

Einstieg in **ABAP**[®]



Karl-Heinz Kühnhauser

Inhalt

Vorwort	11
1 ABAP und erste Schritte im SAP-System	15
1.1 Architektur des SAP-Systems im Überblick	16
1.1.1 Technische Architektur	16
1.1.2 Betriebswirtschaftlich-organisatorische Architektur	18
1.1.3 Plattformunabhängigkeit	20
1.2 Anwendungsprogramme und Laufzeitumgebung	21
1.2.1 Workprozesse	21
1.2.2 Struktur von ABAP-Programmen	23
1.3 Anmelden und Abmelden am System	25
1.3.1 Betriebswirtschaftlicher Modulüberblick	27
1.3.2 ABAP Workbench	29
2 Data Dictionary	33
2.1 Einstieg ins ABAP Dictionary	33
2.1.1 Datenbanktabellen	34
2.1.2 Tabelle anlegen und pflegen	35
2.2 Datenelemente und Domänen	39
2.2.1 Datenelement anlegen	39
2.2.2 Domäne anlegen	43
2.2.3 Datenelement prüfen und aktivieren	47
2.2.4 Technische Einstellungen der Tabelle pflegen	49
2.3 Datensätze erfassen	51
2.3.1 Datensätze eingeben	51
2.3.2 Tabelleninhalt anzeigen	52
3 Programmieren im ABAP Editor	55
3.1 ABAP-Report anlegen	55
3.2 ABAP Editor im Überblick	58
3.2.1 Modi des ABAP Editors	59
3.2.2 Steuerung des ABAP Editors	60
3.3 ABAP-Programme verstehen und bearbeiten	65

3.4	ABAP-Report ausführen	67
3.5	Datenbanktabelle lesen und ausgeben	69
3.6	Aufbereitung von Listen	70
3.6.1	Kettensatz	71
3.6.2	Linien	71
3.6.3	Leerzeilen	71
3.7	Quellcode schreiben und editieren	72
3.7.1	Anmerkungen zum Quellcode	73
3.7.2	Listbild unseres Beispiel-Quellcodes	77

4 Felder und Berechnungen 79

4.1	Report vorbereiten	79
4.2	Felder deklarieren	83
4.2.1	Variablen deklarieren	83
4.2.2	Konstanten deklarieren	87
4.3	Grundrechenarten	88
4.3.1	Kompatible und konvertible Datenobjekte	89
4.3.2	Konvertierungsregeln	90
4.3.3	Besonderheiten bei der Division	91
4.4	Codebeispiel zu Feldern und Berechnungen	92
4.4.1	Anmerkungen zum Quellcode	94
4.4.2	Verbesserte Listaufbereitung	96

5 Modifikation von Zeichenketten 99

5.1	Zeichenketten deklarieren	99
5.2	Zeichenkettenoperationen	101
5.2.1	Zeichenketten verschieben	101
5.2.2	Zeichenketten ersetzen	103
5.2.3	Zeichenketten verdichten	104
5.2.4	Zeichenkettenfelder zusammenziehen	104
5.2.5	Zeichenketten zerlegen	106
5.2.6	Zeichenkettenoperationen mit direkter Positionierung	106
5.3	Codebeispiel zu String-Operationen	108
5.3.1	Anmerkungen zum Quellcode	110
5.3.2	Ausgabe des Quellcodes	113

6 Debugging von Programmen 115

6.1	Überblick	115
6.2	Aufruf des ABAP Debuggers	115
6.3	Arbeiten mit dem ABAP Debugger	118

6.3.1	Felder-Modus	118
6.3.2	Tabellen-Modus	122
6.3.3	Watchpoints-Modus	124
6.3.4	Breakpoints-Modus	126
6.3.5	Statische Breakpoints	129

7 Modifikation von transparenten Datenbanktabellen 131

7.1	Datenbanktabelle kopieren	132
7.2	Ergänzung von Nicht-Schlüsselfeldern	136
7.2.1	Festwerte in Domäne einpflegen	136
7.2.2	Besonderheiten bei Währungs- und Mengenfeldern	138
7.2.3	Fremdschlüssel pflegen	141
7.2.4	Append-Strukturen pflegen	148
7.2.5	Include-Struktur einpflegen	151
7.3	Schlüsselfelder von Tabellen manipulieren	155
7.4	Tabellenfelder löschen	159
7.5	Tabellen löschen	160

