die Maustaste wieder los.

Abb. 1.64: Child Table Window in ein Formular eingefügt.



In dieses Table Window sollen nun Spalten (columns) als "Child Window Objekte" eingefügt werden. Bewegen Sie dazu den Mauszeiger an den linken Rand der Tabelle, er ändert nun die Form, klicken Sie

einmal, um die erste Spalte einzufügen. Bewegen Sie den Mauszeiger an den rechten Rand der eben eingefügten Spalte und fügen Sie durch erneutes klicken eine zweite Spalte ein. Die Spalten haben die Namen "col1" bzw. "col2", Sie können dies mit dem Customizer überprüfen.

Beachten Sie, daß der Customizer verschiedene Informationen anzeigt:

- Klicken Sie mit der rechten Maustaste auf dem **Titel einer Spalte** (untitled), dann sehen Sie den Customizer der Spalte.

- Klicken Sie auf die **Zeilen der Tabelle**, dann sehen Sie den Customizer der Tabelle.

Um die Tabelle unmittelbar nach Ihrem Aufbau mit Datensätzen aus der Tabelle Lager zu füllen, lassen Sie den Customizer der Tabelle anzeigen. Wählen Sie die Option "Edit Actions" und nehmen Sie dort folgenden Eintrag in der Sektion Message Actions vor:

```
◆ Child Table: tbl1
  ◆ Contents
    ◆ Column: col1
      ◇ Message Actions
    ◆ Column: col2
      ◇ Message Actions
  ◇ Functions
  ◇ Window Variables
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Call SalTblPopulate( tbl1,
                             hSql,
                             'select artnr, name from lager into :col1, :col2',
                             TBL_FillNormal )
```

Achten Sie auf die Doppelpunkte vor col1 und col2 (die Zeilenumbrüche sind nicht nötig). Starten Sie die Anwendung, Sie sollten dann folgende Tabelle sehen:

Abb. 1.65: Tabelle als Child Objekt, eingebettet in ein Form Window.

	(untitled)	(untitled)	
	1000	Hammer	
	1001	Kelle	
	1002	Eimer	
	1003	test	

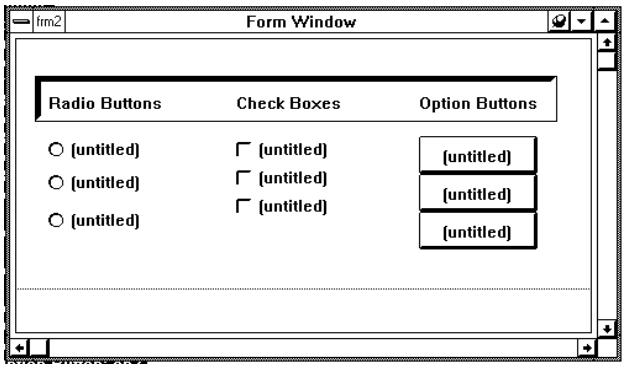
Neben der Möglichkeit, Datensätze aus einer Tabelle darzustellen, können auch andere Datenquellen genutzt werden, z.B. Daten aus Array-Inhalten oder Daten aus DOS-Files (Beispiele hierzu finden sich in der Beispielanwendung "dBASE.APP" im Anhang).

1.3.4.2.3 Objekte, zur Ein- und Ausgabe von logischen Informationen

Zu den hierunter fallenden Objekten zählen Radio Buttons, Check Boxes und Option Buttons. Diese drei Objekte, die alle dazu dienen, "ja/nein-" oder "an/aus-" Zustände zu speichern, sind weitgehend untereinander austauschbar. Das heißt, ob Sie Radio Buttons oder Option Buttons benutzen ist unerheblich, beide haben die gleichen Merkmale. Check Boxes unterscheiden sich von ihnen dadurch, daß eine *Mehrfachauswahl* möglich ist, während beide zuerst genannten Objekte immer nur eine Alternativauswahl zulassen.

Die folgende Abbildung zeigt die drei Objekte nebeneinander als Child Objekte eines Form Windows.

Abb. 1.66: Radio Buttons, Check Boxes und Option Buttons.



Die einzelnen Objekte werden in den folgenden Abschnitten erläutert. Bei allen drei Objektarten ist zu beachten, daß logische Inhalte, so wie Sie mit den Objekten erzeugt werden, nicht direkt in eine Datenbank gespeichert werden können, da dieser (logische) Datentyp dort nicht bekannt ist. Das bedeutet, daß - falls es erforderlich ist, diese Informationen zu speichern - aus dem logischen Inhalt ein numerischer - oder ein Zeicheninhalt anzufertigen ist, bevor gespeichert werden kann. Häufig wird dies mittels einer einfachen If-Anweisung realisiert, die abhängig vom Status z.B. von Radio Buttons das Ziffernpaar 0/1 oder die Zeichen "j/n" verwendet.

Es gilt außerdem zu beachten, daß untereinander angeordnete Radio Buttons oder Option Buttons von SQLWindows als zusammengehörend interpretiert. Sie müssen also nicht zusätzlich durch eine Group Box zusammengefaßt werden. Allerdings bietet die Verwendung einer Group Box zusätzliche Möglichkeiten, die logischen Zustände der darin enthaltenen Radio Buttons zu verändern (hierfür stehen die Funktionen SalGetFirstChild() und SalGetNextChild() zur Verfügung).

1.3.4.2.3.1 Radio Buttons

Radio Buttons werden häufig in einer Gruppe untereinander angeordnet. Der Benutzer soll aus einer möglichen Auswahlmenge genau eine Option auswählen können. Immer, wenn er einen der Radio Buttons anklickt, wird ein anderer, zuvor angewählter, zurückgesetzt.

Radio Buttons kennen nur einen Datentyp (den logischen) und nur zwei Zustände: "TRUE" oder "FALSE". Sind sie angeklickt, dann enthalten Sie den Wert "TRUE", sind sie nicht angeklickt, dann enthalten Sie den Wert "FALSE" (natürlich läßt sich durch Anweisungen wie "Set rb1 = FALSE" dieser Zustand auch im Programmablauf setzen oder ändern).

Um einen Radio Button in einem Formular unterzubringen, klicken Sie zunächst auf dem Radio Button Symbol der Tool Palette.

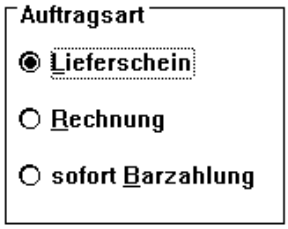
Abb. 1.67: Radio Button Symbol der Tool Palette.

Bewegen Sie den Mauszeiger in das Formular, klicken Sie einmal, um den Radio Button einzufügen. Die angezeigte Beschriftung können Sie einfach mit dem Wort "&Lieferschein" überschreiben.

Wiederholen Sie den erläuterten Vorgang, um insgesamt drei Radio Buttons nacheinander einzufügen, beschriften Sie die folgenden Radio Buttons mit "&Rechnung" bzw. "sofo&rt Barzahlung"; das Et.-Zeichen (&) sorgt dafür, daß das nachfolgende Zeichen mit einem Unterstrich dargestellt wird und dann über eine Tastenkombination (z.B. Alt+L) in der laufenden Anwendung angesteuert werden kann (diese Kurzschlüssel sollten natürlich nicht mit der Menüführung kollidieren; Alt+B ist dort üblicherweise zuständig für das Öffnen des Menüpunktes Bearbeiten).

Die folgende Abbildung zeigt eine Gruppe von drei Radio Buttons, die bei einem System mit einer Auftragsverwaltung Verwendung finden könnte.

Abb. 1.68: Radio Buttons in einer Auftragsverwaltung.



SQLWindows weist nach dem Start der Anwendung dem ersten der drei Radio Buttons den Inhalt "TRUE", den beiden restlichen den Inhalt "FALSE" zu. Auch deshalb setzt man üblicherweise den am häufigsten vorkommenden Fall an erste Stelle.

Sie können durch folgende Konstruktion erreichen, daß ein anderer als der erste Radio Button den Zustand "TRUE" erhält:

```
◆ Radio Button: rb2
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Set rb2 = TRUE
```

Die Zuweisung innerhalb der Sektion "Message Actions" des Radio Buttons rb2 setzt dessen eigenen Zustand auf "TRUE", dabei wird der erste Radio Button rb1 automatisch auf "FALSE" gesetzt.

Es ist wichtig zu wissen, woran SQLWindows erkennt, daß eine Gruppe von Radio Buttons zusammengehört. Stehen die Radio Buttons im Outline Fenster untereinander, dann gehören Sie zusammen, sie müssen dabei nicht unbedingt auf dem Bildschirm untereinander oder nebeneinander stehen, obwohl dies üblicherweise der Fall ist. Fügen Sie in Ihren Anwendungen die Radio Buttons nacheinander ein, so stehen Sie im Outline Fenster immer untereinander, fügen Sie aber zwischendurch ein anderes Objekt ein, dann müssen Sie die Radio Buttons im Outline Fenster untereinander verschieben (einmal mit der linken Maustaste auf die Raute klicken, Alt-Taste drücken, halten und Pfeiltaste hoch oder tief; um die Objektebene zu ändern, Pfeiltaste rechts oder links drücken).

Der folgende Programmcode zeigt drei Radio Buttons untereinander angeordnet.

```
◆ Radio Button: rb1
  ◇ Message Actions
◆ Radio Button: rb2
  ◇ Message Actions
◆ Radio Button: rb3
  ◇ Message Actions
```

Hier sehen Sie auch eine andere Möglichkeit, die Eigenschaften eines Radio Buttons über den Custromizer zu verändern. Sie müssen dazu lediglich auf der hervorgehobenen Fläche vor dem Wort "Radio" einmal mit der rechten Maustaste klicken.

Die folgende Übersicht zeigt die Möglichkeiten des Customizers, die sich dann bieten.

Done	Schließen des Optionsfensters
Object Name	Name des Objekts. Vorgeschlagen rb1, rb2 usw.; oft abzuändern in rbAuftrag, rbLieferschein.
Object Title	Titel des Radio Buttons.
Visible	Sichtbar: ja/nein.
Location and Size	Ort und Größe des Radio Buttons.
Background Color	Hintergrundfarbe.
Text Color	Textfarbe.
Font Name	Schriftart.
Font Size	Schriftgröße.
Font Enhancement	Erweiterung der Schriftart.
Edit Action	Wechsel zum Outline-Fenster.

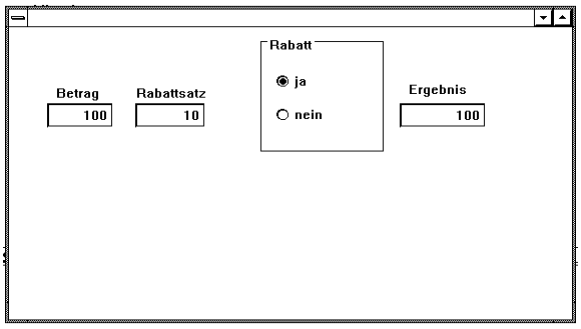
Um nun im Programm auf den Inhalt eines Radio Buttons zugreifen zu können, verwenden Sie üblicherweise eine If-Anweisung, die den Inhalt des Radio Buttons liefert. Da Radio Buttons stets logische Inhalte enthalten, haben Sie zwei Möglichkeiten, die If-Anweisung zu formulieren:

- "If rb1 = TRUE" oder
- "If rb1"

Beispiel

Das folgende Beispiel soll die Verwendung von Radio Buttons erläutern. Zwei Radio Buttons sollen dazu in ein Formular eingefügt werden, die beiden Radio Buttons sollen festhalten, ob bei einer Berechnung mit oder ohne Rabatt gerechnet werden muß. Um tatsächlich eine Berechnung durchführen zu können, brauchen Sie noch zwei Data Fields (df1, df2), beide vom Typ numerisch. Beide Data Fields bleiben zunächst leer. In das erste Data Field soll bei Programmablauf ein Betrag, in das zweite ein Rabattsatz eingetragen werden. Ein drittes Data Field (df3) soll schließlich den berechneten Wert anzeigen. Das Formular wird in der folgenden Abbildung angezeigt.

Abb. 1.69: Formular mit Data Fields und Radio Buttons für Berechnungen.



Die Rechenanweisung für das Beispiel lautet:

Set df3 = df1 - (df1*df2/100)

Die Berechnung soll immer dann stattfinden, wenn der Benutzer bestimmt, ob mit oder ohne Rabatt zu rechnen ist. Der folgende Programmcode zeigt die "Message Actions" der zwei Radio Buttons, das Ereignis "SAM_Click" und die verwendete Rechenanweisung.

```
◆ Radio Button: rb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◆ If rb1
        ◇ Set df3 = df1 -(df1*df2/100)
◆ Radio Button: rb2
  ◆ Message Actions
    ◆ On SAM_Click
      ◆ If rb2
        ◇ Set df3 = df1
```

Falls der Inhalt von rb1 "TRUE" ist (d.h. "Rabatt: ja" gewählt wurde), dann soll der Inhalt des Data Fields "df3" nach der angegebenen Rechenanweisung gesetzt werden. Falls dagegen "rb2" "TRUE" ist (kein Rabatt), dann wird der Inhalt von "df1" in das Ergebnisfeld "df3" kopiert.

Die Aufgabe ist damit gelöst, obwohl noch folgendes zu überlegen ist: Falls *nach* einer Berechnung der Rabattsatz oder der Betrag geändert wird, dann muß die neue Berechnung erzwungen werden, indem zunächst auf den anderen Radio Button und danach wieder auf den ursprünglichen Radio Button geklickt wird, was insgesamt ungeschickt wäre. Dieses Problem läßt sich dadurch lösen, daß die Berechnung nicht nur innerhalb der Message Actions der Radio Buttons, sondern zusätzlich in den Message Actions der Data Fields so definiert wird, daß jedesmal, wenn der Inhalt des Data Fields geändert wird erneut gerechnet wird. So könnte diese Sektion des Data Fields "df2" folgenden Programmcode enthalten:

```
◆ Data Field: df2
  ◆ Message Actions
    ◆ On SAM_AnyEdit
      ◆ If rb1
        ◇ Set df3 = df1 -(df1*df2/100)
      ◆ If rb2
        ◇ Set df3 = df1
```

Nun funktioniert die Sache reibungslos, hat aber einen Makel: Die Berechnung muß mehrfach formuliert werden (für die Data Fields df1 und df2 und für die Radio Buttons). Falls Änderungen vorgenommen werden, dann muß genau darauf geachtet werden, daß die Änderungen überall vorgenommen werden. Da dies in größeren Anwendungen kaum vertretbar ist, würde man einen anderen Weg wählen: Man würde eine Funktion erstellen, die genau diese Rechenanweisung enthält. Diese Funktion wird über ihren Namen immer dann aufgerufen, wenn die Berechnung stattfinden soll. So würden die "Message Actions" der Data Fields und der Radio Buttons nur noch den Aufruf der Funktion beinhalten:

Call BerechneRabatt

Das dazu nötige Vorgehen ist im Kapitel zu den Funktionen erläutert (vgl. auch das folgende Kapitel zu den Check Boxen).

Schließlich ist vorstellbar, daß, zum Speichern des Inhalts eines Radio Buttons, eine Variable vom Typ Zeichen angelegt wird. Abhängig von Einstellungen der Radio Buttons wird dieser Variablen dann ein Inhalt zugewiesen, der aber *nicht* vom Datentyp logisch sondern z.B. vom Datentyp Zeichen ist, denkbar wäre z.B. ein "R" für eine Rechnung, ein "L" für einen Lieferschein und ein "B" für Barzahlung, um das eingangs genannte Beispiel aufzugreifen. Diese Variable kann dann in einer SQL-Insert Anweisung verwendet werden, um eine Spalte in der Tabelle zu füllen (vgl. Kap. 2.2.2).

1.3.4.2.3.2 Check Box

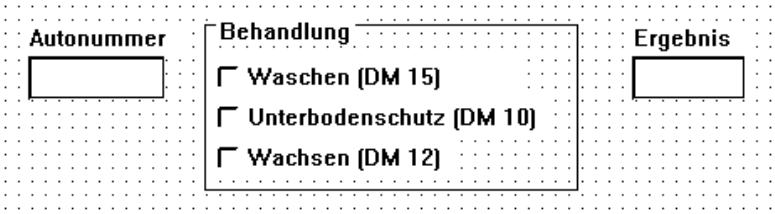
Check Boxes werden ähnlich wie Radio Buttons verwendet: Auch sie enthalten logische Werte, deren Inhalt "TRUE" (Check Box ist angeklickt) oder "FALSE" (Check Box ist nicht angeklickt) annehmen kann. Die aktuellen Einstellungen werden im Programmablauf häufig mit "If "erfragt (s. hierzu vorstehenden Abschnitt zu Radio Buttons).

Allerdings gibt es einen deutlichen Unterschied zwischen Check Boxes und Radio Buttons: Bei einer Gruppe von Check boxen ist es möglich, mehrere Check Boxes anzuwählen, d.h. sie auf den Inhalt "TRUE" zu setzen.

Das folgende Beispiel soll die Verwendung von Check boxen demonstrieren. Um ein anderes, eleganteres Vorgehen als im vorstehenden Abschnitt zu den Radio Buttons zu wählen, wird eine benutzerdefinierte Funktion für die Berechnung gewählt.

Fügen Sie zwei Data Fields, drei Check Boxes (die Group Box um die Check Boxes kann entfallen) in das Formular ein und ordnen Sie gemäß der folgenden Abbildung an.

Abb. 1.70: Check Boxes für die Autopflege.



(Das Feld mit der Autonummer hat keine weitere Bedeutung und dient nur der Illustrierung).

Abhängig von der gewählten Behandlung soll das Ergebnis berechnet werden. Dazu muß das Data Field rechts in den Datentyp Number umgewandelt werden, die Ausrichtung soll rechtsbündig sein (Customizer durch Klick mit der rechten Maustaste öffnen, Data Type und Justify bestimmen).

Jetzt wird die Funktion geschrieben, mittels der eine Berechnung stattfinden soll. Das Problem hierbei ist, alle möglichen Kombinationen der drei Check Boxes zu berücksichtigen. Neben der Möglichkeit daß alle drei Check boxen gewählt oder nicht gewählt sind, gibt es noch einige andere Kombinationen. Die folgende Übersicht zeigt die Möglichkeiten der Verwendung der drei Check Boxes, dabei bedeutet "TRUE", daß die Check Box gewählt (im Programmablauf durch ein Häkchen oder ein x markiert), "FALSE" dagegen, daß die Check Box nicht gewählt ist (im Programmablauf ist die Check Box dann leer).

Check Box 1	**Check Box 2**	**Check Box 3**
FALSE	FALSE	FALSE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	TRUE
TRUE	TRUE	FALSE
FALSE	TRUE	TRUE
TRUE	TRUE	TRUE

Alle logischen Zustände müssen nun in einer neuen benutzerdefinierten Funktion eingebaut werden.

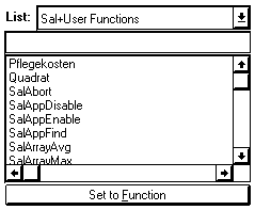
Diese Funktion wird in der Sektion "Global Declarations", Internal Functions" definiert. Die Funktion erhält den Namen "Pflegekosten". Es ist nicht nötig, Variable zu definieren, da die Funktion direkt die Inhalte der Data Fields manipuliert.

```
◆ Actions
  ◆ If NOT cb1 AND NOT cb2 AND NOT cb3
    ◇ Set df2 =  0
  ◆ If cb1
    ◇ Set df2 = 15
  ◆ If cb2
    ◇ Set df2 = 10
  ◆ If cb3
    ◇ Set df2 = 12
  ◆ If cb1 AND cb2 AND NOT cb3
    ◇ Set df2 = 15 + 10
  ◆ If cb1 AND cb3 AND NOT cb2
    ◇ Set df2 = 15 + 12
  ◆ If cb2 AND cb3 AND NOT cb1
    ◇ Set df2 = 10 + 12
  ◆ If cb1 AND cb2 AND cb3
    ◇ Set df2 =  15 + 10 + 12
```

Beachten Sie genau die Klein- und Großschreibung.

Öffnen Sie den Customizer der ersten Check Box, wählen Sie Edit Actions, wählen Sie aus den Outline Options den Eintrag "On SAM_Click" (doppelt klicken), der Eintrag erscheint jetzt eingerückt unter Message Actions. Jetzt wählen Sie "Call" im gleichen Fenster, auch dieser Eintrag wird eingerückt dargestellt. Beachten Sie, daß im anderen Fenster, in dem "Sal+User Functions" angezeigt werden jetzt auch die Funktion mit dem Namen "Pflegekosten" sichtbar ist, so wie das die folgend Abb. zeigt.

Abb. 1.71: Benutzerdefinierte Funktion Pflegekosten im Auswahlfenster.



Klicken Sie zweimal auf dem Eintrag "Pflegekosten", die Funktion wird im Outline Fenster hinter "Call" angeordnet (die Klammern bleiben leer, drücken Sie einmal Enter um die Eingabe zu beenden). Ihre Message Actions Sektion sollte jetzt so aussehen:

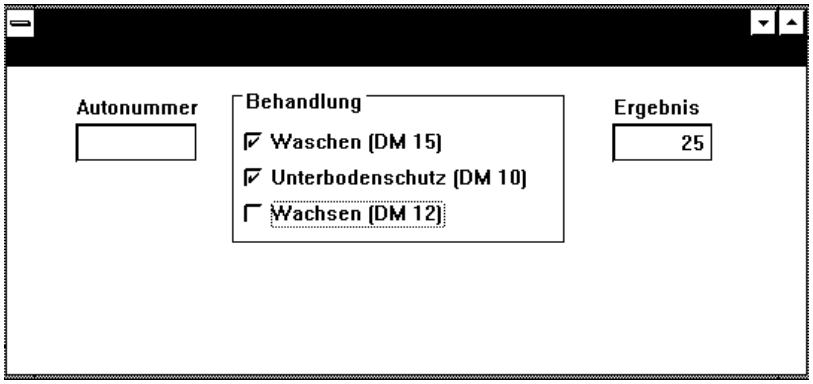
```
◆ Check Box: cb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call Pflegekosten( )
```

Die zwei Zeilen, beginnend mit "On SAM_Click", müssen auch unter den anderen Check boxen eingefügt werden. Sie können hierfür die Zwischenablage benutzen (auf die Raute vor "On SAM_Click" klicken, Strg+Einfg drücken, bei "cb2" und "cb3" jeweils die "Message Actions" öffnen, Shift+Einfg drücken). Ihre fertigen Anweisungen sollten so aussehen:

- Check Box: cb1
  - Message Actions
    - On SAM_Click
      - Call Pflegekosten( )
- Check Box: cb2
  - Message Actions
    - On SAM_Click
      - Call Pflegekosten( )
- Check Box: cb3
  - Message Actions
    - On SAM_Click
      - Call Pflegekosten( )

Damit wird die Funktion jeweils an drei verschiedenen Stellen aufgerufen. Starten Sie jetzt Ihre Anwendung, sie sollte folgendes Bild liefern:

Abb. 1.72: Check Boxes und davon ausgehende Berechnung.



Dadurch, daß eine Funktion für die Berechnung benutzt wird, sind Sie bei Änderungen des Programmcodes (z.B. wenn Sie weitere Pflegepunkte aufnehmen müssen) weitaus schneller fertig, da Sie die Funktion "Pflegekosten()" nur einmal ändern müssen. Gleichzeitig wird Ihre Anwendung weitaus sicherer.

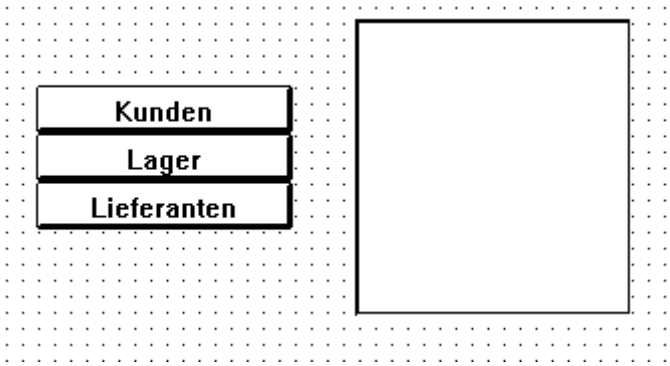
1.3.4.2.3.3 Option Button

Option Buttons werden genauso verwendet wie Radio Buttons. Deshalb werden die Details hierzu nicht mehr wiederholt (vgl. hierzu Kap. 1.3.4.2.3.1.

In Erweiterung zu den bisher genannten Beispielen sollen hier Option Buttons verwendet werden, um eine List Box mit verschiedenen Inhalten zu füllen. Auf diese Weise können auch auf engem Raum Daten aus verschiedenen Tabellen angezeigt werden.

Für das Beispiel benötigen Sie drei Option Buttons und eine List Box. Fügen Sie die Option Buttons nacheinander ein, damit diese von SQLWindows als zusammengehörend interpretiert werden. Danach fügen Sie die List Box ein. Ihr Formular sollte folgendes Aussehen haben.

Abb. 1.73: Option Buttons und List Box.



Ändern Sie die Beschriftung der Option Buttons so ab, daß Kunden, Lager bzw. Lieferanten angezeigt werden.

Nun soll formuliert werden, daß nach dem Klicken auf einem der Option Buttons Datensätze jeweils aus der Kunden-, Lager- bzw. Lieferantentabelle angezeigt werden.

Die dazu nötigen Code-Teile sind wie folgt einzugeben:

```
◆ Option Button: ob1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalListPopulate( lb1, hSql, 'select name from kunden')
◆ Option Button: ob2
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalListPopulate( lb1, hSql, 'select name from lager')
◆ Option Button: ob3
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalListPopulate( lb1, hSql, 'select name from lieferanten')
```

Achten Sie wiederum auf die Schreibweise. Sie können, um den Aufwand gering zu halten, wieder die Zwischenablage verwenden. Nach dem Start der Anwendung werden Sie feststellen, daß dies eine einfache Möglichkeit ist, auf engem Raum Datensätze aus verschiedenen Tabellen darzustellen.

Sie könnten, um eventuelle Pausen beim Lesen großer Datenmengen zu überbrücken, die Funktion SalWaitCursor() verwenden, um die "Sanduhr" von Windows anzeigen zu lassen. Der Funktionsaufruf sollte folgendermaßen durchgeführt werden:

```
◆ On SAM_Click
  ◇ Call SalWaitCursor( TRUE )
  ◇ Call SalListPopulate( lb1, hSql, 'select name from lager')
  ◇ Call SalWaitCursor( FALSE )
```

Schließlich soll hier noch darauf hingewiesen werden, daß Option Buttons mit Bildern versehen werden können. Dabei ist es möglich, für verschiedene Zustände des Option Buttons jeweils verschiedene Bitmap (*.BMP) - oder Icon (*.ICO) - Dateien anzugeben (gedrückt, nicht gedrückt, nicht wählbar).

1.3.4.2.4 Objekte für Benutzer-Aktionen

Der Grund, weshalb die folgenden Objekte nicht zusammen mit Radio Boxes und Option Buttons beschrieben werden liegt darin, daß mit den hier vorgestellten Objekten keine logischen Zustände erzeugt und verarbeitet werden, sondern Aktionen.

So ist beispielsweise der Push Button das wahrscheinlich am häufigsten verwendete Objekt, das für durch den Benutzer ausgelöste Aktionen Verwendung findet (häufig um ein Fenster zu öffnen, einen Datensatz neu einzugeben oder einen Report zu drucken).

Die ebenfalls hier besprochenen Scroll Bars werden dagegen nicht so häufig verwendet. Der Grund liegt wohl darin, daß dieser Objekttyp z.B. in einer List Box oder einem Table Window automatisch angezeigt wird, wenn er benötigt wird.

1.3.4.2.4.1 Push Button

Ein Push Button ist eine Schaltfläche, die es ermöglicht, durch einen einfachen Mausklick eine dem Push Button zugeordnete Aktion auslösen zu lassen. Diese Aktion, die vom Entwickler zugeordnet werden muß, kann das Öffnen eines Fensters, das Speichern von Datensätzen, die Druckausgabe oder ähnliches sein.

Üblicherweise wird durch den Einbau des Ausdrucks

On SAM_Click

in der Message Actions Sektion des Push Buttons eine solche Aktion ausgelöst.

Im folgenden werden zwei Aktionen erläutert, die durch das Drücken eines Push Buttons ausgelöst werden:

- Das Öffnen eines Form Windows sowie
- eine arithmetische Berechnung mit Daten aus einer Tabelle.

Das Speichern von Datensätzen und die Ausgabe eines Reports nach Knopfdruck wird in Teil 2 ausführlich erläutert.

Öffnen eines Form Windows

Das Öffnen eines Fensters nach Klicken auf dem Push Button wird unter Verwendung der Funktion SalCreateWindow() realisiert.

Fügen Sie zunächst ein neues Form Window unter Verwendung der Tool-Palette ein (zunächst F4 drücken, dann auf die Schaltfläche "Form" klicken). Das eingefügte Fenster hat z.B. den Namen "frm2" (rechte Maustaste auf dem Form Window klicken, Object Name auswählen). Schalten Sie die Option "Automatically Create" ab, das heißt, das Fenster soll nicht automatisch bei Start des Programms erzeugt werden (Customizer mit Klicken der rechten Maustaste auf dem Form Window anzeigen lassen), schließen Sie danach das Fenster (auf "Done" klicken).

Jetzt fügen Sie einen neuen Push Button in das Fenster "frmMain" ein: Klicken Sie in der Tool Palette auf das Symbol für den Push Button.

Abb. 1.74: Symbol für Push Button in der Tool Palette.

Als Beschriftung des Push Buttons geben Sie "öffnen" an.

Klicken Sie dann mit der rechten Maustaste auf den neu eingefügten Push Button (sein Name lautet hier "pb2") und wählen Sie die Option "Edit Actions". Lassen Sie mit der F2-Taste die Outline-Options anzeigen, wählen Sie durch einen Doppleklick die Zeile "On SAM_Click" aus, danach "Call" im gleichen Fenster. Jetzt können Sie im Fenster der Outline Options den Eintrag "SalCreateWindow" suchen (Rollbalken benutzen). Klicken Sie zweimal darauf, um den Eintrag zu übernehmen, und ergänzen Sie wie folgt:

```
Pushbutton: pb2
    Message Actions
        On SAM_Click
          Call SalCreateWindow( frm2, frmMain )
```

"frm2" ist der Name des zu öffnenden Fensters, der Eigentümer soll "frmMain" sein.

Lassen Sie das Programm starten (F7-Taste) und testen Sie den Push Button. Sie sehen, daß er funktioniert, allerdings ist mehrfaches Öffnen des Formulars möglich, das muß natürlich unterbunden werden.

Um das **mehrfache Öffnen eines Form Windows zu verhindern**, bestehen zwei Möglichkeiten:

- In der Sektion "Design Time Settings" zu Beginn einer Anwendung kann die Option "Reject Multiple Window Instances" auf "Yes" gesetzt werden; damit verhindert SQLWindows selbst das mehrfache Erzeugen von Objekten.

- Sie selbst deaktivieren den Push Button, nachdem er betätigt wurde und verhindern damit das erneute Anzeigen des Formulars.

Die erste Möglichkeit ist verlockend, da durch diese Einstellung das mehrfache Erzeugen eines Objekts wirkungsvoll verhindert wird. Allerdings meldet Ihre Anwendung beim Versuch des erneuten Aktivierens, z.B. eines Form Windows, einen Fehler, was im Programmablauf nicht zu gebrauchen ist.

Die zweite Möglichkeit, nach der Sie selbst die entsprechenden Anweisungen vornehmen, ist etwas aufwendiger, aber insgesamt besser, da der deaktivierte Push Button nach einem ersten Klick mit grauer Beschriftung dargestellt wird und nicht mehr benutzbar ist.

Hierzu wird der vorstehende Programmtext durch die in Fettdruck hervorgehobene Zeile ergänzt:

```
Pushbutton: pb2
  Message Actions
        On SAM_Click
              Call SalCreateWindow( frm2, frmMain )
              Call SalDisableWindow( hWndItem )
```

Dies bedeutet, daß das gerade aktivierte Objekt (hier ist das der Push Button) deaktiviert (disable) wird, nachdem das Fenster geöffnet wurde (alternativ könnte in der Klammer der Funktion "SalDisableWindow" auch "pb2" stehen).

Jetzt haben Sie allerdings ein neues Problem, da nach Schließen des neu geöffneten Fensters der Push Button weiterhin deaktiviert bleibt. Um ihn zu reaktivieren, müssen Sie entscheiden, wann und wo dieses Reaktivieren stattfinden soll. Sinnvollerweise sollte das Reaktivieren des Push Buttons dann stattfinden, wenn das Fenster wieder geschlossen wird, also der gleiche Zustand wie vor dem ersten Drücken des Push Buttons wieder eintritt. Diese Aktion wird in der Message Actions Sektion des Form Windows definiert.

Geben Sie in der Message Actions Sektion des Fensters frm2 folgendes ein (mit der rechten Maustaste auf das Fenster "frm2" klicken, "Edit Actions" auswählen, Outline Options mit F2 anzeigen lassen, zuerst die Anweisung "On SAM_Destroy", danach "Call" auswählen, den Rest tippen). Die Anweisung sollte folgendermaßen aussehen.:

```
Form Window: frm2
  Message Actions
        On SAM_Destroy
              Call SalEnableWindow( frmMain.pb2 )
```

"SAM_Destroy" ist die Nachricht, die beim "Zerstören" des Fensters gesendet wird.
"SalEnableWindow()" ist eine Funktion, die bei einem Objekt (hier dem Push Button) die Aktivierung herbeiführt. In Klammern angegeben sind der Name des Objekts ("pb2") und der Ort, an dem dieses Objekt zu finden ist ("frmMain"). Verbunden sind diese zwei Angaben mit einem Punkt, das ist die korrekte Adressierung oder vollständige Referenz des Objekts in SQLWindows.

Arithmetische Berechnung mit Daten aus einer Tabelle

Fügen Sie einen Push Button in das Fenster frmMain ein. Überschreiben Sie den Titel (untitled) mit dem Wort "rechne" (ohne Anführungszeichen), schließen Sie mit Drücken der Enter-Taste ab. Zur Kontrolle

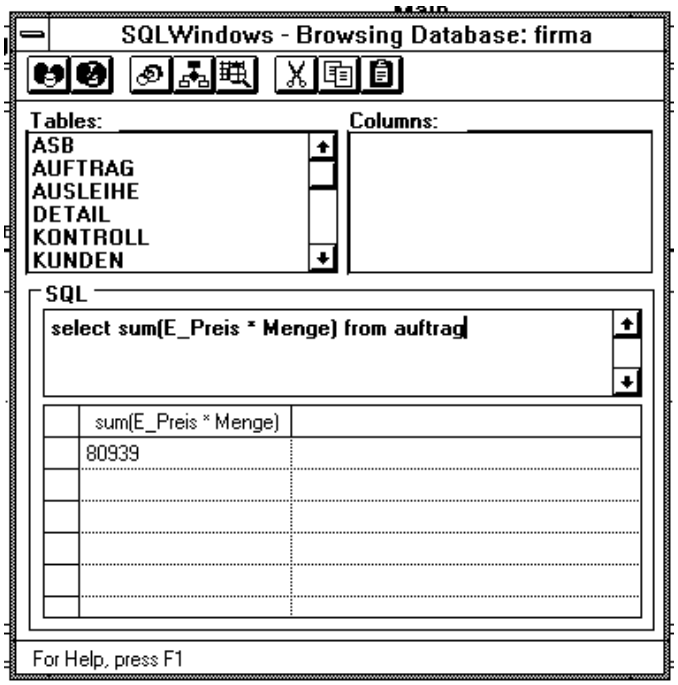
klicken Sie mit der rechten Maustaste, um das Push Button Fenster zu öffnen und um den Namen des Push-Buttons zu erfahren (er sollte "pb1" lauten).

Um eine Berechnung anzustellen, soll in dem folgenden Beispiel auf die Tabelle "Auftrag" der geöffneten Datenbank FIRMA zugegriffen werden (kontrollieren Sie, ob die Datenbank zu Beginn der Anwendung geöffnet wird). Nach dem Drücken des Push Buttons soll die Gesamtsumme aller verkauften Waren errechnet werden. Die dazu verwendete Rechenanweisung lautet: "E_Preis* Menge" (zwei Spalten aus der Tabelle Auftrag). Da diese Berechnung mit allen Datensätzen vorgenommen werden soll, muß die SQL-Anweisung folgendermaßen formuliert werden:

select sum(E_Preis * Menge) from auftrag

Soweit die Formel in SQL, die Sie über den Menüpunkt "View", "Database Browser" testen sollten. Die folgende Abbildung zeigt diese praktische Möglichkeit, auf die Daten zuzugreifen.

Abb. 1.75: Database Browser mit Sql-Anweisung (ausführen lassen mit Alt+E).



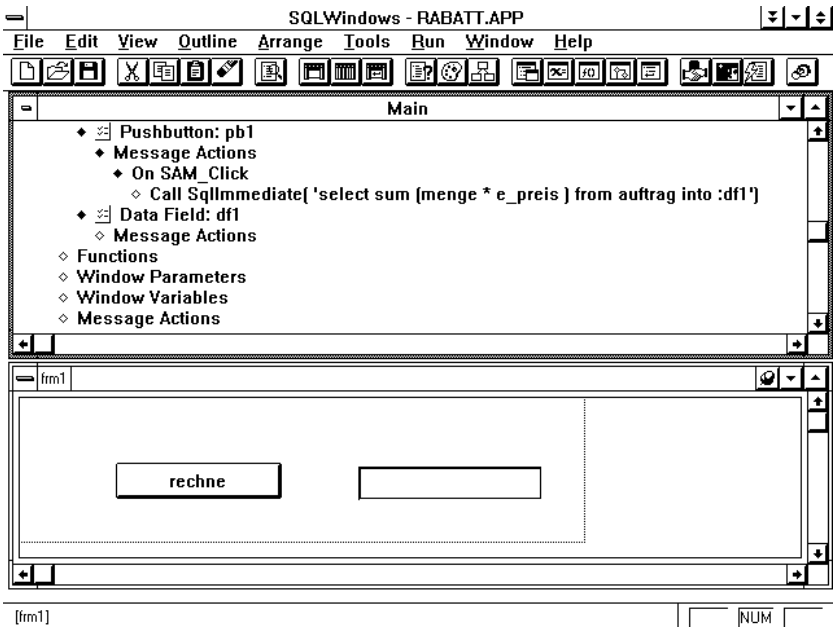
Als nächstes muß definiert werden, *wo* das Ergebnis dieser Berechnung anzuzeigen ist. Dazu wird hier ein Data Field benutzt. Fügen Sie ein solches Data Field mittels der Tool-Palette ein, kontrollieren Sie den Namen (er sollte df1 lauten).

Nun klicken Sie mit der rechten Maustaste auf den Push Button, um die Option "Edit Actions" auszuwählen. Aktivieren Sie mit F2 die Outline-Options und wählen Sie mit einem Doppelklick die Zeile "On SAM_Click" aus. Sie wird in das Outline Fenster eingefügt. Danach klicken Sie zweimal auf der Anweisung "Call", die ebenfalls (eingerückt) eingefügt wird. Ihre Schreibmarke befindet sich jetzt hinter dem Wort "Call", Sie können nun den Funktionsaufruf eingeben. In unserem Fall soll die Funktion "SqlImmediate" benutzt werden, um sofort (immediate) eine SQL-Operation durchzuführen. Geben Sie das folgende ein, achten Sie bei dem Namen der Funktion "SqlImmediate" auf Groß- und Kleinschreibung und beachten Sie die einfachen Anführungszeichen an Anfang und Ende der Sql-Anweisung:

Call SqlImmediate('select sum(e_preis * menge) from auftrag into :df1')

Bestätigen Sie die Eingabe mit Enter und starten Sie die Anwendung. Das Ergebnis könnte ungefähr so aussehen:

Abb. 1.76: Push Button und Data Field, Berechnung unter Verwendung der Funktion SqlImmediate().



Häufig wird die Berechnung etwas komfortabler zu gestalten sein, etwa indem mittels weiterer Data Fields eine Auftragsnummer oder eine Kundennummer erfragt wird. Die SQL-Anweisung würde dann um die "where - Klausel" erweitert. Bei Verwendung von einem weiteren Data Field für die Auftragsnummer ("df2") könnte die Anweisung dann so aussehen:

Call SqlImmediate('select sum(menge*e_preis) from auftrag where aufnr = :df2 into :df1')

Sie sehen, daß das nicht besonders schwierig ist. Da die vorstehende SQL-Anweisung beliebig komplex sein kann, können Sie auch komplizierte Berechnungen mit einem Minimum an Aufwand vornehmen lassen.

Die folgende Übersicht zeigt den Inhalt und die kommentierten Optionen des Customizers (mit der rechten Maustaste auf den Push Button klicken)

Option	Erklärung
Done	Schließen des Optionsfensters.
Object Name	Name des Push-Buttons. Vorgeschlagen wird immer pb1, pb2 usw. Oft werden diese Namen dann abgeändert in pbEnde, pbSichern usw.
Object Title	Titel des Push-Buttons, z.B. "Ende" oder "sichern". Das unterstrichene Zeichen für die Kurzwahl des Push-Buttons erhalten Sie indem das Zeichen "&" voranstellen, also z.B. "&Ok", "Dru&ck".
Visible	Sichtbar: ja/nein.
Location and Size	Ort und Größe des Push-Buttons.
Picture Contents	Angabe eines Bildes als Bestandteil des Push-Buttons. Möglich sind Bitmaps (*.BMP) oder Icons (*.ICO). Sie können dazu die von Windows bekannten Bilder benutzen oder die im Verzeichnis GUPTA liegenden Dateien verwenden.
Picture Transparent Color	Die einem Bild zugeordnete transparente (durchsichtige) Farbe.
Keyboard Accelerator	Funktionstasten, Enter- oder Escape-Taste, die dem Push Button zugeordnet werden können, damit auch über diese die Push Button Aktion ausführbar ist.
Background Color	Hintergrundfarbe des Push Buttons.
Text Color	Textfarbe des Push-Buttons.
Font Name	Name der verwendeten Schriftart.
Font Size	Größe der verwendeten Schriftart.
Font Enhancement	Erweiterungen der Schriftart.
Edit Actions	Wechsel zu Outline-Fenster.

1.3.4.2.4.2 Scroll Bars

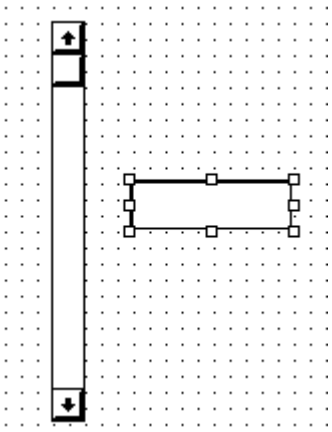
Scroll Bars können verwendet werden, um Eingaben durch den Benutzer zu verarbeiten oder um eine Ausgabe an den Benutzer anzuzeigen. Üblicherweise werden sie für Eingaben benutzt, weshalb das unten folgende Beispiel auch darauf Bezug nimmt. Dabei wird nicht zwischen horizontalen oder vertikalen Scroll Bars unterschieden, da beide über die gleiche Funktionalität verfügen.

Um eine Scroll Bar einzufügen, klicken Sie zunächst auf das dafür vorgesehene Symbol in der Tool Palette.

Abb. 1.77: Scroll Bar Symbol in der Tool Palette.

Fügen Sie dann das Objekt in das Form Window ein und stellen Sie ein Data Field neben die Scroll Bar; das Form Window sollte der folgenden Abbildung in etwa entsprechen.

Abb. 1.78: Scroll Bar (und Data Field) in einem Form Window.



Die Scroll Bar kann mit dem angezeigten Knopf oder durch Klicken zwischen Knopf und oberem bzw. unterem Rand bewegt werden. Dabei entspricht die Position am oberen Rand dem Wert 0, die Position am unteren Rand dem Wert 100. Das Data Field soll dazu benutzt werden, den Wert der aktuellen Position anzuzeigen. Hierfür wird die Nachricht

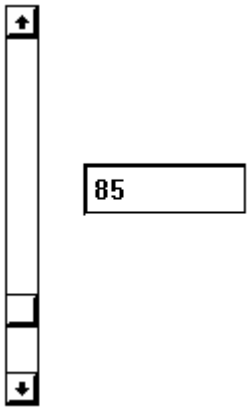
SAM_ScrollBar

ausgewertet. Diese Nachricht wird dann geschickt, wenn Sie den Knopf der Scroll Bar bewegen. Geben Sie in der Message Actions Sektion der Scroll Bar folgendes ein:

```
◆ Vertical Scroll Bar: vsb1
  ◆ Message Actions
    ◆ On SAM_ScrollBar
      ◇ Set df1 = vsb1
```

Starten Sie jetzt die Anwendung und bewegen Sie den Knopf. Die folgende Abbildung zeigt die Scroll Bar mit dem an das Data Field übergebenen Wert .

Abb. 1.79: Scroll Bar mit Data Field.



Die Schrittweiten, die bei Klicken z.B. auf den mit Pfeil versehenen Flächen (hier Schrittweite 1) benutzt wird, kann verändert werden. Hierzu wird die Funktion "SalScrollSetRange()" verwendet. Nähere Informationen hierzu erhalten Sie vom Hilfesystem.

Durch Verwendung vordefinierter Variablen aus SQLWindows, können Sie z.B. feststellen, wann der untere oder obere Rand der Scroll Bar erreicht wurde. Hierfür müssen Sie die Variable "wParam" auswerten. Sie enthält zwar einen numerischen Inhalt, doch in Verbindung mit ebenso vordefinierten Konstanten (die alle mit "SB_" für Scroll Bar beginnen) ist das sehr einfach. Der folgende Code-Teile zeigt die Prüfung, ob der untere Rand erreicht wurde und sendet, falls dies geschehen ist, eine Textfolge an die Statuszeile (dafür wird die Funktion "SalStatusSetText()" benutzt). Der eben formulierte Programmcode muß dazu um folgende Zeilen ergänzt werden:

```
◆ Vertical Scroll Bar: vsb1
  ◆ Message Actions
    ◆ On SAM_ScrollBar
      ◇ Set df1 = vsb1
      ◆ If wParam = SB_Bottom
        ◇ Call SalStatusSetText( frmMain, 'Sie sind am Ende' )
```

Anmerkung: Sie müssen, um die Statuszeile anzeigen zu lassen, eventuell über den Customizer die Eigenschaft des Form Windows "Accesories Enabled" auf "Yes" stellen. Testen Sie die Anwendung!

1.3.4.3 Menu-Objekte

Unter Menu-Objekten werde solche Objekte verstanden, die dazu dienen, ein Menüsystem aufzubauen. Diese Menüsysteme können dabei zur Steuerung einer Anwendung innerhalb von Form Windows, Table Windows und MDI-Windows eingebaut werden.

SQLWindows stellt zwei grundsätzliche Möglichkeiten zur Verfügung, wie ein Menü erstellt werden kann:

- Das Menü kann über die Inhalte der Options Bar zusammengestellt oder
- das Menü kann über einen speziellen Menügenerator angefertigt werden.

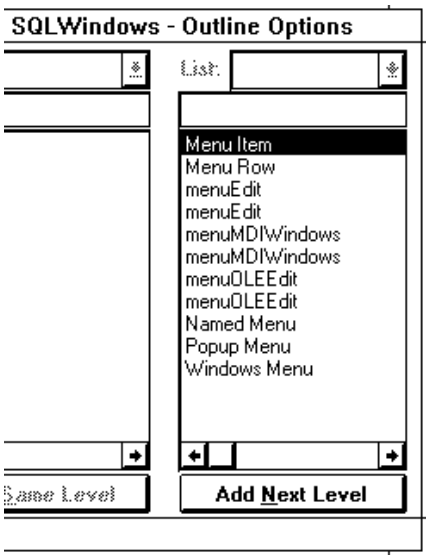
Beide Verfahren werden im folgenden erläutert.

Zunächst werden allerdings noch einige Begriffe erklärt, die in SQLWindows benutzt werden, wenn mit Menüführungen gearbeitet wird. In den folgenden Kapiteln wird dann gezeigt wie Menus erstellt werden und zu benutzen sind.

Begriffe aus SQLWindows für die Arbeit mit Menus

In der folgenden Abbildung finden sich die in diesem Zusammenhang wichtigen Begriffe, wie sie in den Outline Options zu sehen sind:

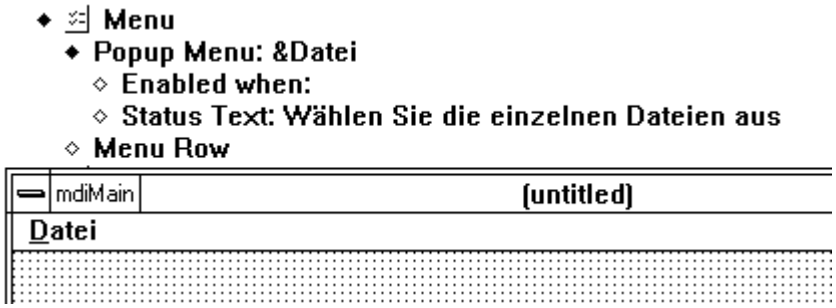
Abb. 1.80: Inhalt der Outline Bar.



Die angezeigten Begriffe haben folgende Bedeutung:

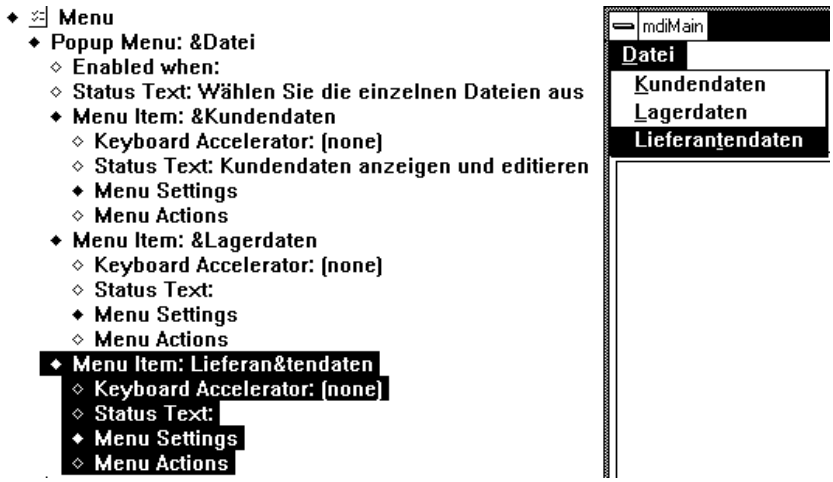
- **Popup Menu**: Ein Menüpunkt, der in der Menüleiste sichtbar ist (z.B. "Datei"), der andere Menü-Objekte (Menu Rows, Menu Separators und Menu Columns) enthält.

Abb. 1.81: Popup Menu "Datei" mit Status Text.



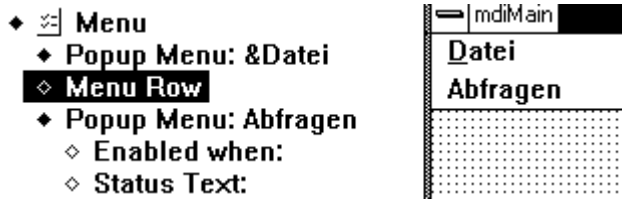
- **Menu Item**: Ein Menüpunkt, der einzeln oder als Inhalt eines Popup Menüs auftreten kann.

Abb. 1.82: Popup Menu (Datei) mit untergeordneten Menu Items.



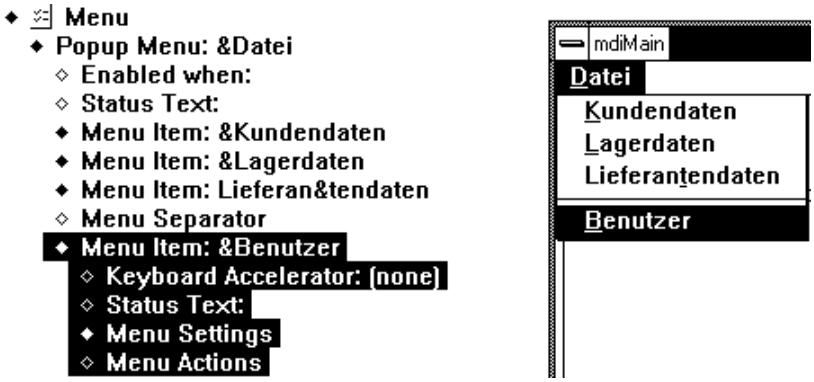
- **Menu Row**: Eine Reihe von Menüpunkten, die horizontal angeordnet sind. Durch das Einfügen einer Menu row (d.h. dieses Schlüsselwortes, das nicht angezeigt wird), kann eine zusätzliche Zeile von Menüpunkten erzeugt werden.

Abb. 1.83: Menu Row zwischen zwei Popup Menus:



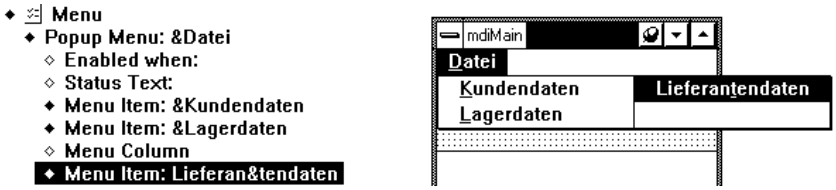
- **Menu Separator**: Eine waagerechte Abgrenzungslinie, die inhaltlich zusammengehörende Menüpunkte eines Popup Menus voneinander abgrenzt.

Abb. 1.84: Menu Separator (vor "&Benutzer").



- **Menu columns**: Gruppiert Menü-Einträge in vertikale Gruppen.

Abb. 1.85: Menu Column (vor Menu Item "Lieferantendaten")



Vorgefertigte Popup Menüs von SQLWindows

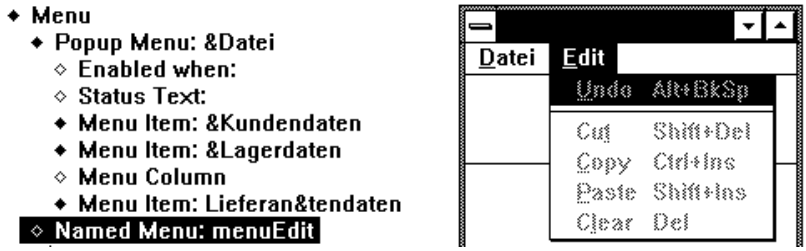
Einige fertige Menüs werden bereits von SQLWindows bereitgestellt, die es erlauben, einen Menüpunkt einzubinden, ohne diesen selbst zu erstellen (sie werden auch als "named Menus" bezeichnet, da sie unter einem Namen gespeichert sind). Hierzu zählt vor allem der Menüpunkt "Edit", in deutschen Anwendungen auch "Bearbeiten" genannt, der die Benutzung der Zwischenablage sowie der DDE/OLE-Mechanismen

ermöglicht, wobei zu beachten ist, daß die hinter den Menüpunkten stehenden Aktionen bereits vorprogrammiert sind. Um z.B. Datensätze aus einer Tabelle in die Zwischenablage zu kopieren, müssen Sie diese Datensätze nur markieren und die Copy-Option des Menüpunktes "Edit" (bzw. Bearbeiten) benutzen.

Die folgenden fertigen Popup Menüs können Sie in Ihre Anwendung einbinden. Sie sollten aber beachten, daß die dann angezeigten Menu Items noch ins Deutsche übersetzt werden müssen.

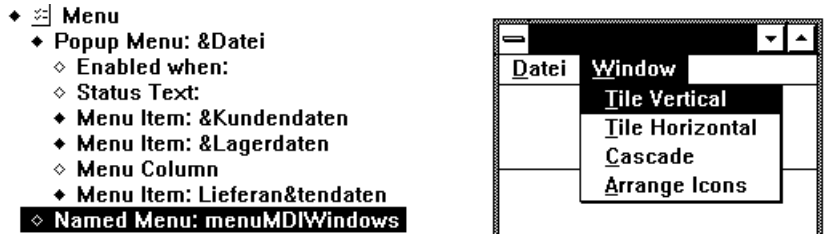
- **Menu Edit**: Ein einfaches Menü, das die üblichen Windows Werkzeuge (Undo, Cut, Copy, Clear) enthält.

Abb. 1.86: Menu Edit neben einem selbst erstellten Menüpunkt.



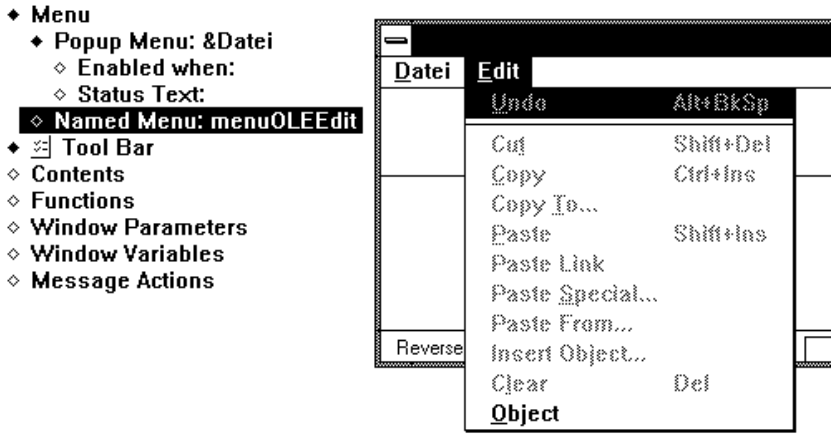
- **Menu MDIWindows**: Ein Menüpunkt, der dazu dient, die innerhalb eines MDI-Windows angezeigten Windows anzuordnen.

Abb. 1.87: Menu MDIWindows.



- **MenuOLEEdit**: Hierbei handelt es sich um ein ausführliches Menü das die für das Verwenden der Zwischenablage und der DDE/OLE-Mechanismen nötigen Menüpunkte bereithält.

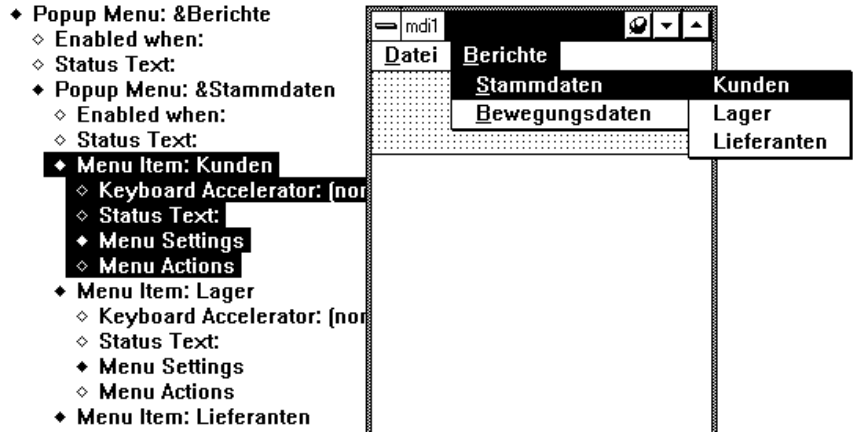
Abb. 1.88: Named Menu: menuOLEEdit.



- **Named Menus**: Neben den vorstehend genannten Named Menus, die für Sie bereits vordefiniert sind, können Sie auch selbst Menüsysteme erstellen und unter einem eigenen Namen speichern. Hiermit kann die wiederholte Definition von immer wieder zu verwendenden Menüpunkten vereinfacht werden. Die "Named Menus" müssen in der Sektion Global Declarations unter Verwendung der bereits genannten Objekte erstellt werden und können dann genauso wie die von SQLWindows zur Verfügung gestellten Menüs verwendet werden (das Verfahren hierzu ist in den mitgelieferten Handbüchern ausführlich dargestellt und wird hier nicht weiter erläutert).

- **Cascading Menu**: Unter diesem Menütyp versteht man ein Menü, das an einen Eintrag in einem anderen Menü gebunden ist. Wählt der Benutzer im ersten Menü den entsprechenden Eintrag, dann öffnet sich ein zweites Menü versetzt davon und zeigt weitere Wahlmöglichkeiten an. Genaugenommen handelt es sich um zwei Popup Menüs, die eingerückt werden müssen, wie das die folgende Abbildung zeigt.

Abb. 1.89: Cascading Menu.



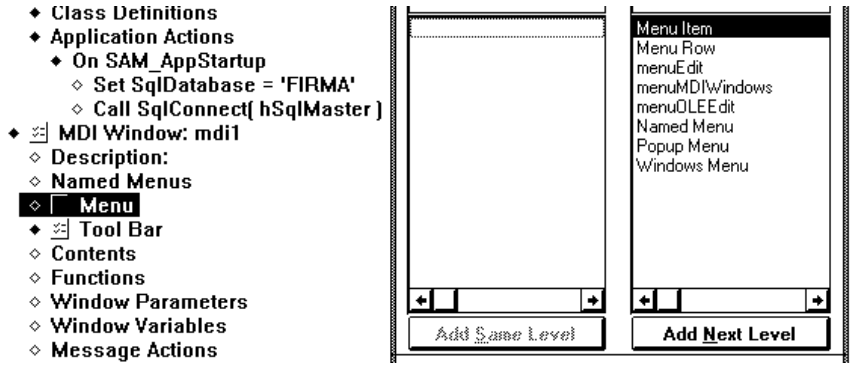
Zu beachten ist, daß die innerste Ebene aus "Menu Items" besteht (z.B. Menu Item: Kunden), daß aber davor zwei verschiedene Popup Menus angeordnet sind (Stammdaten bzw. Berichte).

1.3.4.3.1 Menu unter Verwendung der Options Bar selbst erstellen

Das Erstellen eines Menüs wird hier unter Verwendung der Options Bar demonstriert. Um das Beispiel nachzuvollziehen, benötigen Sie ein Table Window, ein Form Window oder ein MDI-Window (hier wird ein MDI-Window benutzt).

Lassen Sie sich den Inhalt des Outline-Fensters anzeigen, klicken Sie einmal auf dem Eintrag "Menu" des MDI-Fensters und drücken Sie die F2-Taste, um sich das Fenster mit den Outline Options anzeigen zu lassen.

Abb. 1.90: Outline-Fenster mit "Menu"-Eintrag, Fenster mit Outline-Options.



Es soll jetzt ein Menüpunkt mit dem Namen "Datei" eingefügt werden. Klicken Sie hierzu zweimal (mit der linken Maustaste) auf dem Eintrag "Popup Menu" (im Fenster mit den Optionen für "Add next level"). Der Eintrag wird im Outline-Fenster eingefügt, anschließend blinkt die Schreibmarke. Geben Sie die Zeichenfolge "&Datei" ein (das "&"-Zeichen sorgt dafür, daß das nächste folgende Zeichen unterstrichen dargestellt wird). Drücken Sie dann die Enter-Taste, der Eintrag "Datei" ist damit erstellt.

Nun sollen unter dem eingegebenen Menüpunkt die Worte "Kundendaten" und "Lagerdaten" angeordnet werden. Hierzu klicken Sie zweimal (linke Maustaste) auf den Eintrag "Menu Item" (wiederum im Fenster mit den Optionen für "Add next level"; beachten Sie, daß der jetzt eingefügte Text eingerückt unter dem Text für das zuvor eingefügte Popup Menü stehen muß. Geben Sie rechts von dem angezeigten "Menu Item: " die Zeichenfolge "&Kundendaten" ein, drücken Sie danach die Enter-Taste. Klicken Sie (zweimal) in dem Fenster mit den "Add same level" - Optionen auf den Eintrag "Menu Item", um auf gleicher Ebene wie zuvor jetzt den Eintrag "&Lagerdaten" einzufügen, drücken Sie auch danach die Enter-Taste.

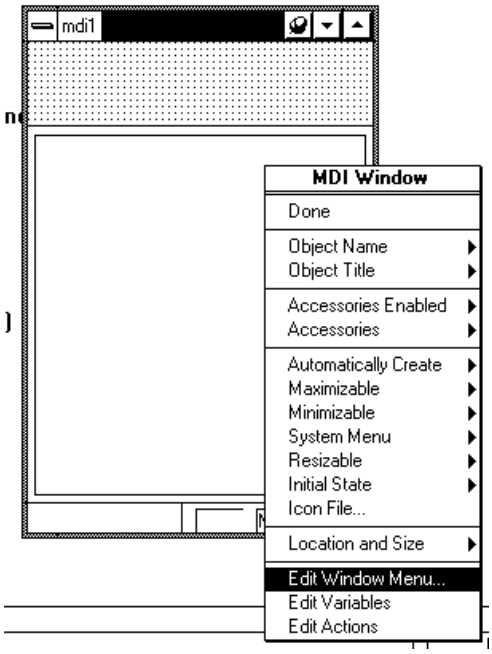
Kontrollieren Sie die Eintragungen, indem Sie sich das Formular anzeigen lassen (ggf. mit Strg+1) und auf den Menüpunkt "Datei" klicken.

Die anderen Objekte, die zum Erstellen eines Menüs Verwendung finden, lassen sich genauso einfügen. Sie sollten lediglich darauf achten, daß die verschiedenen Ebenen innerhalb eines Menüsystems eingehalten werden. Nachträglich können Sie mit Alt-Pfeiltasten links, rechts, hoch, tief die Eintragungen verschieben. Sie können die Eintragungen auch über die Zwischenablage verschieben.

1.3.4.3.2 Menü unter Verwendung des Menügenerators erstellen lassen

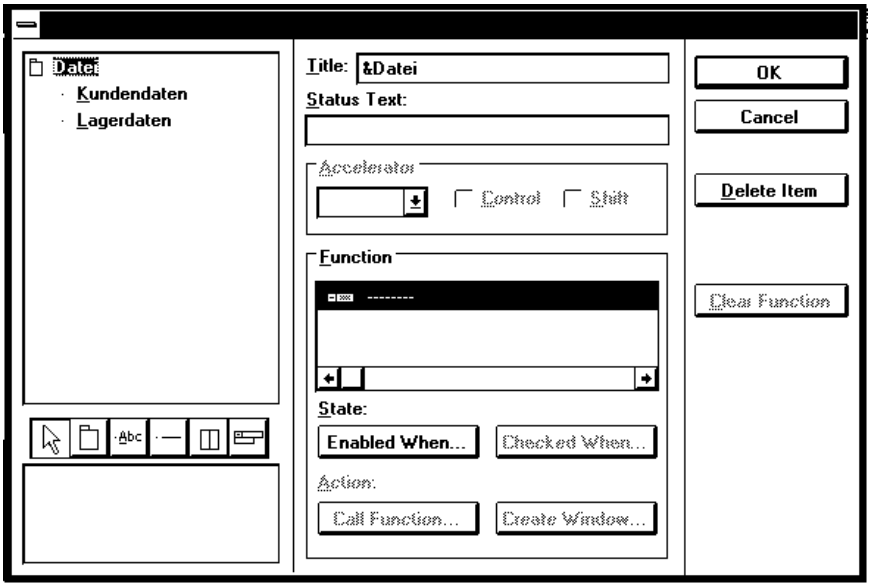
In dem angezeigten Window muß der Customizer geöffnet werden, indem mit der rechten Maustaste auf dem Hintergrund des Fensters geklickt wird.

Abb. 1.91: Customizer mit der Option "Edit Window Menu".



Aus dem Customizer muß die Option "Edit Window Menu" gewählt werden, SQLWindows aktiviert dann (ab der Version 5.0) automatisch den Menügenerator.

Abb. 1.92: Menügenerator von SQLWindows.



Angezeigt ist hier das "von Hand" eingefügte Popup Menü "Datei" mit den zwei eingerückten Menüoptionen "Kundendaten" und "Lagerdaten".

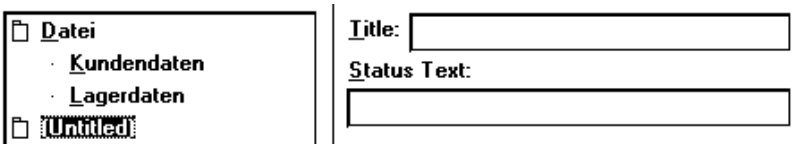
Links unten sind die diversen Möglichkeiten angezeigt, wie Sie ein Menü anlegen können. Im folgenden soll ein weiterer Menüpunkt "Berichte" eingefügt werden.

Klicken Sie zunächst den zweiten Option Button von links an, um ein Popup Menü einzufügen.

Abb. 1.93: Option Button für die Eingabe eines Popup Menüs.

Bewegen Sie dann den Mauszeiger nach oben, an den linken Rand (unter das Kästchen von "Datei") und klicken Sie einmal (links), auf diese Weise wird ein neues Popup Menü eingefügt. Falls das neu eingefügte Menü unter der eingerückt stehenden Zeile "Lagerdaten" angeordnet wird, dann löschen Sie es mit Hilfe der "Delete-Item"-Taste und wiederholen Sie das Einfügen.

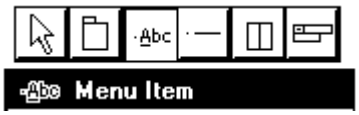
Abb. 1.94: Eingefügtes Popup Menü.



Jetzt tragen Sie neben "Title" den Titel des Popup Menüs ein ("&Berichte"). Der Eintrag für den Statustext, der angezeigt wird, wenn der Menüpunkt gewählt wird, kann entfallen. Drücken Sie aber *nicht* die Enter-Taste um die Eingabe zu bestätigen, da sonst das Fenster geschlossen wird (ggf. Fenster erneut über den Customizer öffnen).

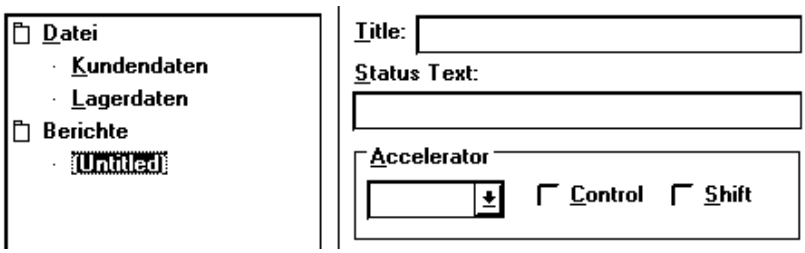
Jetzt soll (eingerückt) unter den Menüpunkt "Berichte" das Wort "Kundendaten" eingetragen werden. Klicken Sie zunächst auf den dritten Option Button von links, um die "Menu Items" einzufügen.

Abb. 1.95: Option Button für Menu Items.



Klicken Sie jetzt unter dem Namen des Popup Menüs "Berichte", aber nicht am linken Rand, sondern eingerückt dort, wo das zu tippende Wort "Kundenliste" später erscheinen soll.

Abb. 1.96: Neuer Menüpunkt unter "Berichte".

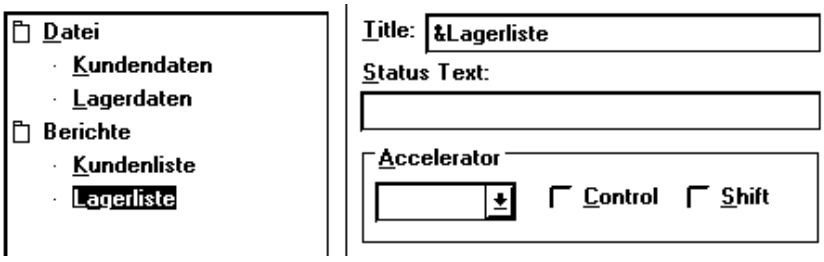


Geben Sie jetzt bei "Title" den Begriff "&Kundenliste" ein (drücken Sie *nicht* die Enter-Taste) und wiederholen Sie den Vorgang für den Eintrag "Lagerliste":

- Klicken auf den dritten Option Button von links;
- Klicken unter dem eingefügten Wort "Kundenliste";
- Eintrag des Wortes "Lagerliste".

Das fertige Popup Menü hat folgendes Aussehen:

Abb. 1.97: Popup Menu "Berichte".



Beachten Sie auch, daß Sie hier über die "Accelerator"-Combo Box einen Kurzschlüssel für die Menüoption vereinbaren können (z.B. Reservieren der F8-Taste für das Anzeigen und der Shift+F8-Taste für das Drucken der Kundendaten).

Wie Sie sehen, können Sie auch die beiden Verfahren (manuelle Erstellung und Benutzung des Menü-Generators) mischen.

1.3.4.3.3 Menü benutzen

Üblicherweise benutzen Sie Menüpunkte nicht direkt. Das heißt, daß es schon möglich ist, auf die Wahl eines Menüpunktes eine Aktion folgen zu lassen (z.B. eine Druckausgabe), doch viel häufiger öffnen Sie z.B. ein Formular, um in diesem Formular z.B. Kundendaten anzeigen zu lassen. Dies soll in dem folgenden kurzen Beispiel demonstriert werden. Das Beispiel wird so angelegt sein, daß die notwendigen Eingaben in dem Outline-Fenster gemacht werden, doch ist es genauso möglich, über den Push Button "Call Function" des Menu Generators das gleiche zu tun.

Nach Klicken auf der Menüoption "Kundendaten" soll ein Formular mit dem Namen "frmKunden" geöffnet werden. Lassen Sie ein neue Anwendung anfertigen ("File", "New"), fügen Sie ein MDI-Window ein und fügen Sie in die Sektion "Menu" des MDI-Windows folgenden Eintrag ein:

- Menu
  - Popup Menu: &Datei
    - Enabled when:
    - Status Text:
    - Menu Item: &Kundendaten
      - Keyboard Accelerator: (none)
      - Status Text:
      - Menu Settings
      - Menu Actions
    - Menu Item: &Lagerdaten
      - Keyboard Accelerator: (none)
      - Status Text:
      - Menu Settings
      - Menu Actions
    - Menu Separator
    - Menu Item: Be&enden
      - Keyboard Accelerator: (none)
      - Status Text:
      - Menu Settings
      - Menu Actions

Dem Menüpunkt "&Kundendaten" soll nun die Aktion zum Öffnen des Fensters "frmKunden" zugeordnet werden. Hierfür legen Sie zunächst ein neues Fenster an (über Tool-Palette, auf "Form Window" klicken) und geben diesem Fenster den Namen "frmKunden" (über Customizer, "Object Name"). Schalten Sie dann (im Customizer) die Option "Automatically Create" auf "No" um, so daß das Formular nicht automatisch angelegt wird.

Fügen Sie dann folgenden Inhalt (von Hand oder unter Verwendung der Outline Options) ein:

- Menu Item: &Kundendaten
  - Keyboard Accelerator: (none)
  - Status Text:
  - Menu Settings
  - Menu Actions
    - Call SalCreateWindow(frmKunden, hWndMDI)

Die Funktion SalCreateWindow() sorgt dafür, daß das Fenster mit dem Namen frmKunden angelegt wird. Der Eigentümer des Fensters soll das aktuelle MDI-Fenster sein, hWndMDI ist der dafür vorgesehene Handle. Beachten Sie, daß der Funktionsaufruf direkt stattfindet, ohne eine Konstruktion wie "On SAM_Click": das ist bei Aktionen, die durch Menüs ausgelöst werden, immer so!

Starten Sie die Anwendung, dann können Sie das Formular "frmKunden" anzeigen lassen, aber Sie können es (abhängig von der Einstellung "Reject Multiple Window Instances" in den Global Declarations) immer wieder anzeigen lassen, da die Menüoption "Kundendaten" nicht deaktiviert wird. Zur Erklärung dazu muß etwas ausgeholt werden: Bisher war oft die Rede von Handles oder davon, daß bei Funktionsaufrufen entweder der Handle oder aber der Name des Objekts angegeben werden muß. Im vorstehenden Beispiel wird der *Name* des Formulars (frmKunden) benutzt, dies funktioniert auch. Aber: Um zu erreichen, daß die

Menüoption nach dem Aufbau des Fensters deaktiviert wird, *muß ein Handle benutzt werden*. Erst durch den Handle (der noch definiert werden muß) kann in der Sektion "Menu Settings" die Deaktivierung einer Menuoption vereinbart werden. Das Verfahren wird im folgenden geschildert:

- Handle definieren (Name z.B. "hWndKundendaten"):

```
◆ Global Declarations
  ◆ Window Defaults
  ◆ Formats
  ◆ External Functions
  ◆ Constants
  ◆ Resources
  ◆ Variables
    ◇ Sql Handle: hSqlMaster
    ◇ Window Handle: hWndKundendaten
```

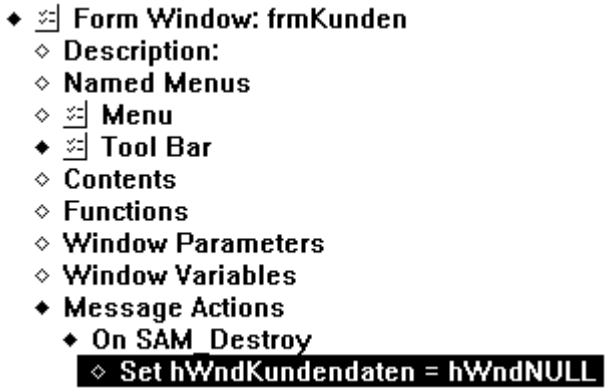
- Funktionsaufruf mit SalCreateWindow, der Rückgabewert dieser Funktion wird im Handle gespeichert, damit erhält der Handle nach Aufruf des Formulars einen Inhalt (wie dieser aussieht ist unerheblich):

```
◆ Menu Item: &Kundendaten
  ◇ Keyboard Accelerator: (none)
  ◇ Status Text:
  ◆ Menu Settings
  ◆ Menu Actions
    ◇ Set hWndKundendaten= SalCreateWindow( frmKunden, hWndMDI )
```

- Eintragung "Enable When" der Menüoption abändern, so daß der Menüpunkt nur dann gewählt werden kann, wenn der Handle keinen Inhalt hat:

```
◆ Menu Item: &Kundendaten
  ◇ Keyboard Accelerator: (none)
  ◇ Status Text:
  ◆ Menu Settings
    ◇ Enabled when: hWndKundendaten = hWndNULL
    ◇ Checked when:
```

- Natürlich müssen Sie dafür sorgen, daß die Menüoption dann wieder erlaubt ist, wenn das Fenster zwischenzeitlich geschlossen wurde. Hierzu geben Sie bei den Message Actions des Form Windows folgendes ein:



- Dadurch erhält (bei dem Entfernen des Formulars frmKundendaten) der Handle hWndKundendaten den gleichen Inhalt des Null-Handles hWndNULL, in anderen Worten, er wird auf NULL zurückgesetzt, wodurch die Menüoption wieder verfügbar wird.

Wird die Aktion SalCreateWindow() durch einen Klick auf einen Push Button ausgelöst, dann kann man den Push Button durch die Funktion SalDisableWindow() deaktivieren, das Sie auf den Menüpunkt "Kundendaten" nicht *direkt* zugreifen können, ist dieses Verfahren aber hier nicht möglich.

2 Entwickeln von Anwendungen

Das Kapitel 2 soll Ihnen die grundlegenden Schritte bei der Entwicklung von Anwendungen aufzeigen. Deshalb werden die hierzu nötigen Schritte detailliert erläutert, ohne dabei nochmals die Eigenschaften von Objekten oder Mechanismen des Einfügens vertieft darzustellen (vgl. dazu Kap. 1).

Zunächst wird erläutert, wie Sie mit den Daten aus nur einer einzelnen Tabelle arbeiten können (Kap. 2.1) danach wird die Verwendung mehrerer Tabellen demonstriert (Kap. 2.2).

In den folgenden Abschnitten wird getrennt nach Einzelsatz bzw. Tabellenmodus erläutert, wie Daten von einer Tabelle gelesen, angezeigt, verändert, gespeichert und auch gedruckt werden können

2.1 Entwickeln einer einfachen Anwendung mit den Daten einer Tabelle

Zunächst soll eine Verbindung zur Datenbank hergestellt werden. Obwohl es prinzipiell möglich ist, mit SQLWindows beliebige Computerprogramme zu erstellen, wird es wohl doch überwiegend so sein, daß Sie mit Daten aus einer Datenbank arbeiten. Diese Daten werden in der Regel mit einem Programm eingegeben, editiert, sortiert, abgefragt und ausgegeben (häufig werden Sie die Anbindung an eine Datenbank in die Startapplikation "Newapp.App" aufnehmen, um sich die Arbeit zu vereinfachen). Im folgenden Abschnitt wird die Anbindung an die Datenbank hergestellt.

2.1.1 Anbinden einer Anwendung an eine Datenbank

Bevor mit Daten aus einer Datenbank gearbeitet werden kann, muß diese Datenbank bekannt gemacht und geöffnet werden. Das Öffnen oder Anbinden an die Datenbank besteht darin, daß Sie der Anwendung mitteilen, daß zu einer Datenbank mit einen bestimmten Namen (im folgenden Beispiel ist es die Datenbank mit dem Namen "FIRMA") eine Verbindung über ein sog. "Handle" hergestellt werden soll. Dieses Handle dient dazu, der Verbindung zur Datenbank einen Namen zu geben. Dieser Name wird immer verwendet, wenn Daten aus dieser Datenbank zu lesen, schreiben, ändern oder drucken sind (dabei ist zu beachten, daß der Name des Handles als Variable vereinbart werden muß).

Neben dem Namen der Datenbank kann auch einem User ein Name gegeben werden; damit erhält er die ihm zugeordneten Rechte. Hierauf wird im folgenden aber nicht eingegangen.

Der folgende Programmcode zeigt das Definieren des Handle in der Sektion "Global Declarations", "Variables" als globale Variable:

```
◆ Variables
  ◇ Sql Handle: hSql
```

Hiermit ist eine Variable vom Typ Sql-Handle vereinbart (häufig finden sich an dieser Stelle noch weitere globale Variablen, die später im Programm Verwendung finden; würden Sie diese Definition unterlassen, erhielten Sie später eine Fehlermeldung).

Über diesen Handle soll nun eine Verbindung zur Datenbank hergestellt werden. Im folgenden Programmcode finden sich die zum Anbinden an eine Datenbank nötigen Schritte, die in der Sektion "Global Declarations", "Application Actions" codiert sind (das ist üblicherweise die Stelle, an der eine Datenbank geöffnet wird):

```
◆ Application Actions
  ◆ On SAM_AppStartup
    ◇ Set SqlDatabase = 'FIRMA'
    ◇ Call SqlConnect(hSql)
```

"On SAM_AppStartup" ist die Nachricht, die bei Start der Anwendung verschickt wird. Eingerückt darunter findet sich die im System für den Zugriff bereits vordefinierte Variable SqlDatabase. Diese Variable erhält den Inhalt 'FIRMA'. Über die Funktion "SqlConnect()", die mittels "Call" aufgerufen wird, wird unter Verwendung des Sql-Handles die Verbindung zwischen der Anwendung und der Datenbank hergestellt.

Häufig findet sich hier auch Programmcode, der beim Beenden der Anwendung auszuführen ist, so wie das im folgenden Programmcode zu sehen ist:

```
◆ Application Actions
  ◆ On SAM_AppStartup
    ◇ Set SqlDatabase = 'FIRMA'
    ◇ Call SqlConnect(hSql)
  ◆ On SAM_AppExit
    ◇ Call SqlDisconnect( hSql )
```

Während mittels der Nachricht "SAM_AppStartup" die Anweisungen beim Start des Programms ausgeführt werden, wird die Nachricht "SAM_AppExit" verwendet, um bei Beenden der Anwendung die Verbindung zur Datenbank zu trennen. Dazu wird die Funktion "SqlDisconnect()" verwendet, bei deren Aufruf der betreffende Handle erneut zu nennen ist.

Würden Sie die Anwendung jetzt starten, könnten Sie über den Task-Manager von Windows (Strg+Esc) feststellen, daß jetzt auch ein Programm mit dem Namen SQLBase läuft. Das ist die Datenbank-Maschine, die die von Ihnen später zu verwendenden SQL-Anweisungen erhält und ausführt.

In den folgenden Abschnitten soll getrennt für Einzelsatzmodus und Tabellendarstellung erläutert werden, wie mit den Daten der Tabelle gearbeitet werden kann. Die Trennung nach diesen beiden Modi wurde deshalb gewählt, weil beide Arten der Darstellung in Ihren späteren Anwendungen eine dominierende Rolle spielen werden. Sie müssen jeweils *vor* Beginn der Codierung entscheiden, welcher Modus für Ihre speziellen Anforderungen besser geeignet ist. Daneben werden Sie häufig eine Kombination von beiden Darstellungsarten benutzen müssen, dann nämlich, wenn Sie Master-Detail Daten darstellen wollen.

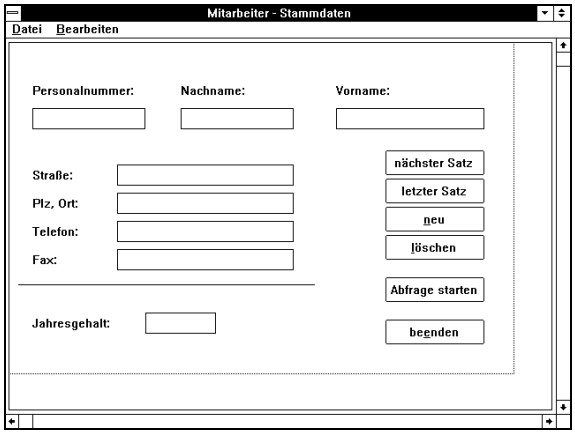
Zu überlegen ist ferner, ob bei einer Tabellendarstellung von Daten auf Einzelsatzdarstellung umgestellt wird, wenn Datensätze editiert werden sollen. Diese andere Form der Darstellung kann das Ändern von Daten auch optisch von der einfachen Anzeige trennen.

2.1.2 Einzelsatzmodus

Der Einzelsatzmodus ist dadurch zu beschreiben, daß immer nur die Informationen eines einzelnen Datensatzes auf dem Bildschirm sichtbar sind. Dabei müssen aber nicht alle Felder eines Datensatzes angezeigt werden: Manche Felder werden vor dem Benutzer verborgen, manche Felder werden zwar angezeigt, sind aber nicht editierbar.

Sinnvoll eingesetzt wird dieser Einzelsatzmodus beispielsweise bei der Neuerfassung von Stammdaten, z.B. bei der Aufnahme eines neuen Artikels in den Artikelstamm oder bei Aufnahme eines neuen Mitarbeiters in den Mitarbeiterstamm, wie es die folgende Abbildung zeigt:

Abb. 2.1: Einzelsatzmodus, Erfassung von Stammdaten.



Die Vorteile des Einzelsatzmodus liegen zum einen in seiner Übersichtlichkeit, da immer nur ein Datensatz zu sehen ist. Zum anderen können viele Objekte zusammen mit dem Einzelsatz angezeigt werden: in der vorstehenden Abbildung sind das Push Buttons, denkbar sind aber auch alle anderen Child Objekte, die SQLWindows kennt.

Von Nachteil ist dieser Modus immer dann, wenn mehrere Sätze betrachtet werden sollen, beispielsweise bei der Prüfung von Bestandsdaten einer Lagertabelle, da dann häufiges Blättern nötig wird (bei der Darstellung von Daten in Tabellenform ist die Kontrolle der Daten wesentlich einfacher). Außerdem erfordert der Einzelsatzmodus einen höheren Programmieraufwand bei der Erstellung der Masken.

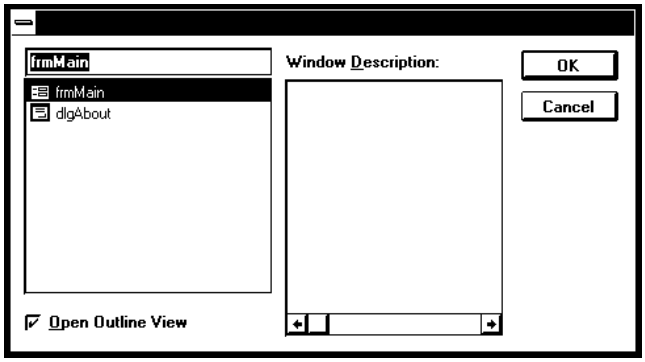
In folgendem Beispiel sollen Datensätze aus der Tabelle "KUNDEN" angezeigt werden. In einem Formular sollen zu den Data Fields Push Buttons zum Blättern bzw. Editieren enthalten sein (dies könnte alternativ auch über Menüpunkte gesteuert werden).

2.1.2.1 Aufnehmen von Data Fields in frmMain

Um das Vorgehen beim Einfügen von Data Fields und dem Füllen eines Data Fields mit Information möglichst prägnant beschreiben zu können, wird zunächst nur ein Data Field eingefügt und weiter verarbeitet. Erst nachdem die zu codierenden Aktionen mit diesem Data Field im Sinne der Anwendung funktionieren, werden die anderen Data Fields in das Formular eingebaut und die Codierung ergänzt.

Lassen Sie sich zunächst das Formular "frmMain" anzeigen. Drücken Sie dazu die Tastenkombination Strg+1 und wählen Sie "frmMain" aus.

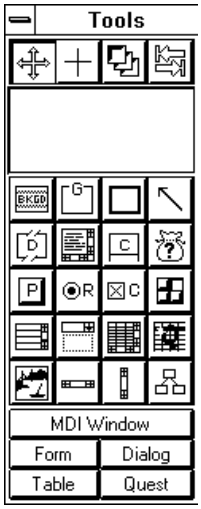
Abb. 2.2: Das Fenster "Show Window" (aktiviert mit Strg+1).



Das Formular "frmMain" sollte leer sein falls Objekte enthalten sind, entfernen Sie diese, indem Sie zunächst mit der linken Maustaste einmal darauf klicken und dann die Entf-Taste drücken.

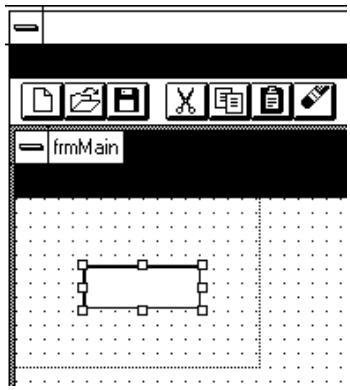
Lassen Sie sich jetzt die Tool-Palette anzeigen, indem Sie die F4-Taste drücken.

Abb. 2.3: Tool-Palette (aktiviert mit F4).



Klicken Sie einmal auf das Symbol für das Data Field (zweite Reihe, ganz links, mit einem "D"), ziehen Sie den Mauszeiger in Ihr Formular und klicken Sie dort einmal (linke Maustaste), um das Data Field einzufügen.

Abb. 2.4: Eingefügtes Data Field im Formular frmMain.



Das eben eingefügte Data Field hat den Namen "df1"; Sie können den Namen anzeigen lassen, indem Sie mit der rechten Maustaste einmal auf dieses Objekt klicken und aus dem daraufhin angezeigten Customizer den Eintrag "Object Name" anzeigen lassen, zusätzlich wird der Name des Objekts, da es markiert ist, am unteren Bildschirmrand in der Statuszeile angezeigt (ggf. die Statuszeile über den Menüpunkt "View", "Status Bar" anzeigen lassen). Der Name "df1" ist von SQLWindows selbst vorgeschlagen worden, nur hier bleibt dieser Name unverändert. In späteren "echten" Anwendungen werden Sie die Namen immer abändern, z.B. in "dfKundenNummer" oder "dfKundenName", Bezeichnungen also, die Sie oder andere Benutzer auch nach einiger Zeit noch verstehen können.

Nachdem Sie dieses Data Field eingefügt haben, müssen Sie innerhalb der Anwendung definieren, was damit zu geschehen hat (vgl. dazu den folgenden Abschnitt).

2.1.2.2 Anzeigen von Datensätzen

Dem eben eingefügten Data Field soll nun der Inhalt einer Tabellenspalte zugeordnet werden. Dazu muß sichergestellt sein, daß ein Sql-Handle definiert wurde und daß über dieses Handle eine Verbindung zur Datenbank hergestellt wird (prüfen Sie das evtl. nochmals indem Sie sich den Inhalt der Sektion "Variables" bzw. "Application Actions" anzeigen lassen).

Um den Inhalt einer Tabellenspalte zu lesen und diesen Inhalt im Data Field "df1" anzeigen zu lassen, muß eine Funktion verwendet werden. Diese Funktion heißt

SqlPrepareAndExecute()

und enthält neben dem Namen des Sql-Handles auch die eigentliche Sql-Anweisung.

Der Name des Sql-Handles lautet **hSql**; er wurde bereits unter "Variables" vereinbart.

Die Sql-Anweisung, eine normale **Select**-Anweisung, enthält den Namen der Spalte, aus der die Information zu lesen ist: hier der Name eines Kunden (Spaltenname: **Name**), den Namen der Tabelle (Tabellenname: **Kunden**) und den Namen des Objekts, in dem angezeigt werden soll ("df1"). Der Funktionsaufruf lautet daher wie folgt:

Call SqlPrepareAndExecute(hSql, 'select Name from Kunden into :df1')

Der Doppelpunkt (vor "df1") muß immer dann verwendet werden, wenn Variable innerhalb von Sql-Anweisungen verwendet werden.

Neben der Möglichkeit, die Funktion "SqlPrepareAndExecute" zu benutzen, steht noch ein Funktionspaar zur Verfügung, mit dem die einzelnen Teilaufgaben separat ausgeführt werden können, nämlich "SqlPreprare()" bzw. "SqlExecute()". Der Vorteil bei der Verwendung dieses Funktionspaars liegt darin, daß es nicht nötig ist, die Sql-Anweisung jedesmal vorzubereiten *und* auszuführen. Genaugenommen ist es geschickter, bei dem Start einer Anwendung die Sql-Select-Anweisung einmal vorzubereiten und sie nur dann auszuführen, wenn es nötig ist, z.B. nach dem Drücken eines Push Button. Hier wurde die "unsaubere"Lösung gewählt, um das Beispiel kurz zu halten.

Das allein genügt zwar für das Lesen, aber noch nicht für die Anzeige der Daten. Die Anzeige kann mit der Funktion

SqlFetchNext()

realisiert werden, die aus einer mit Select erzeugten Ergebnismenge den nächsten anstehenden Satz holt. Dies erscheint zunächst etwas seltsam, ermöglicht aber insgesamt eine außerordentliche Flexibilität, da neben der genannten Funktion für die Anzeige des nächsten Datensatzes auch noch andere Funktionen

("SqlFetchPrevious()" und "SqlFetchRow()") vorhanden sind; die erstgenannte Funktion erlaubt es, rückwärts durch eine Ergebnismenge zu "hüpfen", die letztgenannte erlaubt das Anwählen eines bestimmten Datensatzes anhand seiner Nummer.

Beim "Holen" des nächsten Datensatzes können mehrere Situationen auftreten, die alle mit den genannten Funktionen behandelt werden können: Ein Satz kann seit dem letzten Lesen mittlerweile gelöscht sein; es ist kein weiterer Satz mehr vorhanden, der Datensatz wurde seit dem letzten Lesen geändert und schließlich kann es funktionieren, so daß keine "Störungen" auftreten.

Als Resultat einer "SqlFetch"-Anweisung werden nun diese möglichen Ergebnisse in einer (Receive-) Variablen abgelegt (hier hat sie den Namen "nFetchResultat"), deren Inhalt dann verwendet werden kann, um den Benutzer z.B. über bestimmte Sachverhalte zu informieren. Es ist zwar möglich, eine selbst definierte Variable zu benutzen, man nimmt hierzu aber üblicherweise die in SQLWindows bereits vorformulierten Konstanten. Die folgende Übersicht nennt die dabei möglichen Konstanten, deren numerischen Inhalt und eine kurze Beschreibung:

Konstante	numerischer Inhalt	Beschreibung
FETCH_Ok	0	Nächster Datensatz mit Erfolg gelesen.
FETCH_EOF	1	Keine weiteren Datensätze vorhanden.
FETCH_Update	2	Datensatz wurde seit dem letzten Lesen verändert.
FETCH_Delete	3	Datensatz wurde seit dem letzten Lesen gelöscht.

Um nun den ersten Datensatz anzeigen zu lassen, muß die Funktion "SqlFetchNext()" einmal ausgeführt werden. Der Aufruf geschieht mit der folgenden Anweisung:

SqlFetchNext(hSql, nFetchResultat)

Der Sql-Handle hSql wurde bereits oben erläutert, nFetchResultat ist die oben erwähnte (Receive-) Variable, in der ein Rückgabewert (s. vorstehende Tabelle) gespeichert wird. Sie müssen hierzu die Variable "nFetchResultat" mit dem Typ Number definieren, der folgende Programmausschnitt zeigt das:

- **Variables**
  - **Sql Handle: hSql**
  - **Number: nFetchResultat**

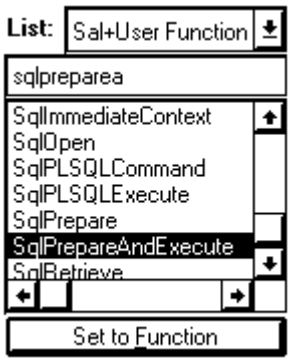
Jetzt muß die Anweisung zum Lesen und Anzeigen des Datensatzes codiert werden. Hier soll die Anzeige erst dann erfolgen, nachdem das Data Field "df1" angelegt wurde.

Klicken Sie also mit der rechten Maustaste auf das Data Field und wählen Sie aus dem Customizer den Eintrag "Edit Actions"; Sie sehen nun die Message Actions von "df1".

Wählen Sie aus den Outline Options (ggf. mit F2 aktivieren) den Eintrag **On SAM_Create**, indem Sie zweimal darauf klicken, der Eintrag wird dadurch eingefügt. Jetzt wählen Sie aus dem gleichen Fenster den Eintrag **Call** (ebenfalls zweimal klicken), auch dieser Eintrag wird als Code eingefügt.

Geben Sie in den Teil der "Outline Options", in dem die "Sal+User Functions" angezeigt werden die Zeichenfolge "sqlpreparea" in die Combo Box ein: die so lautende Funktion wird daraufhin angezeigt: klicken Sie zweimal auf dem Funktionsnamen, auch er wird eingefügt.

Abb. 2.5: Outline Options mit der Funktion "SqlPrepareAndExecute".



Jetzt sehen Sie im Programmcode die Schreibmarke innerhalb der Klammern der eben eingefügten Funktion. Ersetzen Sie den angezeigten Text durch den Namen des Sql-Handles und die Sql-Anweisung. Ergänzen Sie danach den Aufruf der Funktion (Call SqlFetchNext) entsprechend. Der folgende Code-Ausschnitt zeigt die fertigen Message Actions von df1.

```
◆ Message Actions
  ◆ On SAM_Create
    ◇ Call SqlPrepareAndExecute( hSql, 'select name from lager into :df1' )
    ◇ Call SqlFetchNext( hSql, nFetchResultat )
```

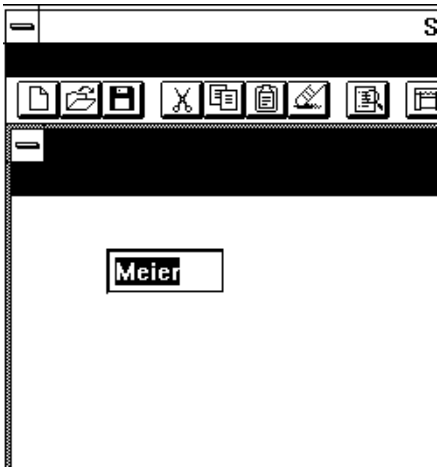
Zur Erläuterung:

Call	Aufruf einer Funktion.
SqlPrepareAndExecute	Funktion, die eine SQL-Anweisung vorbereitet (prepare) und sofort danach ausführt (execute).
hSql	Handle für den Zugriff auf die Datenbank.
select name from kunden into :df1	SQL-Anweisung, erweitert um "into :df1"; into :df1 bedeutet, daß die Ergebnismenge in eine Variable mit dem Namen df1 gespeichert werden soll.
SqlFetchNext()	Funktion für das Lesen des nächsten Datensatzes.
nFetchResultat	Numerische Variable, in die das Ergebnis der Fetch-Operation abgelegt wird.

Natürlich landen nicht alle Namen sämtlicher Kunden sofort in dem einen Data Field. Vielmehr ist es so, daß durch das Ausführen der Funktion SqlPrepareAndExecute() eine "Warteschlange" entsteht, die die Daten enthält. Erst durch den Aufruf der Funktion SqlFetchNext() kann der erste bzw. nächste in der Warteschlange enthaltene Datensatz gelesen und angezeigt werden.

Starten Sie jetzt Ihre Anwendung durch Drücken der Taste F7 (falls die Frage nach dem Sichern gestellt wird, dann erlauben Sie das und geben einen Dateinamen an). Die folgende Abbildung zeigt den Inhalt des Feldes "Name" nach dem Start der Anwendung:

Abb. 2.6: Inhalt eines Feldes nach SqlPrepareAndExecute() sowie SqlFetchNext().