8 Rechnen mit Datum und Zeit, Mengen und Währungen 165

8.1	Felddeklarationen	165
8.2	Rechnen mit Datumsfeldern	167
8.3	Rechnen mit Zeitfeldern	174
8.4	Rechnen mit Mengen- und Währungsfeldern	178
8.5	Codebeispiel zu Datums-, Zeit- und Währungsfeldern	180
8.5.1	Anmerkungen zum Quellcode	186
8.5.2	Ausgabe des Quellcodes	191

9 Daten in einer Datenbanktabelle verändern 193

9.1	Berechtigungskonzept	194
9.2	Sperrkonzept	195
9.3	Open SQL-Anweisungen	198
9.3.1	Neuen Datensatz anlegen	199
9.3.2	Bestehenden Datensatz ändern	201
9.3.3	Datensatz modifizieren	202
9.3.4	Datensatz löschen	203

9.4	Codebeispiel zu INSERT	205
9.4.1	Anmerkungen zum Quellcode	207
9.4.2	Ausgabe des Quellcodes	208
9.5	Codebeispiel zu UPDATE	209
9.5.1	Anmerkungen zum Quellcode	211
9.5.2	Ausgabe des Quellcodes	212
9.6	Codebeispiel zu MODIFY	212
9.6.1	Anmerkungen zum Quellcode	214
9.6.2	Ausgabe des Quellcodes	215
9.7	Codebeispiel zu DELETE	216
9.7.1	Anmerkungen zum Quellcode	218
9.7.2	Ausgabe des Quellcodes	219

10 Programmablaufsteuerung und logische Ausdrücke **221**

10.1	Kontrollstrukturen	221
10.2	Arbeiten mit Mustern	222
10.3	Verzweigungen	225
10.3.1	IF-Struktur	225
10.3.2	CASE-Struktur	228
10.4	Schleifen	230
10.4.1	SELECT-Schleife	230
10.4.2	DO-Schleife	231
10.4.3	WHILE-Schleife	233
10.4.4	Abbrucharweisungen für Schleifen	234
10.5	Logische Ausdrücke	237
10.5.1	Einfache logische Ausdrücke	238
10.5.2	Verknüpfte logische Ausdrücke	240
10.6	Codebeispiel zu IF	243
10.6.1	Anmerkungen zum Quellcode	246
10.6.2	Ausgabe des Quellcodes	248
10.7	Codebeispiel zu CASE	248
10.7.1	Anmerkungen zum Quellcode	250
10.7.2	Ausgabe des Quellcodes	252
10.8	Codebeispiel zu DO und Abbruchbedingungen	253
10.8.1	Anmerkungen zum Quellcode	256
10.8.2	Ausgabe des Quellcodes	257
10.9	Codebeispiel zu WHILE und logischen Ausdrücken	258
10.9.1	Anmerkungen zum Quellcode	261
10.9.2	Ausgabe des Quellcodes	263

11 Selektionsbildschirme 265

11.1	Ereignisse	266
11.1.1	Reihenfolge von Ereignissen	266
11.1.2	Beispiele für Ereignisse	268
11.2	Einfache Selektionen	269
11.2.1	PARAMETERS-Anweisung	269
11.2.2	Zusätze zur PARAMETERS-Anweisung	271
11.3	Komplexe Selektionen	275
11.3.1	SELECT-OPTIONS-Anweisung	276
11.3.2	Mehrfachselektionen	277
11.3.3	Zusätze zur SELECT-OPTIONS-Anweisung	280
11.4	Selektionstexte verwenden	281
11.4.1	Textelemente im Überblick	281
11.4.2	Selektionstexte anlegen	281
11.5	Selektionsbildschirm speichern	284
11.5.1	Selektionsvariante anlegen	285
11.5.2	Report mit Variante starten	290
11.6	Ergänzende Textobjekte	291
11.6.1	Textsymbole anlegen	291
11.6.2	Nachrichten anlegen	293
11.7	Selektionsbildschirm frei gestalten	296
11.7.1	Einzelne Zeile gestalten	296
11.7.2	Zeilenblock gestalten	298
11.8	Codebeispiel zum Selektionsbildschirm (einfache Form)	299
11.8.1	Anmerkungen zum Quellcode	303
11.8.2	Ausgabe des Quellcodes	304
11.9	Codebeispiel zum Selektionsbildschirm (erweiterte Form)	305
11.9.1	Anmerkungen zum Quellcode	310
11.9.2	Ausgabe des Quellcodes	314

12 Interne Tabellen 317

12.1	Sinn und Zweck interner Tabellen	317
12.2	Aufbau und Arten von internen Tabellen	319
12.3	Interne Standardtabelle anlegen	322
12.3.1	Traditionelle Schreibweise mit Kopfzeile	322
12.3.2	Objektorientierte Schreibweise ohne Kopfzeile	323
12.4	Interne Standardtabelle füllen	326
12.4.1	Interne Tabelle mit Kopfzeile füllen	326
12.4.2	Interne Tabelle mit Workarea füllen	331

12.5	Interne Tabelle zeilenweise verarbeiten	335
12.5.1	Interne Tabelle mit Kopfzeile verarbeiten	335
12.5.2	Interne Tabelle mit Workarea verarbeiten	341
12.6	Inhalte von internen Tabellen löschen	344
12.6.1	Interne Tabellen mit Kopfzeile löschen	344
12.6.2	Workareas und interne Tabellen mit Workarea löschen	345
12.7	Codebeispiel zu ITAB mit Kopfzeile	346
12.7.1	Anmerkungen zum Quellcode	351
12.7.2	Ausgabe des Quellcodes im ABAP Debugger verfolgen	354
12.8	Codebeispiel zu ITAB mit Workarea	359
12.8.1	Anmerkungen zum Quellcode	365
12.8.2	Ausgabe des Quellcodes im ABAP Debugger verfolgen	367

13 Programme modularisieren 371

13.1	Überblick	371
13.2	Quelltext-Module	373
13.3	Prozeduren	375
13.3.1	Unterprogramme	375
13.3.2	Funktionsbausteine	387
13.4	Speicherbereiche für Datenübergabe	402
13.4.1	Globales SAP Memory	402
13.4.2	Lokales SAP Memory	403
13.4.3	ABAP Memory	403
13.5	Codebeispiele zur Modularisierung	404
13.5.1	Anmerkungen zum Quellcode	412
13.5.2	Ausgabe des Quellcodes	414
13.6	Codebeispiele zum Aufruf fremder Reports	415
13.6.1	Anmerkungen zum Quellcode	423
13.6.2	Ausgabe des Quellcodes	426

Der Autor 427

Index 429

Vorwort

Dieses Buch will ein Lehr- und Lernbuch für alle Interessierten sein, die in die Welt der ABAP-Programmierung einsteigen möchten. Diesem Ziel ordnet es (fast) alles unter: Es »lügt«, indem es vorgibt, alles wäre ganz unkompliziert. Es vereinfacht und simplifiziert, indem es technische und betriebswirtschaftliche Probleme auf ein Minimum reduziert. Dafür nimmt es Lücken in Kauf, weil es das Thema ABAP nicht in epischer Breite behandeln kann und will, sondern sich auf einen thematischen Kern und den roten Faden hin zum Lernziel beschränkt. Damit möchte Ihnen dieses Buch vor allem eines vermitteln: Erfolgserlebnisse.

Ziel des Buches

Der »Einstieg in ABAP« kann deshalb auch keinen allgemeinen SAP-Grundlagenkurs ersetzen, ebenso wenig vertiefende Literatur zu speziellen ABAP-Themen, und es erhebt auch keinen Anspruch auf Vollständigkeit. Es möchte auch kein Trockenkurs sein, sondern Sie einfach nur zum Lernen und Arbeiten am R/3-System motivieren. Alle dafür relevanten Arbeitsschritte werden ausführlich mit Beispielen, Hintergrundinformationen und Quellcodebeschreibungen erklärt.

Sie werden vom einfachsten ABAP-Report über wichtige Arbeiten im ABAP Dictionary zur Tabellenpflege bis hin zur modularisierten Ablaufsteuerung und komplexen Datenübergabestruktur zwischen Reports geführt. Dabei wird ein einfaches betriebswirtschaftliches Beispiel, eine fiktive Teilnehmerverwaltung, durchgängig beibehalten und immer wieder verwendet, um die Theorie direkt in der Praxis anzuwenden. Der Schwerpunkt dieses Buches liegt so auf der Vermittlung und Aneignung von ABAP-Wissen, nicht auf betriebswirtschaftlichen Zusammenhängen.

Damit ist es für Entwickler geschrieben, in deren Firmen ein SAP-System eingeführt wird. Es ist geschrieben für Berater und Projektleiter, die ABAP-Quellcode lesen und verstehen möchten und einige Änderungen selbst vornehmen wollen. Und es ist geschrieben für Studenten und Auszubildende, die ABAP im Rahmen ihrer Berufsausbildung lernen.