Es ist von größter Bedeutung, daß Sie sich über die Verwendung der Funktion "SqlFetchNext()" Klarheit verschaffen! Insbesondere die Möglichkeiten, den Benutzer darauf hinzuweisen, daß er am letzten oder ersten Satz einer Tabelle angelangt ist, sowie die Möglichkeit, Datensätze zu drucken, hängen von dieser Funktion und den damit zusammenhängenden Konstanten (FETCH_*) ab.

Um nun *andere Spalten aus der Tabelle Kunden in die Maske aufzunehmen* und die entsprechenden Inhalte anzeigen zu lassen, gehen Sie folgendermaßen vor:

- Lassen Sie das Formular frmMain im Entwurf anzeigen (ggf. Strg+1) drücken;
- Lassen Sie über die Tool Palette (Aktivieren mit F4) zwei weitere Data Fields einfügen;
- Öffnen Sie den Customizer von "df1" (mit rechter Maustaste auf "df1" klicken);
- Ändern Sie die Sql-Anweisung ab, so daß folgender Text erscheint:

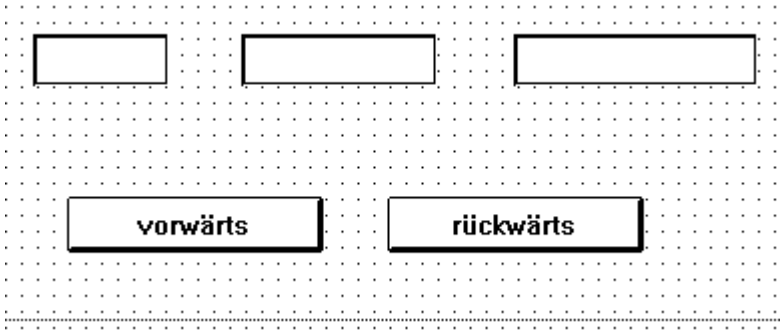
```
▸ On SAM_Create
  ◇ Call SqlPrepareAndExecute( hSql, 'select name, vorname, ort from kunden
    into :df1, :df2, :df3' )
  ◇ Call SqlFetchNext( hSql, nFetchResultat )
```

Lassen Sie dann die Anwendung ausführen. Falls in einem der Felder keine Information angezeigt werden, liegt das eventuell daran, daß in der Tabelle mit Kundendaten überhaupt keine Daten vorhanden sind; Sie können das über die Option "View", "Database Browser" prüfen.

2.1.2.3 Blättern

Um zu blättern, müssen Sie die Funktionen "SqlFetchNext()" bzw. "SqlFetchPrevious()" benutzen. Fügen Sie zunächst zwei Push Buttons ein. Beschriften Sie den ersten ("pb1") mit "vorwärts", den zweiten ("pb2") mit "rückwärts". Folgende Abbildung zeigt das Formular mit den Data Fields und den eingefügten Push Buttons.

Abb. 2.7: Formular frmMain mit Data Fields und Push Buttons.



Klicken Sie nun mit der rechten Maustaste auf den "vorwärts" Push Button. Um den Customizer zu öffnen, wählen Sie "Edit Actions" aus. Nach Drücken des Push Buttons soll vorwärts geblättert werden. d.h. der nächste Datensatz aus der mit Select erzeugten Ergebnismenge soll angezeigt werden. Hierzu wird die bereits oben genannte Funktion "SqlFetchNext()" erneut aufgerufen. Die Message Actions des Push Buttons für das Blättern (vorwärts) sehen dann folgendermaßen aus:

- Message Actions
  - On SAM_Click
    - Call SqlFetchNext(hSql, nFetchResultat)

Fast genauso sieht der Inhalt des Push Buttons aus, wenn Sie rückwärts blättern wollen; nur wird dort die Funktion "SqlFetchPrevious()" verwendet:

- Message Actions
  - On SAM_Click
    - Call SqlFetchPrevious(hSql, nFetchResultat)

Starten Sie die Anwendung und testen Sie die Aktionen.

Ein Aspekt soll hier noch erwähnt werden: Wenn Sie mit den Push Buttons durch die Datensätze blättern, erhalten Sie keine Information darüber, ob Sie am Ende oder am Anfang der Datensätze sind. Nach Drücken des Vorwärts-Push Buttons auf dem letzten angezeigten Datensatz, erfolgt lediglich keine weitere Anzeige. Dies kann man aber so gestalten, daß eine kurze Meldung (in der Statuszeile am unteren Bildschirmrand oder in einer separaten Message Box) dem Benutzer mitteilt, daß er am Ende bzw. am Anfang der Datensätze angekommen ist.

Dies soll im folgenden kurz vorgestellt werden. Dabei prüft das erste Beispiel, ob bei dem vorwärts Blättern der letzte, das zweite Beispiel, ob bei dem rückwärts Blättern der erste Datensatz erreicht wurde.

- Meldung in der Statuszeile (bei vorwärts blättern)

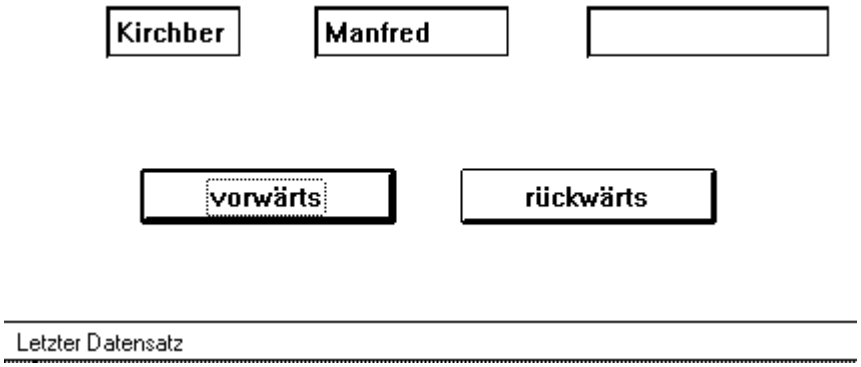
  Hierzu wird die Funktion "SalStatusSetText()" verwendet. Beim Aufruf dieser Funktion müssen der Handle (oder der Name) des Objekts sowie der anzuzeigende Text angegeben werden. Der Inhalt der Message Actions lautet dann:

```
◆ Message Actions
  ◆ On SAM_Click
    ◇ Call SqlFetchNext( hSql, nFetchResultat )
    ◆ If nFetchResultat =  FETCH_EOF
      ◇ Call SalStatusSetText( frmMain, 'Letzter Datensatz' )
```

  Zuerst wird über "SqlFetchNext()" der nächste Satz geholt, danach wird geprüft, ob die Variable nFetchResultat den Inhalt "FETCH_EOF" enthält. Falls dies zutrifft (If...), wird die Funktion "SalStatusSetText()" mit den Parametern "frmMain" und "Letzter Datensatz" aufgerufen. Dabei bedeutet "frmMain", daß die Meldung in dem Fenster mit dem Namen "frmMain" anzuzeigen ist; die in einfachen Anführungszeichen eingeschlossene Zeichenfolge ist der anzuzeigende Text.

  Dies führt zur folgenden Meldung am unteren Bildschirmrand (ggf. über Customizer, "Accessories Enable", "Yes", die Anzeige der Statuszeile gestatten).

Abb. 2.8: Statuszeile eines Form Windows mit angezeigter Meldung.

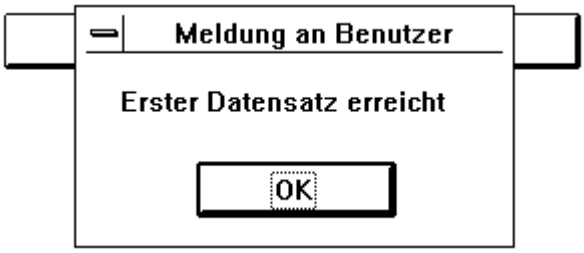


- Um beim rückwärts Blättern eine Meldung in einer Message Box anzeigen zu lassen, codieren Sie folgendes:

```
◆ Message Actions
  ◆ On SAM_Click
    ◇ Call SqlFetchPrevious( hSql, nFetchResultat )
    ◆ If nFetchResultat = FETCH_EOF
      ◇ Call SalMessageBox('Erster Datensatz erreicht',
        'Meldung an Benutzer', MB_Ok )
```

Dies führt dann zur Anzeige einer Message Box mit der angegebenen Meldung ("Erster Datensatz erreicht"), dem angegebenen Fenstertitel ("Meldung an Benutzer") und dem angegebenen Message Button ("MB_Ok", einem OK-Push Button).

Abb. 2.9: Message Box mit angezeigter Meldung.



Beachten Sie, daß in beiden Fällen mit der Konstruktion "FETCH_EOF" gearbeitet wird.

2.1.2.4 Editieren von Datensätzen

Unter dem Editieren von Datensätzen wird hier die Möglichkeit verstanden den

- Inhalt vorhandener Datensätze zu ändern;
- Datensätze zu löschen oder
- neue Datensätze einzugeben.

Getrennt nach diesen drei Möglichkeiten werden in den folgenden Abschnitten einzelne Lösungen beschrieben, wobei der jeweils nötige Programmcode in den einzelnen Kapiteln separat vorgestellt wird. Natürlich können Sie auch alle Möglichkeiten (ändern, löschen und eingeben) in einer umfassenden Funktion abhandeln, doch führt dies zu relativ komplexen Funktionen, die am Anfang nicht leicht zu verstehen sind. Eine solche Funktion, die alle drei hier differenzierten Fälle gemeinsam behandelt, wird am Ende von Kapitel 2.1.3.3.3 vorgestellt.

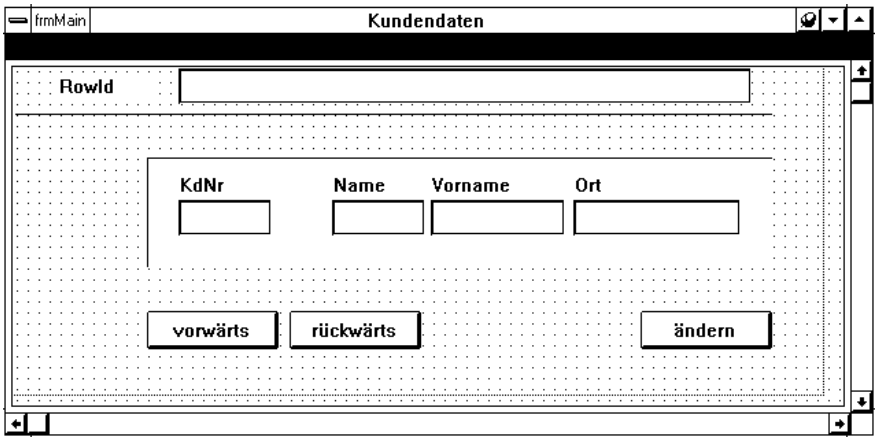
Von zentraler Bedeutung bei den hier geschilderten Vorgängen ist das Vorhandensein einer eindeutigen Identifikationsnummer, "ROWID" genannt, die jedem gelesenen Datensatz von der Datenbank zugewiesen wird (dies ist eine Besonderheit der Datenbank SQLBase, die Sie in vielen anderen Datenbanken nicht finden). Die Datenbank gibt dabei jedem gelesenen Datensatz die Kopie der auf der Festplatte abgelegten "ROWID" mit.

Daneben spielt der bei Stammdaten übliche Primärschlüssel (die Kundennummer bei den Kundendaten oder die Artikelnummer bei den Lagerdaten) eine besondere Rolle, da über diese Nummer Änderungen oder Löschungen programmiert werden können, ohne auf die "ROWID" zurückzugreifen.

2.1.2.4.1 Ändern von Datensätzen

Das Ändern von Datensätzen soll anhand der Daten der Tabelle "Kunden" erläutert werden. Um die Ausgangslage zu erläutern, wird zunächst das hier verwendete Formular ("frmMain") mit den darin enthaltenen Objekten vorgestellt.

Abb. 2.10: Kundendaten (Formular mit Push Button zum Ändern der Daten).



Zu den einzelnen Objekten ist folgendes anzumerken:

- Das Formular ("frmMain") enthält ein Data Field ("dfROWID") für die Anzeige der "ROWID" des Datensatzes. Dieses Data Field wird später verwendet, wenn die Daten mittels einer Update-Anweisung geändert werden.

- Darunter finden sich die Data Fields für Kundennummer ("dfKdnr"), Name ("dfName"), Vorname ("dfVorname") und Ort ("dfOrt"). Das Data Field für die Kundennummer ist vom Typ Number, alle anderen sind vom Typ String. Alle Data Fields sind nicht editierbar, sie dienen nur der Anzeige der Daten.

- Die auf dem Formular befindlichen Push Buttons dienen dem Blättern ("vorwärts", "rückwärts" s. hierzu Kap. 2.1.2.3) sowie der Sicherung von geänderten Datensätzen ("ändern").

Anbindung an die Datenbank

Zunächst wird beim Start der Anwendung an die Datenbank FIRMA angebunden (die dazu nötige Variable für den Sql-Handle muß vorher definiert werden).

```
◆ Application Actions
  ◆ On SAM_AppStartup
    ◇ Set SqlDatabase = 'FIRMA'
    ◇ Call SqlConnect (hSql)
```

Anzeige des ersten Datensatzes

Die Message Actions-Sektionen der Data Fields enthalten im Gegensatz zum oben genannten Beispiel keinen Code, die Anzeige des ersten Datensatzes wird vielmehr über die Message Actions des Formulars (frmMain) durchgeführt. Diese Sektion hat den folgenden Inhalt:

```
◆ Message Actions
  ◆ On SAM_Create
    ◇ Call SqlPrepareAndExecute( hSql,
        'select ROWID, kundennummer, name, vorname, ort
        from kunden
       into :dfROWID, :dfKdnr, :dfName, :dfVorname, :dfOrt'  )
    ◇ Call SqlFetchNext( hSql, nFetchResultat )
```

Die Funktion "SqlPrepareAndExecute()" dient dazu, die Sql-Anweisung vorzubereiten (zu compilen) und auszuführen (execute). Sie enthält den Namen des dazu verwendeten Sql-Handles (hSql) und die eigentliche Sql-Anweisung, erweitert um die Namen der Data Fields in dem Formular. Die folgende Tabelle nennt die dabei verwendeten Namen. Beachten Sie dabei, daß Groß- oder Kleinschreibung bei den Spaltennamen unerheblich ist, daß aber die Schreibweise der Data Fields-Namen genau einzuhalten ist.

Spaltenname aus der Tabelle	Name des Data Fields	Inhalt
ROWID	dfROWID	Identifikationsnummer des Datensatzes
kundennummer	dfKdnr	Kundennummer
name	dfName	Name des Kunden
vorname	dfVorname	Vorname des Kunden
ort	dfOrt	Wohnort des Kunden

Die Namen der Data Fields können über den Customizer (mit rechter Maustaste auf dem Data Field klicken) geändert werden.

Ändern der Datensätze mittels eines zusätzlichen Formulars

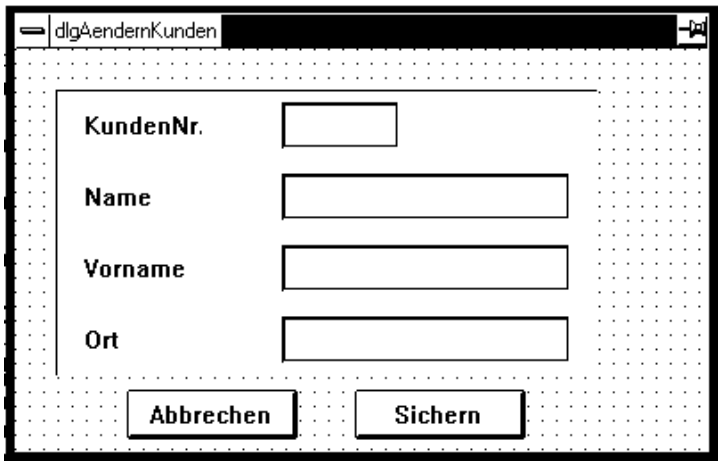
Durch das Drücken des Push Button soll ein zusätzliches Formular geöffnet werden, in dem der Benutzer die Daten ändern kann. Der Push Button ("pb3") beinhaltet dafür den folgenden Code:

```
◆ Pushbutton: pb3
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalModalDialog(dlgAendernKunden, frmMain)
```

Die Funktion "SalModalDialog()" dient dazu eine Dialog Box mit dem Namen **dlgAendernKunden** zu öffnen, Eigentümer dieses Fensters soll das bereits angezeigte Formular "frmMain" sein.

Die folgende Abbildung zeigt das Aussehen der Dialog Box, die über die Tool-Palette angelegt werden kann (der Name wird über den Customizer verändert):

Abb. 2.11: Dialog Box "dlgAendernKunden" mit Data Fields und Push Buttons.



Die Data Fields in dieser Dialog Box erhalten bei Erschaffung der Dialog Box den gleichen Inhalt wie die Data Fields im Formular frmMain. Der folgende Programmcode zeigt den dazu verwendeten Inhalt der Message Actions dieser Dialog Box, der erforderlich ist , um die Daten aus dem Formular "frmMain" in die Dialog Box zu kopieren.

```
◆ Dialog Box: dlgAendernKunden
  ◇ Description:
  ◆ Tool Bar
  ◇ Contents
  ◇ Functions
  ◇ Window Parameters
  ◇ Window Variables
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Set dlgAendernKunden.dfKdnr = frmMain.dfKdnr
      ◇ Set dlgAendernKunden.dfName = frmMain.dfName
      ◇ Set dlgAendernKunden.dfVorname = frmMain.dfVorname
      ◇ Set dlgAendernKunden.dfOrt = frmMain.dfOrt
```

Die in dieser Dialog Box verwendeten Namen für Data Fields entsprechen jenen, die im Formular frmMain verwendet werden. Deshalb werden die Data Fields mit ausführlicher Referenz ("fully qualified Reference") z.B. **dlgAendernKunden.dfKdnr** genannt.

Nachdem die Daten des aktuellen Datensatzes - nach Drücken des Push Button "Ändern" - in der Dialog Box angezeigt werden, kann der Benutzer die Daten ändern.

Nach Änderung der Daten, bieten sich zwei Möglichkeiten: Entweder sind die Änderungen zu verwerfen oder aber zu sichern. Hierzu werden in der Dialog Box zwei Push Buttons eingefügt, einer um abzubrechen, ein weiterer um zu sichern:

- Push Button für Abbrechen (Änderungen werden nicht gesichert)

  Der folgende Code zeigt den Inhalt der Message Actions des Push Buttons, der dann gedrückt wird, wenn nicht zu sichern ist:

```
◆ Pushbutton: pb1                    ! Abbrechen
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalEndDialog( dlgAendernKunden,0 )
```

  Lediglich das Schließen der Dialog Box ist hier angeordnet, sonst ist keine weitere Aktion nötig (die Data Fields enthalten nach dem Schließen der Dialog Box zwar noch immer ihre Daten, doch bei dem nächsten Aufruf zur Änderung der Daten werden die alten Inhalte mit neuen Inhalten überschrieben).

- Push Button für Sicherung der Daten.

  In den Message Actions ist eine Sql-Anweisung ("update") enthalten, die die veränderten Daten in die Datenbank zurückschreibt. Dazu wird Bezug genommen auf die oben erwähnte "ROWID", anhand der die Inhalte der Data Fields zugeordnet werden können. Die Update-Anweisung wird mittels der Funktion "SqlImmediate()" ausgeführt, die sofort die Änderungen vornimmt:

```
◆ Pushbutton: pb2                    ! Speichern
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SqlImmediate( 'update kunden
        set name = :dlgAendernKunden.dfName,
        vorname = :dlgAendernKunden.dfVorname,
        ort = :dlgAendernKunden.dfOrt
        where ROWID = :dfROWID' )
```

  Damit werden die geänderten Daten in die Datenbank zurückgeschrieben. Die Zuordnung der geänderten Daten zur Datenbank (bzw. der in ihr enthaltenen Tabelle) findet dabei über die "ROWID" statt.

Nach Änderung der Daten in der Datenbank-Tabelle "Kunden" sind die ursprünglichen Inhalte des Formulars frmMain natürlich noch unverändert. Deshalb sollen über eine weitere Select-Anweisung die Kundendaten erneut gelesen und (im Formular frmMain) angezeigt werden.

```
◇ Call SqlImmediate( 'select
  ROWID, Kundennummer,name, vorname, ort
  from kunden
  into
  :frmMain.dfROWID, :frmMain.dfKdnr, :frmMain.dfName,
  :frmMain.dfVorname, :frmMain.dfOrt
  where kundennummer  = :frmMain.dfKdnr')
```

Hierbei gilt zu beachten, daß nur die Inhalte des geänderten Datensatzes über den Inhalt der Spalte "Kundennummer" gelesen und angezeigt werden. Die Verwendung von "ROWID" ist nicht möglich, da ein geänderter Datensatz nach der Änderung eine neue ROWID erhält.

Schließlich muß die Dialog Box geschlossen werden, so daß der Benutzer mit seiner Arbeit fortfahren kann.

◇ **Call SalEndDialog(dlgAendernKunden, 0)**

Damit sind die Änderungen geschrieben und die neuen Daten werden im Formular frmMain angezeigt.

Eine andere Möglichkeit, den Inhalt von Datensätzen zu ändern, findet sich in Kapitel 3 und der Beispielapplikation "MUFORM.APP", die im Verzeichnis \GUPTA\SAMPLES enthalten ist.

2.1.2.4.2 Löschen von Datensätzen

Das Löschen von Datensätzen wird unter Verwendung der gleichen Anwendung erläutert wie das im vorhergehenden Abschnitt erläuterte Ändern von Datensätzen. Hierzu wird lediglich ein zusätzlicher Push Button benötigt, der die Aufschrift "Löschen" erhalten soll.

Wie bereits oben erwähnt, können in einer Anwendung mehrere Sql-Handles verwendet werden, die auf die gleiche Datenbank zugreifen. Der Grund, weshalb in diesem Beispiel ein zusätzlicher Sql-Handle eingeführt und benutzt wird, liegt darin, daß bei mehreren Sql-Handles jeder für genau eine Aufgabe verwendet werden kann. So wird in Anwendungen häufig ein Sql-Handle ausschließlich für die Select-Anweisung, ein zweiter ausschließlich für die Update-Anweisung, ein dritter für die Delete-Anweisung und ein vierter schließlich für die Insert-Anweisung verwendet. Der hierbei entstehende Vorteil liegt darin, daß die einzelnen Sql-Anweisungen nicht jedesmal neu kompiliert und ausgeführt werden müssen, sondern bleiben vorkompiliert und werden nur noch für die Ausführung aufgerufen. Dies kann Gewinne in der Geschwindigkeit mit sich bringen (wenn dies auch in den hier vorgestellten kleinen Anwendungen keine Rolle spielt).

Die Variables-Sektion einer Anwendung mit mehreren Sql-Handles enthält folgenden Inhalt:

- **Variables**
  - ◇ **Sql Handle: hSqlSelect**
  - ◇ **Sql Handle: hSqlDelete**
  - ◇ **Boolean: bOk**
  - ◇ **Number: nFetchResultat**

Achten Sie vor allem auf die zwei Sql-Handles: einer davon ("hSqlSelect") wird ausschließlich für die Sql-Select Anweisung benutzt, der andere ("hSqlDelete") ausschließlich für die Sql-Delete Anweisung.

Eine weitere Variable ("bOk") wird hier verwendet, um den Rückgabewert der Funktionen "SqlPrepare()" bzw. "SqlExecute()" zu speichern (damit kann später im Programmablauf geprüft werden, ob diese Funktionen erfolgreich durchgeführt werden konnten).

Das Löschen selbst ist sehr einfach codierbar: Zunächst soll der Benutzer mit einer Dialog Box gefragt werden, ob tatsächlich gelöscht werden soll. Falls die Antwort "nein" lautet, passiert nichts, falls die Antwort dagegen "ja" lautet, wird der gerade angezeigte Datensatz gelöscht.

Der folgende Programmcode zeigt den Inhalt der Message Actions des Push Buttons im Formular "frmMain", der zum Löschen eines angezeigten Datensatzes verwendet wird:

```
◆ Pushbutton: pb4                    ! löschen von Datensätzen
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalMessageBeep(0)
      ◆ If IDOK= SalMessageBox( 'Datensatz wirklich löschen?',
        'Löschen', MB_OkCancel)
        ◇ Set bOk = SqlPrepare(hSqlDelete,
          'delete from kunden where kundennummer =:dfKdnr')
        ◇ Set bOk = SqlExecute(hSqlDelete)
        ◆ If bOk
          ◇ Call SqlCommit(hSqlDelete)
          ◇ Call SqlPrepareAndExecute( hSqlSelect,
            'select ROWID, Kundennummer, Name, Vorname, Ort
             from Kunden
            into :dfROWID, :dfKdnr, :dfName, :dfVorname, :dfOrt' )
          ◇ Call SqlFetchNext( hSqlSelect, nFetchResultat )
```

Zur Erläuterung:

- Zunächst wird ein Signalton ausgegeben ("SalMessageBeep()").

- Falls der Rückgabewert ("IDOK", eine im System vorhandene Konstante) der Message Box "TRUE" ist, dann wird die Funktion "SqlPrepare()" aufgerufen, die die in Klammern angegebene Sql-Anweisung vorbereitet. Diese Funktion "SqlPrepare()" liefert selbst einen logischen Wert zurück, der in der oben erwähnten Variablen "bOk" gespeichert wird.

- Die Funktion "SqlExecute()" wird aufgerufen, um die kompilierte Sql-Anweisung unter Verwendung des Handles "hSqlDelete" auszuführen. Auch der hierbei zurückgelieferte logische Wert wird in "bOk" gespeichert.

- Falls der Inhalt der Variablen "bOk" "TRUE" ist ("If bOk"), wird die Funktion "SqlCommit()" aufgerufen, die die letzten Änderungen fixiert.

- Jetzt werden über den anderen Sql-Handle ("hSqlSelect") unter Verwendung der Funktion "SqlPrepareAndExecute()" die Daten erneut eingelesen und der nächste Datensatz aus der Ergebnismenge angezeigt ("SqlFetchNext()").

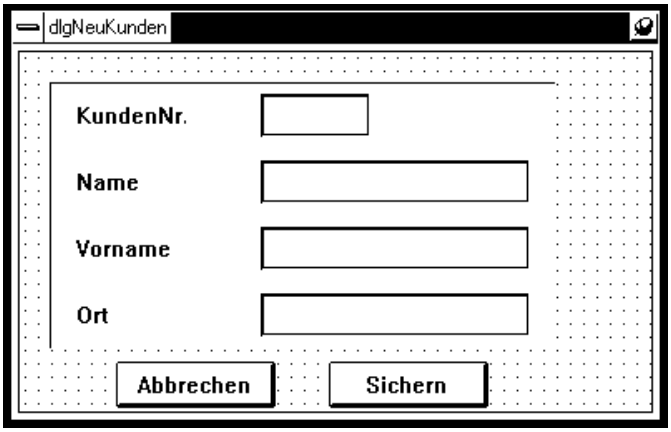
2.1.2.4.3 Einfügen von Datensätzen

Um einen neuen Datensatz in die Tabelle mit den Kundendaten einzugeben, soll ebenso wie beim Ändern und Löschen der Daten eine Dialog Box verwendet werden. Kopieren Sie dazu die oben erzeugte Dialog Box (Name: "dlgAendernKunden") und benennen Sie die neu eingefügte Dialog Box als "dlgNeuKunden" (im Outline Text "dlgAendernKunden" markieren, Strg+Einfg danach Shift+Einfg drücken, anschließend den Namen ändern).

- Dialog Box: dlgAendernKunden
- Dialog Box: dlgNeuKunden

Die Dialog Box "dlgNeuKunden" enthält Data Fields, die mit Ausnahme der Kundennummer vom Benutzer gefüllt werden können.

Abb. 2.12: Dialog Box für die Neueingabe von Kundendaten.



Die Kundennummer spielt in Anwendungen immer eine besondere Rolle. Üblicherweise wird eine Kundennummer *nicht* vom Benutzer gewählt, sondern vom System selbst vorgeschlagen, häufig ist das Data Field nicht einmal editierbar. Um nun eine neue Kundennummer vorzuschlagen, muß eine separate Sql-Anweisung ausgeführt werden. In dieser Sql-Anweisung wird die höchste vorhandenen Kundennummer in der Tabelle mit Kundendaten gesucht, diese Zahl um 1 erhöht und das Ergebnis (die neu vorgeschlagene Kundennummer) in das dafür vorgesehene Data Field geschrieben.

Diese Sql-Anweisung soll nach Erschaffen der Dialog Box ausgeführt werden und wird deshalb in die Message Actions der Dialog Box gestellt.

- Message Actions
  - On SAM_Create
    - Call SqlImmediate('select max(kundennummer)+1from kunden into :dfKdnr')

Hiermit wird die neu vorgeschlagene Kundennummer in das Feld "dfKdnr" geschrieben, die anderen Felder können vom Benutzer editiert werden.

Die Codierung des Push Buttons zur Sicherung der Daten eines neuen Kunden ergibt folgenden Code:

```
◆ Message Actions
  ◆ On SAM_Click
    ◇ Call SqlImmediate( 'insert into  kunden
      (kundennummer, name, vorname, ort)
      values
      (:dlgNeuKunden.dfKdnr,
      :dlgNeuKunden.dfName,
      :dlgNeuKunden.dfVorname,
      :dlgNeuKunden.dfOrt) ' )
    ◇ Call SqlExecute( hSqlSelect )
    ◇ Call SqlFetchNext( hSqlSelect, nFetchResultat )
    ◇ Call SalEndDialog( dlgNeuKunden, 0 )
```

Zunächst wird über die Funktion "SqlImmediate()" eine Sql-Insert-Anweisung ausgeführt. In ihr sind dabei die Spalten der Tabelle Kunden als Ziel und die Data Fields aus dem Formular "frmMain" als Datenquelle enthalten.

Danach wird die Select-Anweisung, die bereits früher vorbereitet wurde (über "SqlPrepare") erneut ausgeführt (über "SqlExecute()") und mittels "SqlFetchNext()" der nächste Satz aus der Datenmenge angezeigt.

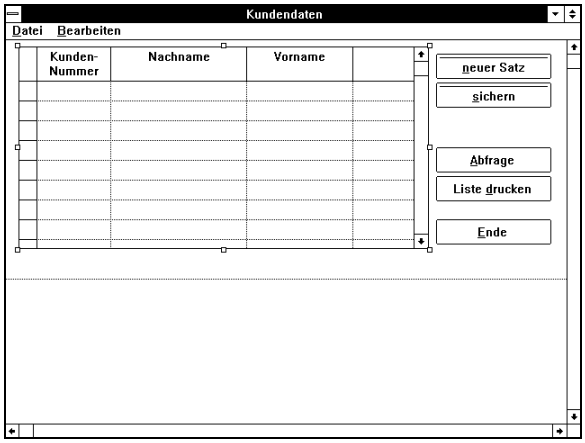
Schließlich wird über "SalEndDialog()" die Dialog Box geschlossen.

2.1.3 Tabellenmodus

Der Tabellenmodus läßt sich dadurch charakterisieren, daß in einem vorgegebenen Bereich (der einzelne Zeilen enthält) immer mehr als ein Datensatz sichtbar ist, wobei Felder gleichen Inhalts immer untereinander angeordnet sind, so daß der Eindruck einer Liste entsteht. Die folgende Abbildung zeigt ein für die Darstellung von Kundendaten verwendetes Child Table Objekt (alternativ ist es auch möglich, ein Top-Level Table Objekt zu benutzen).

Neben den angezeigten Spalten für Kundennummern, Namen und Vornamen der Kunden ist eine auf dem Bildschirm unsichtbare Spalte für die "ROWID" enthalten. Rechts neben den Spalten sind Push Buttons zu sehen, die zur Steuerung verwendet werden (alternativ könnte die Steuerung auch über Menüpunkte erfolgen).

Abb. 2.13: Tabellenmodus für Daten aus der Tabelle Kunden.



Der Tabellenmodus ist immer dann von Vorteil, wenn man sich einen Überblick über bestehende Daten verschaffen möchte. Dadurch, daß immer mehrere Sätze zu sehen sind, ist es einfach, auch große Datenmengen anzuzeigen und zu untersuchen.

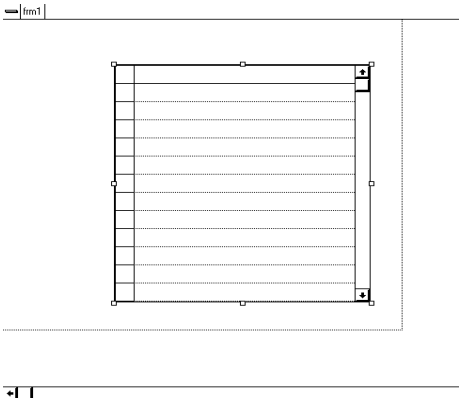
Von Nachteil ist der Modus dann, wenn viele Spalten anzuzeigen sind, da dann die Spalten vor dem Betrachter seitlich verschoben werden (allerdings besteht die Möglichkeit, einen Teil der Spalten am linken Bildschirmrand zu fixieren, so daß z.B. die Kundennummer und der Name des Kunden immer sichtbar sind, s. hierzu die Funktion "SalTblSetLockedColumns()").

Häufig findet eine Kombination der beiden Anzeigemodi (Tabelle und Einzelsatz) statt, z.B. um bei angezeigter Tabelle einen neuen Satz im Einzelsatzmodus einzugeben (vgl. dazu auch den vorstehenden Abschnitt). Die Kombination von Einzelsatz und Tabelle ist insbesondere dann sinnvoll, wenn Daten aus zwei Tabellen, die miteinander in Beziehung stehen, in *einem* Formular angezeigt werden sollen; so z.B. die Daten eines Lieferanten (im Einzelsatzmodus) und die Übersicht der von diesem Lieferanten gelieferten Artikel (in Tabellenform); man spricht dabei von Master-Detail-Beziehungen. Hier soll der Tabellenmodus zunächst nur mit Daten aus einer einzigen Tabelle vorgestellt werden, die Anzeige einer Master-Detail-Beziehung findet später statt.

2.1.3.1 Tabelle in frmMain einbinden

In der folgenden Abbildung ist ein Child Table Objekt in das Formular "frmMain" eingebunden dargestellt. Die Tabelle, deren Namen hier "tbl1" lautet, enthält zunächst noch keine Spalten.

Abb. 2.14: Eingefügte Tabelle, ohne Spalten.



Es sollen vier Spalten eingefügt werden. Die folgende Übersicht zeigt die Spaltennamen und den dafür vorgesehenen Inhalt.

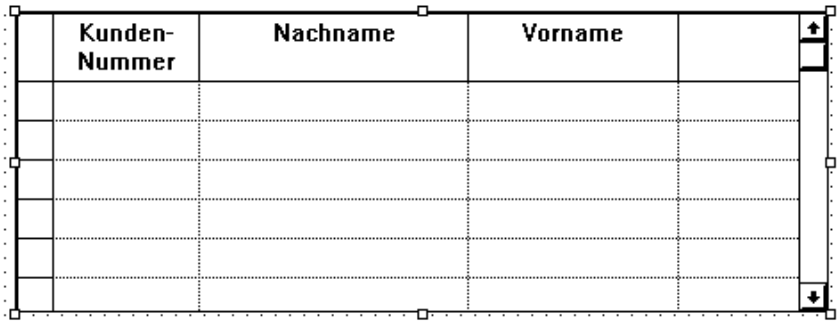
Spaltenname	Inhalt
colROWID	ROWID der Tabelle, von der Datenbank geliefert
colKdnr	Kundennummer
colName	Name
colVorname	Vorname

Beachten Sie folgende Punkte beim Einfügen der Spalten in die Tabelle:

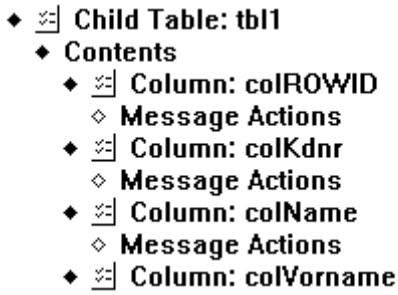
- Änderungen der Objekte sind mit dem Customizer vorzunehmen.
- SQLWindows schlägt als Name der Spalten "col1", "col2" usw. und stets den Datentyp String vor. Sie müssen die Namen aller Spalten und den Typ bei der Spalte für die Kundennummer ändern (Typ: "Number").
- Ändern Sie jeweils die Überschrift der Spalten (vorgeschlagen ist immer "untitled"); Sie können einen Zeilenumbruch in der Überschrift mit Strg+Enter einfügen.
- Passen Sie die Breite der Spalten an, indem Sie mit dem Mauszeiger auf die Trennungslinie zwischen den Spaltennamen gehen, dort klicken, halten und die Maus zur Seite ziehen.

Die fertige Tabelle sollte folgendes Aussehen haben:

Abb. 2.15: Tabelle mit vier eingefügten Spalten.



Nicht angezeigt (die Eigenschaft "Visible" wurde auf "No" umgestellt) ist die Spalte "colROWID". Die Übersicht aller eingefügten Spalten ergibt sich aus dem Inhalt des Outline-Fensters.



Beachten Sie, daß die *Message Actions der einzelnen Spalten keinen* Inhalt haben. Die für das Füllen der Tabelle mit Daten nötigen Schritte werden in den *Message Actions der Tabelle* definiert!

2.1.3.2 Anzeigen von Datensätzen

Um die Datensätze aus der Kundentabelle in dem Child Table Objekt anzeigen zu lassen, müssen Sie die Funktion "SalTblPopulate()" mit der einer Sql-Anweisung ("select...") aufrufen.

Diese Funktion "SalTblPopulate()" hat folgende allgemeine Syntax:

SalTblPopulate(hWndTbl, hSql, strSelect, nMethod)

Die in Klammern enthaltenen Parameter bedeuten folgendes:

- **hWndTbl** ist der Window Handle (oder der Name) des Tabellen Objekts, hier: "tbl1".
- **hSql** ist der Sql-Handle, der für das Lesen der Daten verwendet wird. Dieser Handle muß in der Anwendung als Variable definiert werden:

```
◆ Variables
  ◇ Sql Handle: hSql
```

Außerdem muß über diesen Handle mittels der Funktion "SqlConnect()" eine Verbindung zur Datenbank "FIRMA" hergestellt werden, üblicherweise am Anfang der Anwendung:

```
◆ Application Actions
  ◆ On SAM_AppStartup
    ◇ Set SqlDatabase = 'FIRMA'
    ◇ Call SqlConnect(hSql)
  ◆ On SAM_AppExit
    ◇ Call SqlDisconnect( hSql )
```

In dem Ausschnitt ist auch definiert, daß beim Beenden der Anwendung ("On SAM_AppExit") die Verbindung wieder getrennt werden soll.

- **strSelect** ist die eigentliche Select-Anweisung, eine Zeichenfolge (deshalb der Präfix "str"). Hier kann anstelle der getippten Sql-Anweisung eine zuvor definierte Variable verwendet werden, die den Text der Sql-Anweisung beinhaltet.

- **nMethod** ist die Methode, mit der die Tabelle mit Daten zu füllen ist. Dafür stehen mehrere Verfahren zur Verfügung, häufig wird hier eine dafür im System vordefinierte Konstante mit dem Namen "TBL_FillNormal" verwendet (die anderen Möglichkeiten, eine Tabelle mit Daten zu füllen, finden sich im Hilfesystem bzw. in den mitgelieferten Handbüchern).

Wie bereits oben erwähnt, wird das Füllen der Tabelle mit Daten in den Message Actions der Tabelle definiert. Diese Sektion erhält folgenden Inhalt:

```
◆ Child Table: tbl1
  ◆ Contents
    ◆ Column: colROWID
    ◆ Column: colKdnr
    ◆ Column: colName
    ◆ Column: colVorname
  ◇ Functions
  ◇ Window Variables
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Call SalTblPopulate(tbl1,
         hSql, 'select ROWID, KUNDENNUMMER, NAME, VORNAME
        from KUNDEN
        order by KUNDENNUMMER
         into :colROWID, :colKdnr, :colName, :colVorname',
        TBL_FillNormal )
```

Nach Empfang der Nachricht "On SAM_Create" wird die Funktion "SalTblPopulate()" aufgerufen. Der Funktionsaufruf enthält den Namen der Tabelle ("tbl1"), den Namen des Handles ("hSql"), die eigentliche Select-Anweisung, um den Zusatz "order by KUNDENNUMMER" erweitert, um eine Sortierung der Datensätze nach der Kundennummer zu erhalten, sowie die Angabe, wie die Tabelle zu füllen ist ("TBL_FillNormal").

Das genügt, um die Datensätze aus der Tabelle KUNDEN in einem Tabellenobjekt anzeigen zu lassen. Nach dem Start bietet sich folgendes Bild (die angezeigten Datensätze müssen nicht identisch sein).

Abb. 2.16: Im Table Objekt "tbl1" angezeigte Datensätze aus der Tabelle "Kunden".

	Kunden-Nummer	Nachname	Vorname	
	10014	Dürr	Michel	
	10015	Klirr	Werner	
	10019	Meyer	Mitse	

In den folgenden Abschnitten soll erläutert werden, wie die angezeigten Daten editiert werden können. Dabei wird zum einen dargestellt, wie neue Daten eingegeben und zum anderen, wie vorhandene Daten geändert bzw. gelöscht werden können.

2.1.3.3 Editieren von Datensätzen

Wie bereits bei der Vorstellung der Einzelsatzdarstellung wird auch hier getrennt nach

- Eingeben neuer Datensätze,
- Ändern bzw.
- Löschen vorhandener Datensätze.

Auch in Ihren Anwendungen sollten Sie diese einzelnen Fälle getrennt codieren.

2.1.3.3.1 Eingeben von Datensätzen

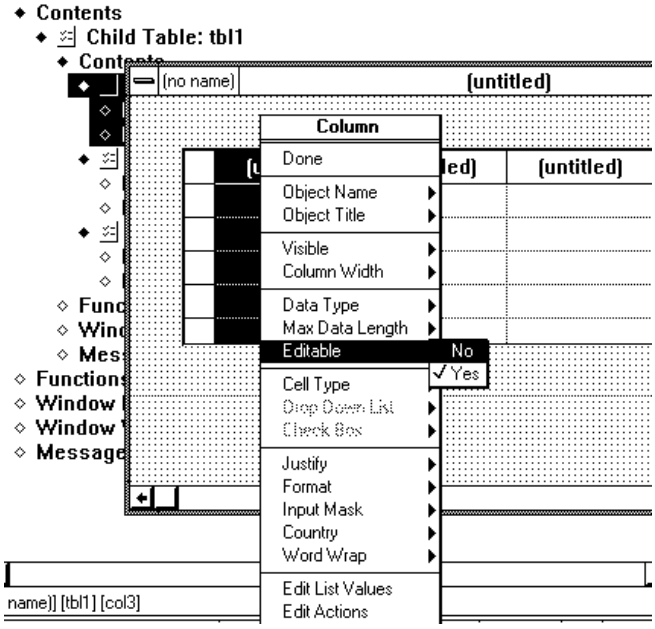
Das Eingeben von neuen Datensätzen soll hier unter Verwendung des oben eingefügten Child Table Objekts erläutert werden. Dabei soll die Tabelle zur Neueingabe von Datensätzen waagerecht geteilt werden, so daß vorhandene Datensätze im oberen Teil angezeigt werden und die neu einzugebenden Datensätze im unteren Fensterteil erscheinen.

Anmerkung zu der Vergabe einer neuen Kundennummer

Bei Neueingabe eines Kunden soll der Benutzer mit dem Feld "Kundennummer" nicht in Berührung kommen. Dieses Feld soll für den Benutzer gesperrt werden, damit er den dort angezeigten Inhalt nicht ändern kann. Es muß außerdem gewährleistet sein, daß keine doppelten Kundennummern vorkommen. Deshalb ist es bei der Neueingabe von Kunden besser, dem Anwendungsprogramm die Aufgabe zu übertragen, eine neue Kundennummer vorzuschlagen. Im einfachsten Fall nimmt es dazu die Tabelle, sucht mit einer "Select max"-Anweisung den höchsten Wert in der Spalte Kundennummer und addiert 1 dazu (wie bereits in Kapitel 2.1.2.4.3 erläutert wurde). Sperren Sie deshalb die Spalte mit der Kundennummer, so daß der Benutzer den Inhalt nicht ändern kann (über den Customizer, Editable: No).

Dieses Beispiel läßt sich auf viele Situationen übertragen, beispielsweise bei einer Auftragserfassung, bei der das Anwendungsprogramm selbst eine neue Auftragsnummer vorschlägt.

Abb. 2.17: Sperren einer Tabellenspalte für die Bearbeitung.



Im folgenden wird erläutert werden, wie Sie einen neuen Datensatz einfügen und eine neue Kundennummer für den neu anzulegenden Kunden erhalten.

Einfügen eines neuen (leeren) Datensatzes

Das Einfügen eines neuen Datensatzes soll über einen Push Button ausgelöst werden. Fügen Sie dazu einen Push Button in das Formular ein und beschriften Sie ihn mit dem Titel "neuer Satz".

Um die Tabelle in zwei Hälften zu teilen, muß

- die Tabelle waagerecht geteilt werden,
- ein neuer leerer Datensatz eingefügt und
- die Schreibmarke in die erste Spalte der Tabelle gesetzt werden.

Hierfür wird jeweils eine Funktion verwendet, sie sind in der folgenden Übersicht vorgestellt.

Funktionsname	Inhalt
SalTblDefineSplitWindow()	Teilt das Tabellenfenster in zwei Hälften
SalTblInsertRow()	Fügt in die Tabelle einen neuen leeren Satz ein
SalTblSetFocusCell()	Setzt die Schreibmarke in das erste Feld

Bei dem Aufruf der Funktion **SalTblDefineSplitWindow()** muß angegeben werden, welches Tabellenobjekt zu teilen ist ("tbl1"), wieviel Zeilen im unteren Fenster darzustellen sind (hier: 3) und ob der Benutzer die Grenzlinie zwischen den beiden Fensterhälften verschieben darf (hier soll das erlaubt sein: "TRUE").

Der Funktionsaufruf erhält dadurch folgendes Aussehen:

```
◇ Call SalTblDefineSplitWindow( tbl1, 3, TRUE )
```

Nach der Teilung des Fensters soll ein neuer Datensatz mit der Funktion **SalTblInsertRow()** eingefügt werden. Diese Funktion erhält den Namen des Tabellenobjekts ("tbl1") und die Anzahl der einzufügenden Zeilen. Man benutzt nicht eine selbst definierte Zahl, sondern üblicherweise eine Konstante, die von SQLWindows hierfür vorgesehen ist und die jeweils die aktualisierte Nummer der Zeile enthält: **TBL_MinRow**. Da die Nummer der Zeile weiterverwendet wird (um die Schreibmarke in die "richtige" Zeile zu stellen, speichert man den von der Funktion SalTblInsertRow() zurückgegebenen Wert in einer numerischen Variablen ab, die hier den Namen **nNewRow** erhält.

```
◆ Variables
  ◇ Number: nNewRow
```

In dieser Variablen wird der Rückgabewert der aufgerufenen Funktion gespeichert:

```
◇ Set nNewRow=SalTblInsertRow(tbl1, TBL_MinRow)
```

Jetzt kann die Funktion **SalTblSetFocusCell()** dazu benutzt werden, die Schreibmarke in die erste zu editierende Spalte zu setzen (dabei ist aber zu beachten, daß die erste Spalte, die der Benutzer editieren soll, *nicht* die erste Spalte des Table Objekts ist, da die erste Spalte die "ROWID" enthält und nicht sichtbar ist; die zweite Spalte aber, die "Kundennummer", soll nicht editierbar sein). Somit ergibt sich als erste Spalte die Spalte mit dem Namen der Kunden, ihr Name ist "colName".

Außerdem kann angegeben werden, wieviel Zeichen des Feldinhaltes markiert werden sollen (mit einer Anfangs- und einer Endeposition). Da hier in einem neuen Datensatz keine Zeichen enthalten sind und nichts markiert werden soll, wird jeweils "-1" angegeben, d.h. es soll nichts markiert werden. Der Funktionsaufruf hat damit folgendes Aussehen:

```
◇ Call SalTblSetFocusCell(tbl1, nNewRow,colName,-1, -1)
```

Die Message Actions Sektion des Push Buttons hat nun insgesamt folgenden Inhalt:

```
◆ Pushbutton: pb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalTblDefineSplitWindow( tbl1, 3, TRUE )
      ◇ Set nNewRow=SalTblInsertRow(tbl1, TBL_MinRow)
      ◇ Call SalTblSetFocusCell(tbl1, nNewRow,colName,-1, -1)
```

Damit können Sie das Programm bereits starten. Es fehlen jedoch die vorgeschlagene neue Kundennummer und die Möglichkeit, die eingegebenen Datensätze abzuspeichern.

Vorschlag einer neuen Kundennummer

Um eine neue Kundennummer vorzuschlagen, muß eine Sql-Anweisung ausgeführt werden, die den Wert der höchsten vorhandenen Kundennummer ermittelt und diese Zahl um 1 erhöht. Diese Sql-Anweisung wird mit der Funktion "SqlImmediate()" ausgeführt, der von ihr ermittelte Wert wird in die Spalte für die Kundennummer geschrieben:

```
◇ Call SqlImmediate( 'select max(KUNDENNUMMER)+1 into :colKdnr from Kunden' )
```

Nun sind zwei verschiedene Situationen denkbar:

- Der Benutzer gibt den ersten neuen Satz ein, dann ermittelt die Sql-Anweisung die richtige neue Kundennummer aus der Tabelle mit den Kundendaten.

- Der Benutzer gibt einen zweiten (dritten usw.) Datensatz ein. Da die neu eingegebenen Daten zu diesem Zeitpunkt aber noch nicht auf der Festplatte gespeichert sind, liefert die Sql-Anweisung für alle neuen Sätze, die nacheinander eingegeben werden, immer den gleichen Wert.

Um zu vermeiden, daß es zu Problemen kommt, wird nach dem *ersten* Aufruf der Funktion "SqlImmediate()" der von ihr zurückgelieferte Wert in eine zusätzliche Variable ("nNeueNummer", sie muß zuvor unter "Variables" definiert werden) gespeichert.

◇ Number: nNeueNummer

Diese Variable hat ursprünglich den Inhalt 0, der ihr nach Aufbau der Tabelle zugewiesen wurde.

```
◆ Child Table: tbl1
  ◆ Contents
    ◆ Column: colROWID
    ◆ Column: colKdnr
    ◆ Column: colName
    ◆ Column: colVorname
  ◇ Functions
  ◇ Window Variables
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Set nNeueNummer = 0
```

Bei jedem weiteren Datensatz, der neu eingegeben wird, wird nun geprüft, ob der Inhalt der Variablen "nNeueNummer" 0 ist (das ist nur bei dem ersten neu eingegebenen Datensatz der Fall), falls der Inhalt nicht 0 ist (d.h. es wurde bereits ein Datensatz eingegeben), dann wird der Inhalt der Variablen nNeueNummer um 1 erhöht und dieser Wert (das ist dann die neue richtige Kundennummer) in die Spalte "colKdnr" geschrieben. Dabei entsteht folgende Konstruktion:

```
◆ Pushbutton: pb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalTblDefineSplitWindow( tbl1, 3, TRUE )
      ◇ Set nNewRow=SalTblInsertRow(tbl1, TBL_MinRow)
      ◆ If nNeueNummer = 0                    ! d.h. erster Aufruf
        ◇ Call SqlImmediate( 'select max(KUNDENNUMMER)+1 into :colKdnr from Kunden' )
        ◇ Set nNeueNummer = colKdnr
      ◆ Else ! folgende Sätze, nicht der erste
        ◇ Set nNeueNummer = nNeueNummer + 1
        ◇ Set colKdnr = nNeueNummer
      ◇ Call SalTblSetFocusCell(tbl1, nNewRow,colName,-1, -1)
```

Beachten Sie, daß mittels einer If-Struktur die zwei oben genannten Fälle getrennt werden ("If nNeueNummer = 0"). Damit ist gewährleistet, daß auch mehrere nacheinander eingegebene Kunden eine eindeutige Kundennummer erhalten.

Abspeichern der neu eingegebenen Datensätze

Hierfür soll ein zusätzlicher Push Button eingefügt werden. Zusätzlich müssen einige Variable definiert werden, die für eine bessere Übersicht sorgen und Änderungen erleichtern. Diese Variablen enthalten die Sql-Anweisungen, hier zunächst eine Sql-Anweisung mit "Insert", um Datensätze in die Datenbank einzufügen ("strInsertKunden"), und eine weitere für das Füllen der Tabelle, um die Daten erneut anzuzeigen ("strPopulateKunden").

Da in den folgenden Abschnitten zusätzliche Variablen für das Ändern vorhandener Daten ("Update") und das Löschen von Datensätzen ("Delete") verwendet werden, wird hier eine Übersicht aller dazu nötigen Variablen gezeigt:

```
◆ Variables
  ◇ String: strInsertKunden
  ◇ String: strPopulateKunden
  ◇ String: strUpdateKunden
  ◇ String: strDeleteKunden
```

Der folgende Programmausschnitt zeigt den Inhalt der Message Actions des Push Buttons für die Sicherung von neu eingegebenen Datensätzen.

```
◆ On SAM_Click
  ◇ ! hier werden Strings für die Sql-Anweisungen benutzt
  ◇ Set strInsertKunden =
     'insert into KUNDEN
    (KUNDENNUMMER, NAME, VORNAME)
     values
    (:colKdnr, :colName, :colVorname)'
  ◇ Set strPopulateKunden =
     'select ROWID, KUNDENNUMMER, NAME, VORNAME
    from KUNDEN order by NAME
     into :colROWID, :colKdnr, :colName, :colVorname'
  ◇ ! die Zweiteilung des Fensters wird aufgehoben
  ◇ Call SalTblDefineSplitWindow( tbl1, 0, TBL_Split_Adjustable )
  ◇ ! jetzt muß der Satz/die Sätze in die Datenbank geschrieben werden
  ◇ Call SqlPrepare(hSql, strInsertKunden)
  ◇ Call SalTblDoInserts( tbl1, hSql, TRUE )
  ◇ ! die Änderungen werden fixiert
  ◇ Call SqlImmediate('commit')
  ◇ ! die Datensätze werden erneut gelesen und angezeigt
  ◇ Call SalTblPopulate(tbl1, hSql,strPopulateKunden , TBL_FillNormal )
```

Die Begriffe bedeuten im einzelnen:

- Zunächst werden nach dem Klick mit der Maus auf dem Push Button die Variablen **strInsertKunden** bzw. **strPopulateKunden** mit den Select-Anweisungen gefüllt.

- Die Funktion **SalTblDefineSplitWindow()**, die bereits oben benutzt wurde, um das Fenster zu teilen, wird hier erneut aufgerufen, um die Teilung des Fensters aufzuheben.

- Mit **SqlPrepare()** wird die Sql-Anweisung vorbereitet, die Funktion **SalTblDoInserts()** schreibt die angehängten Datensätze in die Tabelle "Kunden" auf die Platte.

- **SqlImmediate('commit')** fixiert die Änderungen, so daß sie nicht mehr zurückgenommen werden können.

- **SalTblPopulate()** liest die Daten nach der Übergabe an die Datenbank erneut ein und zeigt sie an.

Die folgende Abbildung zeigt neu eingegebene Datensätze in einem Table Objekt.

Abb. 2.18: Neu eingegebene Datensätze.

	Kunden-Nummer	Nachname	Vorname
	10014	Dürr	Michel2
	10015	Klirr	Werner2
	10019	Meyer	Mitse
	10020	Miller	Lite
	10021	Buddi	Liteb
→	10059	First	
→	10060	Second	
→	10061	Third	

Beachten Sie, daß dieses Verfahren nur sinnvoll in Anwendungen verwendet werden kann, die nicht in einem Netzwerk eingesetzt werden. Hinweise, wie Sie eine Erfassung von Daten im Netz codieren müssen, finden Sie in Kapitel 3.2.

2.1.3.3.2 Ändern von Datensätzen

Der Benutzer hat grundsätzlich die Möglichkeit, die in einer Tabelle angezeigten Datensätze zu ändern (aus Sicht des Programmierers muß dazu zunächst nichts codiert werden). Das Ändern der Daten muß nur durch das Klicken in einer Zelle (d.h. dem Schnittpunkt einer Zeile und einer Spalte) eingeleitet werden. Dieses Ändern von angezeigten Inhalten löst ein internes Signal aus, das dazu führt, daß die Zeile, in der eine Änderung stattfand, markiert wird, so wie dies die folgende Abbildung zeigt.

Abb. 2.19: Geänderte Inhalte einer Zeile.

	Kunden-Nummer	Nachname	Vorname
	10014	Dürr	Michel
	10015	Klirr	Werner
✓	10019	Meyer2	Mitse
	10020		

Der Programmierer kann nun anhand dieser Markierung genau diejenigen Datensätze herauspicken, die diese Markierung ("ROW_Edited") haben, alternativ können auch alle Datensätze gesichert werden, so als wären alle verändert worden (bei großen Datenmengen oder einem Einsatz im Netz wäre dies ungeschickt, da viele Daten über das Netz geschickt werden, obwohl eventuell nur wenige Datensätze verändert wurden).

In dem folgenden Beispiel werden nur die Datensätze gesichert, die geändert wurden. Hierzu wird die Variable **strUpdateKunden** benutzt, in der die Sql-Update-Anweisung gespeichert ist. Beachten Sie, daß der dargestellte Programmcode dem im vorigen Kapitel dargestellten sehr ähnlich ist.

```
◆ Pushbutton: pb2
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Set strUpdateKunden = 'update KUNDEN set
        VORNAME = :colVorname,
        NAME = :colName,
        KUNDENNUMMER = :colKdnr
         where ROWID = :colROWID'
      ◇ Set strPopulateKunden = 'select ROWID, KUNDENNUMMER, NAME, VORNAME
                  from KUNDEN order by NAME
                   into :colROWID, :colKdnr, :colName, :colVorname'
      ◇ Call SalTblDefineSplitWindow( tbl1, 0, TBL_Split_Adjustable )
      ◇ Call SqlPrepare(hSql,strUpdateKunden)
      ◇ Call SalTblDoUpdates( tbl1, hSql, TRUE )
      ◇ Call SqlImmediate('commit')
      ◇ Call SalTblPopulate(tbl1, hSql,strPopulateKunden , TBL_FillNormal )
```

Zur Erläuterung:

- Zuerst werden die Variablen **strUpdateKunden** und **strPopulateKunden** mit den SQL-Anweisungen gefüllt.
- **SalTblDefineSplitWindow()** wird - wie bereits oben - zum Entfernen der Fensterteilung benutzt.
- **SqlPrepare()** wird benutzt, um die Sql-Anweisung, die in der Variablen "strUpdateKunden" enthalten ist, vorzubereiten.

- **SalTblDoUpdates()** schreibt die geänderten Datensätze zurück in die Kundentabelle.
- **SqlImmediate('commit')** wird verwendet, um die Änderungen in der Datenbank zu fixieren.
- **SalTblPopulate()** schließlich liest die Daten erneut ein und zeigt sie in der Tabelle "tbl1" an.

2.1.3.3.3 Löschen von Datensätzen

Das Löschen von Datensätzen wird hier so ermöglicht, daß der Benutzer auf dem Rand vor der ersten angezeigten Spalte klicken und anschließend die Entf-Taste drücken muß (alternativ wäre es möglich, einen weiteren Push Button hierfür einzusetzen). Damit setzt er das Attribut "ROW_MarkDeleted", anhand der SQLWindows erkennen kann, welche Sätze gelöscht werden müssen. Das tatsächliche Löschen der Datensätze wird dann wieder durch das Drücken eines Push Buttons ausgelöst.

Die folgende Abbildung zeigt einen zur Löschung vorgesehenen Datensatz.

Abb. 2.20: Datensatz, zur Löschung vorgesehen.

	Kunden-Nummer	Nachname	Vorname
	10014	Dürr2	Michel
	10015	Klirr	Werner

Um Datensätze, die zur Löschung vorgesehen sind, aus der Tabelle zu entfernen, ist folgende Codierung notwendig:

```
◆ Pushbutton: pb2
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Set strDeleteKunden =  'delete from KUNDEN where ROWID = :colROWID'
      ◇ Set strPopulateKunden = 'select ROWID, KUNDENNUMMER, NAME, VORNAME
                 from KUNDEN order by NAME
                  into :colROWID, :colKdnr, :colName, :colVorname'
      ◇ Call SalTblDefineSplitWindow( tbl1, 0, TBL_Split_Adjustable )
      ◇ Call SqlPrepare(hSql, strDeleteKunden)
      ◇ Call SalTblDoDeletes(tbl1, hSql, ROW_MarkDeleted)
      ◇ Call SqlImmediate('commit')
      ◇ Call SalTblPopulate(tbl1, hSql,strPopulateKunden , TBL_FillNormal )
```

Zur Erläuterung:

- Zunächst wird die Variable **StrDeleteKunden** mit der Sql-Delete Anweisung gefüllt, die Variable strPopulateKunden dient dem erneuten Lesen und Anzeigen der Daten.

- **SalTblDefineSplitWindow()** setzt das Fenster in den ungeteilten Zustand zurück.
- **SqlPrepare()** bereitet die Sql-Anweisung für das Löschen der Daten vor.
- **SalTblDoDeletes()** führt die Löschung durch.
- **SqlImmediate('commit')** fixiert die Änderungen
- **SalTblPopulate()** liest alle Datensätze und zeigt sie erneut an.

Damit Sie für spätere Anwendungen alle Möglichkeiten des Editierens (neu einfügen, ändern, löschen) an einer Stelle zentral codieren können, können Sie alle vorstehenden Aktionen zusammenfassen und z.B. nach Drücken eines Push Buttons mit der Aufschrift "sichern" durchführen lassen. Der folgende Programmcode faßt alle Aktionen zusammen, die nach dem Drücken eines "sichern"-Pushbuttons ausgeführt werden könnten, dabei sind die Insert-, Update- und Delete-Anweisungen berücksichtigt.

Zunächst werden die Variablen mit Ihren Inhalten gefüllt:

```
◆ Pushbutton: pb2
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ ! Variablen mit Inhalten füllen
      ◇ Set strInsertKunden = 'insert into KUNDEN (KUNDENNUMMER, NAME, VORNAME)
         values (:colKdnr, :colName, :colVorname)'
      ◇ Set strUpdateKunden = 'update KUNDEN set
        VORNAME = :colVorname,
        NAME = :colName,
        KUNDENNUMMER = :colKdnr
         where ROWID = :colROWID'
      ◇ Set strDeleteKunden =  'delete from KUNDEN where ROWID = :colROWID'
      ◇ Set strPopulateKunden = 'select ROWID, KUNDENNUMMER, NAME, VORNAME
                 from KUNDEN order by NAME
                  into :colROWID, :colKdnr, :colName, :colVorname'
```

Unmittelbar daran schließen sich die eigentlichen Aktionen an:

```
◇ ! Teilung des Fensters aufheben
◇ Call SalTblDefineSplitWindow( tbl1, 0, TBL_Split_Adjustable )
◇ ! Neue Datensätze einfügen
◇ Call SqlPrepare(hSql, strInsertKunden)
◇ Call SalTblDoInserts( tbl1, hSql, TRUE )
◇ ! Veränderte Datensätze sichern
◇ Call SqlPrepare(hSql,strUpdateKunden)
◇ Call SalTblDoUpdates( tbl1, hSql, TRUE )
◇ ! zur Löschung vorgesehene Sätze entfernen
◇ Call SqlPrepare(hSql, strDeleteKunden)
◇ Call SalTblDoDeletes(tbl1, hSql, ROW_MarkDeleted)
◇ !
◇ ! jetzt muß mittels einer commit-Anweisung die Änderung fixiert werde
◇ Call SqlImmediate('commit')
◇ ! und schließlich werden die Daten werden erneut eingelesen
◇ Call SalTblPopulate(tbl1, hSql,strPopulateKunden , TBL_FillNormal )
```

Durch die Verwendung der oben genannten Variablen können Sie deutlich kürzere Code-Teile schreiben. Außerdem sind in nahezu allen zukünftigen Datenbank-Anwendungen sehr ähnliche programmtechnische Probleme zu lösen, so daß Sie nach einer einmaligen Codierung die Teile leicht an die neuen Verhältnisse anpassen können.

2.1.4 Ausgeben von Datensätzen mit einem Report

Viele Benutzer haben nach den ersten Gehversuchen in SQLWindows noch Schwierigkeiten mit der Programmierung von Report-Ausgaben. Dies liegt daran, daß hier etwas mehr zu codieren ist als z.B. in der xBase-Welt. Es ist zwar insgesamt nicht besonders schwierig, eine Druckausgabe zu programmieren, doch bieten sich mehr Möglichkeiten, Fehler zu machen, insbesondere dadurch, daß ein Report-Template, Funktionen ("SalReport*") und Nachrichten ("SAM_Report*") zu verarbeiten sind.

Das erwähnte Report-Template, eine Datei die Platzhalter für die zu druckenden Informationen bereitstellt, wird mit dem Programm **Report Windows** angefertigt, das über den Programm-Manager oder aus SQLWindows heraus gestartet werden kann (auch aus Quest heraus kann dieses Programm aufgerufen werden).

Abb. 2.21: Report Windows Symbol.

Die mit Report Windows anzulegenden Templates können auf zwei Arten erzeugt werden:

- Erstens unter direkter Verwendung des Programms Report Windows;

- oder zweitens (indirekt) zur Laufzeit einer mit SQLWindows erstellten Anwendung über die Funktion "SalReportCreate()".

Beide Verfahren werden vorgestellt.

Bezüglich der Datenherkunft unterscheidet SQLWindows bei der Arbeit mit Reports **zwei Fälle**, die direkt mit den oben genannten Funktionen zusammenhängen:

- Im ersten Fall müssen die Daten zuerst vom Datenträger gelesen werden (s. hierzu Kap. 2.1.4.1, dabei wird erläutert, wie ein Report-Template selbst erstellt wird);

- im zweiten Fall sind die Daten bereits (mit einer SQL-Select Anweisung) geladen worden und in einer Tabelle auf dem Bildschirm sichtbar (s. hierzu Kap. 2.1.4.2, dabei wird der Report-Template von SQLWindows selbst zur Laufzeit erzeugt).

Die erste Variante tritt am häufigsten auf, erfordert mehr Codierung, die zweite Variante ist einfacher zu programmieren, aber weniger flexibel.

Es stehen grundsätzlich drei Funktionen zur Verfügung, die in Verbindung mit Reports benutzt werden. Die folgende Übersicht nennt diese drei Funktionen, die nachfolgend verwendet werden.

Funktionsname	Zweck
SalReportCreate()	Legt eine Report-Datei (Template) mit Ausgabefeldern an.
SalReportView()	Zeigt eine Report-Datei (Template) in der Druckvorschau.
SalReportPrint()	Druckt eine Report-Datei (Template).

Die Funktion **SalReportCreate()** erlaubt es, einen Report-Template zur Laufzeit, z.B. durch Drücken eines Push Buttons zu erzeugen. Diese Variante kann dann in Anwendungen eingebaut werden, wenn der Benutzer Daten drucken möchte, die dem Entwickler unbekannt sind (so können beispielsweise mit einer Select-Anweisung die Daten irgendeiner Tabelle verarbeitet werden).

Die Funktion **SalReportView()** erlaubt das Anzeigen von zu druckenden Daten auf dem Bildschirm in einer Seitenvorschau. Mittels verschiedener Push Buttons, die zu den Daten angezeigt werden, kann in den angezeigten Daten geblättert oder aus diesem Fenster heraus gedruckt werden.

SalReportPrint() ist die Funktion, die für den Druck von Daten verwendet wird.

2.1.4.1 Daten von der Platte lesen und anzeigen

Das Template soll in diesem Beispiel durch den Benutzer erstellt, die Daten sollen vom Datenträger gelesen werden. Im folgenden wird nur ein einziges Feld in den Report aufgenommen, um die Erläuterungen kurz zu halten.

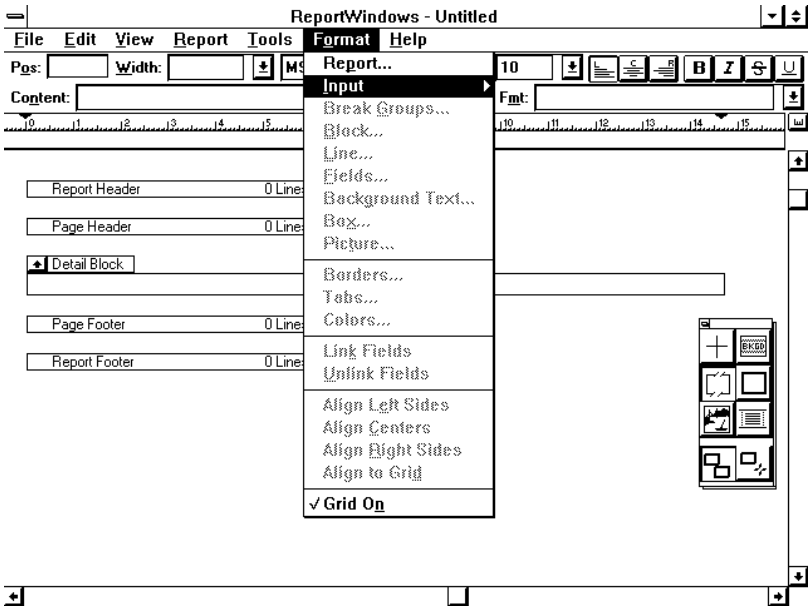
2.1.4.1.1 Template anlegen

Das Template wird mit dem Programm "Report Windows" angelegt. Das Programm befindet sich in der gleichen Programmgruppe wie SQLWindows. Sie können es über den Programm Manager starten oder innerhalb von SQLWindows den Push Button für dessen Aufruf anklicken.

Abb. 2.22: Report Windows Push Button in SQLWindows (oben rechts).

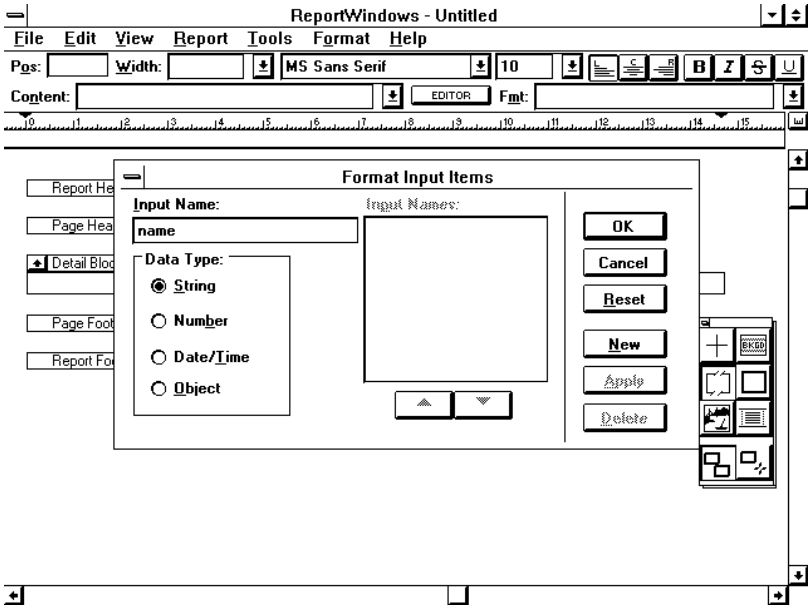
Nachdem Report Windows gestartet ist, klicken Sie in Report Windows auf den Menüpunkt Format, um das Format der Input-Variablen (Eingabe-Variablen) festzulegen, achten Sie dabei besonders auf die Groß- bzw. Kleinschreibung.

Abb. 2.23: Report Windows, Menu Format, Input Item.



Wählen Sie aus dem Menüpunkt Format die Option "Input Items", geben Sie den Namen der Eingabevariablen an ("name") stellen Sie sicher, daß der Datentyp "String" eingestellt ist.

Abb. 2.24: Report Windows, Input Item "name" vom Typ String.

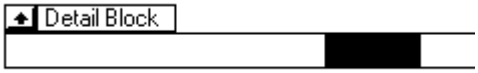


Bestätigen Sie durch Klicken auf "OK" (falls Sie später zusätzliche Input Items angeben wollen, müssen Sie in diesem Fenster auf den Push Button "New" drücken und andere Inputs Items definieren).

Nachdem das oben angezeigte Fenster geschlossen ist, führen Sie den Mauszeiger in die lange graue Zeile des darauf angezeigten Templates, die mit *Detail Block* überschrieben ist. Ihr Mauszeiger verändert sich jetzt zu einem Rechteck. Das deutet an, daß mit einem Mausklick jetzt Felder eingefügt werden können (Sie können durch Klicken mit der rechten Maustaste die anderen möglichen Einfügeoptionen, z.B. neue Zeile, Bild, Hintergrundtext, anzeigen lassen).

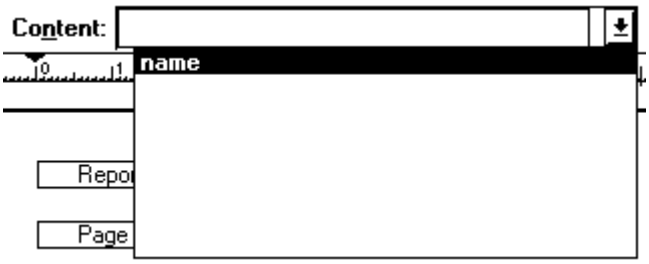
Fügen Sie ein Feld ein, indem Sie einmal (mit der linken Maustaste) klicken.

Abb. 2.25: Detail Block mit eingefügtem (leerem) Datenfeld.



Danach öffnen Sie die Combo Box rechts von dem Wort **Contents** am oberen Bildschirmrand. Dort finden Sie das von Ihnen definierte Input Item "name".

Abb. 2.26: Contents Combo Box mit Input Item.



Klicken Sie einmal auf das Wort "name", um dieses Input Item an das eben eingefügte (noch markierte) Feld zu binden. Jetzt schließen Sie den Report, das Template ist erstellt (speichern Sie unter dem Dateinamen "**LISTE.QRP**").

Im nächsten Abschnitt soll einem Push Button die Anzeige eines Reports in der Vorschau zugewiesen werden.

2.1.4.1.2 Anzeige in der Seitenvorschau

Um die Daten in der Seitenvorschau anzuzeigen, müssen einige separate Aktionen codiert werden. Hierzu wird dieser Abschnitt wie folgt gegliedert:

- Funktionen und Nachrichten.
- Push Button Aktionen (einfaches Verfahren).
- Verbessertes Verfahren durch die Benutzung zusätzlicher Variablen.
- Separates Form Window für die Druckvorschau.

Funktionen und Nachrichten

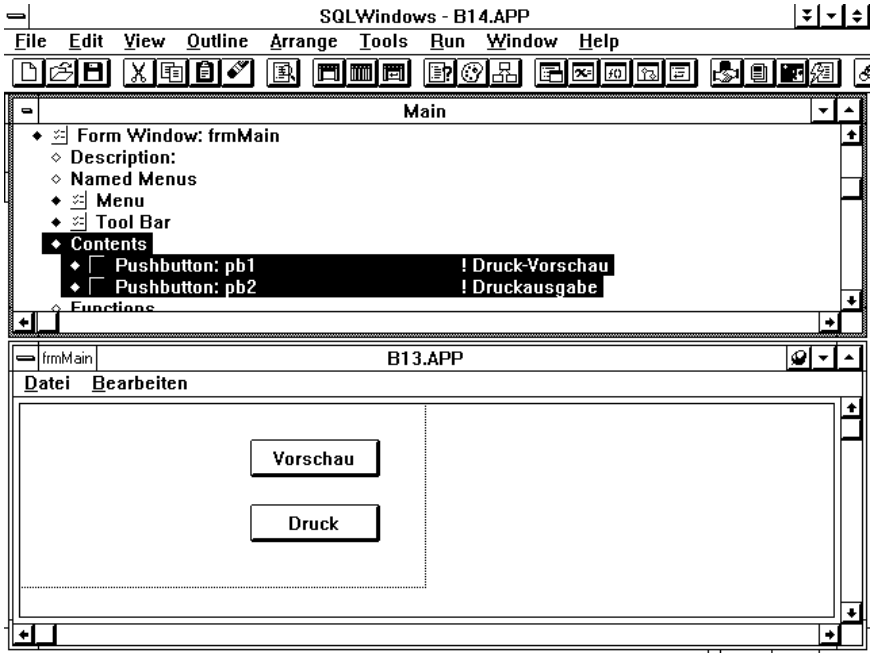
Um die Daten in der Vorschau anzuzeigen, müssen Sie die Funktion "SalReportView()" benutzen und die Nachrichten verwalten, die SQLWindows beim Lesen und Drucken der Daten verwendet (sie beginnen alle mit "SAM_Report*"). Dabei ist es von **größter Wichtigkeit**, diese Nachrichten an der *richtigen* Stelle zu verwalten. Prinzipiell kann zwar jedes Fenster diese Nachrichten verwalten, aber üblicherweise benutzen Sie das Fenster, in dem der Druck gestartet wird (hier also das Formular "frmMain").

Die folgende Übersicht faßt die Nachrichten zusammen, die Sie in Ihrem Code verarbeiten müssen.

Art der Nachricht	Zweck
SAM_ReportStart	Wird nur einmal bei Start des Reports gesendet. Hier wird üblicherweise die Funktion **SqlPrepare** aufgerufen, um die Daten mittels SQL-Anweisung zu lesen. Der unter dieser Nachricht angeordnete Programmcode wird nur einmal ausgeführt.
SAM_ReportFetchInit	Aufruf unmittelbar nach SAM_ReportStart. Falls die Daten aus einer Datenbank kommen, wird hier die Funktion **SqlExecute()** aufgerufen. Falls als Rückgabewert ein logisches "FALSE" zurückgegeben wird, erfolgt der Abbruch des Reports.
SAM_ReportFetchNext	Hier wird üblicherweise die Funktion SqlFetchNext() aufgerufen, um den nächsten anstehenden Datensatz zu lesen. Falls der Satz erfolgreich gelesen wurde, wird "TRUE" zurückgegeben, andernfalls (keine weiteren Sätze vorhanden) wird "FALSE" zurückgegeben.
SAM_ReportFinish	Wird gesendet, nachdem alle Datensätze gelesen wurden. Hier könnte z.B. der Push Button für die Druckausgabe wieder aktiviert werden.

Für das Beispiel soll eine Anwendung angefertigt werden, die aus einem Fenster ("frmMain") und zwei Push Buttons ("pb1" und "pb2") besteht. Der erste Push Button wird für die Vorschau der Daten benutzt, der andere für den eigentlichen Druck. Das Formular könnte folgendes Aussehen haben (vergewissern Sie sich, daß die Datenbank "FIRMA" bei dem Start der Anwendung geöffnet wird):

Abb. 2.27: Das Formular "frmMain" mit zwei eingefügten Push-Buttons.



Push Button Aktionen

Klicken Sie mit der rechten Maustaste auf den Push Button für die Vorschau. Um diesem Push Button die Druckvorschau mittels der Funktion "SalReportView()" zuzuordnen, wählen Sie die Option "Edit Actions" aus.

Wählen Sie aus den Outline Options zunächst "On SAM_Click" aus. Suchen Sie dann aus dem Fenster, in dem die Funktionen angezeigt werden, die Funktion "SalReportView()" aus und lassen Sie diese Funktion einfügen. Sie hat folgende Syntax (die einzelnen Optionen sind wegen besserer Lesbarkeit untereinander angeordnet:

```
◆ Message Actions
  ◆ On SAM_Click
    ◇ Call SalReportView(
      Window_Handle,
      Window_Handle,
      String,
      String,
      String,
      Number )
```

Die innerhalb der Klammern anzugebenden Parameter haben folgende Bedeutung bzw. folgenden Inhalt:

Call SalReportView(	Aufruf der Funktion.
Window_Handle,	Der Name des Fensters, das die SAM_Report* Nachrichten verarbeitet (hier **frmMain**).
Window_Handle,	Fenster für das Anzeigen der Vorschau (hier: **hWndNULL**, d.h. die Funktion bastelt sich selbst ein solches Fenster).
String,	Der Name des Report-Templates (hier: **LISTE.QRP**).
String,	Liste mit **Variablennamen**, aus denen die Daten für den Template zu holen sind (müssen noch definiert und mit einer SQL-Anweisung mit Information aus der Datenbank gefüllt werden.
String	Liste mit den Input-Items aus dem Report-Template (hier: **name**).
Number)	Falls ein Fehler auftritt, wird die Nummer des Fehlers in dieser **Variablen** gespeichert (wird der Wert im Voraus auf 1 gesetzt, so wird auch von der EXE-Datei eine Fehlermeldung ausgegeben).

Hier nun die veränderte Fassung, die für das vorliegende Beispiel anzufertigen ist:

```
◆ Pushbutton: pb1                    ! Druck-Vorschau
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalReportView(
        frmMain,
        hWndNULL,
        'LISTE.QRP',
        ':strName',
        'name',
        nError )
```

Beachten Sie auch die Variable "nError", die für die Fehlerbehandlung notwendig ist, sie muß als numerische Variable definiert sein.

Sie können das Programm damit bereits starten, bekommen aber nur ein leeres Fenster zu sehen.

Jetzt müssen die "SAM_Report*"- Nachrichten verwaltet werden, über die der Druckbeginn, das Lesen des nächsten Datensatzes und das Druckende gesteuert wird (daß eben ein leeres Fenster angezeigt wurde liegt genau daran, daß die Datensätze - obwohl sie bereits zur Verfügung standen - nicht gelesen wurden).

Versuchen Sie zunächst zu verstehen, *wo* diese "SAM_Report*"- Nachrichten verwaltet werden müssen: In der Sektion Message Action *des Fensters, in dem der Report aufgerufen wird*, hier also in der Sektion "Message Actions" von "frmMain" (würde diese Codierung in einem anderen Fenster angeordnet werden, dann würde SQLWindows den Code nicht finden und könnte ihn auch nicht ausführen). Die folgende Abbildung zeigt die Message Actions des Fensters "frmMain".

```
◆ Form Window: frmMain
  ◇ Description:
  ◇ Named Menus
  ◇ Menu
  ◆ Tool Bar
  ◆ Contents
    ◆ Pushbutton: pb1
      ◆ Message Actions
        ◆ On SAM_Click
          ◇ Call SalReportView(frmMain, hWndNULL, 'Liste.qrp', ':strName', 'name', nError)
  ◇ Functions
  ◇ Window Parameters
  ◇ Window Variables
  ◆ Message Actions
    ◇ On SAM_ReportStart
    ◇ On SAM_ReportFetchInit
    ◇ On SAM_ReportFetchNext
    ◇ On SAM_ReportFinish
```

Die Nachrichten (SAM_Report*) wurden bereits oben genannt und beschrieben. Deshalb wird hier nur der Programmcode vorgestellt, den Sie unter "SAM_Report*" eingeben müssen.

```
◆ Message Actions
  ◆ On SAM_ReportStart
    ◇ Call SqlPrepare( hSql, 'select Name from lager into  :strName' )
    ◇ Call SqlExecute( hSql)
    ◇ Return TRUE
  ◆ On SAM_ReportFetchInit
    ◇ Call SqlPrepare( hSql, 'select Name from lager into :strName' )
    ◇ Call SqlExecute(hSql)
    ◇ Return TRUE
  ◆ On SAM_ReportFetchNext
    ◆ If SqlFetchNext(hSql, nFetch)
      ◇ Return TRUE
    ◆ Else
      ◇ Return FALSE
  ◇ On SAM_ReportFinish
```

Drei Funktionen werden verwendet, um auf die Nachrichten des Reports zu reagieren:

- **SqlPrepare()** dient dazu, die vorbereitete Sql-Anweisung ("select...") zu kompilieren.
- **SqlExecute()** führt die zuvor kompilierte Anweisung aus.
- **SqlFetchNext()** liefert den nächsten Datensatz aus der mit "Select" erhaltenen Ergebnismenge.

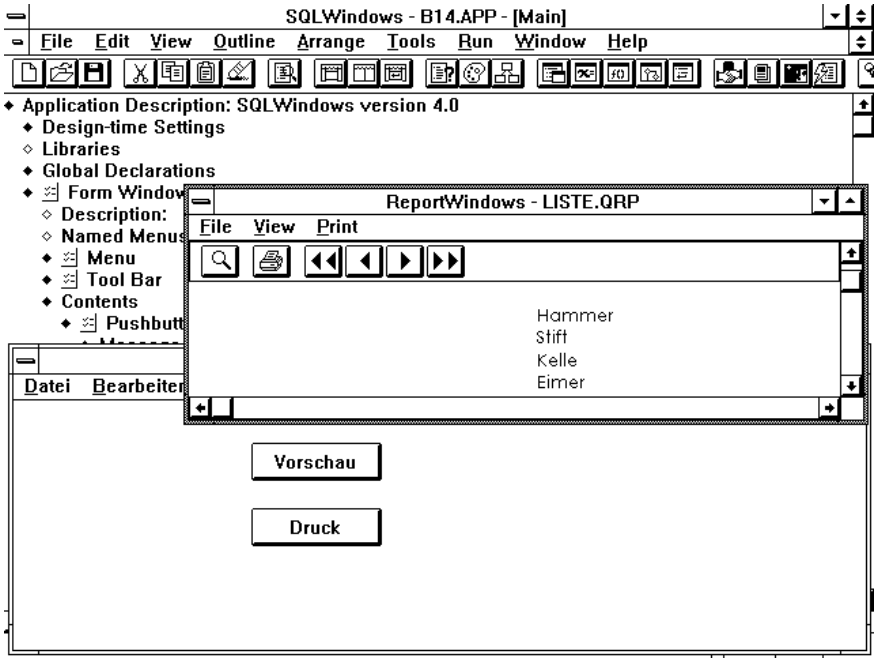
Die genannten Funktionen werden bei der Verarbeitung der oben angezeigten Nachrichten ausgeführt, sie beinhalten einen Sql-Handle ("hSql"), eine Sql-Anweisung ("select...") sowie eine Variable ("nFetch"), in der die Satznummer des nächsten zu druckenden Satzes gespeichert wird (diese Variable muß als numerische Variable definiert sein).

Anmerkung zu dem verwendeten Handle:

Hier wird ein Handle mit dem Namen "hSql" benutzt, der bereits mehrfach in anderen Beispielen Verwendung fand. Denken Sie daran, daß Sie üblicherweise einen Handle für das Drucken von Daten reservieren sollten, der auch nur dann Verwendung findet, wenn Daten zu drucken sind (also nicht z.B. um ein Table Window mit Daten zu füllen).

Nachdem alles codiert wurde, können Sie sich das Ergebnis anzeigen lassen:

Abb. 2.28: Vorschau der Druckausgabe.



In dem Vorschau-Fenster können Sie die Daten durch Klicken auf das Drucker-Symbol ausdrucken lassen; die anderen Schaltflächen erlauben das Zoomen des Fensters bzw. das Blättern.

Verbessertes Verfahren durch die Benutzung von Variablen

Das eben geschilderte Verfahren ist zwar etwas aufwendig, aber die folgenden Hinweise sollen helfen den Aufwand für die Codierung zu verringern. Im Grunde sind für alle zu druckenden Reports die gleichen Angaben nötig, weshalb man z.B. nicht den Namen des Report Templates in den Aufruf der Funktion "SalReportView()" einträgt, sondern hierfür zusätzliche Variable definiert, die vor Aufruf der Funktion mit dem Namen der Datei gefüllt wird. Konsequent verwirklicht bedeutet dies, daß eine Gruppe von Variablen eingeführt wird, die je nach den zu druckenden Daten mit den Angaben zu Template-Namen aber auch Input Items usw. gefüllt werden. Der folgende Code zeigt die hierfür nötigen Variablen und die vorgesehenen Inhalte:

```
◆ Variables
  ◇ String: strReport     ! Name der Templatedatei: Liste.QRP
  ◇ String: strBindVars   ! Variablen in SqlWindows die an das Template übergeben werden
  ◇ String: strInputs     !  Variablen im Template die Daten empfangen
  ◇ Number: nError        ! Variable für Fehlerbehandlung
```

Die Variablen können entweder global oder lokal definiert werden.

Durch die Verwendung der Variablen ändert sich das Aussehen des Programmcodes:

```
◆ Pushbutton: pb1                    ! Druck-Vorschau
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Set strReport = 'LISTE.QRP'
      ◇ Set strBindVars = ':strName'
      ◇ Set strInputs = 'name'
      ◇ Set nError = 1
      ◇ Call SalWaitCursor(TRUE)
      ◇ Call SalReportView(
        frmMain,
        hWndNULL,
        strReport,
        strBindVars,
        strInputs,
        nError)
      ◇ Call SalWaitCursor(FALSE)
```

Hierdurch wird der Funktionsaufruf von "SalReportView()" nahezu universal verwendbar, da für eine Anpassung nur noch die Variablen-Inhalte zu ändern sind.

Um nun zusätzliche Felder aus der Tabelle mit den Lagerdaten zu drucken, muß nur noch das Report-Template angepaßt werden (zusätzliche Input Items müssen definiert werden); weiterhin müssen die vorstehenden Variablen mit anderen Inhalten gefüllt werden. Außerdem können zusätzlich zu den Spalten aus einer Tabelle auch "normale" Variablen-Inhalte gedruckt werden. Der folgende Programmcode weist darauf hin:

Set strBindVars = 'strName, wert, text'

wobei "strName" und "text" Zeichenvariable darstellen und "wert" für eine numerische Variable steht. Die Variable, die die Input Items des Report-Templates enthält, muß auch angepaßt werden. Sie erhält den Inhalt

Set strInputs = 'NAME, wert, text'

wobei wiederum genau auf die Schreibweise zu achten ist.

Separates Form Window für die Druckvorschau

Schließlich kann zusätzlich zu dem oben verwendeten Formular "frmMain" ein separates Form Window (z.B. mit dem Namen "frmReport") eingeführt werden, das ausschließlich für Druckausgaben in Ihrer Anwendung benutzt wird (dies verschafft einen besseren Überblick).

Der folgende Programmcode zeigt den Inhalt des Outline-Fensters für ein Formular "frmReport", in deren Message Actions die "SAM_Report*"- Nachrichten verarbeitet werden.

```
Form Window: frmReport
  Message Actions
    On SAM_ReportStart
      Call SqlPrepare(hSql, strReportSelect)
    On SAM_ReportFetchInit
      Call SqlPrepare(hSql, strReportSelect)
      Return SqlExecute(hSql)
    On SAM_ReportFetchNext
      If SqlFetchNext(hSql, nFetch)
        Return TRUE
      Else
        Return FALSE
    On SAM_ReportFinish
      Set hWndReport = hWndNULL
```

Sie haben hierdurch zwar insgesamt mehr Programmcode, aber eine deutlich bessere Ordnung, da - abgesehen von dem Füllen der Variablen für die Druckausgabe - alle Aufgaben in Verbindung mit dem Druck in einem Formular ("frmReport") codiert werden können.

2.1.4.1.3 Drucken

Der folgende Programmtext, dem Push Button "pb2" zugeordnet, erlaubt die Druckausgabe (ohne dabei die Daten in der Vorschau anzuzeigen). Hierzu wird die Funktion

SalReportPrint()

benutzt.

Die in dem vorstehenden Kapitel gemachten Anmerkungen zu der Verwendung von Variablen sind hier berücksichtigt, das heißt, es werden innerhalb der Funktion "SalReportPrint()" nur Variablennamen benutzt, die zuvor deklariert wurden und unmittelbar vor der Druckausgabe mit Werten gefüllt wurden.

Übersicht der deklarierten Variablen:

```
Variables
   Sql Handle: hSql
   Number: nRow
   String: strName
   Number: nFetch
   Number: nError
   String: strReport
   String: strBindVars
   String: strInputs
   Number: wert
   String: text
```

Der Programmcode für die Druckausgabe (nach Drücken des Push Buttons "pb2") besteht aus folgenden Zeilen:

```
Pushbutton: pb2
  Message Actions
    On SAM_Click
      Set strReport =                Name des Report Templates
'LISTE.QRP'                          SQLWindows-Variable, die übergeben wird
      Set strBindVars =              Name des Input Items innerhalb Liste.qrp
':strName'                           Variable für Fehlerbehandlung
      Set strInputs = 'name'         Funktionsaufruf
      Set nError = 1                 Formular, in dem Messages behandelt werden
      Call SalReportPrint(           Variable für Template-Namen
        frmMain,                     Variable, die zum Template geschickt werden
        strReport,                   Variable im Template, die die Daten erhalten
        strBindVars,                 Anzahl der Kopien je Seite
        strInputs,                   Konstante für Druck aller Seiten
        1,                           Beginn Druckausgabe (hier: Seite 1)
        RPT_PrintAll,                Ende  Druckausgabe (hier: Seite 1)
        1,                           Variable für Fehlerbehandlung
        1,
        nError)
```

Insgesamt ähnelt der Aufbau sehr dem zuvor geschilderten. Allerdings müssen bei dem Funktionsaufruf von "SalReportPrint()" zusätzliche Parameter benutzt werden.

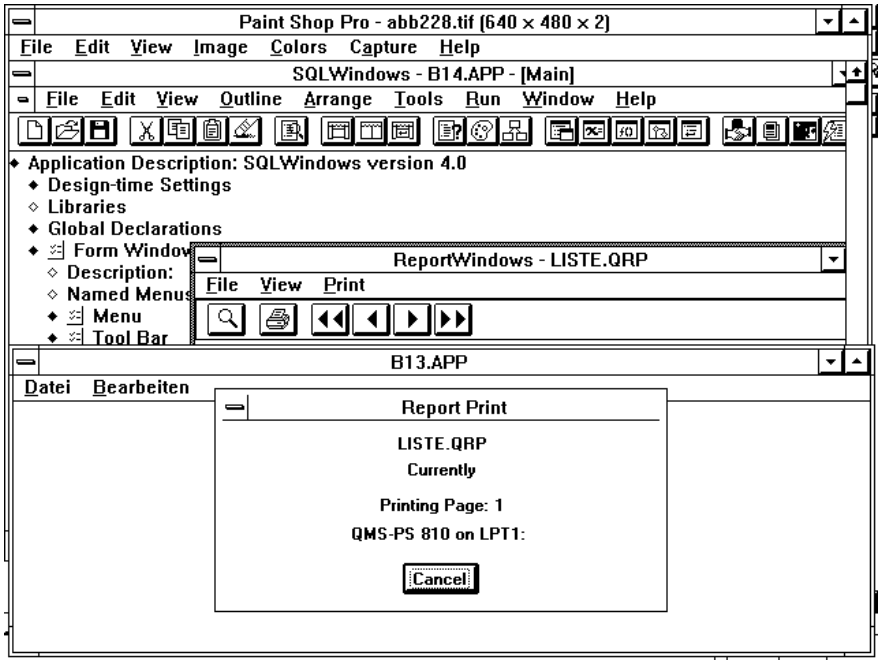
- Nach "strInputs" kann die **Anzahl der Kopien** je Seite angegeben werden (oben ist eine 1 eingetragen).
- **RPT_PrintAll** definiert, daß alle Druckseiten gedruckt werden sollen. Alternativ kann z.B. mit "RPT_PrintRange" eine Auswahl der zu druckenden Seiten angegeben werden

- Nach RPT_PrintAll folgt **zweimal "1"**, das bedeutet, daß die Druckausgabe von Seite 1 bis Seite 1 gehen soll. Da aber zuvor bereits angewiesen wurde, daß alle Seiten zu drucken sind, ist es bedeutungslos, was in *diesem* Fall hier angegeben wird (bei Verwendung von "RPT_PrintRange" dagegen müssen hier die Seitennummern für die erste und letzte zu druckende Seite angegeben werden).

- **nError** erhält eine Zahl, die für einen eventuell aufgetretenen Fehler steht. Die dabei vorhandenen Möglichkeiten können über das Hilfe-System von SQLWindows angezeigt werden.

Nach Drücken des Push Buttons erhalten Sie das übliche, vom Druckmanager erzeugte Fenster, das Sie über den Druckfortschritt informiert, bzw. Ihnen mitteilt, falls es zu Problemen kommt.

Abb. 2.29: Druckausgabe, Nachricht des Druckmanagers.



Solange das über den Druckfortschritt informierende Fenster angezeigt bleibt, können Sie nicht weiter arbeiten (ein Druck auf den Cancel-Pushbutton unterbricht den Ausdruck).

2.1.4.2 Daten aus einer angezeigten Tabelle drucken

Falls die zu druckenden Daten in einer Tabelle (Top Level oder Child Level) angezeigt werden, bietet SQLWindows zusätzliche Funktionen an, die einen Ausdruck deutlich einfacher gestalten als die zuvor dargestellten Methoden.

Es handelt sich insgesamt um drei Funktionen, deren Namen sehr an die bereits genannten Funktionsnamen erinnern, doch enthalten die im folgenden vorzustellenden Funktionen immer das Wort **"Table"**. Die folgende Übersicht nennt die drei Funktionen und gibt eine kurze Erläuterung; im Anschluß daran erläutern Beispiele die Verwendung der Funktionen.

Funktionsname	Zweck
SalReportTableCreate()	Legt eine Report-Datei mit Ausgabefeldern an.
SalReportTableView()	Zeigt eine Report-Datei in der Druckvorschau.
SalReportTablePrint()	Druckt einen Report.

Unter Report-Template ist die schon im vorigen Kapitel beschriebene Report-Datei zu verstehen, die Sie selbst mit Report Windows anlegen können (oder durch den Aufruf der Funktion "SalReportTableCreate()" anlegen lassen können).

Interessant hierbei ist, daß Sie sich nicht um das Lesen der Datensätze kümmern müssen. Das heißt, die Nachrichten ("On SAM_ReportStart()", "On SAM_ReportFetchInit()", "On SAM_ReportFetchNext()", "On SAM_ReportFinish()") brauchen Sie *nicht* zu behandeln. Daraus ergibt sich, daß die Druckausgabe mit Hilfe dieser Funktionen sehr einfach und kurz zu codieren ist. Das folgende Beispiel erläutert die Druckausgabe anhand von Daten, die bereits in einer Tabelle auf dem Bildschirm sichtbar sind.

Hierzu benötigen Sie ein Table Window, das Sie am einfachsten neu anlegen; die folgenden Erläuterungen beschreiben das Vorgehen.

Um das Beispiel nachzuvollziehen:

- Fertigen Sie eine neue Anwendung an.
- Prüfen Sie, ob die Variable "hSql" definiert wurde und ob mittels "SqlConnect()" an die Datenbank "FIRMA" angebunden wird.
- Fügen Sie ein neues Child-Table Objekt ein (Tool-Palette benutzen), definieren Sie *keine* Spalten.
- Um die Tabelle mit Daten zu füllen: Öffnen Sie den Customizer (mit rechter Maustaste auf das Tabellen-Objekt klicken), wählen Sie Edit Actions und geben Sie folgenden Inhalt ein:

```
◆ Child Table: tbl1
  ◇ Contents
  ◇ Functions
  ◇ Window Variables
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Call SalTblPopulate(
         tbl1,
        hSql,
        'select * from kunden',
        TBL_FillNormal )
```

- Lassen Sie das Programm ausführen, es sollten alle Spalten und alle Zeilen der Tabelle mit den Kundendaten angezeigt werden.

Als nächstes wird ein Push Button in das Formular "frmMain" eingefügt werden, nach dessen Drücken ein Report Template angelegt werden soll.

2.1.4.2.1 Template anlegen lassen

Fügen Sie einen neuen Push Button in Ihr Formular frmMain ein, öffnen Sie den Customizer, wählen Sie die Option "Edit Actions" und geben Sie den folgenden Programmcode ein:

```
◆ Pushbutton: pb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalReportTableCreate(
        'test.qrp',
        tbl1,
        nError)
```

Hiermit wird angewiesen, daß nach dem Drücken des Push Buttons die Funktion "SalReportTableCreate()" aufgerufen wird. Als Parameter erhält die Funktion:

- **'test.qrp'** als Dateiname für das neu anzulegende Report-Template.
- **tbl1**, der Name des Table-Objekts, in dem die zu druckenden Daten stehen.
- **nError**, eine Variable, in der ein eventuell auftretende Fehler gespeichert werden (diese Variable muß definiert sein).

Lassen Sie das Programm starten, klicken Sie einmal auf den Push Button, dann erkennen Sie am Aufleuchten der Kontrollampe für die Festplatte, daß ein Schreibvorgang stattfindet.

Nachdem das Report-Template angelegt wurde, passiert nichts weiter, auch das wiederholte Klicken auf dem Push Button führt lediglich zur erneuten Erzeugung der Template-Datei. Wie die Daten mit dem Template angezeigt werden, wird in den beiden folgenden Abschnitten erläutert.