Zielgruppen

Um den Wissenstransfer optimal ausschöpfen zu können, sollten Sie allerdings einige Voraussetzungen mitbringen: Ideal wäre es sicherlich, sich mit SAP-Grundlagen wie der Navigation im Hauptmenü schon vorab beschäftigt zu haben. Auch Erfahrungen mit allgemeiner Programmierlogik aus anderen Programmiersprachen, Makros oder Skripten ist für den Lernerfolg förderlich. Ideal wäre auch der Zugang zu einem R/3-System mit den entsprechenden Berechtigungen – hier genügt auch das Mini-

Systemvoraussetzungen

SAP-System, das verschiedenen Büchern von SAP PRESS (<http://www.sap-press.de>) beiliegt oder von SAP-Kunden über die Website der SAP (<http://www.sap.com>) kostengünstig erworben werden kann. Das Mini-SAP-System stellt die ABAP-Entwicklungsumgebung bereit – und mehr benötigen Sie für den Lernerfolg mit diesem Buch nicht.

Kapitelübersicht und Aufbau

Kapitel 1 vermittelt Ihnen das für den Einstieg in ABAP nötige Wissen über die organisatorische und technische Architektur von SAP-Systemen und die Entwicklungsvoraussetzungen. Ferner wird die Arbeitsteilung zwischen Laufzeitumgebung und Anwendungsprogrammen gezeigt sowie die Struktur von ABAP-Reports.

Kapitel 2 liefert einen Überblick über Sinn und Zweck des ABAP Dictionary. Am Beispiel einer transparenten Tabelle lernen Sie das Anlegen einer Tabelle bis zum Erfassen und Anzeigen von Tabelleneinträgen und alle nötigen Arbeiten mit Datenelementen, Domänen und technischen Einstellungen.

In **Kapitel 3** legen Sie Ihren ersten ABAP-Report an, pflegen seine Eigenschaften, erstellen den Quellcode und führen ihn aus. Gleichzeitig lernen Sie die ersten ABAP-Anweisungen und die nötige Syntax kennen.

Das »Grundrechnen mit Variablen« üben Sie in **Kapitel 4**, betrachten die Eigenschaften von Datenobjekten und den Unterschied zwischen Kompatibilität und Konvertibilität. Gleichzeitig lernen Sie einige Kniffe für eine bessere Listaufbereitung kennen.

In **Kapitel 5** arbeiten Sie mit Zeichenketten und führen Stringoperationen in verschiedenen Varianten aus. Sie suchen Teilstrings und modifizieren Zeichenketten.

Kapitel 6 ist dem Verfolgen des Programmablaufs und dem Suchen von Programmfehlern gewidmet. Dafür arbeiten Sie mit dem ABAP Debugger und seinen wichtigsten Modi.

In **Kapitel 7** erweitern und modifizieren Sie eine transparente Tabelle, lernen, was ein Fremdschlüssel ist und bauen Prüfungen für die Tabelleninhalte ein. Dazu arbeiten Sie mit Festwerten in der Domäne, Wertetabellen und Prüftabellen. Außerdem lernen Sie Besonderheiten bei Währungs- und Mengefeldern kennen.

In **Kapitel 8** deklarieren Sie Datums- und Zeitfelder, verarbeiten sie und betrachten ihre besonderen Eigenschaften. Zum Verständnis dienen viele Beispiele, die auch im ABAP Debugger verfolgt und beschrieben werden.

Den Inhalt einer Datenbanktabelle modifizieren Sie in **Kapitel 9**, fügen neue Zeilen über ABAP-Anweisungen ein und ändern und löschen bestehende Zeilen. Auch die Risiken beim Löschen und der Datenmanipulation werden gezeigt.

Das Anwendungsbeispiel wird langsam komplexer: In **Kapitel 10** treffen Sie Fallunterscheidungen, bauen Kontrollstrukturen und Verzweigungen ein und lernen logische Ausdrücke kennen.

In **Kapitel 11** lernen Sie, einen Report mit Eingabewerten für den Programmablauf in einem Selektionsbildschirm zu versorgen. Sie erarbeiten sich einfache und komplexe Selektionen und Selektionstexte und gestalten den Selektionsbildschirm völlig frei. Außerdem lernen Sie, was Textsymbole, Nachrichten und Varianten bedeuten, insbesondere in Zusammenhang mit dem Selektionsbildschirm.

Kapitel 12 zeigt interne Tabellen, ihre Bedeutung und verschiedenen Formen. Sie erfahren, was eine Kopfzeile ist, lernen den Unterschied zu einer Workarea und wichtige Anweisungen zum Verarbeiten der Tabellenzeilen kennen. Zum besseren Verständnis wird auch in diesem Arbeitsschritt besonders Wert auf die Ablaufverfolgung im ABAP Debugger gelegt.

Kapitel 13 dient schließlich der Modularisierung von Programmen. Sie erfahren, was Quelltextmodule, interne und externe Unterprogramme sowie Funktionsbausteine sind. Ein weiterer wichtiger Punkt dieses Kapitels ist die Übergabe von Daten und Datenstrukturen an gerufene Module.

Um Ihnen Zeit raubende Tipparbeit zu ersparen, können Sie die Quellcodes zu allen Beispiellistings von der Website zum Buch unter <http://www.sap-press.de> herunterladen.

**Download der
Codebeispiele**

Nicht versäumen möchte es, mich bei meiner Frau und bei meinen Kindern zu bedanken. Sie haben es mir durch viel Verzicht auf unser Familienleben und durch häusliche »Entlastungstätigkeiten« ermöglicht, dieses Buch zu schreiben.

Danksagung

Allen Lesern wünsche ich viel Spaß und viel Erfolg beim Lesen und Lernen!

Karl-Heinz Kühnhauser
Nürnberg, im Juli 2005

10 Programmablaufsteuerung und logische Ausdrücke

Entscheidungen zu treffen ist schwer. Gesunder Menschenverstand und logische Ausdrücke in der Programmierung scheinen manchmal widersprüchlich zu sein: Warum tut der Rechner nie das, was man meint? Hier lernen Sie die richtigen Regeln im Umgang mit dem Programmablauf.

Noch im letzten Kapitel blieb nichts anderes übrig, als beispielsweise den Returncode der DELETE-Anweisung in die Liste zu schreiben. Damit hätte sich der Programmablauf überhaupt nicht geändert, wenn ein Fehler aufgetreten wäre und der Returncode einen anderen Wert als Null zurückgeliefert hätte, sondern es wäre die gleiche Liste produziert worden, nur mit kaum sinnvollen Ergebnissen.

Für die Praxis ist es sinnvoller, eine Fallunterscheidung zu treffen, um beispielsweise Fehlerfälle zu erkennen und gesondert zu behandeln. Vielleicht ist es in bestimmten Fällen sogar besser, gar keine Liste mehr zu erzeugen und stattdessen das Programm mit einer Meldung zu beenden. In diesem Kapitel lernen Sie das Handwerkszeug für diese Fälle: die *Programmablaufsteuerung* und die *logischen Ausdrücke*.

**Fallunter-
scheidungen**

10.1 Kontrollstrukturen

Ein Programm hat eine Aufgabe zu lösen, die wiederum aus Teilaufgaben besteht. Einige Teilaufgaben sind in jedem Fall zu lösen, andere nur unter bestimmten Bedingungen, manche alternativ. Zu jeder Teilaufgabe gehören Anweisungen, die wiederum in Anweisungsblöcken zusammengefasst sind. Ein Anweisungsblock kann auch nur aus einem einzigen Kommando, beispielsweise einer WRITE-Anweisung bestehen, andererseits auch komplexe Prozesse beinhalten.

Unter welchen Randbedingungen ein Anweisungsblock durchlaufen wird, regeln Anweisungen für die Programmablaufsteuerung, die so genannten *Kontrollstrukturen*. Sie enthalten die Bedingung und die dazugehörige Aktion. Die Programmablaufsteuerung hat die Aufgabe, die Teilaufgaben sinnvoll durch Kontrollstrukturen zu verbinden. Es gibt Kontrollstrukturen für Fallunterscheidungen und für Wiederholungen.

**Bedingung und
Aktion**

Das mag abstrakt klingen, ist es aber nicht, denn solche Kontrollstrukturen existieren genauso im täglichen Leben: Wir gehen ins Kino, wenn wir Zeit haben, jemand mitgeht und uns der Film interessiert – sonst bleiben wir zu Hause und lesen. Nur sagt das niemand folgendermaßen: `IF` Zeit und Begleitung und Interesse, `THEN` Kino, `ELSE` zu Hause bleiben und lesen, `ENDIF`.