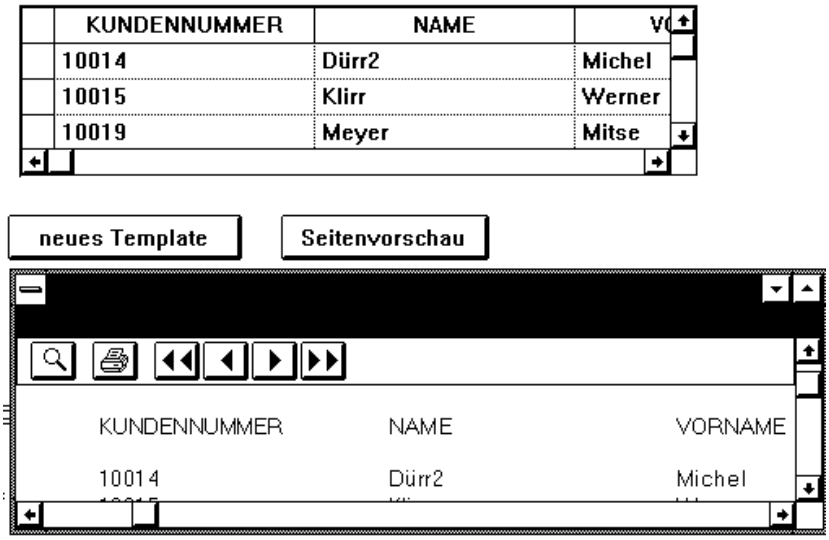
2.1.4.2.2 Anzeige in der Seitenvorschau

Über die Funktion "SalReportTableView()" kann eine Seitenvorschau unter Verwendung des oben erzeugten Report-Templates ausgeführt werden. Fügen Sie auch hierfür einen Push Button ein, und weisen Sie ihm folgende Aktion zu:

```
◆ Pushbutton: pb2
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalReportTableView(
        tbl1,
         hWndNULL,
        'Test.Qrp',
        nError)
```

Nach dem Drücken des Push Button erhalten Sie folgende Anzeige:

Abb. 2.30: Daten aus einem Tabellen-Objekt, angezeigt in der Seitenvorschau.



Hierbei ist anzumerken, daß der besonderes Reiz dieses Verfahrens darin liegt, daß beliebige Daten nach ihrer Anzeige in einem Tabellenobjekt gedruckt werden können.

2.1.4.2.3 Bericht drucken

Schließlich soll ein dritter Push Button die Druckausgabe ermöglichen. Weisen Sie dem neu einzufügenden Push Button folgende Aktion zu:

```
◆ Pushbutton: pb3
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalReportTablePrint(
        tbl1,
        'test.qrp',
        hWndNULL,
        nError)
```

Die Angaben, die hier vorzunehmen sind, entsprechen denen des vorhergehenden Abschnitts. Zusätzlich allerdings erwartet diese Funktion nach dem Namen der Template-Datei einen Array (Inhalte mit "RPT_PrintParam*" Informationen), der Angaben zu den Druckoptionen beinhaltet. Hier wurde an Stelle des Arrays der Name des Handles "hWndNULL" gesetzt, um auf die möglichen Angaben zu den einzelnen Inhalten verzichten zu können.

Falls Sie drucken, erhalten Sie das in Windows übliche Fenster, das Sie über den laufenden Druckjob informiert.

2.1.5 Abfragen von Daten

Das Abfragen von Daten innerhalb einer Anwendung ist der Teil, der am wenigsten standardisiert werden kann. Jeweils anhand den Wünschen des Benutzers muß entschieden werden, auf welche Art die vorhandenen Daten abgefragt werden sollen.

Dabei stehen Ihnen zur Abfrage von Daten grundsätzlich folgende Möglichkeiten offen:

- Verarbeiten einer vom Benutzer selbst eingegebenen Sql-Anweisung.
- Verwenden von Data Fields, in die der Benutzer Angaben zu den gewünschten Informationen eingibt (z.B. ein Feld, in das eine Kundennummer eingegeben werden kann, um die Umsätze dieses einen Kunden anzeigen zu lassen).
- Verwenden eines horizontal zu teilenden Tabellenobjekts, in dem (häufig zusammengefaßte) Daten angezeigt werden.
- Verwenden eines Quest-Table-Windows (vgl. dazu Kap. 1.3.4.1.5).

Deutlich flexibler, aber auch mit deutlich höheren Programmieraufwand, können Sie eine Oberfläche codieren, die dem Benutzer verschiedene Möglichkeiten zur Auswahl von Tabellen, Spalten, Operatoren und Verknüpfungen bietet. Sie laufen dann jedoch Gefahr, die Funktionalität von Quest nachzuprogrammieren, was eigentlich nicht der Sinn Ihrer Programmierarbeit sein sollte.

Die folgenden Beispiele sollen Anregungen zu diesem Thema liefern.

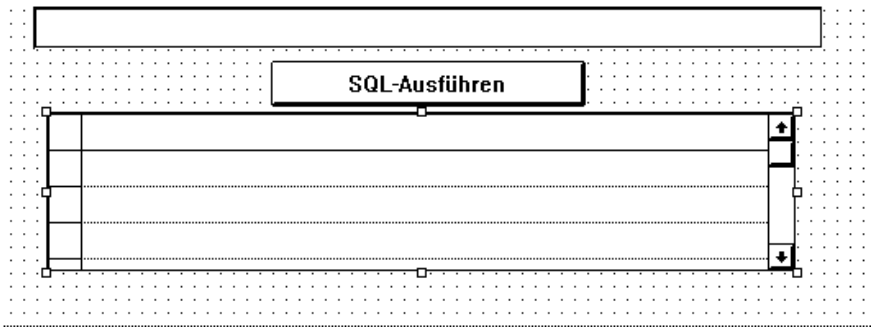
Verarbeiten einer vom Benutzer getippten Sql-Anweisung

Das ist eine sehr einfach zu programmierende und dennoch sehr leistungsfähige Variante, wenn es darum geht, dem Benutzer eine möglichst variable Möglichkeit zur Datenabfrage zu bieten. Allerdings wird vorausgesetzt, daß der Benutzer die Sprache "Sql" beherrscht.

Sie brauchen dazu ein Data Field, einen Push Button und ein Child-Table Window, das keine Spalten enthält.

Auf dem Bildschirm angeordnet könnte folgendes Formular entstehen:

Abb. 2.31: Data Field (oben) für die Eingabe einer Sql-Anweisung, Push Button (Mitte) nach dessen Drücken die Sql-Anweisung ausgeführt wird und Table Objekt (unten), in dem die Ergebnismenge angezeigt wird.



Das Data Field (vom Typ String) wird für die Eingabe einer Sql-Anweisung verwendet. Drückt der Benutzer anschließend den Push Button, läßt er die Funktion "SalTblPopulate()" ausführen, die als Inhalt jetzt nicht eine vordefinierte Sql-Anweisung enthält, sondern den Namen des Data Fields "df1" (dieses Data Field kann wie eine Variable verwendet werden). Der Inhalt der Message Actions von "pb1" lautet:

```
◆ Pushbutton: pb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalTblPopulate( tbl1, hSql, df1, TBL_FillNormal )
```

Innerhalb der Klammern der Funktion wird Bezug genommen auf "df1", das Data Field in dem die vom Benutzer eingegebene Sql-Anweisung steht. Da die Tabelle "tbl1" *keine* vorgegebenen Spalten enthält und die Belegung der Spalten dynamisch (d.h. zur Laufzeit der Anwendung) erfolgt, braucht man sich keine Gedanken über Zahl oder Formatierung der Spalten zu machen. Da *alle* (!) Sql-Anweisungen damit ausführbar sind und die Weiterverarbeitung der damit erzeugten Daten völlig problemlos ist (z.B. durch die Möglichkeit, die Daten mittels der Funktionen "SalReportTable*" zu drucken), erhält der Benutzer hiermit ein sehr leistungsfähiges Modul.

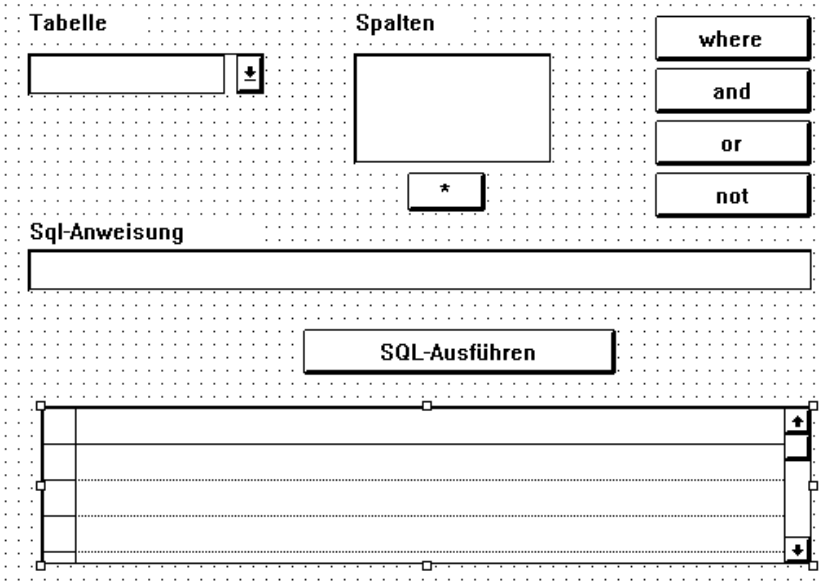
Verwenden von Data Fields, in die der Benutzer Angaben zu den gewünschten Informationen eingibt

Dieses Verfahren ähnelt dem zuvor dargestellten sehr. Es ist jedoch benutzerfreundlicher, da der Benutzer nicht auf Sql-Kenntnisse angewiesen ist, verlangt aber auf der anderen Seite großen Aufwand bei der Codierung.

Sie erstellen ein Formular, in dem die Bedingung(en), die zur Auswahl der Daten führt, mittels der Maus zusammengeklickt werden.

Durch die Verwendung einiger Child-Objekte könnte folgendes Formular entstehen:

Abb. 2.32: Formular für die Abfrage von Daten.



Um die Combo Box mit den Namen der Tabellen zu füllen, ist folgender Inhalt der Message Actions nötig:

```
◆ Message Actions
  ◆ On SAM_Create
    ◇ Call SalListPopulate( cmb1,hSql,
      'select name from systables' )
  ◆ On SAM_Click
    ◇ Call SalListPopulate( lb1, hSql,
      'select * from syscolumns where tbname = :cmb1' )
```

Hiermit wird nach dem Erzeugen der Combo Box dieses Objekt mit dem Inhalt der Spalte "Name" aus der Tabelle "Systables" gefüllt. Das bedeutet, daß alle in der Datenbank Firma enthaltenen Tabellennamen in der Combo Box angezeigt werden.

Durch Klicken auf die Combo Box, d.h. durch die Auswahl eines Tabellennamens durch den Benutzer, werden die in der ausgewählten Tabelle enthaltenen Spalten in der daneben stehenden List Box (mit dem Namen "lb1") angezeigt.

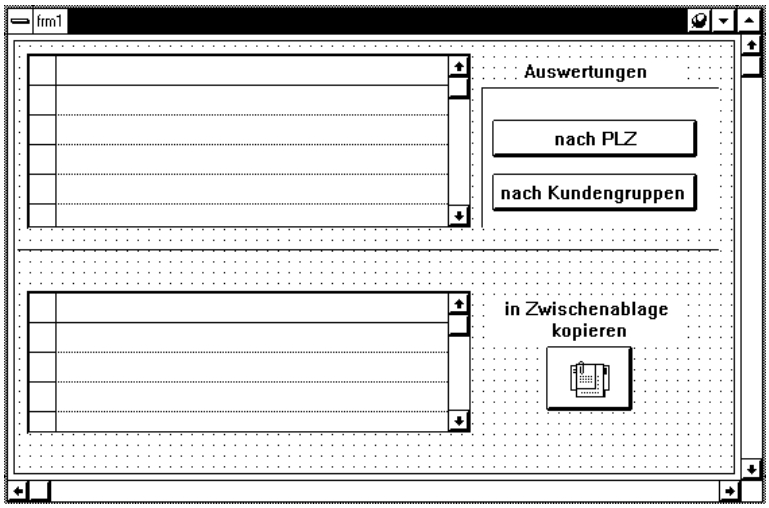
Die anderen Push Buttons dienen dazu, Sql-Anweisungen zusammenzustellen. Durch Verwendung des Concatenate Operators ("||") werden dabei die einzelnen Bestandteile der Sql-Anweisung aneinander gehängt (z.B. "Set df1 = df1 || cmbOperator").

Verwenden eines Tabellenobjekts, in dem Daten angezeigt werden

Im folgenden Beispiel soll ein Formular dazu benutzt werden, in einem ersten Tabellen-Objekt Kundendaten anzuzeigen. Durch das Drücken verschiedener Push Buttons können verschiedene Auswertungen gestartet werden, die diesen zugeordnet sind. Die durch die Auswertung erhaltenen Daten werden in einem zweiten Tabellenobjekt angezeigt. Ein zusätzlicher Push Button soll dazu verwendet werden, die angezeigten Daten in die Zwischenablage zu kopieren, damit alle anderen Windows-Anwendungen die Daten übernehmen und eventuell weiterverarbeiten können. Dabei werden in keiner der beiden Tabellen einzelne Spalten definiert, so daß ein dynamischer Aufbau stattfinden kann.

Die Oberfläche dieser Anwendung hat folgenden Aufbau:

Abb. 2.33: Zwei Tabellenobjekte mit der Möglichkeit, oben angezeigte Tabellendaten auszuwerten und das Ergebnis dieser Auswertung in die Zwischenablage zu kopieren.



Die einzelnen Objekte enthalten folgenden Code:

- Der Inhalt der Message Actions des ersten (oberen) Tabellenobjekts lautet:

```
◆ Child Table: tbl1
  ◇ Contents
  ◇ Functions
  ◇ Window Variables
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Call SalTblPopulate( tbl1,
         hSql ,'select * from kunden',
        TBL_FillNormal )
```

  Nach dem Aufbau von "tbl1" sollen mittels einer Select-Anweisung alle Datensätze der Kundentabelle angezeigt werden.

- Die in deer Abb. oben rechts stehenden zwei Push Buttons enthalten die Sql-Anweisungen, die zum Auswerten und Anzeigen der Daten in der unten stehenden Tabelle verwendet werden.

  Inhalt der Message Actions des ersten Push Buttons:

```
◆ Pushbutton: pb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalTblPopulate(
        tbl2,
        hSql,
        'select plz, count(plz) from kunden group by plz',
        TBL_FillNormal )
```

  Nach dem Drücken des Push Buttons wird die Sql-Anweisung ausgeführt, die die Datensätze nach der Postleitzahl gruppiert und die Anzahl der Kunden mit gleicher Postleitzahl in der unteren Tabelle ("tbl2") anzeigt.

  Inhalt der Message Actions des zweiten Push Buttons:

```
◆ Pushbutton: pb2
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalTblPopulate(
        tbl2,
        hSql,
        'select kundengruppe, count(kundengruppe) from kunden
         group by kundengruppe',
        TBL_FillNormal )
```

  Durch Drücken dieses Push Buttons wird nach dem Inhalt des Feldes "Kundengruppe" ausgewertet und die Daten in der Tabelle "tbl2" angezeigt.

- Die unten angezeigte Tabelle ("tbl2") hat keinen Inhalt in den Message Actions.

- Der dritte Push Button (unten rechts) soll dazu verwendet werden, die angezeigten Daten in die Zwischenablage zu kopieren. Hierzu enthalten die Message Actions folgenden Inhalt:

  ◆ Pushbutton: pb3
  ◆ Message Actions
  ◆ On SAM_Click
  ◇ Call SalTblCopyRows(tbl2, 0, 0)

  Die hier verwendete Funktion "SalTblCopyRows()" kopiert Zeilen einer Tabelle in die Zwischenablage. Damit der Benutzer die zu kopierenden Zeilen nicht markieren muß, werden die zwei Parameter nach dem Tabellennamen ("tbl2") auf "0" gesetzt (natürlich ist es auch möglich, nur vom Benutzer markierte Datensätze in die Zwischenablage zu kopieren, hierzu werden die Einstellungen der "ROW_Flags" benutzt, s. hierzu Hilfesystem von SQLWindows).

Mit dieser Methode können Sie die Daten übergreifend an beliebige Programme unter Windows (z.B. Excel oder Winword) übergeben.

2.2 Entwickeln von Master-Detail-Anwendungen

Unter Master-Detail-Anwendungen versteht man Verbindungen von Tabellen, die über gemeinsame Schlüsselfelder hergestellt werden. Eine solche Verbindung hat ihren Sinn in der Anforderung an die Datenbank, möglichst wenig redundante (sich wiederholende) Information zu speichern.

Zwei Beispiele sollen verwendet werden, um die Benutzung mehrerer, zueinander in Beziehung stehender Tabellen zu verdeutlichen. Zunächst wird in einer Anwendung erläutert, wie Sie Daten aus der Lieferantentabelle und Daten aus der Lagertabelle in Beziehung bringen und gemeinsam anzeigen lassen können (Kapitel 2.2.1).

In dem darauf folgenden Kapitel (2.2.2) soll eine einfache Auftragserfassung zeigen, wie neu einzugebende Daten dargestellt und auf verschiedene Tabellen verteilt gespeichert werden können.

2.2.1 Beziehung zwischen zwei Tabellen

Die in diesem Kapitel verwendeten Tabellen haben die Namen "Lager" bzw. "Lieferant".

- Die Tabelle "Lager" beinhaltet alle Daten zu den am Lager befindlichen Artikel, so z.B. Artikelnummer, Artikelname usw.

- Die Tabelle "Lieferant" enthält alle Informationen zu den einzelnen Lieferanten, also z.B. die Lieferantennummer, Name und Anschrift, Ansprechpartner usw.

Beide Tabellen enthalten, gemäß den in Kapitel 1.1 geschilderten Vorgaben zur Normalisierung von Daten, ein Feld mit der Lieferantennummer. Da von jedem Lieferanten mehrere Artikel geliefert werden, besteht also die Möglichkeit über eine 1:n-Verknüpfung die Daten zu kombinieren, unabhängig davon, daß es sich dabei um die Verknüpfung von zwei Tabellen mit Stammdaten handelt (typischerweise werden 1:n-Verknüpfungen mit Master-Detail-Daten angelegt). Die Lieferantennummer in der Lieferantentabelle stellt dabei den 1-Zweig der Verknüpfung, die Lieferantennummer in der Lagertabelle den n-Zweig der Verknüpfung.

In der späteren Darstellung soll diese Beziehung so realisiert werden, daß in einem Formular oben der Lieferant (Einzelsatz) und darunter in einer Tabelle die von ihm gelieferten Artikel angezeigt werden.

Die Erläuterung geschieht gemäß folgender Reihenfolge:

- Sql-Handles;

- Variables;

- Anzeigen der Lieferantendaten (Master);

- Anzeige der Lagerdaten (Detail);

- Editieren von Detail-Daten und
- Drucken von Master-Detail-Daten.

Sql-Handles

In dieser Anwendung werden mehrere Tabellen verwendet. Deshalb sollen hier auch mehrere Sql-Handles für den Zugriff auf diese Tabellen benutzt werden. Dies entspricht dem üblichen Vorgehen bei der Arbeit mit mehreren Tabellen.

Der folgende Programmcode nennt die verschiedenen Sql-Handles:

```
◆ Variables
  ◇ Sql Handle: hSqlMaster
  ◇ Sql Handle: hSqlDetail
  ◇ Sql Handle: hSqlMasterDetail
```

Die Handles bedeuten im einzelnen:

- **hSqlMaster:** Daten der Master-Tabelle (hier die Daten der Lieferanten);
- **hSqlDetail:** Daten der Detail-Tabelle (hier die Daten der am Lager befindlichen Artikel);
- **hSqlMasterDetail:** spätere Report-Ausgabe (Master- und Detaildaten verknüpfend).

Variables

Neben den Sql-Handles werden auch Variable benötigt. Hier sollen Variable für neu einzufügende Datensätze und Variable für Select-, Update-, Insert- und Delete-Anweisungen benutzt werden, so wie das bereits oben erwähnt wurde.

```
◆ Variables
  ◇ String: strInsertLager        ! einfügen neuer Datensätze
  ◇ String: strUpdateLager        ! ändern vorhandender Datensätze
  ◇ String: strPopulateLager      ! anzeigen vorhandener Datensätze
  ◇ String: strDeleteLager        ! löschen markierter Datensätze
  ◇ !
  ◇ Number: nNewRow               ! für SalTblInsertRow()
```

Anzeigen der Lieferanten-Daten

Zwei Spalten aus der Tabelle mit Lieferantendaten sollen in einer Combo Box angezeigt werden. Hierzu muß zunächst eine Combo Box in das Formular eingefügt werden, die für das Anzeigen der Daten folgenden Inhalt der Message Actions erhält:

```
◆ Combo Box: cmb1                ! Master-Daten (Lieferanten)
  ◇ List Initialization
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Call SalListPopulate(
        cmb1,
        hSqlMaster, "select liefnr || '     ' ||name from lieferant")
```

Als Besonderheit ist hier lediglich darauf hinzuweisen, daß die ab SQLWindows Version 5.0 möglichen doppelten Anführungszeichen verwendet werden. Falls Sie eine ältere Version benutzen, müssen Sie das abändern (z.B. in 'select liefnr, name from lieferanten').

Die Combo Box wird, unter Verwendung des Handles "hSqlMaster" mit den Inhalten der Spalten "Liefnr" (Nummer des Lieferanten) und "Name" (Name des Lieferanten) gefüllt. Zwischen den beiden Spalten sollen einige Leerzeichen für bessere Lesbarkeit sorgen.

Beachten Sie: Die an sich *numerische* Lieferantennummer wird automatisch in eine Zeichenkette umgewandelt, Sie müssen deshalb später aus der angezeigten Zeichenkette die ersten fünf Zeichen extrahieren, um damit die Lieferantennummer zu erhalten (dies wird unten entsprechend erläutert).

Anzeige der Lagerdaten (Detail)

Die Combo Box wird verwendet, um die auf dem Formular untergebrachte Tabelle mit Daten zu füllen. Hier die Aktion, die ausgelöst wird, wenn der Benutzer aus der Combo Box einen Lieferanten ausgewählt hat:

```
◆ Combo Box: cmb1                ! Master-Daten (Lieferanten)
  ◇ List Initialization
  ◆ Message Actions
    ◆ On SAM_Create
    ◆ On SAM_Click
      ◇ Set nKdnr = SalStrToNumber(SalStrLeftX(cmb1,5))
      ◇ Call SalTblPopulate( tbl2, hSqlDetail,
         'select rowid, lieferant, artnr, name, ekpr, vk1
        from lager into :colROWID, :col1, :col2, :col3, :col4, :col5
        where lager.lieferant = :nKdnr',
        TBL_FillNormal )
```

Zunächst werden aus dem angezeigten String (bestehend aus der Kundennummer und dem Namen des Lieferanten) die ersten fünf Zeichen extrahiert (d.h. gelesen und vom Rest getrennt) und in einer numerischen Variable ("nKdnr") abgelegt (diese Variable muß vorher definiert werden).

Danach wird mittels der Funktion "SalTblPopulate()" (unter Verwendung des Handles "hSqlDetail") die Tabelle mit Datensätzen aus der Tabelle Lager gefüllt. Dabei wird eine Bedingung verwendet, die nur solche Datensätze auswählt, deren Inhalt in der Spalte "lieferant" dem Inhalt der numerischen Variable "nKdnr" entspricht ("...where lager.lieferant = :nKdnr").

Um die ausgewählten Daten in der Tabelle anzeigen zu können, werden dort folgende Spalten eingefügt:

- Child Table: tbl2
  - Contents
    - Column: colROWID
    - Column: col1
    - Column: col2
    - Column: col3
    - Column: col4
    - Column: col5

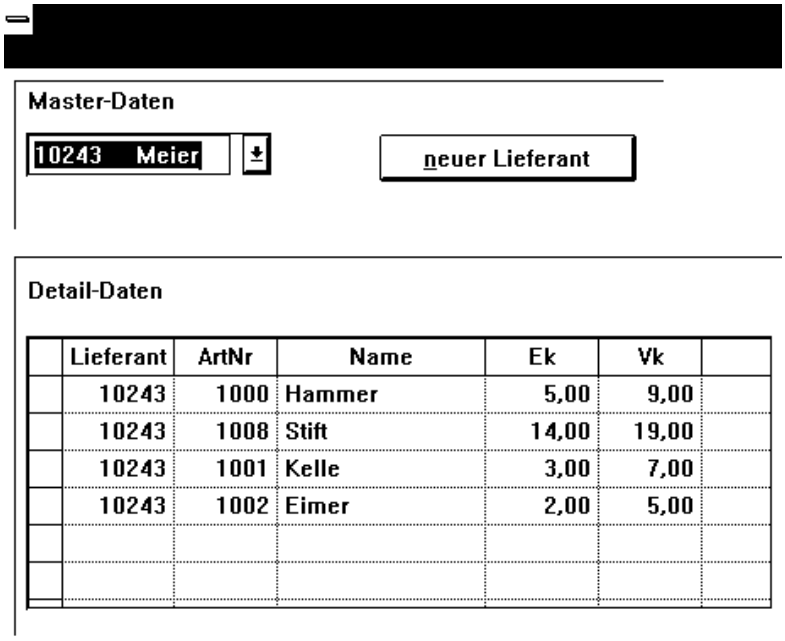
Die Spalte "colROWID" (vom Typ String) dient dazu, die von der Datenbank verwaltete "ROWID" zu speichern, die Spalten "col1" - "col5" haben folgende Datentypen und vorgesehene Inhalte:

Name der Spalte	Datentyp	Inhalt
col1	Number	Lieferant
col2	Number	Artikelnummer
col3	String	Name
col4	Number	Einkaufspreis
col5	Number	Verkaufspreis

Zusätzlich wurden für die Spalten der Preise ("col4" und "col5") Anzeigeformate vereinbart, so daß die dort anzuzeigenden Beträge mit Dezimalstellen und rechtsbündig formatiert werden (über den Customizer).

Wählt nun der Benutzer einen Lieferanten aus der Combo Box aus, so werden die ausgewählten Datensätze in der unteren Tabelle ("tbl2") durchgeführt. Die folgende Abbildung zeigt das Ergebnis einer solchen Auswahl.

Abb. 2.34: Anzeige von Datensätzen aus der Lager-Tabelle nach der Auswahl eines Lieferanten.



Die Spalte mit der Lieferantennummer in der Tabelle wird nur zur Kontrolle angezeigt, ihre Inhalte sind nicht veränderbar.

Editieren von Detail-Daten

Unter dem Editieren von Datensätzen ist, wie schon bereits früher ausgeführt wurde, die Neueingabe, das Ändern und das Löschen von Datensätzen zu verstehen. Da die einzelnen Verfahren hierzu bereits geschildert wurden sollen hier nur kurz die Inhalte der Message Actions genannt werden.

Zwei Push Buttons werden hier verwendet, um Datensätze neu einzugeben bzw. um geänderte Daten zu sichern.

- Push Button für die Eingabe eines neuen Artikels:

```
◆ Pushbutton: pb2                          ! neuer Artikel
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalTblDefineSplitWindow( tbl2, 3, TRUE )
      ◇ Set nNewRow =  SalTblInsertRow( tbl2, TBL_MinRow )
      ◇ Set col1 = nKdnr
      ◇ Call SalTblSetFocusCell( tbl2, nNewRow, col2, -1, -1)
```

Wichtig hierbei ist, daß der Inhalt der numerischen Variable "nKdnr" verwendet wird, um die Spalte "col1" mit einem Inhalt zu versehen (die Lieferantennummer wird in diese Spalte geschrieben). Dies ist notwendig, da später *nur* über diesen Inhalt herausgefunden werden kann, von welchem Lieferant dieser neu eingegebene Artikel stammt.

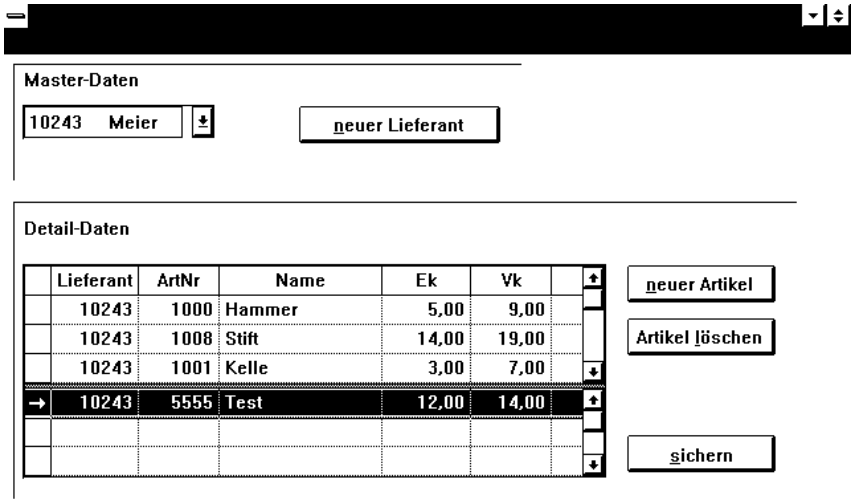
- Push Button für die Sicherung der Datensätze (neuer, geänderter und gelöschter Daten):

```
◆ Pushbutton: pbSichern                    ! sichern
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Set strInsertLager = 'insert into Lager (lieferant, artnr, NAME, ekpr, vk1)
         values (:col1, :col2, :col3, :col4, :col5)'
      ◇ Set strUpdateLager = 'update Lager set
        artnr = :col2,
        NAME = :col3,
        ekpr = :col4
         where ROWID = :colROWID'
      ◇ Set strDeleteLager =  'delete from lager where ROWID = :colROWID'
      ◇ Set strPopulateLager = 'select ROWID, lieferant, artnr, name, ekpr, vk1
                  from lager
                   into :colROWID, :col1, :col2, :col3, :col4, :col5'
      ◇ Call SalTblDefineSplitWindow( tbl2, 0, TBL_Split_Adjustable )
      ◇ Call SqlPrepare(hSqlDetail, strInsertLager)
      ◇ Call SalTblDoInserts( tbl2, hSqlDetail, TRUE )
      ◇ Call SqlPrepare(hSqlDetail,strUpdateLager)
      ◇ Call SalTblDoUpdates( tbl2, hSqlDetail, TRUE )
      ◇ Call SqlPrepare(hSqlDetail, strDeleteLager)
      ◇ Call SalTblDoDeletes(tbl2, hSqlDetail, ROW_MarkDeleted)
      ◇ Call SqlImmediate('commit')
      ◇ Call SalTblPopulate(tbl2, hSqlDetail, strPopulateLager , TBL_FillNormal )
```

Beachten Sie, daß hierbei wieder Variable ("strInsertLager" etc.) benutzt werden, um den Aufwand für die Eingabe zu reduzieren.

Nach dem Start der Anwendung erhalten Sie folgenden Bildschirm:

Abb. 2.35: Master-Detail Daten in einem Formular mit neuem Datensatz.



Um zu verhindern, daß der Benutzer durch Betätigen der Tab-Taste in der letzten Spalte erneut einen Datensatz anhängt, wurden die Message Actions der Tabelle "tbl2" um folgenden Eintrag erweitert:

- Message Actions
  - On SAM_EndCellTab
    - Return FALSE

Die Nachricht "SAM_EndCellTab" wird dann verschickt, wenn in der letzten Tabellenspalte die Tabulatortaste gedrückt wird. Die Rückgabe von "FALSE" verhindert, daß die Schreibmarke aus dem Feld heraus bewegt werden kann.

Drucken von Master-Detail-Daten

Bei der Ausgabe von Daten aus dem oben erstellten Master-Detail-Formular können drei Fälle getrennt werden:

- Daten werden nur aus der Master-Tabelle ausgegeben,
- Daten werden nur aus der Detail-Tabelle ausgegeben bzw.
- Daten werden aus beiden Tabellen kombiniert und zusammen ausgegeben.

Da die ersten beiden Fälle bereits früher erörtert wurden (vgl. Kap. 2.1.3) und sich nicht von den hier geschilderten Verhältnissen unterscheiden, wird im folgenden nur der dritte Fall, die kombinierte Ausgabe von Daten aus zwei Tabellen erläutert. Außerdem wird an dieser Stelle nur die Funktion "SalReportView()" benutzt, d.h. die Daten werden nur auf dem Bildschirm in der Seitenvorschau angezeigt.

Die Vorgehensweise für das Drucken der Daten aus zwei Tabellen ist prinzipiell die gleiche wie bei den in Kapitel 2.1.4 geschilderten Verfahren und besteht aus folgenden Teilschritten:

- Template anlegen;
- Push Button in Fenster einbauen;
- "SAM_Report*"- Nachrichten behandeln.

Template erzeugen

Zunächst soll das Template erzeugt werden, das die Daten anzeigen soll. Starten Sie hierfür ReportWindows aus SQLWindows heraus (Push Button oben).

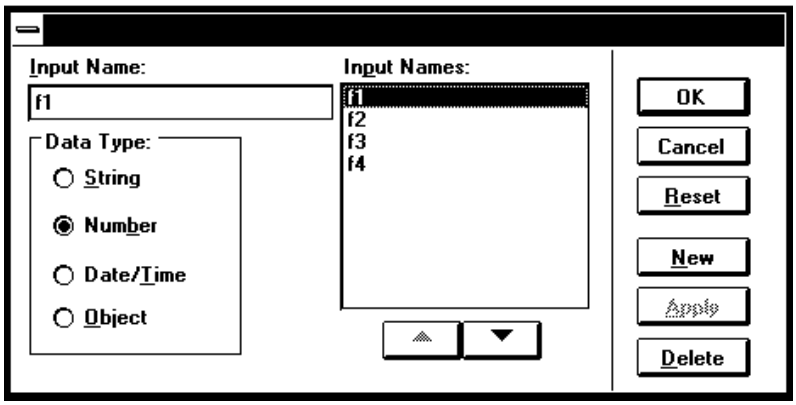
Abb. 2.36: Report Windows, Start aus SQLWindows heraus.

Öffnen Sie den Menüpunkt "Format", wählen Sie die Option "Input" und danach "Input Items". In dem darauf erscheinenden Fenster geben Sie die in der folgenden Übersicht dargestellten Input Items ein; beachten Sie dabei die Reihenfolge (mit "f1" beginnend), die Schreibweise und den Datentyp.

Inhalt	Name Tabellenspalte	Name Input Item	Datentyp
Artikelnummer	LAGER.ARTNR	f1	Number
Artikelname	LAGER.NAME	f2	String
Nummer des Lieferanten	LAGER.LIEFERANT	f3	Number
Name des Lieferanten	LIEFERANT.NAME	f4	String

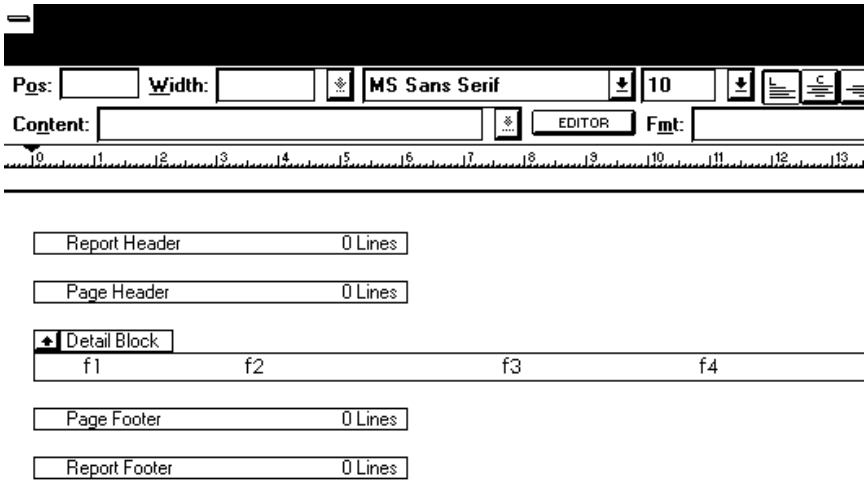
Nachdem Sie alle Input Items definiert haben, sollte Ihre Maske zur Definition der Input Items folgendes Aussehen haben.

Abb. 2.37: Report Windows, Definition von vier Input Items (f1-f4).



Nachdem Sie die Input Items definiert haben, müssen diese so im Report eingebaut werden, daß sie mit Daten gefüllt werden können. Hierzu fügen Sie, nachdem Sie das oben angezeigte Fenster geschlossen haben, zuerst ein leeres Data Field in den Detail Block ein, klicken danach auf den Pfeil der Combo Box (Content), und wählen den ersten Namen ("f1"). Danach wiederholen Sie die Schritte bis alle vier Input Items in den Detail-Block aufgenommen sind. Der fertige Report sollte ungefähr das folgende Aussehen haben:

Abb. 2.38: Report Template mit Input Items.



Speichern Sie das Template dann unter dem Namen "Mastdet.qrp" ab und schließen Sie das Modul Report Windows.

Zurück in SQLWindows muß jetzt ein Push Button in das Formular eingebaut werden, nach dessen Drücken die Tabellendaten gedruckt werden sollen. Fügen Sie hierzu einen Push Button in das vorhandene Fenster frmMain ein, beschriften Sie den Push Button (hier wurde "Master und Detail drucken" eingegeben), und weisen Sie anschließend dem Push Button folgende Aktion zu:

```
◆ Pushbutton: pb6                ! Master und Detail drucken
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Set sBindVars = ':nArtnr, :sArtName, :nLieferant, :sLieferName'
      ◇ Set sIntoVars = 'f1, f2, f3, f4'
      ◇ !
      ◇ Call SalReportView(
        frmMain,
        hWndNULL,
        'MASTDET.QRP',
        sBindVars,
        sIntoVars,
        nError )
```

Die Variablen **sBindVars** und **sIntoVars** müssen natürlich vorher definiert werden. Sie dienen der Speicherung der Bind-Variablen, also der Variablen, die nach einer Sql-Select-Anweisung für die Übergabe der Daten verwendet werden, sowie der Variablen, die in dem Report-Template als Input Items verwendet werden. Beachten Sie, daß diese Variablen mit den in vorliegendem Formular benutzten Variablen nichts zu tun haben! Dies ist auch der Grund, weshalb bei Bearbeitung der "SAM_Report*" - Nachrichten ein separater Handle ("hSqlMasterDetail") benutzt wird.

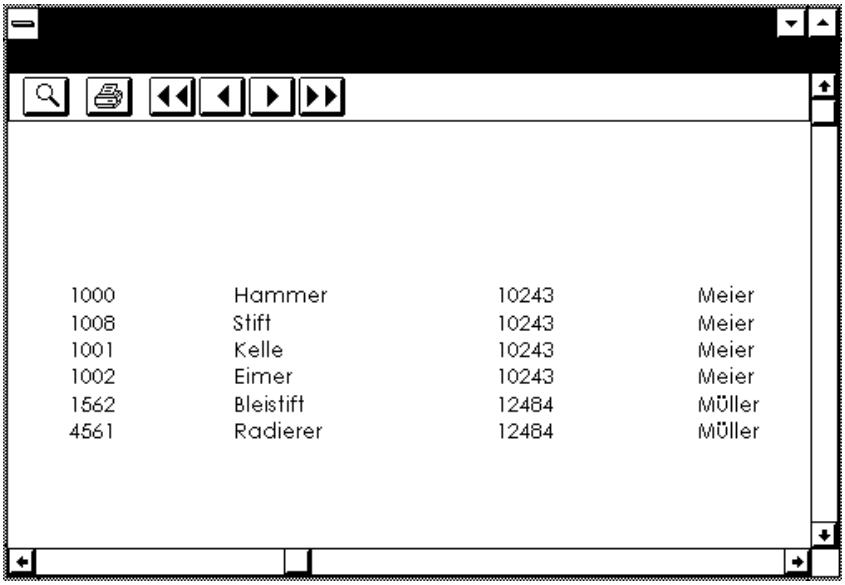
Jetzt müssen die "SAM_Report*"- Nachrichten in der Sektion Message Actions des Fensters frmMain behandelt werden. Tragen Sie dort folgendes ein (beachten Sie, daß die numerische Variable "nArtnr" noch in der Sektion Global Declarations definiert werden muß):

```
◆ On SAM_ReportStart
  ◇ Call SqlPrepare( hSqlMasterDetail,
    'select lager.artnr, lager.name,
                lager.lieferant, lieferant.name
    from lager, lieferant
    into ' ||  sBindVars ||
    ' where lager.lieferant = lieferant.liefnr')
  ◇ Call SqlExecute( hSqlMasterDetail)
  ◇ Return TRUE
◆ On SAM_ReportFetchInit
  ◇ Call SqlPrepare( hSqlMasterDetail,'select lager.artnr, lager.name,
                lager.lieferant, lieferant.name
    from lager, lieferant
    into ' ||  sBindVars ||
    ' where lager.lieferant = lieferant.liefnr')
  ◇ Call SqlExecute(hSqlMasterDetail)
  ◇ Return TRUE
◆ On SAM_ReportFetchNext
  ◆ If SqlFetchNext(hSqlMasterDetail, nFetch)
    ◇ Return TRUE
  ◆ Else
    ◇ Return FALSE
◇ On SAM  ReportFinish
```

Der einzige Unterschied zu dem bereits früher vorgestellten Verfahren liegt darin, daß hier die Daten aus zwei Tabellen gelesen werden (s. Select-Anweisung).

Als Ergebnis erhalten Sie die Menge *aller* Datensätze aus der Lagertabelle, deren Spalten mit den Lieferantennummern den gleichen Inhalt haben wie die Lieferantennummern aus der Lieferantentabelle.

Abb. 2.49: Master- und Detaildaten im Vorschau-Fenster.

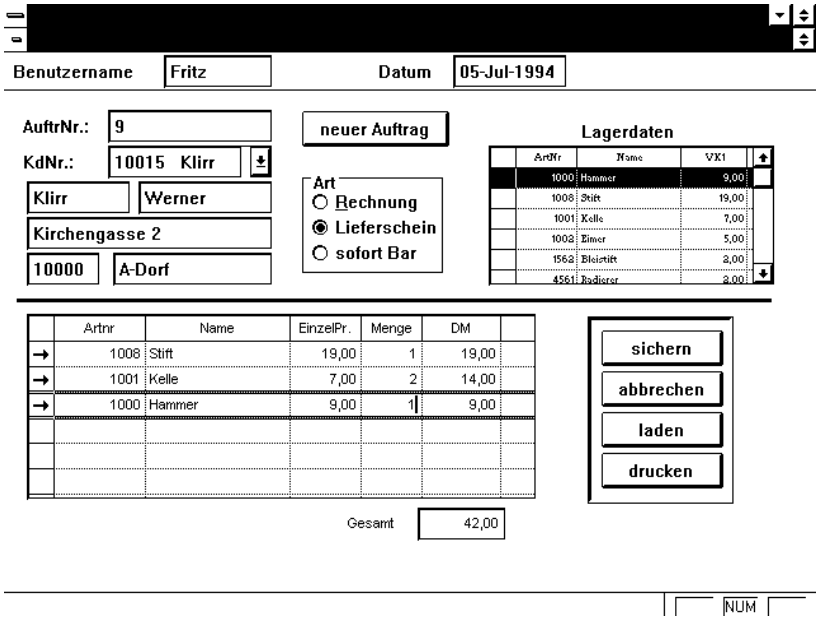


Sie können den Report ohne große Mühe an weitere Vorstellungen anpassen, indem Sie die Sql-Select-Anweisung ändern.

2.2.2 Auftragserfassung

In diesem Beispielprogramm soll mit einfachen Mitteln demonstriert werden, wie mit geringem Aufwand eine Auftragsverwaltung mittels SQLWindows erstellt werden kann. Hierzu soll ein Formular erzeugt werden, das ausgehend von den Daten einer Kunden- und Lagertabelle neue Daten (Auftragsdaten) erzeugt. Das Formular ist dabei genauso aufgebaut wie ein Rechnungsformular.

Abb. 2.40: Auftragserfassung, Formular mit Kundendaten (links oben) Lagerdaten (rechts oben), und den eigentlichen Detail-Daten des Auftrags (unten).



Zur Erläuterung des Programms werden die einzelnen Bestandteile getrennt vorgestellt, auf die Beschreibung, wie die einzelnen Objekte eingefügt werden, wird hier verzichtet.

Alle Objekte sind in einem Form Window eingebettet; hier handelt es sich um ein MDI-Child Formular, möglich ist aber ebenso ein einfaches Form Window zu benutzen.

Variable

Folgende Variable werden benutzt:

```
◆ Variables
  ◇ Sql Handle: hSqlMaster          ! für Kundendaten
  ◇ Sql Handle: hSqlLager           ! für Lagerdaten
  ◇ Sql Handle: hSqlDetail          ! für Auftragsdaten (Details)
  ◇ Number: nNewRow                 ! für neuen Satz in Tabelle tbl2
  ◇ Number: nKdnr                   ! für SqlImmediate bei cmb1
  ◇ String: strInsertAuftrag        ! für einzelne Datensätze
  ◇ String: strInsertKontroll       ! eine Zeile je Auftrag
  ◇ String: sArt                    ! für Rechnung/Lieferschein/Bar
```

Verschiedene Sql-Handles werden für den Zugriff auf die Tabellendaten verwendet. Sie werden dort, wo sie verwendet werden, besprochen.

Da z.B. die Möglichkeit der *Auftragskorrektur* hier nicht codiert wurde, müßten dafür noch weitere Sql-Handles eingeführt werden.

Objekte

Combo Box für Anzeige der Kundennummer bzw. des Kundennamens. Die Combo Box, die zur Auswahl eines Kunden bei der Auftragserfassung verwendet wird, enthält folgende Message Actions:

```
◆ Combo Box: cmb1
  ◇ List Initialization
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Call SalListPopulate(
        cmb1,
        hSqlMaster,
        "select kundennummer || '   ' || name from kunden" )
    ◆ On SAM_Click
      ◇ Set nKdnr = SalStrToNumber(SalStrLeftX(cmb1, 5))
      ◇ Call SqlImmediate
        ( 'select name,vorname, strasse, plz, ort
        from kunden
        into :df1, :df2, :df3, :df4, :df5
        where kundennummer = :nKdnr')
```

Nach dem Erzeugen der Combo Box wird diese unter Verwendung der Funktion "SalListPopulate()" mit zwei Tabellenspalten ("kundennummer", "name") gefüllt. Zwei Spalten wurden deshalb gewählt, weil die Zuordnung ausschließlich nach der Kundennummer nicht sehr anschaulich ist, da sich kaum jemand alle Kundennummern merken kann. Hier werden die in SQLWindows ab Version 5.0 möglichen doppelten Anführungszeichen verwendet, die es erlauben, auch die einfachen Anführungszeichen eingebettet zu verwenden.

Nach der Auswahl eines Kunden aus der angezeigten Liste ("On SAM_Click"), wird eine Sql-Anweisung mittels der Funktion "SqlImmediate()" ausgeführt, die die restlichen Informationen zu dem Kunden aus der Tabelle mit den Kundendaten liest und sie in den Data Fields "df1" bis "df5" anzeigt (dies dient nur der Anschaulichkeit, Änderungen der angezeigten Daten bleiben folgenlos).

Push Buttons

Der **Push Button für einen neuen Auftrag** hat die Aufgabe, eine neue Auftragsnummer vorzuschlagen und diese in dem Data Field "dfAuftragNummer" anzuzeigen. Hierfür wird aus der Kontrolltabelle die größte vorhandene Auftragsnummer ermittelt, deren Wert um 1 erhöht und danach in das Data Field "dfAuftragNummer" geschrieben. Diese neue Auftragsnummer wird später für das Sichern der Daten verwendet. Nachfolgend ist der Inhalt der Message Actions dieses Push Buttons ("pb5") dargestellt.

```
◆ Pushbutton: pb5                          ! neuer Auftrag
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SqlImmediate(
        'select max(aufnr)+1
        from kontroll
        into :dfAuftragNummer'  )
```

Der **Push Button für das Sichern der Daten** dient dazu, die vom Benutzer neu erfaßten Daten eines Auftrags in die Tabellen für die Auftragsdaten (bzw. Kontrolldaten) zu schreiben. Hierzu enthält die Message Actions folgende Zeilen:

```
◆ Pushbutton: pb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ ! Bestimmung ob Rechnung, Lieferschein oder Sofortrechnung
      ◆ If rb1
      ◆ If rb2
      ◆ If rb3
      ◇ ! Schreiben in die Tabelle mit Kontrolldaten
      ◇ Call SqlImmediate(strInsertKontroll)
      ◇ Call SqlImmediate('Commit')
      ◇ ! Schreiben in die Tabelle mit Auftragsdaten
      ◇ Call SqlPrepare(hSqlDetail, strInsertAuftrag)
      ◇ Call SalTblDoInserts( tbl2, hSqlDetail, TRUE )
      ◇ Call SqlImmediate('Commit')
      ◇ ! Löschen der angezeigten Daten, Ende der Auftragserfassung
      ◇ Call SalTblReset( tbl2 )
      ◇ Set df1 = ''
      ◇ Set df2 = ''
      ◇ Set df3 = ''
      ◇ Set df4 = ''
      ◇ Set df5 = ''
      ◇ Set df6 = 0
      ◇ Set dfAuftragNummer = 0
```

Zunächst wird geprüft, welcher der drei vorhandenen Radio Buttons ("rb1" - "rb3") von dem Benutzer gewählt wurde (oben sind die Radio Buttons geschlossen, nachfolgend sind ihre Inhalte dargestellt):

```
◆ Pushbutton: pb1
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ ! Bestimmung ob Rechnung, Lieferschein oder Sofortrechnung
      ◆ If rb1
        ◇ Set sArt = 'R'
      ◆ If rb2
        ◇ Set sArt = 'L'
      ◆ If rb3
        ◇ Set sArt = 'B'
```

Falls der erste Radio Button gewählt wurde (das ist die Voreinstellung), handelt es sich bei dem Auftrag um eine Rechnung, weshalb die String-Variable "sArt" mit einem "R" gefüllt wird; analog wird bei den anderen beiden Radio Buttons verfahren und die Variable ggf. mit dem Zeichen "L" (Lieferschein) bzw. mit "B" (sofort Bar) gefüllt. Die etwas seltsam anmutende Konstruktion wird deshalb nötig, weil die Datenbank SQLBase keine logischen Datentypen kennt. Sie können also den Status eines Radio Buttons (der entweder "TRUE" oder "FALSE" ist) nicht *direkt* in die Tabelle schreiben, sondern müssen eine Umwandlung vornehmen.

Hier wurde der logische Inhalt der Radio Buttons in eine Zeichenkette (einen String), die nur ein Zeichen enthält, umgewandelt. Sie können aber genauso eine numerische Variable benutzen, deren Inhalt danach in eine (ebenfalls numerische) Spalte der Tabelle abgespeichert werden kann.

Danach wird mit einer "SqlImmediate" - Anweisung die in der Variablen "strInsertKontroll" gespeicherte Sql-Anweisung ausgeführt, die folgenden Inhalt hat:

```
◇ Set strInsertAuftrag = 'insert into auftrag
   (AUFNR, ARTNR, MENGE, E_PREIS )
  values
   (:dfAuftragNummer, :tbl2.col1, :tbl2.col4, :tbl2.col5) '
◇ Set strInsertKontroll = 'insert into kontroll
  (AUFNR, KDNR, DATUM, GESAMT, ART)
  values
  (:dfAuftragNummer, :nKdnr, :dfDatum, :df6, :sArt)'
```

Mit dieser Sql-Insert-Anweisung wird die neue Auftragsnummer zusammen mit der Kundennummer, dem Datum der Auftragserfassung, der Gesamtsumme und der Auftragsart in die Tabelle "Kontroll" gespeichert.

Anschließend wird eine weitere Sql-Anweisung dazu benutzt, die Detail-Daten in die Tabelle "Auftrag" zu schreiben; hierzu wird die Variable "strInsertAuftrag" benutzt.

Schließlich werden die angezeigten Data Fields gelöscht. Das Formular kann nun für eine erneute Auftragserfassung benutzt werden.

Radio Buttons für Rechnung/Lieferschein/Sofort Bar

Die Radio Buttons enthalten keine weiteren Anweisungen innerhalb der Message Actions. Ihr Inhalt, der entweder "TRUE" oder "FALSE" ist, wird ausgewertet, nachdem der Benutzer den "Sichern" Push Button gedrückt hat.

Child Table für Lagerdaten

Dieses Table-Objekt dient der Auswahl von Artikeln. Die in der Tabelle "Lager" enthaltenen Daten werden (z.T.) angezeigt, um eine leichte Auswahl für die Auftragserfassung zu ermöglichen. Die Message Actions der Tabelle haben folgenden Inhalt:

```
◆ Child Table: tbl1
  ◆ Contents
  ◇ Functions
  ◇ Window Variables
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Call SalTblPopulate(
        tbl1,
        hSqlLager,
        'select artnr, name, vk1 from lager into :col1, :col2, :col3',
        TBL_FillNormal )
```

Die Tabelle für Lagerdaten enthält drei Spalten ("col1" - "col3"), die mittels der Funktion "SalTblPopulate()" gefüllt werden. Hierbei ist lediglich zu beachten, daß die Datentypen der Spalten "col1" und "col3" auf "Numeric" umgestellt werden. Alle Spalten sind nicht editierbar.

Daneben soll ermöglicht werden, daß der Benutzer durch einen Doppelklick auf den Inhalt der ersten Spalte den angeklickten Artikel nach unten (in die eigentliche Auftragstabelle) kopieren kann. Hierzu enthält die Message Actions *der ersten Spalte (col1)* folgenden Inhalt:

```
◆ Child Table: tbl1
  ◆ Contents
    ◆ Column: col1
      ◇ List Values
      ◆ Message Actions
        ◆ On SAM_DoubleClick
          ◇ Call SalTblInsertRow( tbl2, TBL_MaxRow )
          ◇ Set tbl2.col1 = tbl1.col1
          ◇ Set tbl2.col2 = tbl1.col2
          ◇ Set tbl2.col3 = tbl1.col3
```

Nach einem Doppelklick werden die Inhalte der Spalten "col1", "col2", "col3" (in der die Lagerdaten angezeigt werden) nach unten in die Tabelle mit den Detaildaten ("tbl2") kopiert.

Child Table für Auftragsdaten (unten)

Dieses Tabellen-Objekt erlaubt das Zusammenstellen der Daten von Artikeln, die im Rahmen des aktuellen Auftrags abgegeben werden. Nach dem Aufbau des Tabellenobjekts werden zunächst die Variablen "strInsertAuftrag" bzw. "strInserKontroll" mit den Inhalten gefüllt, die später beim Drücken des Push Buttons für das Sichern verwendet werden:

```
◆ Child Table: tbl2
  ◆ Contents
  ◇ Functions
  ◇ Window Variables
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Set strInsertAuftrag = 'insert into auftrag
          (AUFNR, ARTNR, MENGE, E_PREIS )
        values
          (:dfAuftragNummer, :tbl2.col1, :tbl2.col4, :tbl2.col5) '
      ◇ Set strInsertKontroll = 'insert into kontroll
        (AUFNR, KDNR, DATUM, GESAMT, ART)
        values
        (:dfAuftragNummer, :nKdnr, :dfDatum, :df6, :sArt)'
```

Die Tabelle enthält einige **Spalten**, die im folgenden beschrieben werden.

Die erste Spalte **col1** für die Artikelnummer wird zum einen dadurch mit einer Artikelnummer gefüllt, daß der Benutzer doppelt auf einer Artikelnummer in der Tabelle mit den Lagerdaten klickt. Zum anderen soll es aber möglich sein, daß der Benutzer selbst einen neuen Datensatz einfügt, um eine selbst getippte Artikelnummer einzugeben. Hierfür enthält die erste Spalte folgende Message Actions:

```
◆ Child Table: tbl2
  ◆ Contents
    ◆ Column: col1
      ◇ List Values
      ◆ Message Actions
        ◆ On SAM_DoubleClick
          ◇ Set nNewRow =  SalTblInsertRow( tbl2, TBL_MaxRow )
          ◇ Call SalTblSetFocusCell( tbl2, nNewRow, col1, 0, 0 )
        ◆ On SAM_AnyEdit
          ◇ Call SqlImmediate(
            'select name,vk1
            from lager
            into :tbl2.col2, :tbl2.col3
            where artnr = :col1' )
          ◇ Call SalUpdateWindow( tbl2)
```

Durch einen Doppelklick in der ersten Spalte wird zunächst eine neue Zeile eingefügt ("SalTblInsertRow()"), danach wird die Schreibmarke in diese erste Spalte gesetzt ("SalTblSetFocusCell()") und der Benutzer kann die Artikelnummer eingeben.

Durch das Editieren des Data Fields wird die Nachricht "SAM-AnyEdit" verschickt. Aus der Tabelle mit den Lagerdaten sollen daraufhin der Name und der Verkaufspreis ("VK1") desjenigen Artikels gelesen (und in die entsprechenden Spalten der Tabelle "tbl2" eingefügt werden), dessen Artikelnummer eingegeben wurde.

Schließlich soll der Inhalt der Tabelle aufgefrischt, d.h. die Daten erneut angezeigt, werden.

Die Spalte **col2**, in der der Artikelname angezeigt wird, enthält keine weiteren Anweisungen.

Die Spalte **col3**, in der der Preis des Artikels angezeigt wird (er wird aus der Lagertabelle kopiert und hier eingefügt, bleibt aber editierbar), enthält folgende Message Actions:

```
◆ Column: col3
  ◇ List Values
  ◆ Message Actions
    ◆ On SAM_AnyEdit
      ◇ Set col5 = col4 * col3
      ◇ Set df6 = SalTblColumnSum( tbl2, 5, 0, 0 )
```

Falls der Benutzer den Inhalt der Spalte verändert (indem er einen anderen Preis einträgt) wird der Inhalt der Spalte "col5" neu berechnet. Der Inhalt dieser Spalte ergibt sich aus dem Produkt der Spalten "col4" und "col3" (Menge * Einzelpreis).

Ebenso soll der Gesamtrechnungsbetrag, der sich aus der Summe aller Inhalte von Spalte "col5" ergibt, neu berechnet werden (allerdings darf hier nicht der Name der Spalte eingetragen werden, vielmehr muß die Nummer der Spalte, von links gezählt, eingetragen werden).

Die Spalte **col4** (Menge) hat den gleichen Inhalt der Message Actions wie die Spalte col3, da auch nach Änderung der abgegebenen Menge eine Neu-Berechnung stattfinden soll.

```
◆ Column: col4
  ◇ List Values
  ◆ Message Actions
    ◆ On SAM_AnyEdit
      ◇ Set col5 = col4 * col3
      ◇ Set df6 = SalTblColumnSum( tbl2, 5, 0, 0 )
```

3 Vertiefung

Mit diesem Kapitel soll der Programmierer angesprochen werden, der bereits über erste Erfahrungen im Umgang mit SQLWindows verfügt. Erläutert werden soll zunächst das objektorientierte Programmieren (Kap. 3.1), weiterhin werden Besonderheiten der Programmierung von Netzwerkanwendungen (Kap. 3.2) behandelt. Schließlich soll ein Programm vorgestellt werden, mit dessen Hilfe ein Fremdformat (hier: "DBF") gelesen werden kann (Kap. 3.3).

3.1 OOP (Object Oriented Programming)

In einem einführenden Buch zur Programmierung von Datenbankanwendungen die Objektorienierte Programmierung zu behandeln, ist eigentlich ein Widerspruch, da diese Thematik dem fortgeschrittenen Programmierer zugedacht ist. Trotzdem soll hier der Versuch einer kurzen Einführung unternommen werden, um auch dem Anfänger zu zeigen, in welche Richtung sich die Programmierung mit SQLWindows entwickeln wird.

Objekte und Quickies

Genaugenommen arbeiten Sie - ohne daß Sie etwas dazu tun müßten - in SQLWindows immer mit Objekten (Top- und Child-Level), die einen Teilaspekt der OOP darstellen. Hinzu kommen die *Quick-Objekte*, die Sie in die Lage versetzen, mit einem Minimum an Aufwand lauffähige Anwendungen zu erstellen. Drei Arten von Quick Objects sind in SQLWindows enthalten

- Quick Forms, die Formulare gestalten und sofort mit der Datenquelle verbinden;
- Quick Graphs, die es erlauben Business Graphic in SQLWindows-Anwendungen einzubinden, und
- Quick Menus, die es ermöglichen, ein Menüsystem anzufertigen.

Diese Module sind sehr einfach zu bedienen und zielen darauf ab, mit einem Minimum an Aufwand echte Client-Server-Anwendungen in kürzester Zeit zu codieren worin ein generelles Anliegen der OOP besteht.

Verallgemeinert können die *grundsätzlichen Ziele der OOP* folgendermaßen formuliert werden:

- Schnellere und einfachere Programmentwicklung;
- Erzeugung und Wiederverwendung von globalem Code, der - einmal codiert - immer wieder verwendet werden kann.

Vor- und Nachteile der OOP

Alle anderen Eigenschaften der OOP lassen sich aus den genannten Punkten ableiten, so z.B. folgende *Vorteile*:

- Zeitgewinn durch Wiederverwendung;
- stabilerer Code durch Wiederverwendung bereits getesteter, fehlerfreier Module;
- kompakterer Code.

Aber kein Feuer ohne Rauch; natürlich müssen durch die OOP auch *Nachteile* in Kauf genommen werden. Dies sind:

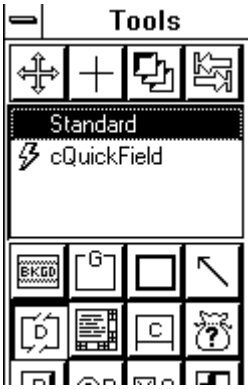
- höhere Anforderungen an den Entwickler bei der Erstellung von Anwendungen;
- durch Auslagerung von Code ist die Fehlersuche evtl. schwieriger, da zusätzlich in anderen Dateien gesucht werden muß.

Beispiel

Ein Beispiel soll erläutern, wie Sie die durch OOP zur Verfügung gestellten Mechanismen nutzen können.

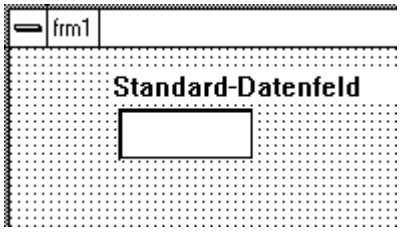
Von zentraler Bedeutung in der OOP ist der Begriff *Klasse*. Eine Klasse ist eine Struktur, in der Daten und Funktionen gespeichert werden können. Wenn Sie z.B. in einer SQLWindows-Anwendung ein Data Field in ein Formular einfügen, dann lassen Sie über die Tool-Palette zunächst die Standard-Klasse der Datenfelder anzeigen (eine andere haben Sie, abgesehen von den Quick-Klassen, nicht).

Abb. 3.1: Tool-Palette mit gewähltem Data Field und angezeigten Klassen (Standard-Klasse und Quick-Klasse).



Normalerweise fügen Sie ein Data Field ein, indem Sie zunächst auf den Data Field-Push Button klicken, danach den Mauszeiger in das Formular bewegen und dann einmal die linke Maustaste betätigen.

Abb. 3.2: Eingefügtes Data Field der Standard-Klasse.

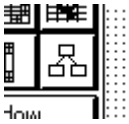


Sie haben damit ein Data Field der Standard-Klasse eingefügt, das bestimmte Eigenschaften hat. So ist ein frisch eingefügtes Data Field der Standard-Klasse z.B. grundsätzlich vom Datentyp String, hat eine bestimmte Breite und andere Eigenschaften, die über den Customizer eingestellt werden können (allerdings verändern Sie damit nicht die Klasse selbst, sondern allein das eingefügte Objekt).

Nun soll eine zusätzliche Klasse erzeugt werden, die (abweichend von der Standard-Klasse) andere Eigenschaften haben soll.

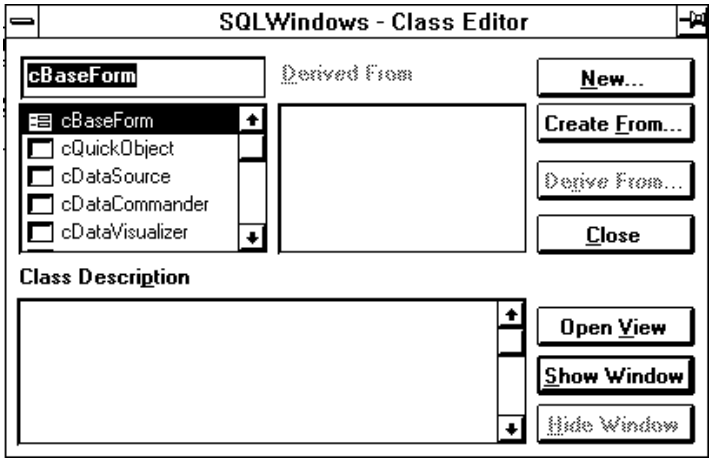
Um eine neue Klasse anzulegen muß der sogenannte Class-Editor gestartet werden. Zwei Möglichkeiten stehen Ihnen zu diesem Zweck zur Verfügung: , Sie können den Class-Editor entweder über den Menüpunkt "Tools", "Class Editor" starten oder indem Sie in der Tool-Palette den entsprechenden Push Button drücken, so wie es die folgende Abbildung zeigt.

Abb. 3.3: Tool-Palette mit Push Button für Class-Editor.



Nachdem Sie den Push Button gedrückt haben erhalten Sie ein Formular zur Verwaltung von Klassen.

Abb.: 3.4: Class-Editor.

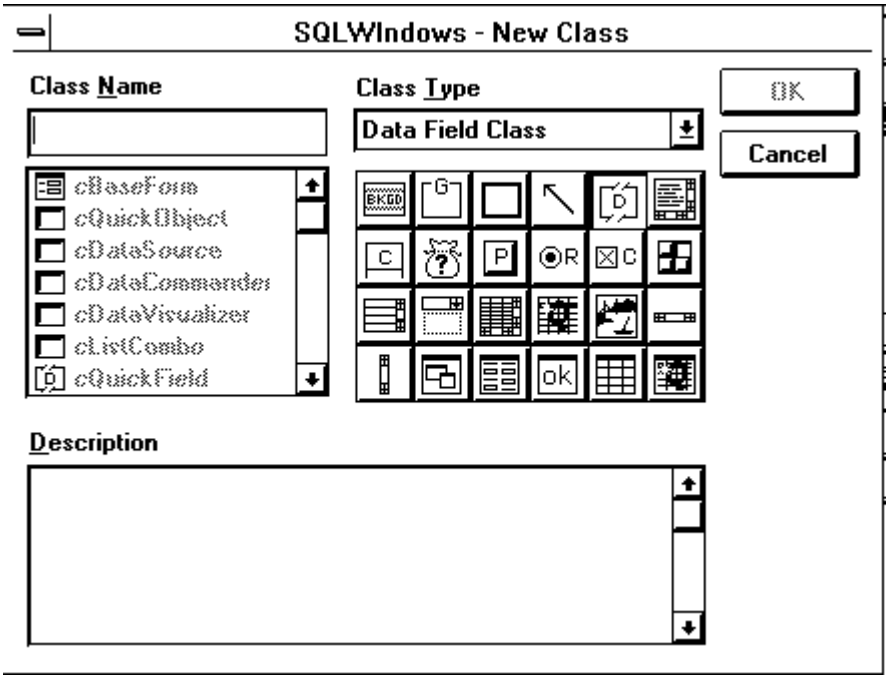


Es bestehen zwei Möglichkeiten, wie eine Klasse angefertigt werden kann: Entweder wird eine *neue* Klasse angelegt oder eine Klasse wird von einer (oder mehreren) vorhandenen Klassen *abgeleitet*. Hierfür sind rechts die Push Buttons "New" für neue bzw. "derive from" für eine abgeleitete Klasse vorgesehen.

In der links oben angezeigten List Box findet sich eine Übersicht aller vorhandenen Klassen (von Ihnen auszuwählen, wenn Sie eine Klasse *ableiten* möchten).

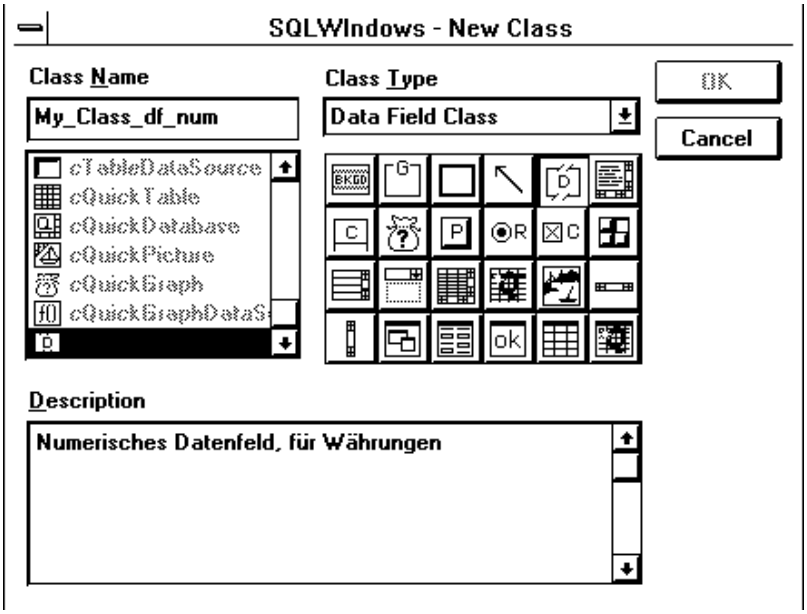
In diesem Beispiel soll eine ganz *neue Klasse* erzeugt werden: Klicken Sie dazu auf den Push Button "New", ohne eine Auswahl in der List Box zu treffen. Sie erhalten dann folgendes Formular:

Abb. 3.5: SQLWindows New Class für das Erzeugen einer neuen Klasse.



Wählen Sie hier aus der Combo Box (oben rechts) den Eintrag "Data Field Class" oder klicken Sie auf den dafür vorgesehenen Push Button (unten), tragen Sie danach in dem Namen für die Klasse (oben links) den Namen für die neue Klasse ein, er soll "My_Class_df_num" lauten, tragen Sie unten bei "Description" eine kurze Beschreibung ein, so wie es folgende Abbildung zeigt.

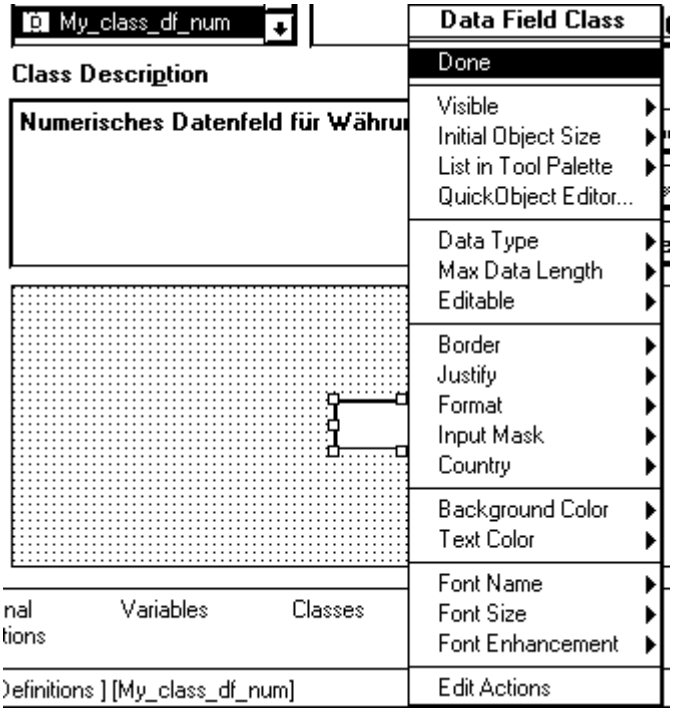
Abb. 3.6: Neuer Klassenname und Beschreibung.



Jetzt schließen Sie das Formular (OK-Push Button), Sie kehren damit zurück in den Class-Editor. Nachdem Sie die Art der Klasse ("Data Field") und den Namen der Klasse ("My_class_df_num") definiert haben, sollen nun deren Eigenschaften festgelegt werden.

Klicken Sie entweder auf den Push Button "Show Window" oder klicken Sie zweimal auf den angezeigten Namen der neu erzeugten Klasse. Daraufhin wird das Formular erweitert und es erscheint ein Data Field, mit dem Sie - ähnlich wie Sie das bereits bei "normalen" Data Fields getan haben - die Eigenschaften festlegen können.

Abb. 3.7: Festlegen der Eigenschaften einer Klasse mit dem Customizer.

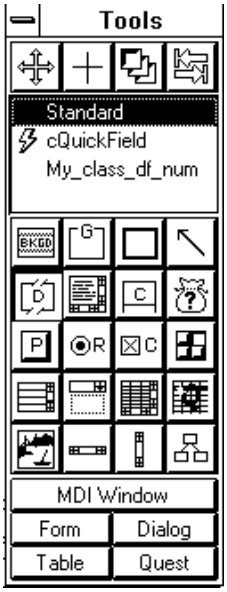


Das Ändern der Eigenschaften bezieht sich nun auf die Klasse selbst (und nicht wie früher auf das einzelne Objekt), d.h. wenn Sie nun die Feldbreite vergrößern oder den Datentyp ändern, dann ändern Sie die Eigenschaften der Klasse. Fügen Sie später ein Data Field dieser neuen Klasse in Ihre Anwendung ein, dann hat dieses Data Field genau die an dieser Stelle festgelegten Eigenschaften.

Um dies zu demonstrieren, verbreitern Sie das Data Field, legen den Data Type "Number" fest, wählen Justify "Right" und Format "Currency". Schließen Sie anschließend das Formular, indem Sie auf den "close" Push Button drücken. Damit ist die Definition einer neuen Klasse beendet.

Lassen Sie jetzt ein neues Form Window in Ihre Anwendung einfügen (Tool-Palette), klicken Sie anschließend auf den Data Field Push Button, dann werden Sie feststellen, daß innerhalb der Tool-Palette jetzt drei Klassen angezeigt werden: die **Standard-Klasse**, die **cQuickField-Klasse** und die **My_class_df_num-Klasse**. Die folgende Abbildung zeigt dies.

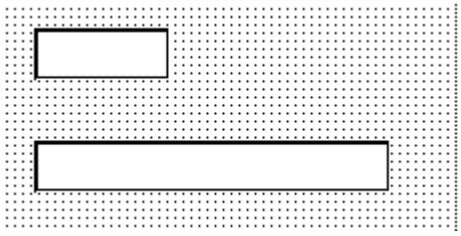
Abb. 3.8: Tool-Palette mit drei Klassen für Data Field.



Um ein Data Field in Ihr Formular einzufügen, klicken Sie zunächst auf der Standard-Klasse, bewegen dann den Mauszeiger in das Formular und drücken die linke Maustaste. Das erste Standard-Data Field ist damit eingefügt.

Jetzt klicken Sie erneut auf das Data Field-Symbol in der Tool-Palette, wählen aber diesmal die von Ihnen erzeugte Klasse ("My_Class_df_num") und fügen ein zweites Data Field in Ihr Formular ein.

Abb. 3.9: Data Field der Standard-Klasse (oben) und Data Field der selbst definierten Klasse (unten).



Zur Kontrolle können Sie sich mit dem Customizer die Eigenschaften beider Data Fields anzeigen lassen: Das obere enthält die Standard-Einstellungen, das untere dagegen weist die durch Sie veränderten Einstellungen (s.o.) auf.

Der folgende Code-Ausschnitt zeigt den Inhalt des Outline-Windows von SQLWindows.

- Form Window: frm1
  - Description:
  - Named Menus
  - Menu
  - Tool Bar
  - Contents
    - Data Field: df1
      - Message Actions
    - My_class_df_num: df2
      - Message Actions

Beide Data Fields können gleichrangig benutzt werden.

Zusammenfassend

Da Sie mit diesem Verfahren nicht nur Objekte (wie z.B. Data Fields) "kopieren" und mit neuen Eigenschaften ausstatten können, sondern zusätzlich auch Programmcode "verpacken" können, sind Sie in der Lage, außergewöhnlich leistungsfähige Module zu schaffen, deren Eigenschaften nur ein einziges Mal zu definieren sind. So ist es denkbar, daß Sie ein Table Window erzeugen, in dem alle notwendigen Code-Teile für die Eingabe, Sicherung und Löschung der Daten enthalten sind. In zukünftigen Anwendungen wäre dann nur noch diese von Ihnen erzeugte Klasse aufzurufen; zu codieren ist dann nur noch wenig.

Begriffe

Im folgenden sollen einige Bergriffe erläutert werden, die bei der weiterführenden Arbeit mit OOP gebraucht werden

Begriff	Erläuterung
Vererbung	Eine neue Klasse erbt die Eigenschaften einer vorhandenen Klasse, ohne daß diese Eigenschaften nochmals definiert werden müßten.
Polymorphismus	Es existiert eine Menge gleichnamiger Funktionen. Bei einem Funktionsaufruf wird automatisch die für das betreffende Objekt richtige Funktion ausgewählt. Dies erlaubt es, eine Gruppe gleichartiger Objekte (z.B. mehrere Form Windows) gemeinsam zu bearbeiten, ohne die sonst nötigen Fallunterscheidungen zu treffen.
Methode	Funktion einer Klasse, die eine Schnittstelle zur Umwelt herstellt.
Objekt	Instanz einer Klasse, eine Variable. Einzufügen mit: • Objekt in der Tool-Palette anklicken, • Klasse wählen, • Objekt einfügen.
Klasse	Eine Struktur, die Daten und Funktionen beinhaltet. Üblicherweise als Prototyp vorhanden. Durch Ableitung entstehen aus einer vorhandenen (Base-) Klasse neue Klassen-Objekte (s. Objekt).

3.2 Arbeiten im Netz

Es macht einen grundlegender Unterschied, ob Sie mit SQLWindows eine Anwendung für einen einzelnen Benutzer entwickeln oder ob Sie eine Anwendung entwickeln, die in einem Netzwerk lauffähig sein soll. Der Unterschied ergibt sich dadurch, daß im zweiten Fall grundsätzlich jeder Benutzer die in der verwendeten Datenbank gespeicherten Daten ändern kann (die Einschränkung dieses Rechtes durch SQL-Anweisungen sollen hier keine Rolle spielen). Deshalb muß bei einer Änderung vorhandener Daten immer geprüft werden, ob

- die Daten überhaupt geändert werden können (evtl. sind die Datensätze gesperrt) und
- die zu ändernden Daten eventuell schon von einem anderen Benutzer verändert wurden.

Voraussetzung für die Arbeit im Netz ist das Vorhandensein einer Multi-User Version von SQLBase und die Anpassung einer Anwendung an die besonderen Anforderungen, die sich durch den Zugriff mehrerer Benutzer auf den Datenbestand ergeben. Um die Arbeit im Netz zu *simulieren*, brauchen Sie allerdings nicht die genannte Multi-User Version von SQLBase. Es ist auch möglich, die Zugriffe mit der im Lieferumfang von SQLWindows oder Quest enthaltenen Single-User Multi-Tasking fähigen Datenbank (DBWSERVR.EXE) zu testen. Diese Datenbank kann (auf einem Rechner) mehrfach gestartet werden und bietet Ihnen alle Möglichkeiten für eine Test-Umgebung. Üblicherweise wird für Testzwecke die Datenbank einmal gestartet, die Entwicklungsumgebung SQLWindows (mit der zu testenden Anwendung) dagegen zweimal. Mit dem Task-Manager von Windows können Sie dann zwischen den beiden Anwendungen umschalten oder auch die zwei Anwendungen nebeneinander anzeigen lassen.

Folgende Begriffe sind in SQLWindows bei der Erstellung von netzwerkfähigen Anwendungen von zentraler Bedeutung:

- Isolation Level und die Funktion "SqlSetIsolationLevel()";

- die Sql-Anweisungen "commit" bzw. "rollback" und

- "ROWID", die eindeutige Bezeichnung einer Zeile aus der Datenbank-Tabelle (vgl. Kap. 2.1.2.4.1).

Anhand dieser Begriffe soll das Thema erläutert werden. Anschließend wird ein Beispiel zeigen, wie die Codierung bewerkstelligt werden kann.

3.2.1 Die Funktion SqlSetIsolationLevel()

Als Isolation Level wird in SQLWindows die Möglichkeit beschrieben, Daten und deren Änderung durch einen Benutzer von einem oder mehreren anderen Benutzern zu isolieren. Im Grunde heißt dies nur, wie stark Daten, die bei einem Benutzer angezeigt werden, von anderen Benutzern bzw. deren Möglichkeit die Daten zu ändern, zu trennen sind.

Die allgemeine Syntax der Funktion lautet:

Call SqlSetIsolationLevel(hSql, 'RR')

Dieser Funktionsaufruf, der häufig unmittelbar nach der Anbindung an die Datenbank mittels "SqlConnect()" anzufinden ist, beinhaltet den Namen des Handles, das für die Verbindung zur Datenbank vereinbart wurde ("hSql") und die Art des Isolation Levels, hier RR (=Read Repeatability). Dies ist die Voreinstellung (d.h. ohne den Aufruf der Funktion "SqlSetIsolationLevel()" wird diese Einstellung automatisch von SQLWindows gewählt). Auch bei einer Einzelplatzanwendung ist dies die Standardeinstellung. Hiermit werden die Daten, nachdem sie auf dem Bildschirm angezeigt sind, für alle anderen Benutzer solange gesperrt, bis ein "commit" sie wieder freigibt. Dabei ist es unerheblich, ob der Benutzer diese Daten ändert oder nicht, solange sie angezeigt werden und "commit" nicht an die Datenbank geschickt wurde, kann kein anderer die Daten lesen.

Daraus kann geschlossen werden, daß diese Einstellung kaum in einer Netzwerkumgebung eingesetzt werden wird. Sie bietet zwar maximale Sicherheit, behindert aber die Arbeit.

Um in einem Netz zu arbeiten, bieten sich insgesamt vier Möglichkeiten, den Isolation Level festzulegen:

- **RR** (Read Repeatability): Alle angezeigten Datensätze (einer oder viele) bleiben für andere Benutzer solange gesperrt, bis ein "commit" die Daten wieder freigibt. Dabei ist es unerheblich, ob die Daten geändert wurden! Diese Einstellung ist die Voreinstellung und wird in Einzelplatzanwendungen verwendet.

- **RL** (Release Locks): Keine automatische Sperren, der Programmierer muß die Sperren selbst setzen. Empfohlen für die Tabellen-Anzeige von Datensätzen, minimaler Netzwerkverkehr.

- **RO** (Read Only): Es werden keine Datensätze gesperrt. Sie können alle gelesen, nicht aber verändert werden (DDL- und DML-Operationen von Sql sind nicht möglich).

- **CS** (Cursor Stability): Nur die Datensätze, die gerade auf dem Bildschirm angezeigt werden, sind für die anderen Benutzer gesperrt (shared lock). Andere, zuvor angezeigte Datensätze sind für die anderen

Benutzer nicht gesperrt. Aufwendig für Netzwerke, da ständig Meldungen zwischen Client und Server hin- und hergeschickt werden müssen.

Es läßt sich zwar keine Empfehlung für alle Fälle treffen, doch wird häufig die Benutzung der **RL**-Einstellung **sinnvoll** sein. Der folgende Code-Auschnitt zeigt ihre Codierung:

```
◆ Application Actions
  ◇ ! Verbindung zur Datenbank
  ◆ On SAM_AppStartup
    ◇ Set SqlDatabase = 'FIRMA'
    ◇ Call SqlConnect(hSql)
    ◇ Call SqlSetIsolationLevel( hSql, 'RL' )
```

Dabei sind die Situationen durchaus nicht immer auf Anhieb richtig einzuschätzen, so daß vor der Codierung zunächst eine sorgfältige Analyse stattfinden sollte, in welcher Weise die Sicherheit oder Konsistenz der Daten zu gewährleisten ist. Denn nicht immer müssen Sperrungen nötig sein; es muß aber grundsätzlich geprüft werden, ob z.B. nach einer Insert- oder Update-Operation die Daten überhaupt in der Datenbank angekommen sind. Der Erfolg kann geprüft werden, indem der Rückgabewert z.B. der Funktion "SalTblDoDeletes()" in einer logischen Variablen gespeichert wird ("Set bOk = Call SalTblDoDeletes(..)"). Abhängig von deren Inhalt wird dann eine Meldung an den Benutzer angezeigt.

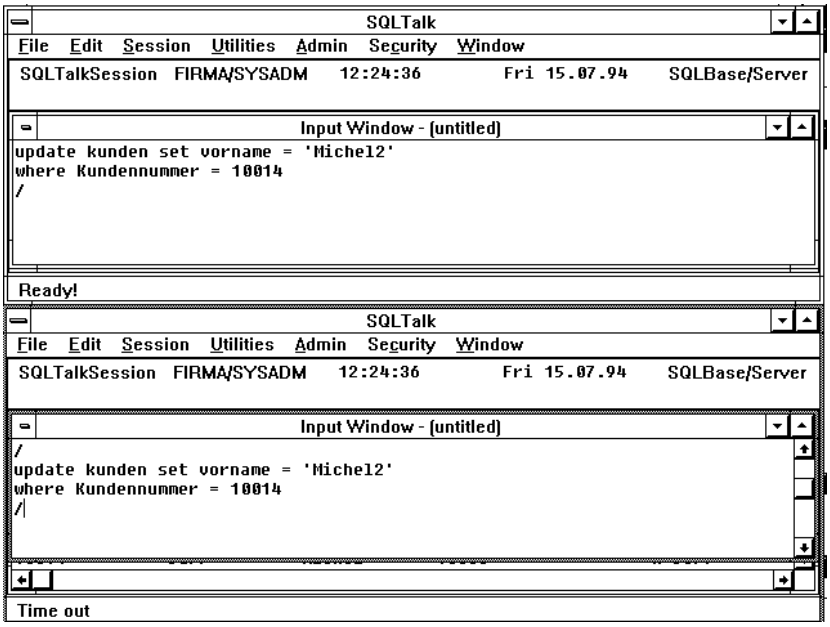
3.2.2 Commit, Rollback

Bereits im vorigen Abschnitt wurde darauf hingewiesen, daß die Commit - und Rollback-Anweisungen in einer Netzwerk-Umgebung von großer Bedeutung sind. Sie können durchaus eine funktionierende Einzelplatzanwendung schreiben, die ohne diese Anweisungen auskommt, dagegen *müssen* Sie in einem Netzwerk diese Anweisungen benutzen.

Zur Verdeutlichung soll ein kurzes Beispiel einen Fehler erzeugen, der durch das Fehlen der Commit-Anweisung verursacht wird. "SQLTalk" muß hierfür zweimal gestartet werden, in beiden Sitzungen muß dann über F2 eine Verbindung zur Datenbank Firma hergestellt werden.

Danach soll in beiden Sitzungen der gleiche Datensatz geändert werden, so wie das die folgende Abbildung darstellt:

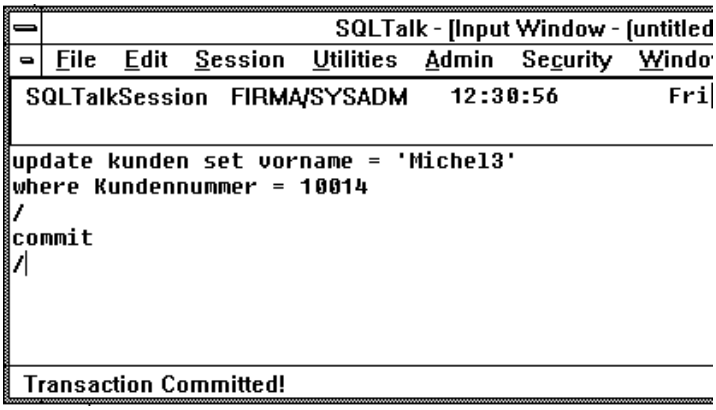
Abb. 3.10: SQLTalk, Time out-Fehler nach wiederholtem Versuch, einen Datensatz zu ändern (SQLTalk wurde dazu zweimal gestartet).



Der "Time out" - Fehler (unten links angezeigt) wird dann erzeugt, wenn zum zweitenmal versucht wird, die Daten der gleichen Tabelle zu ändern. Dabei ist es unerheblich, ob die gleiche Sql-Anweisung zweimal oder zwei verschiedene Sql-Anweisungen ausgeführt werden; ebenso ist es ohne Belang, ob sich die Änderung auf den gleichen Datensatz oder auf verschiedene Datensätze bezieht.

Um zu vermeiden, daß es hierzu kommt, muß nach einer Anweisung zur Änderung von Daten ("update", "insert", "delete"), die Änderung jeweils mit "commit" fixiert werden. Wenn Sie dies nach den jeweiligen Änderungen tun, erhalten Sie keine "Time out" - Fehlermeldungen.

Abb. 3.11: Erfolgreiche Änderung von Daten durch Verwendung der "commit" - Anweisung.



Beachten Sie auch die Meldung "Transaction Commited" am unteren Bildrand, die Ihnen mitteilt, daß die Daten gesichert wurden. Mit diesem Mechanismus kann sichergestellt werden, daß Änderungen an den Daten sicher durchgeführt werden.

Manchmal aber soll überhaupt nichts geändert werden, dann z.B. wenn ein angezeigter Datensatz zwischenzeitlich durch einen anderen Benutzer verändert wurde. Ob das in Ihren Anwendungen der Fall sein wird, muß jeweils geprüft werden. Die folgende Übersicht liefert (getrennt für Stamm- und Bewegungsdaten) hierzu Anhaltspunkte.

	Sql-Anweisung		
	insert	update	delete
Tabellenart Stammdaten (Master)	zuerst neuen Primärschlüssel bilden, sofort sichern, andere Felder später mit update einfügen	unter Verwendung des Primärschlüssels (oder der ROWID)	unter Verwendung des Primärschlüssels (oder der ROWID)
Bewegungsdaten (Detail)	einfaches insert mit folgendem commit	nur mit ROWID möglich	unter Verwendung des Fremdschlüssels oder der ROWID

Zur Erinnerung sind in den folgenden Übersichten die Tabellen (Kunden, Lager, Lieferanten, Kontroll einerseits und Auftrag andererseits) kurz dargestellt.

Folgende Tabellen enthalten Primärschlüssel. Bei Ihnen ist es von besonderer Bedeutung, daß die Schlüssel (Kundennummer, Artikelnummer, Lieferantennummer) eindeutig sein müssen!

Bei allen Tabellen wird eine Datenänderung über die "ROWID" oder den Primärschlüssel realisiert:

Tabelle	Besonderheit
Kunden	Primärschlüssel ist die Kundennummer.
Lager	Primärschlüssel ist die Artikelnummer.
Lieferanten	Primärschlüssel ist die Lieferantennummer.
Kontroll	Je Auftrag wird ein Datensatz gesichert, der die Auftragsnummer, die Kundennummer und zusammenfassende Informationen zum Auftrag enthält (aber nicht die Daten der abgegebenen Artikel). Primärschlüssel ist die Auftragsnummer.

Die Datenänderung wird im folgenden über die "ROWID" oder den Fremdschlüssel (z.B. die Auftragsnummer) verwirklicht:

Tabelle	Besonderheit
Auftrag	Detaildaten der einzelnen Aufträge, hier werden Fremdschlüssel (Auftragsnummer, Kundennummer, Artikelnummer) und die abgegebenen Mengen gespeichert.

Bei den dargestellten Tabellen kommen Änderungen und Neue Eingaben von Daten in Betracht. Hierzu müssen einige Mechanismen entwickelt werden, die gewährleisten, daß z.B. die Primärschlüssel tatsächlich eindeutig sind.

Hier soll erinnert werden, daß die in dem früheren Kapitel 2.1.2.4.3 geschilderte Vorgehensweise bei der Vergabe von Kundennummern, die lokal orientiert war, im Netz nicht zu gebrauchen ist. Dort waren neue Kundennummern lokal, also auf dem einzelnen Rechner hochgezählt worden. Würden bei dieser Codierung in einem Netz zwei Benutzer neue Kunden eingeben, dann würde, nachdem zunächst die in der Tabelle vorhandene größte Kundennummer festgestellt wurde (mit der Anweisung **SqlImmediate('select max(kundennummer)+1 into :df1**') jeder der beiden Benutzer für sich hochzählen, d.h. es würden identische Kundennummern für unterschiedliche Kunden vergeben. Dies muß natürlich verhindert werden (analog gilt dies für alle Tabellen, in denen Daten mit Primärschlüsseln verwaltet werden)!

Bei der Auftragserfassung ist die Sachlage ähnlich: Nachdem eine neue Auftragsnummer erstellt wurde (indem aus der Kontrolltabelle die Nummer des letzten Auftrags um eins erhöht wurde), muß diese neue Auftragsnummer sofort in die Tabelle mit den Kontrolldaten geschrieben werden (obwohl der Auftrag noch gar nicht fertig erstellt ist), so daß ein anderer Benutzer, der ebenfalls einen neuen Auftrag erfaßt (wiederum indem er die Nummer des letzten Auftrags ermittelt und die Zahl um eins erhöht) tatsächlich eine neue, bisher nicht benutzte Auftragsnummer erhält.

Zusammenfassend kann also festgestellt werden, daß die Erzeugung neuer Primärschlüssel ein sofortiges Sichern des Primärschlüssels zur Folge haben muß, so daß andere Benutzer, die das gleiche tun, ebenfalls neue, eindeutige Primärschlüssel erhalten.

Daneben muß bei Korrektur der Daten immer dafür gesorgt werden, daß die korrigierten Daten richtig zugewiesen werden, daß z.B. nach der Änderung einer Kunden-Telefonnummer tatsächlich nur die Daten bei

genau diesem Kunden verändert werden. Dies ist aber durch die vorhandenen Primärschlüssel nicht besonders schwierig. Etwas komplexer ist dagegen die Änderung der Detail-Daten eines Auftrags, wenn z.B. nach einer fehlerhaften Buchung eine Artikelnummer verändert wurde, oder wenn einem Auftrag eine oder mehrere zusätzliche Zeilen angefügt werden. In diesen Fällen können die bereits vorhandenen Daten gelöscht oder verändert und die neuen Daten eingefügt werden. Am einfachsten wird es wohl sein, alle in der Tabelle gespeicherten Detail-Daten, nachdem sie auf dem Bildschirm zur Änderung angezeigt werden, in der Tabelle zu löschen und danach die aktualisierten Daten neu in die Tabelle einzufügen (dabei ist aber zu beachten, daß z.B. die Lagerbestände aktualisiert werden müssen). In Kap. 3.2.3 ist dies als Beispiel dargestellt.

3.2.3 Beispiel

3.2.3.1 Stammdaten aus Netz2.app

Nach dem Drücken eines ersten Push Buttons wird ein neuer Datensatz so angelegt, daß die neue Kundennummer aus der Kundentabelle geholt und sofort gespeichert wird. Dieser neue Datensatz hat zunächst nur einen Inhalt in der Spalte mit der Kundennummer, die restlichen Inhalte werden in einem späteren Schritt eingefügt!

```
◆ Pushbutton: pb1                                   ! neuer Satz
  ◆ Message Actions
    ◆ On SAM_Click
      ◇ Call SalTblDefineSplitWindow( tbl1, 3, TBL_Split_Adjustable )
      ◇ Set nNewRow=SalTblInsertRow(tbl1, TBL_MinRow)
      ◇ Call SqlImmediate( 'select max(KUNDENNUMMER)+1
        into :colKdnr from Kunden'  )
      ◇ Call SqlImmediate( 'insert into kunden
        (kundennummer)
        values
        (:colKdnr)'  )
      ◇ Call SqlImmediate( 'commit' )
      ◇ Call SalTblSetFocusCell(tbl1, nNewRow,colName,-1, -1)
      ◇ Set nAnzahlNeueSaetze =nAnzahlNeueSaetze + 1
```

Die wichtigste Zeilen in dem vorstehenden Code-Ausschnitt sind die folgenden:

- **Call SqlImmediate('select max(kundennummer)+1) into :colKdnr from Kunden')**, diese Zeile liefert eine neue Kundennummer und schreibt diese in die Spalte "colKdnr" der Tabelle.

- **Call SqlImmediate('insert into kunden (kundennummer) values (:colKdnr)')**, hiermit wird die neu erzeugte Kundennummer sofort in die Tabelle Kunden gespeichert, so daß kein anderer Benutzer die gleiche Kundennummer erzeugen kann.

Der folgende Programmausschnitt zeigt den Code, der dafür sorgt, daß beliebig viele neue Kunden im Netz richtig gesichert werden (die Erläuterung folgt im Anschluß daran).

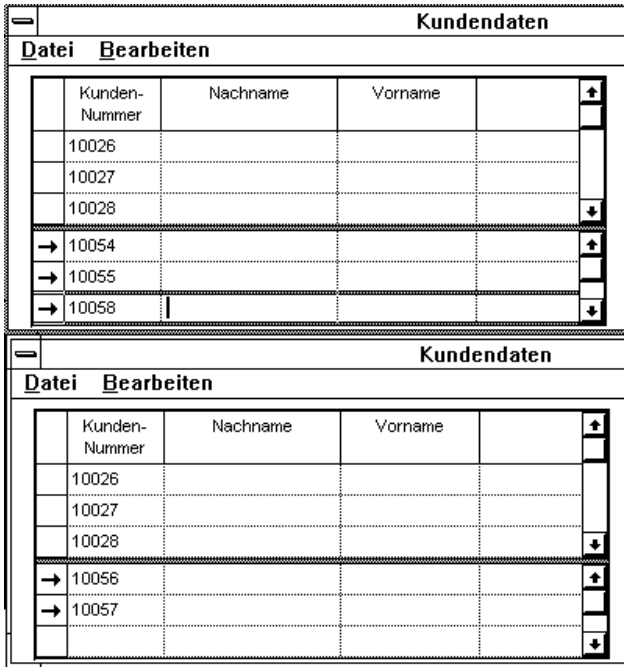
```
◆ Pushbutton: pb5                          ! sichern im Netz
  ◆ Message Actions
    ◆ On SAM_Create
      ◇ Set strPopulateKunden =
          'select ROWID, KUNDENNUMMER, NAME, VORNAME
                  from KUNDEN order by kundennummer
                   into :colRowId, :colKdnr, :colName, :colVorname'
    ◆ On SAM_Click                  ! unten sichern
      ◇ Set nCurrentRow = TBL_MinSplitRow
      ◆ While nAnzahlNeueSaetze > 0
        ◇ Call SalTblSetFocusRow(tbl1, nCurrentRow)
        ◇ Call SqlImmediate('update kunden
          set vorname =:colVorname, name = :colName
          where kundennummer = :colKdnr')
        ◇ Call SqlImmediate('commit')
        ◇ Set nAnzahlNeueSaetze = nAnzahlNeueSaetze -1
        ◇ Set nCurrentRow=nCurrentRow+1
      ◇ Call SalTblDefineSplitWindow( tbl1, 0, TBL_Split_Adjustable )
      ◇ Call SalTblPopulate(tbl1, hSql,strPopulateKunden , TBL_FillNormal )
```

Die Aktionen sind einem Push Button (pb5) zugeordnet. Von grundlegender Bedeutung ist die Verwendung der While-Schleife, in der die neu eingegebenen Datensätze abgearbeitet werden. Dabei wird eine Variable ("nAnzahlNeueSaetze") benutzt, in der die Anzahl der neu eingegebenen Kunden gespeichert wurde (s. hierzu den davor stehenden Code-Ausschnitt). Innerhalb der Schleife werden die Datensätze jeweils mit einer "Update" - gefolgt von einer "Commit" - Anweisung in die Tabelle geschrieben.

Zu beachten ist hierbei, daß *Änderungen* bereits angezeigter Daten (in dem oberen Teil der Tabelle) mitgesichert werden (dies nur um den Programmcode möglichst kurz zu halten.

Die folgende Abbildung zeigt das Programm zur Neueingabe von Kundendaten im Netz, nachdem es zweimal gestartet wurde. Zu beachten ist, daß die neu vorgeschlagenen Kundennummern (im oberen und unteren Fenster) zu eindeutigen Primärschlüsseln führen (ignorieren Sie dabei, daß die Spalten für Vor- und Nachnamen keine Zeichen enthalten).

Abb. 3.12: Anwendung, zweimal gestartet mit neuen Kundennummern.



Beachten Sie die in den zwei Fenstern neu vorgeschlagenen Kundennummern (im unteren Teil der Tabellen) unterschiedlich sind, so wie das von einer Anwendung im Netz verlangt wird.

3.2.3.2 Bewegungsdaten

Die Verarbeitung von zu sichernden Bewegungsdaten im Netz ist dem für Stammdaten geschilderten Verfahren sehr ähnlich. Deshalb sollen hier nur Stichworte das Verfahren zur Sicherung dieser Daten zusammenfassen:

- Neue Auftragsnummer erzeugen: **Call SqlImmediate('select max(aufnr) into :dfAufnr from kontroll')**, neue Auftragsnummer sofort in der Tabelle mit Kontrolldaten speichern.
- Erfassung der **Bewegungsdaten** in einem Table Window.
- Speichern der Bewegungsdaten mit **SalTblDoInserts()**, danach mittels **Commit** fixieren.
- Speichern der **Kontrolldaten** mit **update**, anschließen fixieren mit **Commit**.