Strukturanfang und -ende

Eine Kontrollstruktur hat einen Strukturanfang und ein Strukturende. Im gerade genannten Beispiel ist der Strukturanfang die Anweisung `IF` und das Strukturende die Anweisung `ENDIF`: Beide Kontrollstrukturanweisungen gehören immer zusammen wie Klammern in der Mathematik. Dazwischen stehen die Anweisungsblöcke für die Teilaufgaben. Kontrollstrukturen verzweigen zu den relevanten Anweisungsblöcken oder wiederholen bestimmte Anweisungen in einer Schleife.

10.2 Arbeiten mit Mustern

Die Anweisungen für die Kontrollstrukturen müssen wie alle anderen ABAP-Anweisungen auch in den Quellcode geschrieben werden. Unter Umständen kann es die Arbeit erleichtern, dabei mit *Mustern* zu arbeiten, denn sie fügen vorgefertigte Strukturen ab der Cursorposition in den Quellcode ein.

Muster einfügen

Um ein solches Muster einzufügen, positionieren Sie den Cursor in der Zeile, ab der das Muster eingefügt werden soll, und drücken den **Muster**-Button wie in Abbildung 10.1 gezeigt.

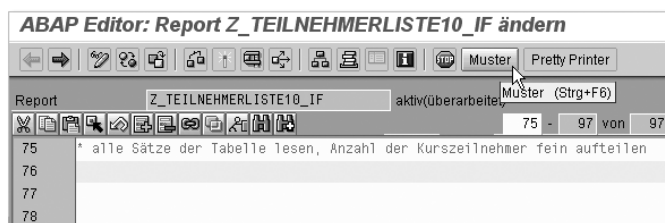


Abbildung 10.1 Muster einfügen

Es erscheint ein Fenster, in dem häufig verwendete Muster angeboten werden, beispielsweise die `SELECT`-Anweisung. Andere Muster wie etwa die `IF`-Struktur erhalten Sie über die Zeile **Anderes Muster**. Da eine `IF`-Struktur eingefügt werden soll, geben Sie dort »IF« ein (siehe Abbildung 10.2) und klicken auf den **Weiter**-Button. Das System fügt dann die komplette `IF`-Strukturbeschreibung in den Quellcode ein (siehe Abbildung 10.3).

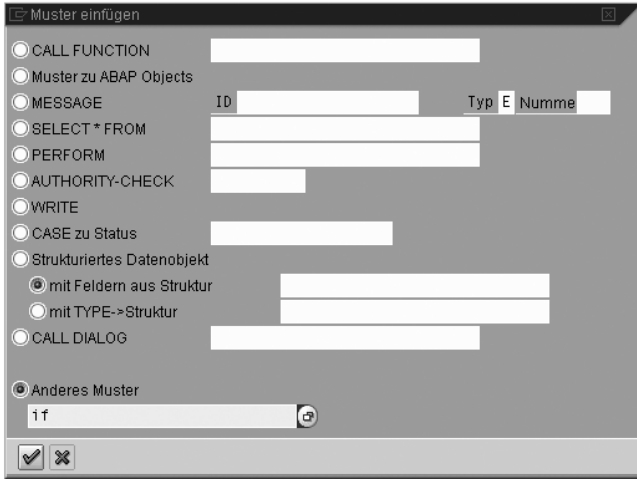


Abbildung 10.2 Auswahlfenster für Muster



Abbildung 10.3 In den Quellcode per Muster eingefügte IF-Struktur

Die Zeilen, die nicht benötigt werden, löschen Sie oder machen Sie zur Kommentarzeile. Die Teile, die fehlen, also Bedingungen und Anweisungsblöcke, ergänzen Sie. Für den Fall, dass Sie noch nicht sicher sind, wie das Muster genau zu bezeichnen ist, können Sie nach dem Fokussieren des Feldes **Anderes Muster** den **Hilfe**-Button drücken (siehe Abbildung 10.4). Sie erhalten dann ein weiteres Fenster mit möglichen Mustern angezeigt: Es beginnt mit verschiedenen Mustern für Kommentarzeilen, enthält die Anweisungen für die Kontrollstrukturen dieses Kapitels sowie weitere Muster (siehe Abbildung 10.5).