Damit werden die Daten zuverlässig in die Auftragstabelle bzw. Kontrolltabelle geschrieben. Zu überlegen ist, ob nach erfolgreicher Sicherung der Daten dies in der Statuszeile angezeigt werden sollte.

4 Anhang

Aufgabe des Anhangs ist es, in übersichtlicher Form stets erneut vorkommende Begriffe knapp und konzentriert auf ihren maßgeblichen Gehalt zu erläutern. Hier finden Sie Angaben zu der in diesem Buch verwendeten Datenbank "FIRMA", eine Übersicht wichtiger Begriffe die in Zusammenhang mit SQL bzw. SQLWindows stehen, Befehle der Sprache SQL, Funktionen der Sprache SAL und die in SQLWindows verwendeten Nachrichten.

4.1 Datenbank "FIRMA", Struktur und Datensätze der verwendeten Tabellen.

Die Datenbank FIRMA enthält eine Reihe von Tabellen, deren Spalten sich zum einen durch die Anforderungen des Kunden (bzw. der bei ihm zu verarbeitenden Daten) und aus Gründen der Normalisierung der Daten ergeben. So ist die folgende Übersicht nicht als allgemeine Lösung, sondern als Ansatz zu verstehen, aus dem heraus einzelne individuelle Lösungen zu erarbeiten sind. In den folgenden Abschnitten werden neben den Tabellenstrukturen auch jeweils einige Datensätze genannt.

Achten Sie darauf, daß bei Ihrer späteren Arbeit keine Sonderzeichen verwenden, auch dann nicht, falls dies einmal möglich sein sollte. Das heißt, daß für den Spaltennamen "STRASSE" auf keinen Fall das Zeichen "ß" verwenden, sondern "ss". Verzichten Sie bei Spaltennamen auch auf Umlaute.

Beachten Sie außerdem, daß die Strukturen der Tabellen nur für das vorliegende Buch gewählt wurden. Im allgemeinen müssen Sie die Spaltenbreiten für "echte" Anwendungen deutlich vergrößern, außerdem ist zu prüfen, wie eine maximal mögliche Anzahl von Information in den Stammdaten (Kunden-, Lager- und Lieferantendaten) bei möglichst geringer Redundanz in den Tabellen der Buchungsdaten (Auftragsdaten, Kontrolldaten) zu speichern sind.

4.1.1 Kundentabelle

Die Kundentabelle enthält die Informationen zu den Kunden.

Spaltenname	Information	Datentyp	Länge
KDNR	Kundennummer	integer	
NAME	Name des Kunden	character	10
VORNAME	Vorname des Kunden	character	10
PLZ	Postleitzahl des Wohnorts	character	5
STRASSE	Name der Strasse	character	10
ORT	Wohnort	character	10
TELEFON	Telefonnummer	character	10
KUNDENGRUPPE	Nummer der Kundengruppe	small integer	

Primärschlüssel: KDNR, Fremdschlüssel: KUNDENGRUPPE.

Folgende Datensätze könnten in der Kundentabelle enthalten sein:

Kdnr	Name	Vorname	Plz	Ort	Strasse	Telefon	Kundengruppe
100	Kohler	Erich	5000	Kusel	Gasse 1	012-554	1
101	Mopper	Karl	42481	Kehl	Gasse 3	082-112	2
102	Locker	Hans	12345	Alsfeld	Weg 3	045-122	1

Genaugenommen müßte in einer weiteren Tabelle definiert sein, was unter der Kundengruppe 1 oder 2 zu verstehen ist (z.B. Großkunden oder Privatkunden).

4.1.2 Lagertabelle

In dieser Tabelle werden alle am Lager befindlichen Artikel erfaßt. Sie hat folgende Struktur:

Spaltenname	Information	Datentyp	Länge	Dezimal
ARTNR	Artikelnummer	integer		
NAME	Name des Artikels	character	10	
EKPR	Einkaufspreis	decimal	10	2
VK1	Verkaufspreis (Kundengruppe 1)	decimal	10	2
VK2	Verkaufspreis (Kundengruppe 2)	decimal	10	2
VK3	Verkaufspreis (Kundengruppe 3)	decimal	10	2
LIEFERANT	Nummer des Lieferanten	integer		

Primärschlüssel: ARTNR, Fremdschlüssel: LIEFERANT.

In dem einführenden Kapitel wurden in diese Tabelle folgende Sätze eingegeben:

Artnr	Name	Ekpr	Vk1	Vk2	Vk3	Lieferant
1000	Hammer	12,00	19,52	17,36	16,52	10020
1001	Kelle	11,70	12,20	11,90	10,10	10085
1002	Tafel	21,50	40,20	35,20	29,60	18452

Die in der Spalte Lieferant angegebene Nummer wird in der Tabelle mit den Lieferantendaten definiert.

Es ist denkbar, daß eine zusätzliche Tabelle für Sonderpreise eingerichtet wird. Damit könnten jedem Kunden beliebig viele solcher Preise zugeordnet werden (diese Tabelle würde nur drei Spalten enthalten: Artikelnummer, Kundennummer, Sonderpreis).

4.1.3 Lieferantentabelle

Die Lieferantentabelle enthält die Stammdaten der Lieferanten. Wichtig hierbei der Inhalt der Spalte "Liefnr", die die Nummer des Lieferanten enthält. Über sie wird die Information aus der Lagertabelle mit dieser Tabelle verknüpft.

Spaltenname	Information	Datentyp	Länge
LIEFNR	Nummer des Lieferanten	integer	
NAME	Name des Lieferanten	character	10
TELEFON	Telefonnummer	character	10

Primärschlüssel: LIEFNR.

Diese Tabelle enthält folgende Datensätze:

Liefnr	Name	Telefon
100000	Meier	089-123
120220	Keller	056-332
124842	Müller	012-123

Die Struktur einer solchen Tabelle hat oft große Ähnlichkeit mit der Struktur von Kundentabellen.

4.1.4 Auftragstabelle

In dieser Tabelle werden ausschließlich die in einem Auftrag enthaltenen Artikelnummern, Stückzahlen, Einzelpreise und die Auftragsnummer gespeichert. Sie dient damit nicht nur der Speicherung der in einem Auftrag enthaltenen Daten, sondern hilft auch Statistiken, z.B. nach Kunden zusammengefaßte Umsätze anzufertigen.

Spaltenname	Information	Datentyp	Länge	Dezimal
AUFNR	Nummer des Auftrags	integer		
ARTNR	Artikelnummer	integer		
MENGE	abgegebene Menge	integer		
E_PREIS	Einzelpreis	decimal	10	2

Die Daten für diese Tabelle werden im Kapitel 2 des Handbuchs erzeugt (sie können über die Sql-Anweisung "select * from auftrag" z.B. in "SQLTalk" angezeigt werden). Folgende Datensätze könnten in dieser Tabelle enthalten sein.

Aufnr	Artnr	Menge	E_preis
1	1000	1	12,00
1	1001	2	19,00
1	4002	1	87,00
2	1003	1	14,00
2	2005	1	19,00
3	2001	1	16,00
3	1001	1	14,00

Beachten Sie, daß die Gesamtsumme eines Auftrags nicht in dieser Tabelle sondern in der Kontroll-Tabelle gespeichert wird (ebenso die Auftragsart, d.h. ob Rechnung, Lieferschein oder Barverkauf)

4.1.5 Kontrolltabelle

Die Tabelle mit den Kontrolldaten enthält für jeden gebuchten Auftrag (der ja seinerseits aus mehreren Zeilen besteht) genau eine Zeile. Hierin wird gespeichert, für welchen Kunden der Auftrag erfaßt wurde, wann der Auftrag gebucht wurde und ob es sich um eine Rechnung, einen Lieferschein oder um eine Sofort-Barzahlung handelt, also Eigenschaften, die *nicht* den einzelnen Artikeln, sondern dem Auftrag insgesamt zugeordnet werden.

Spaltenname	Information	Datentyp	Länge	Dezimal
AUFNR	Nummer des Auftrags	integer		
KDNR	Kundennummer	integer		
DATUM	Datum des Auftrags	date		
GESAMT	Gesamtbetrag	decimal	10	2
ART	Lieferschein/Rechnung	character	1	

Primärschlüssel: AUFNR, Fremdschlüssel: KDNR.

Folgende Datensätze sind - nach Speicherung der Detail-Daten in der Auftragstabelle - in dieser Kontrolltabelle enthalten.

Aufnr	Kdnr	Datum	Gesamt	Art
1	100	12.5.94	137,00	R
2	101	12.5.94	33,00	R
3	102	13.5.94	30,00	L

Bei konsequenter Normalisierung müßte in einer weiteren Tabelle definiert sein, was unter "R" und "L" (bzw. "B") zu verstehen ist.

Auch die Daten dieser Tabelle werden durch in Kapitel 2 erzeugt. Sie können sich über die Sql-Anweisung "select * from kontroll" die dabei angelegten Daten anzeigen lassen.

4.2 Begriffe

Abfragesprache	Sprache, die dazu dient, Informationen aus einer Datenbank abzufragen. Dazu wird ein festgelegter Wortschatz verwendet, der im Fall der Abfragesprache SQL unabhängig von der verwendeten Plattform (DOS/Windows, OS/2, VMS etc.) und sogar unabhängig von der verwendeten Datenbank-Software (SQLBase, Oracle, Informix, Omnis etc.) ist.
ANSI	Abkürzung für American National Standard Institute. Eine nationale Behörde der Vereinigten Staaten, die technische Standards für Hard- und Software festlegt. Diese Behörde ist vergleichbar dem Deutschen Institut für Normung e.V. (DIN).
Back-End	s. Front-End.
Bind-Variable	Variable (häufig eine Gruppe von Variablen), die dazu benutzt wird, die Daten zwischen der Datenbank und der Anwendung hin und her zu schicken. Auch beim Druck von Reports müssen Bind-Variables verwendet werden, um die Daten zu dem Report-Template zu schicken.
Cache	Zwischenspeicher, der nach dem Lesen der Daten von einer Datenbank mit diesen gefüllt wird. Auch geänderte Daten werden zunächst im Cache gespeichert.
Client	Computer oder Anwendungsprogramm, der oder das Ressourcen benutzt, die von einem anderen Computer (oder Anwendungsprogramm) zur Verfügung gestellt werden.
Client-Server-Modell	Aufteilung der Datenbank und der Bedieneroberfläche auf zwei Module. Im Back-End (der Datenbank) werden die Daten verwaltet, das Front-End (die Bedieneroberfläche) erlaubt das Definieren, Eingeben und Verändern der Daten. Client und Server müssen zwar nicht getrennt sein, häufig ist es aber so, daß die Daten auf dem Server im Netz von vielen Clients benutzt werden.
Commit	Sql-Anweisung, die zur "Fixierung" der zuletzt durchgeführten Änderung in der Datenbank führt. Danach können diese Änderungen nicht mehr mit einer "Rollback"-Anweisung zurückgenommen werden. Ein Commit entfernt gleichzeitig alle vorhandenen Sperren ("locks"), so daß in einem Netzwerk die anderen Benutzer mit den Daten arbeiten können.
Cursor	Der Begriff "cursor" wird in zwei verschiedenen Bedeutungen verwendet. Zum einen ist es die englische Bezeichnung für den Mauszeiger, zum anderen existiert in der Sprache Sql der Begriff "cursor" dafür, daß eine durch Select erzeugte Ergebnismenge einen Namen erhält (durch diese Ergebnismenge kann man sich satzweise bewegen).
Data Dictionary	In diesem "Wörterbuch" werden Informationen zu Tabellen, Views, Benutzern, Zugriffsrechten etc. abgelegt. Man könnte es auch als Sammlung von Tabellen über Tabellen bezeichnen. Ein anderer Begriff hierfür ist System-Katalog.
Datenbank	Sammlung von Daten mit dem Ziel, diese Daten einem oder mehreren Benutzern zur Verfügung zu stellen.

Datenbank-Administrator	Person, die besondere Rechte innerhalb der Datenbank besitzt. Nur sie kann entscheiden wer z.B. Zugriff auf welche Daten erhält.
Datenbank-modelle	Formale Beschreibung der Daten und ihrer Beziehungen untereinander. Man unterscheidet verschiedene Organisationsformen: • hierarchische, • relationale, • semantische, • objektorientierte und • Netzwerk-Datenbankmodelle.
Datenintegrität	Bei redundanten Daten (z.B. mehrfach erfaßten Adressen) ist es oft nur schwer möglich, eine Änderung durchzuführen. Die Daten sind häufig nicht auf dem gleichen Stand, die Datenintegrität ist gefährdet.
Datentyp	Der Datentyp definiert, welcher Art die Daten sind, die in der Spalte einer Tabelle gespeichert werden sollen. Im einfachsten Fall wird entschieden ob Text (Datentyp Zeichen) oder numerische Information (Datentyp Numerisch) abgelegt werden sollen. Andere Datentypen können Datumswerte, Zeitwerte oder Objekte beinhalten.
DBMS	Abkürzung für Datenbank-Management-System, eine Software, die Daten einer Datenbank einem oder mehreren Benutzern zur Verfügung stellt.
DCL	Abkürzung für Data Control Language. Dieser Name steht für die Gruppe von Anweisungen bezüglich des Anlegens und Löschens von Datenbankrechten (s. Befehle: grant, revoke).
DDL	Abkürzung für Data Definition Language. Dies sind Befehle zum Anlegen und Löschen von Datenbankobjekten (Tabellen, Views; s. Befehle: create, drop).
DML	Abkürzung für Data Manipulation Language. SQL-Anweisungen zum Aktualisieren und Durchsuchen von Tabellendaten; s. Befehle: select, insert, delete, update).
Embedded SQL	SQL-Anweisungen, die eingebettet (embedded) in anderen Programmiersprachen (z.B. C) verwendet werden.
Entität	Summe aller Merkmalsträger, oder Eigenschaften: In einer Kundentabelle die Menge aller Kundennummern, Vornamen, Nachnamen usw., die den Kundenstamm charakterisieren. Abgeleitet ist der Begriff "Entity Relationship", das die Grundlagen für die inhaltliche Entwurfsarbeit bei der Anlage von Tabellen definiert.
Feld	Kleinstes Element einer Tabelle, das eine Information enthält. Die Menge aller Felder untereinander ergibt die Spalte einer Tabelle, die Menge aller Felder einer Zeile ergibt einen Datensatz.
Foreign Key	Fremdschlüssel. Feldinhalt einer Tabelle, der bei einer anderen Tabelle als Primärschlüssel verwendet wird. Foreign Keys werden verwendet, um Tabellen zu verknüpfen (s.a. Primary Key).

Front-End	Software, die auf dem Rechner eines Clients für die Datenverarbeitung eingesetzt wird und die mit einem Back-End, d.h. einer Software die auf dem Server läuft, zusammenarbeitet (dabei können Client und Server durchaus auf einem einzigen Rechner installiert sein).
Hierarchische Datenbank-modelle	Eine Datenbank, organisiert als Baumstruktur. Die Relationen zwischen den einzelnen Datenmengen sind eindeutig und fest definiert.
Handle	Referenz auf ein Objekt. Etwas unsauber mit "Griff" zu übersetzen. Erlaubt die eindeutige Identifikation eines Objekts. Häufig werden bei SAL-Funktionsaufrufen Handles verlangt. Oft kann man auch anstelle eines Handles den Namen eines Objekts angeben.
Index	Eine Ordnungsstruktur, die zusätzlich zu einer Tabelle angelegt wird. Damit wird die Zugriffszeit reduziert. Dieser Index sortiert jedoch die Daten der Tabelle nicht.
Integriertes Datenbanksystem	Gegenteil von Client-Server-Systemen: Die Bedieneroberfläche und die eigentliche Datenbank befinden sich auf ein- und demselben Rechner (so z.B. bei dBASE).
Isolation Level	Umfang, mit dem geregelt wird, wie Datenbank-Operationen eines Benutzers von den Datenbank-Operationen eines anderen Benutzers isoliert werden. Es sind insgesamt vier Isolation Levels bekannt (RR: Read Repeatability; CS: Cursor Stability; RL: Release Locks; RO: Read only).
Kartesisches Produkt	Bei der Verbindung zweier (oder mehrerer) Tabellen in einer Abfrage werden (falls nicht einschränkende Bedingungen angegeben sind) alle möglichen Kombinationen aller Zeilen angezeigt: Enthält eine Tabelle 5 Zeilen, die andere Tabelle 3 Zeilen, so werden insgesamt 15 Zeilen angezeigt. Üblicherweise kann man mit dieser Kombination der Datensätze nichts anfangen, weshalb Bedingungen (where...) benutzt werden, um die Tabellen gemäß den Vorstellungen zu verbinden.
lParam	s. wParam.
Normalisierung	Damit wird das Verfahren umschrieben, mit dem man Tabellen auf Redundanz prüft. Gemeint ist, daß mehrfache Nennungen der gleichen Information vermieden werden sollen. Beim Aufbau einer Lagertabelle stellen Sie z.B. fest, daß 200 Artikel den gleichen Lieferanten haben. Anstatt Namen, Adresse, Telefonnummer des Lieferanten jedesmal bei der Erfassung eines Artikels einzugeben, erstellen Sie ein System von zwei zu verknüpfenden Tabellen. In der einen erfassen und verwalten Sie Ihre Artikel, jeweils versehen mit einer Lieferantennummer. In einer zweiten Tabelle erfassen und verwalten Sie Ihre Lieferanten. Jeder Lieferant wird nur einmal erfaßt. Redundanz ist damit (bis auf die notwendige Wiederholung der Lieferantennummer in der Lagertabelle) auf ein Minimum reduziert. Ebenso werden Informationen, die errechnet werden können (Preise zzgl. Mehrwertsteuer etc.) nicht abgelegt, man errechnet diese Informationen bei Bedarf neu.

OLE	Object Linking and Embedding, ein Mechanismus unter Windows, der es ermöglicht ein Objekt, d.h. eine Quell-Anwendung (z.B. Paintbrush mit einem Bild) in eine Ziel-Anwendung (z.B. Winword oder eine mit SQLWindows erstellte Anwendung) einzubetten.
Primary Key	Primärschlüssel. Feldinhalt, der einen Datensatz eindeutig kennzeichnet. Beispiele für Primärschlüssel finden sich in Tabellen mit Stammdaten: Kundennummern in Kundentabellen, Artikelnummern in Lagertabellen usw. (s. auch Foreign Key).
Query	Eine Query (dt.: Abfrage) ist die Anweisung an die Datenbank, aus einer vorhandenen Menge von Information (abgelegt in Tabellen oder Views) eine Teilmenge auszuwählen.
RDBMS	Abkürzung für Relationales Datenbank Management System.
Redundante Daten	Mehrfach vorhandene gleiche Daten, z.B. eine Adresse, die in verschiedenen Systemen mehrfach gespeichert ist. In relationalen Datenbanken ist ein völliger Verzicht auf Redundanz nicht möglich. Schlüsselfelder (z.B. Kundennummern), mit denen Tabellen verknüpft werden, enthalten redundante Daten.
Referentielle Integrität	Referentielle Integrität ist ein Mechanismus der Datenbank, der verhindern soll, daß Änderungen zu inkonsistenten Daten führen. So könnte z.B. das Löschen eines Kunden aus der Stammdaten-Tabelle dazu führen, daß die dem Kunden zugeordneten Datensätze in einer Detail-Tabelle nicht mehr zuzuordnen sind. Die Datenbank wacht deshalb über die Tabellen, so daß dieser Zustand nicht eintreten kann, oder sorgt dafür, daß bei Löschung der Stammdaten automatisch die zugeordneten Detail-Daten mitgelöscht werden (s. Befehle: on delete restrict, on delete cascade, on delete set null).
Relationale Datenbank	Datenbank, die mehrere Tabellen beinhaltet, deren Informationen über Relationen verknüpft werden können. Diese Relationen können auch wieder aufgehoben werden. Da die Suche nach Information in großen Datenmengen oft viel Zeit in Anspruch nimmt, wird häufig ein Index zur schnelleren Suche benutzt.
Rollback	s. Commit.
Router	Eine Software, die den Zugriff auf andere Datenbanken (z.B. Oracle, DB2 etc.) ermöglicht.
ROWID	Identifikation (ID) einer Zeile (ROW). Eine intern von der Datenbank SQLBase verwendete Bezeichnung einer Zeile. Sie ist nicht sichtbar, kann aber mit einer select-Anweisung gelesen werden. Die "ROWID" wird häufig verwendet, um herauszufinden, ob ein Datensatz in einer Netz-Umgebung von einem anderen Benutzer geändert wurde, da jede Änderung des Datensatzes eine veränderte "ROWID" zur Folge hat (dies bewerkstelligt SQLBase).
SAL	Abkürzung für SQLWindows Application Language, die Sprache, die bei der Programmierung von Anwendungen mit SQLWindows verwendet wird.
Satz	Zeile einer Tabelle, bestehend aus verschiedenen Feldern.
Server	Ein Datenbanksystem, das den clients die Daten zur Verfügung stellt, auch Back-End (s. dort) oder engine genannt.

Spalte	Die in einer Tabelle senkrecht untereinander stehenden Felder einer Spalte mit gleicher Überschrift.
SQL	Abkürzung für Structured Query Language, einer Abfragesprache zur Verwaltung von Daten in Form von Tabellen. Die Betonung liegt hierbei auf "Abfrage" der Sprache sind z.B. Anweisungen zum Erstellen von Eingabeformularen unbekannt. Die Sprache kann "stand alone" oder eingebettet (embedded) verwendet werden.
SQLBase	SQL-fähige relationale Datenbank der Firma Gupta.
SQLTalk	Programm, das die Benutzung der Datenbank SQLBase erlaubt. Neben der Verwendung von SQL-Anweisungen können auch Daten, die in einigen Fremdformaten vorliegen, importiert werden.
SYSADM	Abkürzung für Systemadministrator.
Systemkatalog	s. Data Dictionary.
Tabelle	Die grundlegende Form der Abspeicherung von Daten in einer Datenbank. Die Tabelle besteht aus Zeilen und Spalten, deren Datentyp vor Eingabe der Daten festgelegt werden muß. Anders formuliert handelt es sich um eine zweidimensionale Matrix, die mit Information gefüllt ist..
Timeout	Einstellbares Zeitintervall, das festlegt, wie lange auf das Gelingen einer Operation zu warten ist.
Transaktions-verarbeitung	Möglichkeit, eine Anweisung (oder eine Gruppe von Anweisungen) rückgängig zu machen. Stürzt eine Anwendung z.B. während einer Geld-Überweisung ab, kann der Betrag auf dem Quell-Konto schon abgebucht sein, ohne auf dem Ziel-Konto zu erscheinen. Bei Transaktionsverarbeitung kann mit einer "rollback"-Anweisung der alte Zustand vor der letzten durchgeführten aber mißglückten Operation wieder restauriert werden (s. Commit, Rollback).
View	Während eine Tabelle tatsächlich vorhandene Daten verwaltet, werden Views als virtuelle Tabellen betrachtet. Die View wählt z.B. einen Teil der Spalten von einer Tabelle aus und präsentiert sie so als wären diese eine "richtige" Tabelle. Damit können für Benutzer, die nicht alle Spalten einer Tabelle sehen dürfen, diese Spalten ausgeblendet werden. Der Benutzer merkt hiervon nichts, das Aussehen ist identisch. Tatsächlich wird bei der Speicherung einer View aber nur die *Art* gespeichert, wie die View zu erzeugen ist (d.h. die hierzu nötige SQL-Anweisung), nicht aber die anzuzeigenden Daten. Diese finden sich immer nur in der Tabelle.
wParam	wParam (und lParam) sind Parameter, die bei der Benutzung von Nachrichten (Messages) verwendet werden. Dabei bezeichnet "w" bzw. "l" die Länge des Parameters (w = word, 16 bits; l=long word, 32 bits).

4.3 SQLTalk und SQL

Das Programm SQLTalk, das in dem einführenden Kapitel 1.2 verwendet wurde, um eine Datenbank und mehrere Tabellen anzulegen, erlaubt die Benutzung aller SQL-Anweisungen, nicht nur in Verbindung mit der Datenbank "SQLBase" sondern z.B. auch in Verbindung mit DB2.

Im folgenden sollen die in SQLTalk möglichen Anweisungen genannt und kurz beschrieben werden; ausführliche Hinweise ergeben sich aus den Handbüchern SQLBase, SQLTalk Language Reference Manual bzw. SQLBase, Database Administrator´s Guide (beide sind im Lieferumfang von SQLWindows enthalten).

Die Gliederung orientiert sich an folgenden Punkten:

- Kommandos, die eine Verbindung zur Datenbank herstellen, beenden und die Eingabe von Betriebssystem-Anweisungen erlauben.
- Kommandos für die Steuerung der Umgebung.
- Kommandos für die Report-Ausgabe.
- Kommandos für das Speichern von Prozeduren und Kommandos.
- Elemente der Sprache SQL.

Im folgenden werden die einzelnen Kommandos beschrieben. Beachten Sie, daß die Schreibweise (Groß- und Kleinschreibung) im Gegensatz zu den Elementen der Sprache SAL (SQLWindows Application Language) beliebig ist.

- Kommandos, die eine Verbindung zur Datenbank herstellen, beenden und die Eingabe von Betriebssystem-Anweisungen erlauben:

Kommando	Erläuterung
connect	Stellt die Verbindung zur Datenbank her.
disconnect	Trennt die Verbindung zur Datenbank.
exit	Beendet SQLTalk.
use	Wählt einen neuen Cursor (Arbeitsbereich).
$	Erlaubt die Ausführung von Betriebssystem-Kommandos.

- Kommandos für die Steuerung der Umgebung:

Kommando	Erläuterung
backup	Erstellt eine Sicherungskopie einer Datenbank (oder der Log-Files oder beides).
copy	Kopiert eine Datenbank oder eine Tabelle.
load	Importiert eine oder mehrere Tabellen in die aktuelle Datenbank.
release log	Erzeugt ein neues Log-File.
reorganize	Reorganisiert die Datenbank (entfernt Fragmentierungen).
restore	Stellt eine mit backup gesicherte Datenbank wieder her.
rollforward	Stellt eine Datenbank wieder her.
set server	Stellt eine Verbindung zu einem Server her.
unload	Exportiert Daten aus einer Datenbank in ein Fremdformat.

- Kommandos für die Report-Ausgabe:

Kommando	Erläuterung
break	Definiert Break-Variable für Report-Ausgabe.
btitle	Legt einen Kopf für die Report-Ausgabe fest.
column	Stellt Spalten-Attribute ein.
compute	Stellt zusammenfassende Berechnungen an.
left	Beginnt mit der Daten-Ausgabe am linken Rand.
print	Druckt einen Report (auf dem Drucker oder in eine Datei).
right	Beginnt mit der Daten-Ausgabe am rechten Rand.
ttitle	Legt einen Report-Fuß fest.

- Kommandos für das Speichern von Prozeduren und Kommandos:

Kommando	Erläuterung
erase	Löscht eine gespeicherte SQL-Anweisung.
execute	Führt eine gespeicherte SQL-Anweisung aus.
store	Speichert eine vorkompilierte SQL-Anweisung.
pause	Wartet auf die Drücken der Enter-Taste.
remark	Erlaubt die Eingabe einer Bemerkung.
run	Führt eine SQLTalk Kommando-Datei aus.
save	Speichert die Report-Umgebung oder eine SQL-Anweisung.
prepare	Bereitet eine SQL-Anweisung vor.
perform	Führt eine SQL-Anweisung aus.
set	Setzt Optionen.
show	Zeigt den Inhalt einer Variablen oder eines Attributs an.

- Elemente der Sprache SQL:

Kommando	Erläuterung
alter database	Ändert die Art, in der eine Datenbank gespeichert ist.
alter dbarea	Ändert die Größe des Datenbank-Speicherbereichs.
alter password	Ändert ein Passwort.
alter stogroup	Ändert die Einstellungen der Speicher-Aufteilung auf verschiedenen Laufwerken.
alter table	Ändert die Struktur einer Tabelle.
check database	Prüft die Integrität der Datenbank.
comment on	Ersetzt oder fügt einen neuen Kommentar an eine Tabelle, View oder Spalte hinzu.
commit	Fixiert Änderungen so, daß sie nicht zurückgenommen werden können.
create database	Erzeugt eine Datenbank.
create dbarea	Erzeugt einen Speicherbereich für eine Datenbank.
create index	Erzeugt einen Index für eine Tabelle oder View.
create stogroup	Erstellt eine Speichergruppe für eine Datenbank.
create synonym	Erzeugt einen zusätzlichen Namen für eine Tabelle oder View.
create table	Erzeugt eine Tabelle.
create view	Erzeugt eine View (virtuelle Tabelle).
deinstall database	Entfernt eine Datenbank aus der Server-Liste (kein Zugriff mehr möglich).
delete	Löscht eine oder mehrere Spalten aus einer Tabelle.
drop	Entfernt ein Objekt (Table, View, Index) aus dem Systemkatalog.
drop database	Löscht eine Datenbank.
drop dbarea	Löscht den Speicherbereich einer Datenbank.
drop stogroup	Löscht die Speichergruppe einer Datenbank.
grant	Gewährt Datenbank-Rechte.
insert	Fügt Zeilen in eine Tabelle ein.
install database	Fügt eine Datenbank der Server-Liste hinzu.
label	Ändert oder fügt neue Beschreibungen in Tabellen oder Spalten hinzu.
revoke	Entfernt Datenbank-Rechte.
rollback	Nimmt die in der Datenbank vorgenommenen Änderungen zurück.
savepoint	Legt innerhalb einer Transaktion einen Speicherpunkt fest.
select	Ermöglicht die Abfrage von Tabellen oder Views.
set default stogroup	Legt die voreingestellte Speichergruppe für neu anzulegende Datenbanken fest.
union	Vermischt die Ergebnisse von zwei oder mehr Select-Abfragen.

update	Ändert die Inhalte von Spalten in Tabellen oder Views.
update statistics	Frischt die Indexe auf.

4.4 Funktionen von SQLWindows

Die Funktionen von SQLWindows werden im folgenden in einer Übersicht dargestellt. Hierzu sind sie nach Kriterien geordnet:

Allgemeines
- Array-Handling, numerische Funktionen
- Farbe und Schriftarten
- Datum
- Edit-Funktionen
- Datei-Management
- Formate und Validierung
- Zeichenketten, Datentyp-Konvertierung

Windows
- DDE (Dynamic Data Exchange)
- OLE (Objekt Linking and Embedding)
- Drag und Drop

SQLWindows
- Debugging
- Window Management
- Messages (Nachrichten)

SQL
- SQL

Objekte
- Dialog Box
- List Box und Combo Box
- MDI (Multiple Document Interface)
- Picture
- Quest
- Report
- Druck
- Scroll Bar
- Table Window

Sonstiges
- System

Im einzelnen:

- Array-Handling:
  Diese Funktionen dienen der Bearbeitung von Array-Inhalten.

Name	Bedeutung
SalArrayAvg	Mittelwertberechnung von Array-Elementen.
SalArrayMax	Maximalwert aus Array-Elementen.
SalArrayMin	Minimalwert aus Array-Elementen.
SalArraySum	Summe von Array-Elementen.
SalQueryArrayBounds	Erstes und letztes Array-Element anzeigen.
SalSetArrayBounds	Festlegen der Anzahl von Elementen.

- Numerische Funktionen:
  Diese Funktionen erlauben die Bearbeitung von numerischen Variablen.

Name	Bedeutung
SalNumberAbs	Absolutwert-Berechnung.
SalNumberArcCos	Arc-Kosinus-Berechnung.
SalNumberArcSin	Arc-Sinus-Berechnung.
SalNumberArcTan	Arc-Tangens-Berechnung.
SalNumberArcTan2	Arc-Tangens-Berechnung (2 Werte).
SalNumberCos	Cosinus-Berechnung.
SalNumberCosH	Berechnung von hyperbolischem Cosinus.
SalNumberExponent	Exponential-Funktion.
SalNumberHigh	Ermittelt die ersten 16 Bit eines 32 Bit Wertes.
SalNumberHypot	Länge der Hypotenuse bei rechtwinkligem Dreieck.
SalNumberLog	Natürlicher Logarithmus.
SalNumberLogBase10	Logarithmus zur Basis 10.
SalNumberLow	Ermittelt die letzten 16 Bit eines 32 Bit-Wertes.
SalNumberMax	Liefert den größeren von zwei Werten.
SalNumberMin	Liefert den kleineren von zwei Werten.
SalNumberMod	Liefert den ganzzahligen Anteil nach Division zweier Zahlen (Modulo).
SalNumberPi	Multiplikation einer Zahl mit Pi.
SalNumberPower	Potenzrechnung.
SalNumberRandInit	Startpunkt für Zufallszahlen-Berechnung.
SalNumberRandom	Zufallszahlen-Berechnung.
SalNumberRound	Rundung.
SalNumberSin	Sinus-Berechnung.
SalNumberSinH	Berechnung von hyperbolischem Sinus.
SalNumberSqrt	Quadratwurzel.
SalNumberTan	Tangens-Berechnung.
SalNumberTanH	Hyperbolischer Tangens.
SalNumberToChar	Zeichenberechnung aus ASCII-Wert.
SalNumberToHString	String-Handle aus Zahl.

SalNumberToStr, SalNumberToStrX	Zeichenfolge aus einer Zahl.
SalNumberToWindowHandle	Window Handle aus Zahl.
SalNumberTruncate	Abschneiden eines Zahlenteils.

- Farbe und Schriftarten:
  Mit diesen Funktionen können Farben und Schriftarten manipuliert werden.

Name	Bedeutung
SalColorFrom RGB	Bildung von Farbwert aus Rot, Grün, Blau.
SalColorGet	Liefert Farbe von Window.
SalColorGetSysColor	Liefert Systemfarbe.
SalColorSet	Setzen von Windows-Farben.
SalColorToRGB	Liefert die Farbanteile.
SalFontGet	Liefert Font-Name, Größe und Erweiterungen eines Fensters.
SalFontGetNames	Liefert Name der vorhandenen Schrift-Fonts.
SalFontGetSizes	Liefert verfügbare Größen für Font.
SalFontSet	Setzen von Fonts für ein Fenster.

- Datum:
  Diese Funktionen dienen der Arbeit mit Datumswerten.

Name	Bedeutung
SalDateConstruct	Datumserzeugung.
SalDateCurrent	System-Datum und System-Zeit.
SalDateDay	Tag des Systemdatums.
SalDateHour	Stunde des Systemdatums.
SalDateMinute	Minuten des Systemdatums.
SalDateMonth	Monat des Systemdatums.
SalDateMonthBegin	Datum des ersten Tages im Monat.
SalDateQuarter	Quartal des Systemdatums.
SalDateQuarterBegin	Datum des ersten Tages im Quartal.
SalDateSecond	Sekunden des Systemdatums.
SalDateToString	String aus dem Systemdatum.
SalDateWeekBegin	Daum des letzten Montags.
SalDateWeekday	Nummer des Tages.
SalDateYear	Jahr des Systemdatums.
SalDateYearBegin	Datum des ersten Tages im Jahr.

- Edit-Funktionen:
  Funktionen für Cut- und Paste- Möglichkeiten unter Windows.

Name	Bedeutung
SalEditCanCopyTo	Gestattet die Benutzung der Option "Copy to" (Edit-Menü).
SalEditCanCut	Gestattet die Benutzung der Cut-Option (Edit-Menü).

SalEditCanInsertObject	Gestattet die Benutzung der Option "Insert Object" (Edit-Menü).
SalEditCanPaste	Gestattet die Benutzung der Option "Paste" (Edit-Menü).
SalEditCanPasteFrom	Gestattet die Benutzung der Option "Paste from" (Edit-Menü).
SalEditCanPasteLink	Gestattet die Benutzung der Option "Paste Link" (Edit-Menü).
SalEditCanPasteSpecial	Gestattet die Benutzung der Option "Paste Special" (Edit-Menü).
SalEditCanUndo	Gestattet die Benutzung der Option "Undo" (Edit-Menü).
SalEditClear	Löscht markierte Daten.
SalEditCopy	Kopiert markierte Daten in die Zwischenablage.
SalEditCopyString	Kopiert einen String in die Zwischenablage.
SalEditCopyTo	Kopiert ein Objekt in eine Datei.
SalEditCut	Schneidet markierte Daten aus.
SalEditInsertObject	Fügt ein Objekt ein.
SalEditPaste	Fügt den Inhalt der Zwischenablage ein.
SalEditPasteFrom	Fügt eine Grafik-Datei ein.
SalEditPasteLink	Verbindet ein Bild aus der Zwischenablage mit der Zielanwendung.
SalEditPasteSpecial	Erlaubt das Einbinden eines Bildes und die Änderung des Formates.
SalEditPasteString	Fügt den Inhalt der Zwischenablage (einen String) in den markierten String ein.
SalEditUndo	Zurücknehmen der letzten Änderung.

- Datei-Management
  Für den Zugriff auf Dateien und damit zusammenhängende Einstellungen.

Name	Bedeutung
SalFileClose	Schließt eine offene Datei.
SalFileCopy	Kopiert eine Datei.
SalFileCreateDirectory	Erzeugt ein Verzeichnis.
SalFileGetC	Liest das nächste Zeichen einer offenen Datei, die keine ASCII-Zeichen enthält.
SalFileGetChar	Liest das nächste Zeichen einer offenen ASCII-Datei.
SalFileGetCurrentDirectory	Liefert den aktuellen Pfad.
SalFileGetDateTime	Liefert Datum und Zeit der letzten Datei-Änderung.
SalFileGetDrive	Liefert den Buchstaben des aktuellen Laufwerks.
SalFileGetStr	Liefert die nächste Zeile einer offenen Datei.
SalFileOpen	Öffnet eine Datei.
SalFileOpenExt	Öffnet erstmals bzw. öffnet wiederholt eine Datei.
SalFilePutC	Schreibt ein Zeichen in eine offene Datei (nicht ASCII).
SalFilePutChar	Schreibt ein Zeichen in eine offene ASCII-Datei.
SalFilePutStr	Schreibt einen String in eine offene Datei.
SalFileRead	Liest Zeichen aus einer offenen Datei.
SalFileRemoveDirectory	Löscht ein Verzeichnis.
SalFileSeek	Verschiebt die Lesemarke in einer offenen Datei.
SalFileSetCurrentDirectory	Bestimmt ein neues aktuelles Verzeichnis.
SalFileSetDateTime	Ändert Datum und Zeit einer Datei.
SalFileSetDrive	Bestimmt ein neues aktuelles Laufwerk.

SalFileTell	Zeigt die aktuelle Position der Lesemarke in einer offenen Datei.
SalFileWrite	Schreibt einen String in eine offene Datei.

- Formate und Validierung:
  Zur Einstellung der Anzeigeformate.

Name	Bedeutung
SalFmtCopyProfile	Kopiert Format-Einstellungen.
SalFmtFieldToStr	Kopiert (formatierte oder unformatierte Inhalte) eines Data Fields oder einer Tabellenspalte in einen String.
SalFmtFormatDateTime	Formatiert Datums- und Zeitangaben.
SalFmtFormatNumber	Formatiert numerische Werte.
SalFmtGetFormat	Liefert das eingestellte Format.
SalFmtGetInputMask	Liefert die Eingabemaske.
SalFmtGetParmInt	Liefert eine Zahl für das eingestellte Format.
SalFmtGetParmStr	Liefert einen String für das eingestellte Format.
SalFmtGetPicture	Liefert das Anzeige-Format.
SalFmtGetProfile	Liefert den Profilnamen.
SalFmtIsValidField	Prüft, ob der Inhalt eines Data Fields (oder einer Tabellenspalte) dem Format entspricht.
SalFmtIsValidInputMask	Prüft, ob der Inhalt einer Eingabe-Maske entspricht.
SalFmtIsValidPicture	Prüft die Anzeige von numerischen- oder Datums/Zeit-Eingaben.
SalFmtSetFormat	Legt das Format fest.
SalFmtSetInputMask	Legt die Eingabemaske fest.
SalFmtSetParmInt	Legt das Format anhand einer Zahl fest.
SalFmtSetParmStr	Legt das Format anhand eines Strings fest.
SalFmtSetPicture	Legt das Anzeigeformat zur Laufzeit fest.
SalFmtSetProfile	Legt das Profil fest.
SalFmtStrToField	Kopiert einen String zu einem Data Field oder einer Tabellenspalte.
SalFmtUnmaskInput	Entfernt die Formatierung einer Eingabe.
SalFmtValidateField	Prüft und formatiert den Inhalt eines Data Fields oder einer Tabellenspalte.
SalIsNull	Prüft, ob ein Inhalt vorhanden ist.
SalIsValidDateTime	Prüft, ob ein Data Field, ein Multiline Text Feld oder eine Tabellenspalte einen korrekten Datums/Zeit-Eintrag hat.
SalIsValidDecimal	Prüft eine Dezimalzahl.
SalIsValidInteger	Prüft eine Integer-Zahl.
SalIsValidNumber	Prüft ein numerisches Feld (integer oder decimal).
SalStrIsValidCurrency	Prüft einen String auf korrekte Währungs-Zahl.
SalStrIsValidDateTime	Prüft einen String auf korrekten Datums/Zeit-Wert.
SalStrIsValidNumber	Prüft einen String auf korrekte Zahl.

- Zeichenketten, Datentyp-Konvertierung
  Diese Funktionen dienen der Bearbeitung von Zeichenketten (Strings) und der Konvertierung (Umwandlung) von Datentypen.

Name	Bedeutung
SalFormUnitsToPixels	Wandelt ein Objekt aus der Einheit Form-Units in Bildpunkte (Pixel) um.
SalGetDataType	Liefert den Datentyp eines Objekts.
SalHStringToNumber	Wandelt ein String-Handle in eine Nummer um.
SalPixelsToFormUnits	Wandelt ein Objekt aus Bildpunkten in die Einheit der Form-Units um.
SalStrCompress	Komprimiert einen String (z.B. für Long String Daten oder Bilder).
SalStrFirstC	Liefert das erste Zeichen in einem String.
SalStrGetBufferLength	Liefert die Länge eines String-Buffers.
SalStrLeft, SalStrLeftX	Extrahiert einen Teil aus einer Zeichenkette, von links beginnend.
SalStrLength	Liefert die Länge eines Strings.
SalStrLop	Liefert den ASCII-Wert des ersten Zeichens eines Strings.
SalStrLower, SalStrLowerX	Wandelt die Zeichen eines Strings in Kleinbuchstaben um.
SalStrMid, SalStrMidX	Extrahiert eine Teilzeichenkette mit beliebiger Anfangsposition.
SalStrProper, SalStrProperX	Wandelt die Schreibweise eines Strings um. (erstes Zeichen groß, restliche Zeichen klein).
SalStrRepeat, SalStrRepeatX	Wiederholt (verkettet) einen String.
SalStrReplace, SalStrReplaceX	Ersetzt Zeichen eines Strings durch angegebene Zeichen.
SalStrRight, SalStrRightX	Extrahiert Zeichen aus einer Zeichenkette (von rechts beginnend).
SalStrScan	Sucht Zeichen in einer Zeichenkette.
SalStrSetBufferLength	Legt die Länge eines String-Buffers fest.
SalStrToDate	Wandelt einen String in einen Datum/Zeitwert um.
SalStrTokenize	Wandelt einen bestehenden String aus mehreren Worten in einzelne (Teil-) Strings um.
SalStrToNumber	Wandelt einen String in eine Zahl um.
SalStrTrim, SalStrTrimX	Entfernt aus einem String führende oder folgende Leerzeichen, reduziert mehrere Leerzeichen (oder Tabulatoren) auf ein Leerzeichen.
SalStrUncompress	Wandelt einen mit SalStrCompress umgeformten String in eine nicht komprimierte Form um.
SalStrUpper, SalStrUpperX	Wandelt Zeichen in Großbuchstaben um.

Windows

- DDE (Dynamic Data Exchange):
  Zur Verwendung von DDE-Mechanismen in SQLWindows-Anwendungen.

Name	Bedeutung
SalDDEAddAtom	Fügt dem gemeinsam für DDE genutzten Speicher eine Zeichenfolge hinzu.
SalDDEAlloc	Zuteilung eines Handles für DDE.
SalDDEDeleteAtom	Löscht einen Eintrag im für DDE genutzten Speicher.
SalDDEExtract	Liest Informationen der Parameter wParam, lParam.
SalDDEExtractCmd	Liest den command-string aus einer WM_DDE Nachricht.
SalDDEExtractDataText	Liest Daten aus einer WM_DDE_Data oder WM_DDE_Poke Nachricht.
SalDDEExtractOptions	Liest Optionen aus einer WM_DDE_Advise Nachricht.
SalDDEFindAtom	Findet das einer Zeichenkette zugeordnete Atom.
SalDDEFree	Setzt einen Speicher-Handle zurück.
SalDDEGetAtomName	Zeigt den einem Atom zugeordnete Namen an.
SalDDEGetExecuteString	Liefert den command-String, der durch einen Client verschickt wurde.
SalDDEPost	Sendet eine DDE-Nachricht.
SalDDERequest	Befiehlt, Anfrage an Server Daten zu schicken.
SalDDESend	Schickt eine WM_DDE_Ack-Nachricht als Antwort auf eine WM_DDE_Initiate-Nachricht.
SalDDESendAll	Schickt eine WM_DDE_Initiate Nachricht.
SalDDESendExecute	Schickt eine WM_DDE_Execute-Nachricht an einen Server.
SalDDESentToClient	Schickt Daten zu einem client.
SalDDESetCmd	Füllt den command-string der Nachricht WM_DDE_Execute.
SalDDESetDataText	Füllt die Daten in einer WM_DDE_Data oder einer WM_DDE_Poke-Nachricht.
SalDDESetOptions	Setzt die Optionen in einer WM_DDE_Advise-Nachricht.
SalDDEStartServer	Signalisiert den clients, daß der Server für den Verbindungsaufbau bereit ist.
SalDDEStartSession	Startet eine "hot-link"-Verbindung zwischen Server und client.
SalDDEStopServer	Signalisiert den clients, daß der Server die DDE-Verbindung beenden möchte.
SalDDEStopSession	Beendet die DDE-Verbindung zwischen client und server.

- OLE (Objekt Linking and Embedding):
  Ermöglicht OLE mit Bildern.

Name	Bedeutung
SalOLEAnyActive	Prüft, ob OLE-Server-Anwendungen aktiv sind.
SalOLEAnyLinked	Prüft, ob Anwendungen an ein Objekt gebunden sind.
SalOLEDoVerb	Führt ein angegebenes Verb an der Server-Anwendung aus.
SalOLEFileInsert	Fügt ein Bild in ein OLE-Objekt ein.
SalOLEGetServers	Liefert die Namen der verfügbaren OLE-Server.
SalOLEGetVerbs	Liefert die von einer OLE-Anwendung ausführbaren Verben.

SalOLELinkProperties	Zeigt die Dialog Box an, um die Verknüpfungen in einer Anwendung anzuzeigen bzw. zu ändern.
SalOLEServerInsert	Fügt ein OLE-Objekt in ein Bild ein.
SalOLEUpdateActive	Zwingt den OLE-Server, das editierte Bild erneut anzuzeigen.

- Drag und Drop:
  Funktionen für Drag- und Drop-Mechanismen in Programmen.

Name	Bedeutung
SalDragDropDisableDrop	Unterbindet dropping im drag-Modus.
SalDragDropEnableDrop	Erlaubt dropping im drag-Modus.
SalDragDropGetSource	Liefert den Handle des Source-Fensters und die Position der Maus zu Beginn des Drag-Modus.
SalDragDropGetTarget	Liefert den Handle des Ziel-Fensters und die Position der Maus.
SalDragDropStart	Startet den Drag-Modus.
SalDragDropStop	Beendet den Drag-Modus.
SalDropFilesAcceptFiles	Prüft, ob ein Fenster einen Dateinamen aus dem Windows Datei-Manager erhalten kann.
SalDropFilesQueryFiles	Liefert die Dateinamen der durch das Droppen verschobenen Dateien.
SalDropFilesQueryPoint	Liefert die Position des Mauszeigers zu der Zeit, in der die Dateinamen in das Fenster eingefügt wurden.

SQLWindows

- Debugging:
  Unterstützt die Fehlersuche in SQLWindows-Anwendungen.

Name	Bedeutung
SalCompileAndEvaluate	Kompiliert und wertet einen Ausdruck aus.
SalContextBreak	Setzt einen Break-Point.
SalContextCurrent	Stellt den aktuellen Kontext wieder her.

- Window Management
  Dient der Manipulation von Objekten (Top Level und Child Level).

Name	Bedeutung
SalBringWindowToTop	Bringt ein Formular (Fenster) in den Vordergrund.
SalCenterWindow	Zentriert ein Formular.
SalCreateWindow	Erzeugt ein Formular (Form window, MDI-Window, Modeless Dialog Box, Top-Level Table Window).
SalCreateWindowFrom Str	Erzeugt ein Formular zur Laufzeit einer Anwendung aus einem String heraus.
SalDestroyWindow	Entfernt ein Fenster.
SalDisableWindow	Sperrt die Benutzung eines Objekts (Top-Level oder Child-Level).

SalDisableWindowAndLabel	Sperrt die Benutzung eines Objekts (Top-Level oder Child-Level) und sperrt zusätzlich die Möglichkeit, ein Objekt mit einer Tastenkombination anzuwählen.
SalEnableWindow	Erlaubt die Benutzung eines Objekts.
SalEnableWindowAndLabel	Erlaubt die Benutzung eines Objekts und die Verwendung der dafür vorgesehenen Tastenkombination.
SalFormGetParmNum	Zeigt Informationen zu einem Form Window.
SalGetDefButton	Liefert den Handle des Pushbuttons mit dem Fokus.
SalGetFirstChild	Liefert das erste Child-Level Objekt eines Windows.
SalGetFocus	Liefert das Objekt mit dem Fokus.
SalGetItemName	Zeigt den Namen eines Objekts an.
SalGetMaxDataLength	Zeigt die maximal mögliche Datenlänge eines Data Fields, Multiline-Feldes oder einer Tabellenspalte an.
SalGetNextChild	Liefert das nächste Child-Level-Objekt.
SalGetType	Zeigt den Typ eines Objekts an.
SalGetWindowLoc	Liefert die Koordinaten eines Objekts.
SalGetWindowSize	Liefert die Größe eines Objekts.
SalGetWindowState	Zeigt den Status eines Formulars an.
SalGetWindowText	Liefert den (angezeigten) Titel eines Formulars (oder den Wert als String, z.B. bei Data Field)
SalHideWindow	Macht ein Objekt unsichtbar.
SalHideWindowAndLabel	Macht ein Objekt und die ihm zugeordnete Bezeichnung unsichtbar.
SalInvalidateWindow	Erzwingt die erneute Anzeige eines Objekts.
SalisButtonChecked	Prüft den Status eines Radio Buttons (oder einer Check Box).
SalIsWindowEnabled	Prüft, ob ein Objekt mit Maus oder Keyboard angewählt werden kann.
SalIsWindowVisible	Prüft, ob ein Objekt sichtbar ist.
SalMoveWindow	Verschiebt ein Objekt.
SalParentWindow	Liefert den Handle des zugeordneten Parent-Windows.
SalQueryFieldEdit	Prüft, ob der Inhalt eines Data Fields, Multine Text Fields oder Table Windows geändert wurde.
SalSetDefButton	Setzt einen Pushbutton als Default.
SalSetFieldEdit	Setzt oder löscht den Status der Änderung eines Data Fields, Multine Text Fields oder Table Windows.
SalSetFocus	Setzt den Fokus auf ein Objekt.
SalSetMaxDataLength	Legt die maximale Datenlänge eines Data Fields fest.
SalSetWindowLoc	Verschiebt ein Objekt an Koordinaten.
SalSetWindowSize	Legt die Größe eines Objekts fest.
SalSetWindowText	Legt den Titel eines Objekts fest.
SalShowWindow	Macht ein Objekt sichtbar.
SalShowWindowAndLabel	Macht ein Objekt und dessen Titel sichtbar.
SalStatusGetText	Liest den in der Status Bar eingegebenen Text.
SalStatusSetText	Zeigt einen Text in der Status Bar an.
SalStatusSetVisible	Legt fest, ob der Text in der Status Bar angezeigt wird oder nicht.
SalTBarSetVisible	Legt fest, ob die Tool Bar (eines Form- oder MDI Windows) angezeigt wird.

SalUpdateWindow	Erzwingt die Neu-Anzeige eines Objeks.
SalWindowHandleToNumber	Erzeugt aus einem Window-Handle eine Nummer.

- Messages (Nachrichten):
  Zur Versendung von Nachrichten (vom System- und vom Benutzer- definierte).

Name	Bedeutung
SalPostMsg	Trägt eine Nachricht in ein Fenster ein.
SalSendClassMessage	Verschickt eine Nachricht nach Aufruf der Message Actions einer Objekt-Klasse.
SalSendClassMessageNamed	Wie eben, zu verwenden in Klassen, die das Ergebnis mehrfacher Vererbung sind.
SalSendMsg	Schickt eine Nachricht an ein Fenster.
SalSendMsgToChildren	Schickt eine Nachricht an alle Child-Objekte eines Form Windows.
SalSendValidateMsg	Schickt eine SAM_Validate-Nachricht (an Data Field, Multi-Line Feld oder Tabellenspalte).

SQL

- SQL:
  Funktionen für die Be- und Verarbeitung von Sql-Anweisungen.

Name	Bedeutung
SqlClearImmediate	Hebt die durch SqlImmediate hergestellte Verbindung zur Datenbank wieder auf.
SqlClose	Schließt einen mit Namen versehenen Cursor.
SqlCommit	Übergibt (fixiert) die aktuelle Sql-Transaktion.
SqlConnect	Stellt eine Verbindung zwischen einem Sql-Handle und der Datenbank her.
SqlContextClear	Hebt die durch SqlContextSet getroffenen Vereinbarungen wieder auf.
SqlContextSet	Stellt Vereinbarungen für Funktionsaufrufe (SqlPrepare, SqlFetchNext etc.) her.
SqlDirectoryByName	Liefert die Namen der Datenbanken auf dem angegebenem Server.
SqlDisconnect	Trennt einen Sql-Handle von der Datenbank.
SqlDropStoredCommand	Löscht eine mit "Store" gespeicherte Sql-Anweisung.
SqlError	Liefert den zuletzt aufgetretenen (Sql-) Fehler-Code.
SqlErrorText	Liefert zu einem Fehler eine Beschreibung.
SqlExecute	Führt eine Sql-Anweisung aus.
SqlExecutionPlan	Zeigt den Execution-Plan für kompilierte Sql-Anweisungen.
SqlExists	Prüft, ob eine bestimmte Zeile einer Tabelle vorhanden ist.
SqlExtractArgs	Zeigt Fehler-Informationen an.
SqlFetchNext	Holt die nächste Zeile aus einer mit select erzeugten Ergebnismenge.
SqlFetchPrevious	Holt die vorhergehende Zeile aus einer mit select erzeugten Ergebnismenge.
SqlFetchRow	Holt eine bestimmte Zeile aus der Ergebnismenge.

SqlGetErrorPosition	Liefert die Position, an der in einer Sql-Anweisung ein Fehler aufgetreten ist.
SqlGetErrorText, SqlGetErrorTextX	Liefert den Fehlertext zu einer Sql Fehler-Nummer.
SqlGetModifiedRows	Zeigt die Zahl der Datensätze, die durch Insert, Update oder Delete geändert wurden.
SqlGetParameter	Liefert den numerischen Wert eines Parameters (DBP*) der Datenbank.
SqlGetParameterAll	Liefert den Wert eines Parameters als Wert oder String.
SqlGetResultSetCount	Zeigt die Anzahl der Datensätze in einer Ergebnismenge.
SqlGetRollbackFlag	Prüft, ob eine Rollback-Anweisung erfolgreich war.
SqlImmediate	Bereitet eine Sql-Anweisung vor und führt sie aus.
SqlImmediateContext	Bereitet eine Sql-Anweisung vor, führt sie aus und liest (Fetch*) die Datensätze.
SqlOpen	Gibt einem Cursor einen Namen und führt eine Sql-Anweisung aus.
SqlPrepare	Bereitet eine Sql-Anweisung vor.
SqlPrepareAndExecute	Bereitet eine Sql-Anweisung vor und führt sie aus.
SqlRetrieve	Holt eine mit SqlStore gespeicherte Sql-Anweisung.
SqlSetInMessage	Legt die Größe des Input Message Buffers für die Zahl gelesener Datensätze fest.
SqlSetIsolationLevel	Legt den Isolation Level für die Anwendung fest.
SqlSetLockTimeout	Legt die Zeitgrenze für "Time out"-Fehler fest.
SqlSetOutMessage	Legt die Größe des Output Message Buffers (für Sql-Anweisungen oder neue Datensätze) fest.
SqlSetParameter	Legt einen Datenbank-Parameter mit einer Zahl fest.
SqlSetParameterAll	Legt einen Datenbank-Parameter mit einer Zahl oder einem String fest.
SqlSetResultSet	Schaltet den Result Set Modus an oder aus.
SqlStore	Kompiliert und speichert ein Sql-Kommando.

Objekte

- Dialog Box:
  Zeigen die von Windows bereitgestellten Dialog Boxen für Fontauswahl und Dateien sowie selbst definierte (modale) Dialog Boxen an.

Windows-Dialog-Boxes

Name	Bedeutung
SalDlgChooseColor	Dialog Box zur Farbauswahl anzeigen.
SalDlgChooseFont	Dialog Box zur Auswahl der Fonts anzeigen.
SalDlgOpenFile	Dialog Box für "Datei öffnen" anzeigen.
SalDlgSaveFile	Dialog Box für "Datei speichern" anzeigen.

Selbst definierte Dialog-Boxes

Name	Bedeutung
SalEndDialog	Schließt eine modale Dialog Box.
SalModalDialog	Erzeugt eine modale Dialog Box.

SalModalDialogFromStr	Erzeugt eine modale Dialog Box mit angegebenem Namen.

- List Box und Combo Box:
  Mit diesen Funktionen können Inhalte von List- und Combo Boxen erzeugt und verändert werden.

Name	Bedeutung
SalListAdd	Einen Eintrag anhängen.
SalListClear	Mehrere Einträge entfernen.
SalListDelete	Einen Eintrag entfernen.
SalListFiles	Dateinamen als Einträge hinzufügen.
SalListGetMultiSelect	Liefert die Index-Nummern von gewählten Einträgen (nur List Box).
SalListInsert	Fügt einen einzelnen Eintrag ein.
SalListPopulate	Füllen mit dem Ergebnis einer SQL-Abfrage.
SalListQueryCount	Liefert die Anzahl der Inhalte.
SalListQueryFile	Wählt Dateiname aus.
SalListQueryMultiCount	Liefert die Anzahl gewählter Einträge (nur List Box).
SalListQuerySelection	Index des aktuell gewählten Eintrags.
SalListQueryState	Prüft, ob spezifiziertes Element gewählt ist.
SalListQueryText, SalListQueryTextX	Holt einen Eintrag.
SalListQueryTextLength	Länge eines Eintrags.
SalListRedraw	Steuert den Neu-Aufbau von List Box, Combo Box.
SalListSelectString	Sucht und wählt einen Eintrag.
SalListMultiSelect	Wählt einen Eintrag bzw. nimmt Wahl zurück.
SalListSetSelect	Wählt einen Eintrag anhand des Index-Eintrages aus.
SalListSetTabs	Setzt Tabulatoren.

- MDI (Multiple Document Interface):
  Diese Funktionen gestatten die Manipulation von Child-Windows in einem MDI-Window.

Name	Bedeutung
SalMDIArrangeIcons	Ordnet Symbole am unteren Rand eines MDI-Windows an.
SalMDICascade	Ordnet Fenster in einem MDI-Fenster überlappend an.
SalMDITile	Ordnet Fenster in einem MDI-Fenster nebeneinander an.

- Picture:
  Diese Funktionen erlauben das Manipulieren von Bildern.

Name	Bedeutung
SalPicClear	Löscht das angezeigte Bild.
SalPicGetDescription	Liefert eine Beschreibung des Bildes.
SalPicGetString	Kopiert den Inhalt einer Bilddatei in einen String.
SalPicSet	Fügt den Inhalt einer Ressource in ein Bild.
SalPicSetFile	Fügt den Inhalt einer Bild-Datei in ein Bild.

SalPicSetFit	Bestimmt die Anpassung des Bildes an den verfügbaren Raum.
SalPicSetString	Weist einem Bild den Inhalt eines Strings zu.

- Quest:
  Funktionen, die die Benutzung von Quest-Objekten regeln.

Name	Bedeutung
SalQuestTblApplyEdits	Wendet die insert-, update- und delete-Anwendungen auf die Datenbank an.
SalQuestTblDiscardEdits	Verwirft Änderungen und zeigt den alten Zustand wieder an.
SalQuestTblPopulate	Füllt ein Quest-Window mit Daten.
SalQuestTblReset	Erzwingt die erneute Anzeige von Daten in einem Quest-Window.

- Report:
  Zur Anzeige eines Reports in der Seitenvorschau oder für den Druck.

Name	Bedeutung
SalReportClose	Schließt ein Report-Window.
SalReportCmd	Schickt eine Anweisung an ein Report-Window
SalReportCreate	Erzeugt einen Report.
SalReportDlgOptions	Setzt die Optionen für die Dialog Box der Druck-Einstellungen.
SalReportGetDateTimeVar	Liefert den Wert der Date/Time Report-Variablen.
SalReportGetNumberVar	Liefert den Inhalt einer numerischen Report-Variablen.
SalReportGetObjectVar	Liefert den Inhalt eines Report-Objekts.
SalReportGetStringVar	Liefert den Wert einer Report-String-Variablen
SalReportPrint	Druckt einen Report.
SalReportPrintToFile	Druckt einen Report in eine Datei.
SalReportReset	Zeigt den Report in der Seitenvorschau erneut an.
SalReportSetDateTimeVar	Setzt den Wert einer Date/Time-Variablen.
SalReportSetNumberVar	Setzt den Wert einer numerischen Variablen.
SalReportSetObjectVar	Setzt den Wert eines Report-Objekts.
SalReportSetStringVar	Setzt den Wert einer Zeichenketten-Variablen.
SalReportTableCreate	Erzeugt einen Report aus den Inhalten eines Table-Windows.
SalReportTablePrint	Druckt einen Report mit den Inhalten eines Table-Windows.
SalReportTableView	Zeigt einen Report mit den Inhalten eines Table-Windows in der Seitenvorschau.
SalReportView	Zeigt einen Report in der Seitenvorschau.

- Druck:
  Für den Druck von Objekten.

Name	Bedeutung
SalPrtExtractRect	Liefert die Koordinaten für den Ausdruck.
SalPrtGetDefault	Liefert das voreingestellte Ausgabegerät, den Treiber und den Port.
SalPrtGetParmNum	Liefert den Wert eines Druck-Parameters.
SalPrtPrintForm	Druckt ein Form Window.

SalPrtSetDefault	Legt den Drucker fest.
SalPrtSetParmDefaults	Führt ein Reset am Drucker durch.
SalPrtSetParmNum	Setzt den Wert eines Druck-Parameters.
SalPrtSetup	Zeigt das Fenster mit den Druckereinstellungen an.

- Scroll Bar:
  Für die Steuerung von Scroll-Bar-Aktivitäten.

Name	Bedeutung
SalScrollGetPos	Zeigt die aktuelle Scroll Bar-Position an.
SalScrollSetRange	Legt die für die Anzeige nötigen Werte fest.
SalScrollSetPos	Setzt die Scroll Bar-Position.
SalScrollSetRange	Legt den Bereich für die Scroll Bar fest.

- Table Window:
  Diese Funktionen werden in Verbindung mit Table Windows benutzt.

Name	Bedeutung
SalTblAnyRows	Stellt fest, ob bei einer der angezeigten Zeilen ein Attribut (Flag) gesetzt ist.
SalTblClearSelection	Entfernt die Markierung von Datensätzen.
SalTblColumnAverage	Berechnet den Mittelwert der Werte einer Tabellenspalte.
SalTblColumnSum	Berechnet die Summe der Werte einer Tabellenspalte.
SalTblCopyRows	Kopiert ausgewählte Zeilen in die Zwischenablage.
SalTblCreateColumn	Erzeugt eine Tabellenspalte zur Laufzeit.
SalTblDefineRowHeader	Definiert die Anzeige des Zeilenkopfes (links vom ersten Feld einer Zeile).
SalTblDefineSplitWindow	Definiert die Art, wie eine Tabelle horizontal geteilt wird.
SalTblDeleteRow	Löscht eine Zeile aus der Tabelle (nicht auf dem Datenträger).
SalTblDeleteSelected	Löscht alle ausgewählten Datensätze (auf dem Datenträger und aus der angezeigten Tabelle).
SalTblDestroyColumns	Entfernt die zur Laufzeit erzeugten Spalten.
SalTblDoDeletes	Löscht alle Datensätze mit einem bestimmten Attribut.
SalTblDoInserts	Fügt neue Datensätze von dem Table-Objekt in die Datenbank ein.
SalTblDoUpdates	Schreibt Änderungen in die Datenbank.
SalTblFetchRow	Holt einen Datensatz (der gerade nicht im Datenpuffer des Table-windows steht) in die Tabelle.
SalTblFindNextRow	Sucht vorwärts nach einem Datensatz mit einem bestimmten Attribut.
SalTblFindPrevRow	Sucht rückwärts nach einem Datensatz mit einem bestimmten Attribut.
SalTblGetColumnText	Liefert den Inhalt eines Feldes (einer anzugebenden Spalte).
SalTblGetColumnTitle	Liefert den Titel einer Spalte.
SalTblGetColumnWindow	Liefert den Handle einer Spalte.
SalTblInsertRow	Fügt eine neue Zeile in ein Table-Objekt ein (nicht in der Datenbank).
SalTblKillEdit	Setzt das Attribut "ROW_Edited" in einer neu eingefügten oder geänderten Zeile und beendet das Editieren.
SalTblKillFocus	Entfernt den Fokus einer Zeile.
SalTblPasteRows	Fügt den Inhalt der Zwischenablage ein.

SalTblPopulate	Füllt eine Tabelle mit Daten, die sich aus einer Sql-Anweisung ergeben.
SalTblQueryColumnFlags	Prüft den Status einer Spalte.
SalTblQueryColumnID	Liefert die Nummer einer Spalte.
SalTblQueryColumnPos	Liefert die relative Position einer Spalte.
SalTblQueryColumnWidth	Liefert die Breite einer Spalte.
SalTblQueryContext	Liefert die Nummer einer Zeile.
SalTblQueryFocus	Liefert die Nummer der Zeile, die den Fokus hat.
SalTblQueryLockedColumns	Zeigt die Anzahl gesperrter Spalten an.
SalTblQueryRowFlags	Prüft, ob eine Zeile ein bestimmtes Attribut hat.
SalTblQueryRowHeader	Zeigt den Status des Row-Headers einer Zeile an.
SalTblQueryScroll	Zeigt die Satznummer des Datensatzes, der an erster Stelle der Tabelle steht.
SalTblQuerySplitWindow	Stellt fest, wie ein Table-window horizontal geteilt ist.
SalTblQueryTableFlags	Prüft die Attribute (flags) einer Tabelle.
SalTblQueryVisibleRange	Zeigt die Nummern des ersten und letzten editierbaren Datensatzes.
SalTblReset	Entfernt die Daten einer Tabelle.
SalTblScroll	Blättert zu einer bestimmten Zeile bzw. Spalte.
SalTblSetCellTextColor	Stellt die Farbe eines Feldes ein.
SalTblSetColumnFlags	Setzt oder löscht Attribute einer Spalte.
SalTblSetColumnPos	Verschiebt eine Spalte.
SalTblSetColumnText	Schreibt einen Wert in eine Spalte.
SalTblSetColumnTitle	Setzt eine Spaltenüberschrift.
SalTblSetColumnWidth	Setzt die Spaltenbreite.
SalTblSetContext	Legt die aktive Zeile fest (ohne den Fokus zu setzen).
SalTblSetFlagsAnyRows	Setzt oder löscht Zeilen-Attribute.
SalTblSetFocusCell	Setzt den Fokus auf ein Feld.
SalTblSetFocusRow	Setzt den Fokus auf eine Spalte.
SalTblSetLockedColumns	Fixiert Spalten.
SalTblSetRange	Bestimmt die maximale bzw. minimale Zahl anzuzeigender Zeilen.
SalTblSetRow	Bewegt den Fokus auf eine andere Zeile.
SalTblSetRowFlags	Setzt oder löscht Zeilen-Attribute.
SalTblSetTableFlags	Setzt oder löscht Tabellen-Attribute.
SalTblSortRows	Sortiert die Datensätze nach einem Spalteninhalt.

Sonstiges

- System:
  Dient der allgemeinen Steuerung von SQLWindows-Anwendungen (Programmstart, Programmende, Timer-Verwendung, Cursor-Formen)

Name	Bedeutung
SalAbort	Unterbricht die aktuelle Operation.
SalAppDisable	Unterbindet die Benutzung laufender Anwendungen.

SalAppEnable	Erlaubt die Benutzung laufender Anwendungen.
SalClearField	Entfernt den angezeigten Wert eines Data Fields, Multiline Feldes oder einer Tabellenspalte.
SalCursorClear	Löscht den Mauszeiger (Cursor).
SalCursorSet	Setzt einen (anderen) Mauszeiger.
SalCursorSetFile	Setzt einen anwender-definierten Mauszeiger.
SalCursorString	Setzt einen Mauszeiger nach dem Inhalt eines Strings (einer Bild-Datei).
SalDrawMenuBar	Zeigt die Inhalte der Menu Bar erneut an.
SalGetProfileNumber	Sucht einen in einer Ini-Datei eingetragen Integer-Wert.
SalGetProfileString	Sucht eine in einer Ini-Datei eingetragene Zeichenkette.
SalGetVersion	Zeigt die verwendete SQLWindows-Version an.
SalLoadApp	Startet eine andere Anwendung.
SalLoadAppAndWait	Startet eine andere Anwendung und wartet auf das Beenden dieser Anwendung (um danach zu der ursprünglichen Anwendung zurückzukehren).
SalMessageBeep	Erzeugt einen Signal-Ton.
SalMessageBox	Zeigt eine Message Box an.
SalQuit	Beendet die SQLWindows-Anwendung.
SalSetProfileString	Setzt einen Wert in einer Ini-Datei.
SalTimerKill	Löscht einen zuvor definierten Timer.
SalTimerSet	Erzeugt einen neuen Timer.
SalTrackPopupMenu	Erzeugt ein Popup-Menü während der Laufzeit.
SalWaitCursor	Erzeugt oder entfernt die Stundenglas-Anzeige des Mauszeigers.
SalWinHelp	Ruft das Windows Hilfe-System auf.
SalYieldEnable	Bestimmt, ob der Benutzer länger-andauernde Operationen unterbrechen kann.
SalYieldQueryState	Stellt den über SalYieldEnable festgelegten Status fest.
SalYieldStartMessages	Erlaubt das Unterbrechen und das Senden von Nachrichten.
SalYieldStopMessages	Sperrt die Möglichkeit, zu unterbrechen und Nachrichten zu schicken.

4.5 Nachrichten von SQLWindows

Beachten Sie, daß die im System bereits vorhandenen Variablen (z.B. "hWndForm", "hWndItem", "wParam", "lParam") bei den einzelnen Nachrichten mit Informationen gefüllt werden können. Diese Inhalte können im Programmablauf verwendet werden; näheres hierzu findet sich in dem Technical Reference Manual.

Name	Zeitpunkt	Objekte
SAM_AnyEdit	Nachdem ein Benutzer den Inhalt eines Objekts geändert hat.	Data Field, Multiline text Field, Table Window column.
SAM_AppExit	Wenn alle Fenster geschlossen werden (letzte Nachricht die verschickt wird).	Outline Application Actions
SAM_AppStartup	Start der Anwendung.	Outline Application Actions.
SAM_CacheFull	Zwischenspeicher für zu lesende Daten ist voll, kein weiterer Datensatz kann gelesen werden.	Table Window, Quest Window.
SAM_CaptionDoubleClick	Doppelklicken auf den Spaltennamen einer Tabelle.	Table Window, Quest Window, Spalte einer Tabelle.
SAM_Click	Klicken auf ein Objekt.	Table Window, Table Window column, Push Button, Radio Button, Check Box, Option Button, Combo Button, List Box, Picture (bei Doppelklicken auf einer List Box oder Table Window ebenso).
SAM_Close	Nach Auswahl der Option "Schließen" aus dem Systemmenü oder Doppelklick auf dem Systemmenü (nicht nach SalQuit, SalEndDialog oder SalDestroyWindow).	Form Window, Dialog Box, Top Level Table Window, Top Level Quest Window, MDI Window.
SAM_ColumnSelectClick	Einfaches Klicken auf den Titel einer Spalte.	Table Window, Quest Window, Spalte.
SAM_CornerClick	Einfaches Klicken in der linken oberen Ecke einer Tabelle.	Table Window, Quest Window.
SAM_CornerDoubleClick	Doppeltes Klicken in der linken oberen Ecke einer Tabelle.	Table Window, Quest Window.
SAM_CountRows	Anwählen des letzten Datensatzes (z.B. nach Drücken der Ende-Taste).	Table Window, Quest Window.
SAM_Create	Nachdem ein Objekt erzeugt aber bevor es angezeigt ist.	Top-Level Windows und Child-Windows.

SAM_DDE_Client_Execute	DDE-Client schickt ein Kommando.	Alle Objekte einer DDE-Server-Anwendung.
SAM_DDE_ClientRequest	DDE-Client fordert Daten an.	Alle Objekte einer DDE Server-Anwendung.
SAM_DDE_DataChange	DDE-Server hat Daten für einen DDE-Client.	Alle Objekte einer DDE-Server-Anwendung.
SAM_Destroy	Unmittelbar bevor die Fenster verschwinden.	Top Level Windows und Child-Objekte.
SAM_DoubleClick	Doppelklicken mit der Maus auf einem Objekt.	List Box, Combo Box, Table Window, Quest Window, Spalten, Picture.
SAM_DragCanAutoStart	SQLWindows prüft, ob ein Objekt Auto-Dragging wünscht (z.B. nach Verschieben eines Eintrags).	Top Level Window, Data Field, Multiline Field, List Box, Combo Box, Picture.
SAM_DragDrop	Der Benutzer benutzt die Maus für Drag-Operationen.	Top Level Window, Data Field, Multiline Field, List Box, Combo Box, Picture.
SAM_DragEnd	Der Drag-Modus wird beendet.	Top Level Window, Data Field, Multiline Field, List Box, Combo Box, Picture.
SAM_DragEnter	Der Benutzer zieht die Maus (im Drag-Modus) in ein Zielfenster.	Top Level Window, Data Field, Multiline Field, List Box, Combo Box, Picture.
SAM_DragExit	Der Benutzer zieht die Maus (im Drag-Modus) aus dem Zielfenster hinaus.	Top Level Window, Data Field, Multiline Field, List Box, Combo Box, Picture.
SAM_DragMove	Der Benutzer zieht die Maus (im Drag-Modus) an eine andere Stelle im Zielfenster.	Top Level Window, Data Field, Multiline Field, List Box, Combo Box, Picture.
SAM_DragNotify	Im Drag-Modus erfolgt eine Maus-Aktion.	Top Level Window, Data Field, Multiline Field, List Box, Combo Box, Picture.
SAM_DragStart	Nach dem Start des Drag-Modus.	Top Level Window, Data Field, Multiline Field, List Box, Combo Box, Picture.
SAM_DropDown	Der Benutzer klickt auf den nach unten zeigenden Pfeil einer Combo Box.	Combo Box.
SAM_DropFiles	Der Benutzer droppt einen Dateinamen aus dem Dateimanager auf ein Objekt.	Spalte, Data Field, Multiline Field, List Box, Combo Box, Picture, Custom Controll.
SAM_EndCellTab	Der Benutzer drückt in der letzten Spalte einer Tabelle die Tab-Taste.	Table Window, Quest Window.
SAM_FetchDone	Nachdem mittels SalTblPopulate (TBL_FillAllBackground) das Füllen der Tabelle mit Datensätzen beendet ist.	Table Window, Quest Window.

SAM_FetchRow	Vor jedem Datensatz, der aus der Datenbank in den Cache-Datenpuffer eingelesen wird.	Table Window, Quest Window.
SAM_FetchRowDone	Nachdem ein Datensatz aus der Datenbank in den Cache-Zwischenspeicher eingelesen wurde.	Table Window, Quest Window.
SAM_FieldEdit	Der Benutzer ändert einen angezeigten Wert und bewegt sich aus dem Feld mit der Tab-Taste heraus.	Table Window, Spalte, Data Field, Combo Box, Multiline Field.
SAM_Help	Der Benutzer drückt die F1-Taste.	Top Level Windows.
SAM_KillFocus	Der Benutzer nimmt den Fokus von einem Objekt.	Table Window, Data Field, Multiline Field, Push Button, Radio Button, Check Box, Option Button, Combo Box, List Box, Spalte, Scroll Bars.
SAM_Print	SQLWindows druckt ein Objekt.	Form Window, Data Field, Multiline Field, Combo Box, Push Buttons.
SAM_ReportFetchInit	Wenn Report Windows bereit ist, die erste Seite der Druckausgabe zu formatieren (wird nach SAM_ReportStart verschickt).	Top Level Windows.
SAM_ReportFetchNext	Report Windows ist bereit die nächste Datenzeile zu drucken (wird nach SAM_ReportFetchInit veschickt).	Top Level Windows.
SAM_ReportFinish	Report Windows hat die Report-Ausgabe beendet.	Top Level Windows.
SAM_ReportNotify	Report Windows ist dabei, einen Teil des Reports zu formatieren.	Top Level Windows.
SAM_ReportStart	Nach dem Aufruf der Funktionen SalReportView oder SalReportStart.	Top Level Windows.
SAM_RowHeaderClick	User klickt auf den Zeilen-Kopf (row header).	Table Window, Quest Window.
SAM_RowHeader-DoubleClick	User klickt doppelt auf den Zeilenkopf (row header).	Table Window, Quest Window.
SAM_RowValidate	Benutzer bewegt den Fokus von einer Zeile weg.	Table Window, Quest Window.

SAM_ScrollBar	Benutzer klickt auf einen Teil einer Scroll Bar.	Scroll Bars.
SAM_SetFocus	Benutzer bewegt den Fokus auf ein Objekt.	Table Window, Data Field, Multiline Field, Push Button, Radio Button, Check Box, Option Button, Combo Box, List Box, Spalte, Scroll Bar.
SAM_SqlError	Eine Sql-Anweisung führt zu einem Fehler.	Outline Application Actions.
SAM_Timer	Die (durch SalTimerSet in Millisekunden eingestellte Zeit) ist vergangen.	Top Level Windows.
SAM_Validate	Der Benutzer ändert den Wert eines Objekts und bewegt den Fokus weg.	Data Field, Multiline Field, Combo Box, Spalte.

4.6 Beispiel zu Datenimport

Die folgende Anwendung soll Hinweise zu der Arbeit mit Fremddaten geben. Hierzu wird eine dBASE-Datei geöffnet und der Header der Datei gelesen, in dem die Beschreibung der Datei-Struktur geschrieben steht. Danach wird eine SQL-Anweisung gebaut, mit der eine Tabelle angelegt werden kann, die die gleiche Struktur aufweist wie die DBF-Struktur. Schließlich werden die Daten aus der Datei in einer Tabelle angezeigt (aber nicht in die Datenbank eingelesen.

Dem Beispiel liegt eine Datei mit dem Namen "Test.dbf" zugrunde. Diese Datei, die mit dBASE, Clipper oder FoxPro erzeugt werden kann, hat folgende Struktur:

Feldname	Datentyp	Länge	Dezimalstellen
Persnr	N	5	0
Name	C	10	
Abteilung	C	2	
Gehalt	N	10	2
Eintritt	D	8	
Bonus	L	1	

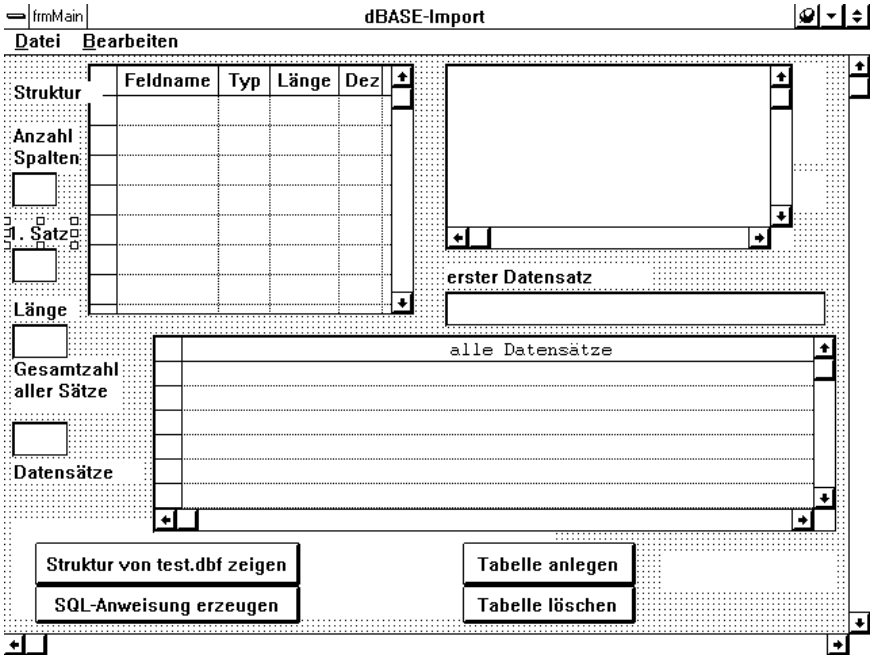
Die Datei enthält 5 Datensätze, auf deren Wiedergabe hier verzichtet wird, da in dem unten folgenden Screen-Shot die Sätze angezeigt sind.

Die Anwendung soll

- die Datei Test.DBF öffnen;
- die Struktur lesen;
- aus der Struktur eine SQL-Create Table-Anweisung erzeugen, die eine identische Tabelle in der geöffneten Datenbank (FIRMA) anlegen kann;
- die Tabelle anlegen und
- die Datensätze anzeigen (die Datensätze werden nicht in die neu angelegte Tabelle geschrieben).

Hierzu wurde ein Formular mit folgendem Inhalt angelegt.

Abb. 4.1: Formular "frmMain" mit eingebetteten Objekten.



Links oben soll nach Drücken eines Push Buttons die Struktur der zu lesenden Datei angezeigt werden, rechts oben soll die ebenfalls nach Drücken auf einen Push Button anzufertigende SQL-Create Table-Anweisung angezeigt werden. In der breiten Tabelle sollen alle Datensätze angezeigt werden.

Die folgende Liste zeigt die vereinbarten Variablen:

Sql Handle: hSql
File Handle: hFile
Number: nChar
Number: anzahl
Number: feldzahl
String: sChar
String: sAlleDaten
!
String: sName
String: sTyp
Number: nLaenge
Number: nDez
Number: nNeuSatz
Number: nZeile
String: strCreateTable
String: sDatatype

```
String: sLaenge
!
Number: nNeueZeile
String: sTeil1
String: sTeil2
Number: nPosition
Number: nStelle                 !für Auftauchen von ASCII Char 13
                                ! als Beginn des Datenbereichs
Boolean: weiter
Number: nAlleZeichen            ! Zähler für alle gelesenen Zeichen
Number: nPosErsterSatz          ! Position des ersten Datensatzes
Number: nAlleSaetze             ! Gesamtzahl aller Datensätze
Number: nLaengeSatz             ! Länge eines Datensatzes
String: strNummer               ! für Array: Umwandlung von Zeilennummer in String
Number: nVglAnzahl              ! für das Lesen der Datensätze in tbl2
Number: nDatenZeile             ! für Anzahl Datensätze in unterer Tabelle
```

Nach dem Start der Anwendung wird zunächst die Verbindung zur Datenbank Firma hergestellt.

```
◆ Application Actions
  ◆ On SAM_AppStartup
    ◇ Set SqlDatabase = 'FIRMA'
    ◇ Call SqlConnect(hSql)
```

Der erste Push Button "Struktur von Test.dbf zeigen" enthält folgenden Inhalt der Message Actions:

```
On SAM_Click
    ! Zuerst die Struktur der DBF erkunden und aufzeichnen
    !
    Set feldzahl = 0
    Set nAlleZeichen = 0            ! Gesamtzahl aller gelesenen Zeichen
    Set anzahl = 0          ! internes Zahlen für Anzahl gelesener Zeichen
    Call SalFileOpen(hFile, 'C:\\dbf_data\\test.dbf', OF_Read)     !Datei öffnen
    !
    ! Vorspann im Header lesen
    !
    While anzahl < 32                           ! die ersten 32 Character lesen
        Set nChar =  SalFileGetChar(hFile)
        If anzahl >= 4 AND  anzahl <= 7         !Byte Nr 4-7
                                                !für Gesamtzahl Sätze
            Set nAlleSaetze = nAlleSaetze + nChar
            Set df5 = nAlleSaetze
        If anzahl = 8 OR anzahl = 9                     ! Byte Nr 8 und 9
                                                ! für Pos. 1. Datensatz
```

```
        Set nPosErsterSatz = nPosErsterSatz + nChar
        Set df3 = nPosErsterSatz
    If anzahl = 10 OR anzahl = 11                ! Byte Nr 10 und 11
                                                 ! für Länge Datensatz
        Set nLaengeSatz = nLaengeSatz + nChar
        Set df4 = nLaengeSatz
    Set anzahl = anzahl + 1
    Set nAlleZeichen = nAlleZeichen +1
!
! Info zu den einzelnen Feldern lesen
!
Set nNeueZeile = 0
While nAlleZeichen < nPosErsterSatz -5 AND nChar != 13
    Set anzahl = 0               ! Anzahl gelesener Zeichen für Feldnamen
    ! eine neue leere Zeile in die Tabelle einfügen
    ! Call SalTblInsertRow(tbl1, nNeueZeile +1)
    Call SalTblInsertRow(tbl1, TBL_MinRow +1)
    Set nNeueZeile = nNeueZeile +1
    ! gleichzeitig muß der Zähler (nNeuSatz), der die neu eingefügten Zeilen
    ! zählt, um eins erhöht werden (das geschieht in der Message-Action-
    ! Sektion der Tabelle 1)
    While anzahl < 11 AND nChar != 13        ! einzelne Zeichen
                                             ! des Feldnamens lesen
        Set nChar =  SalFileGetChar(hFile)
        If nChar = 13
            Break
        If (nChar>=65 AND nChar<=90 AND nChar !=0 AND nChar != 255)
        OR nChar = 95
                                                 ! 95 = Unterstrich
            Set ml2 =  ml2 || SalNumberToChar( nChar )
            Set sName = sName || SalNumberToChar( nChar )
        Set anzahl = anzahl + 1
        Set nAlleZeichen = nAlleZeichen + 1
    ! Feldname in das erste Tabellenfeld schreiben
    Set tbl1.col1 = sName
    Set sName = ''
    Set nNeuSatz = nNeuSatz + 1
    Set df1 = nNeuSatz                       ! Anzahl neuer Sätze in tbl1
    Set nChar =  SalFileGetChar(hFile)           ! ein Zeichen für Feldtyp
    Set ml2 =  ml2 ||' ' || SalNumberToChar( nChar )
    Set sTyp = sTyp || SalNumberToChar(nChar)
    Set tbl1.col2 = sTyp
    Set sTyp = ''
    Set anzahl = 0
    While anzahl < 4 AND nChar != 13             ! vier Zeichen überspringen
```

```
            Set nChar =  SalFileGetChar(hFile)
            Set anzahl = anzahl + 1
            Set nAlleZeichen = nAlleZeichen +1
        !
        Set nChar =  SalFileGetChar(hFile)                  ! ein Zeichen für Feldlänge
        If nChar = 13
            Break
        Set ml2 =  ml2 ||' ' || SalNumberToStrX( nChar,0 )
        Set nLaenge =  nChar
        Set tbl1.col3 = nLaenge                     ! in die Tabelle schreiben
        Set nLaenge = 0                             ! wieder mit 0 belegen
        !
        Set nChar =  SalFileGetChar(hFile)                  ! ein Zeichen für
                                                    ! Dezimalstellen
        Set ml2 =  ml2 ||' ' || SalNumberToStrX( nChar,0 )
        Set ml2 = ml2 || '      '
        Set anzahl = 0
        Set nDez =  nChar
        Set tbl1.col4 = nDez                        ! in die Tabelle schreiben
        Set nDez = 0                                        ! wieder mit 0 belegen
        While anzahl < 14 AND nChar!= 13                    ! 14 Zeichen übergehen
            Set nChar =  SalFileGetChar(hFile)
            Set anzahl = anzahl + 1
            Set nAlleZeichen = nAlleZeichen +1
        Set feldzahl = feldzahl + 1
    ! Ende
! Jetzt muß die Lesemarke an den Beginn des ersten Datensatzes bewegt werden
    Call SalFileSeek ( hFile, 0, FILE_SeekBegin )           ! auf erstes
                                                    ! Zeichen der Datei
    Set nAlleZeichen = 0
    While nAlleZeichen < nPosErsterSatz                     ! auf erstes
                                                    ! Zeichen des Satzes
        Set nChar = SalFileGetChar(hFile)
        Set nAlleZeichen = nAlleZeichen +1
    ! Jetzt wird der erste Satz eingelesen
    Set nVglAnzahl = 0
    While nVglAnzahl < nAlleSaetze                  ! Solange nicht am Ende
                                                    ! der Datei
        Set anzahl = 0                              ! Anzahl gelesener Zeichen
        Call SalTblInsertRow(tbl2,1)
        While anzahl < nLaengeSatz
            Set nChar = SalFileGetChar(hFile)
            Set anzahl = anzahl +1
            ! Set df2 = df2||  SalNumberToChar( nChar)
            Set tbl2.col1 = tbl2.col1 ||  SalNumberToChar( nChar)
```

```
        Set nVglAnzahl = nVglAnzahl +1
! jetzt wird durch die Tabelle 1 "durchgesprungen" und für
! jede dort definierte Spalte die dazugehörigen Daten gelesen
```

Der darunter angeordnete Push Button zur Erzeugung der SQL-Anweisung hat folgenden Inhalt der Message Actions:

```
On SAM_Click
    Set nZeile = 0
    Set strCreateTable = 'Create table test( '          ! Anfang der SQL-Anweisung
    While nZeile < nNeuSatz
        Call SalTblSetFocusRow(tbl1,nZeile)
        ! dBASE-Datentypen werden nach SQL übertragen
        ! der Typ Memo ist hier nicht eingebaut
        If tbl1.col2 = 'N'
            Set sDatatype = 'INTEGER'
            Set sLaenge = SalNumberToStrX(tbl1.col3,0)
            ! SQL-Anweisung wird erstellt
            Set strCreateTable = strCreateTable || tbl1.col1 ||'  ' || sDatatype|| ' ' ||  ', '
        If tbl1.col2 = 'C'
            Set sDatatype = 'CHAR('
            Set sLaenge = SalNumberToStrX(tbl1.col3,0)
            Set strCreateTable = strCreateTable ||
            tbl1.col1 ||'  ' || sDatatype|| ' ' || sLaenge || '), '
        If tbl1.col2 = 'L'
            Set sDatatype = 'CHAR('
            Set sLaenge = SalNumberToStrX(tbl1.col3,0)
            Set strCreateTable = strCreateTable ||
            tbl1.col1 ||'  ' || sDatatype|| ' ' || sLaenge || '), '
        If tbl1.col2 = 'D'
            Set sDatatype = 'DATE'
            Set sLaenge = SalNumberToStrX(tbl1.col3,0)
            Set strCreateTable = strCreateTable || tbl1.col1 ||'  ' || sDatatype|| ' ' || ', '
        ! jetzt in if-Anweisung eingebaut
        ! Set sLaenge = SalNumberToStrX(tbl1.col3,0)
        ! SQL-Anweisung wird erstellt
        ! Set strCreateTable = strCreateTable || tbl1.col1 ||'  ' || sDatatype||' ' || sLaenge || '), '
        Set nZeile = nZeile + 1
    Set strCreateTable = strCreateTable || ')'
    ! hier muß das letzte Komma entfernt werden
    ! dazu werden zwei Teilstrings gebildet
    Set sTeil1 = SalStrLeftX(strCreateTable, (SalStrLength(strCreateTable)-5))
    Set sTeil2 = SalStrRightX(strCreateTable, 5)
    Set nPosition = SalStrScan(sTeil2, ',')
```

```
    Set sTeil2 = SalStrReplaceX(sTeil2, nPosition,1, '')
    Set strCreateTable = sTeil1 || sTeil2
    Set ml3 = strCreateTable
```

Schließlich soll durch Drücken eines weiteren Pushbuttons die Tabelle angelegt werden. Hierzu erthält er folgenden Inhalt:

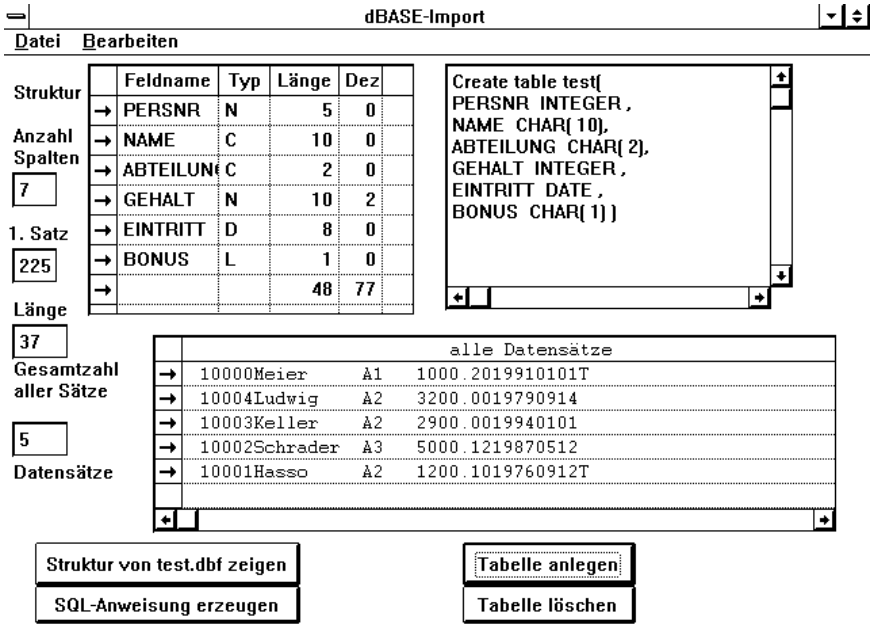
```
On SAM_Click
    Call SqlImmediate(ml3)
```

Der letzte Pushbutton dient nur dazu, eine eventuell vorhandene Tabelle mit gleichem Namen zu löschen, damit das Neuanlegen möglich ist.

```
        On SAM_Click
                Call SqlImmediate('drop table test')
```

Die gestartete Anwendung führt zu folgendem Aussehen des Formulars:

Abb. 4.2: Gestartete Anwendung dBASE.APP mit Struktur der DBF (Data Base File), Sql-Anweisung und Datensätzen.



5 Stichwortverzeichnis

Beachten Sie, daß Begriffe wie "Tabelle" oder "Data Field" in dem folgenden Stichwortverzeichnis nicht aufgenommen sind. Bei solchen allgemeinen Begriffen sollten Sie zunächst einen Blick in die Gliederung werfen.

A

Array 53, 54, 55, 63, 96, 114, 119, 207, 287

B

Bewegungsdaten 2, 3, 4, 6, 10, 245, 249
Bottom-Up 7

C

Client 26, 243, 256, 258, 271, 282
Commit 243, 248, 249, 256, 259, 260
Concatenate 210

D

Datentypen 15, 48, 49, 50, 103, 109, 111, 216, 228, 229, 257, 270

E

Ereignis 8, 62, 127
ERWin 22

F

Fremdschlüssel 246, 251, 252, 255, 257

H

Handle .. 49, 58, 59, 77, 79, 80, 88, 113, 115, 152, 153, 154, 155, 156, 160, 162, 165, 178, 179, 196, 198, 202, 222, 258, 267, 272, 273, 278, 286

I

IDOK 173

L

lParam 258, 260, 271, 281

M

Master-Detail 5, 6, 156, 219
Menu 45, 76, 77, 79, 85, 86, 95, 141, 142, 143, 144, 145, 146, 147, 148, 150, 151, 153, 192, 280
MenuOLEEdit 45, 144
Message Box 67, 165, 166, 173, 280

N

Newapp.App 40, 43, 44, 46, 48, 155
Normalisierung 3, 4, 6, 213, 250, 255, 258

P

Primärschlüssel167, 245, 246, 247, 251, 252, 253, 255, 257, 259

Q

Quick Objects 232

R

Relationen 5, 258, 259
Rollback 243, 256, 259, 260
ROW_MarkDeleted 188
RPT_PrintAll 202, 203

S

SalCreateWindow 58, 62, 63, 70, 71, 77, 78, 79, 83, 121, 133, 134, 135, 152, 153, 154, 272
SalEnableWindow 70, 79, 135, 273
SalEndDialog 83, 175, 275, 281

SalGetFirstChild 123, 273
SalGetNextChild 123, 273
SalListAdd 112, 113, 115, 119, 276
SalListPopulate..... 70, 112, 115, 119, 120, 226, 276
SalListQuerySelection 118, 276
SalListQueryText 118, 276
SalMessageBeep 64, 173, 280
SalMessageBox..... 67, 280
SalModalDialog 83, 169, 275
SalQueryArrayBounds 115, 266
SalReportCreate 70, 191, 277
SalReportPrint 70, 191, 201, 202, 277
SalReportTableCreate 204, 205, 277
SalReportTablePrint 204, 277
SalReportTableView 204, 205, 277
SalReportView .. 70, 191, 194, 196, 199, 200, 219, 277, 283
SalScrollSetRange 140, 278
SalStatusSetText 140, 165, 273
SalTblColumnSum..... 278
SalTblCopyRows 212, 278
SalTblDefineSplitWindow . 70, 182, 185, 187, 189, 278
SalTblDoDeletes..... 189, 243, 278
SalTblDoInserts 185, 249, 278
SalTblDoUpdates 188, 278
SalTblInsertRow 70, 182, 231, 278, 288, 289
SalTblPopulate 70, 87, 88, 178, 180, 186, 188, 189, 208, 215, 229, 279, 283
SalTblSetFocusCell 70, 182, 183, 231, 279
SalTblSetLockedColumns 176, 279
SalWaitCursor 132, 280
SAM_AppExit..... 42, 156, 179, 281
SAM_AppStartup 42, 47, 48, 156, 281
SAM_Click 9, 34, 39, 58, 59, 62, 63, 64, 77, 78, 102, 108, 127, 130, 133, 134, 135, 137, 152, 196, 202, 226, 281, 287, 290, 291
SAM_Destroy..... 79, 80, 109, 135, 282
SAM_EndCellTab 219, 282
SAM_ScrollBar 139, 284
Server 12, 21, 26, 30, 107, 243, 256, 258, 259, 262, 271, 274
SQL 1, 2, 11, 12, 13, 16, 26, 59, 63, 136, 250, 256, 257, 260, 261, 262, 265, 274, 290
SQLBase .. 2, 4, 13, 21, 25, 30, 49, 156, 167, 228, 241, 256, 259, 260, 261
SqlConnect 48, 70, 72, 156, 179, 204, 242, 274
SqlDatabase 48, 156
SqlExecute 70, 160, 172, 173, 175, 195, 198, 201, 274
SqlFetchNext .. 160, 161, 162, 163, 164, 165, 173, 175, 195, 198, 201, 274
SqlFetchPrevious..... 161, 164, 274
SqlFetchRow..... 161, 274
SqlImmediate 62, 68, 70, 106, 107, 108, 137, 171, 175, 183, 185, 188, 189, 226, 228, 246, 247, 249, 274, 275, 291
SqlPrepare 70, 172, 173, 175, 185, 187, 189, 195, 198, 201, 274, 275
SqlPrepareAndExecute... 160, 162, 163, 169, 173, 275
SqlSetIsolationLevel 241, 242, 275
SQLTalk 1, 2, 11, 13, 20, 21, 22, 24, 26, 29, 30, 72, 243, 244, 254, 260, 261, 262
Stammdaten 2, 3, 5, 6, 10, 146, 157, 167, 213, 245, 247, 249, 250, 253, 259

T

Top-Down 7

V

Variable... 37, 42, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 58, 63, 64, 65, 72, 77, 80, 86, 95, 107, 114, 118, 128, 129, 140, 155, 156, 160, 161, 162, 165, 168, 172, 178, 179, 183, 184, 187, 188, 197, 198, 199, 200, 202, 204, 205, 208, 214, 215, 218, 222, 225, 228, 241, 248, 256

W

wParam..... 140, 258, 260, 271, 281