Initiale Anpassungen

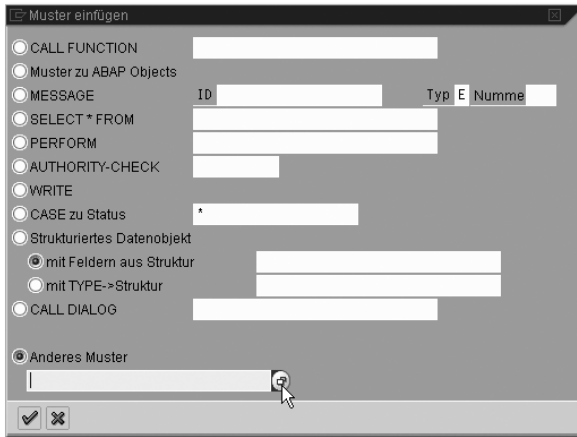


Abbildung 10.4 Andere Muster auswählen

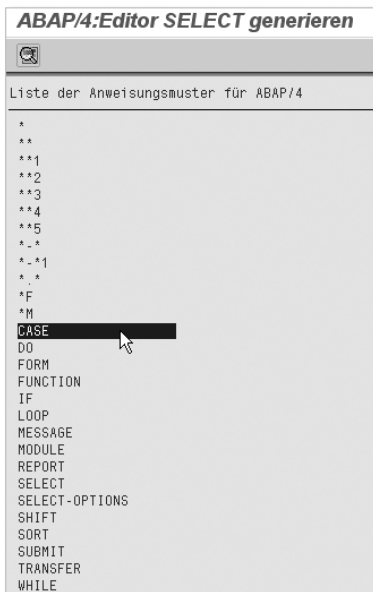


Abbildung 10.5 Listfenster für Muster

Hier wählen Sie das relevante Muster aus – entweder durch Doppelklick oder durch Markieren mit anschließendem Klick auf den **Auswählen**-Button – und es wird daraufhin in das Auswahlfenster übernommen. Der Rest funktioniert analog wie oben beschrieben.

Vollständiges Einfügen

Das Einfügen von Mustern bewirkt immer ein vollständiges Einfügen der Struktur. Wenn es im Einzelfall einfacher und schneller ist, die Struktur an-

weisungen manuell in den Quellcode zu schreiben, spricht nichts dagegen, dies zu tun. Dafür gewährleistet das Einfügen von Strukturen per Muster beispielsweise eine saubere Schachtelung über alle Ebenen. Bei der späteren Arbeit mit Funktionsbausteinen (siehe Kapitel 13) werden Sie die Arbeit mit Mustern ebenso schätzen lernen.

10.3 Verzweigungen

Eine Verzweigung entspricht dem umgangssprachlichen »Entweder–Oder«: Ausschlaggebend, wofür man sich entscheidet, ist eine Bedingung formuliert. Diese Bedingung könnte einfach ein Zustand sein wie »Es regnet.«, oder eine komplexe Bedingung, die aus vielen verschachtelten Bedingungen zusammengesetzt ist, beispielsweise: »Qualitätsklasse 1 gilt als erreicht, wenn die Qualitätskriterien 1 bis n erfüllt sind. Qualitätskriterium 1 ist erfüllt wenn die Qualitätsmerkmale 1 bis n erfüllt sind. Qualitätsmerkmal 1 ist erfüllt, wenn ...«.

In allen Fällen gilt aber, dass bestimmte Aktionen – im Quellcode Anweisungsblöcke – nur dann durchlaufen werden, wenn die Bedingung erfüllt ist. Mag es im umgangssprachlichen Bereich noch Interpretationsspielräume geben, wie eine Bedingung »genau« lautet und wann sie erfüllt ist, können im ABAP-Quellcode nur operationale, also messbare Bedingungen und Zustände verwendet und geprüft werden. Dazu werden Felder und Feldinhalte verglichen oder mit logischen Ausdrücken verknüpft.

Operationale Bedingungen formulieren

10.3.1 IF-Struktur

Die einfachste Kontrollstruktur für eine Verzweigung beginnt in ABAP mit der Anweisung `IF` und einer Bedingung, sie endet mit der Anweisung `ENDIF`. Zwischen den beiden Anweisungen stehen die Kommandos, die ausgeführt werden, falls die Bedingung erfüllt ist. Zu Beginn prüfen wir zunächst nur einen Feldinhalt ab, im Lauf der weiteren Arbeit werden Sie sehen, dass sich für die Bedingungen der `IF`-Struktur beliebig komplexe logische Ausdrücke verwenden lassen.

Angenommen, Sie möchten nur die Teilnehmer des Kurses »PC-Grundlagen« zählen, so könnte dies beispielsweise so aussehen: Sie verarbeiten zunächst alle Sätze der Tabelle ganz normal. Doch immer dann, wenn ein Satz diese besondere Bedingung erfüllt, zählen Sie die Teilnehmerzahl um 1 hoch.

IF ... ENDIF

```
IF zteilnehmer02-zzkurstitel = 'PC-Grundlagen'.  
    tz_pc_grundlagen = tz_pc_grundlagen + 1.  
ENDIF.
```

Hat das Feld für den Kurstitel den Inhalt »PC-Grundlagen«, ist die Bedingung erfüllt und der Anweisungsblock, der hier nur aus einer Zeile besteht, wird ausgeführt. Auf diese Weise wird gezählt, wie viele Personen den Kurs »PC-Grundlagen« besuchen.

ELSE Eine Verzweigung nach dem Prinzip »entweder A oder B« enthält zusätzlich für den Fall »oder« die Strukturanweisung ELSE. Ist die Bedingung der IF-Anweisung erfüllt, wird der erste Anweisungsblock ausgeführt, ansonsten der zweite Anweisungsblock nach der ELSE-Anweisung.

Im Beispiel werden im ersten Anweisungsblock die Teilnehmer für den Kurs »SAP-Grundlagen« gezählt: Im zweiten Anweisungsblock wird ein Zähler für alle restlichen Teilnehmer gebildet. Gezählt wird also in jedem Fall. Nur welcher Zähler erhöht wird, hängt von der Bedingung ab.

```
IF zteilnehmer02-zzkurstitel = 'SAP-Grundlagen'.
    tz_sap_grundlagen = tz_sap_grundlagen + 1.
ELSE.
    tz_sonst = tz_sonst + 1.
ENDIF.
```

ELSEIF Die dritte Möglichkeit, mit der IF-Anweisung zu arbeiten, prüft zunächst wieder eine Bedingung. Ist sie nicht erfüllt, wird eine zweite Bedingung mit ELSEIF geprüft. Ist diese auch nicht erfüllt, wird eine dritte Bedingung mit ELSEIF geprüft usw. Das Spielchen lässt sich unbegrenzt fortsetzen und kann für die restlichen Fälle auch noch mit einer ELSE-Anweisung abgeschlossen werden.

Dazu folgendes Beispiel:

```
IF zteilnehmer02-zzkurstitel = 'Netzwerktechnik'.
    tz_netzwerktechnik = tz_netzwerktechnik + 1.
ELSEIF zteilnehmer02-zzkurstitel = 'PC-Grundlagen'.
    tz_pc_grundlagen = tz_pc_grundlagen + 1.
ELSEIF zteilnehmer02-zzkurstitel = 'SAP-Grundlagen'.
    tz_sap_grundlagen = tz_sap_grundlagen + 1.
ELSE.
    tz_sonst = tz_sonst + 1.
ENDIF.
```

Die erste Bedingung soll dann erfüllt sein, wenn im Feld für die Kursbezeichnung der Wert »Netzwerktechnik« enthalten ist. Im ersten Verarbeitungsblock werden dann die Teilnehmer für den Kurs »Netzwerktechnik« gezählt. Anschließend prüft das System keine weiteren Bedingungen

mehr, sondern geht zu `ENDIF` und setzt das Programm mit der ersten Anweisung nach der `IF`-Struktur fort.

Ist die erste Bedingung nicht erfüllt, wird geprüft, ob die zweite Bedingung erfüllt ist, ob also die Kursbezeichnung »PC-Grundlagen« lautet. Ist die zweite Bedingung erfüllt, wird der zweite Anweisungsblock ausgeführt und der Zähler für die Teilnehmer am Kurs »PC-Grundlagen« erhöht. Wenn die zweite Bedingung ebenfalls nicht erfüllt ist, wird die dritte Bedingung (»SAP-Grundlagen«) geprüft. Trifft sie zu, werden die Teilnehmer dieses Kurses hochgezählt.

Eins, zwei
oder drei

Falls die dritte Bedingung auch nicht erfüllt ist, tritt der »Sonst-Fall« ein und der Zähler für die bis dahin nicht erfassten Kurse und Teilnehmer wird erhöht. Danach springt das System zur Anweisung `ENDIF` und dann zum nächsten Kommando nach der `IF`-Struktur.

Die Anweisung `ELSE` gewährleistet, dass – salopp ausgedrückt – irgendeine Bedingung immer erfüllt ist: Entweder die Daten durchlaufen die Verarbeitung und die Anweisungsblöcke der `IF`- und `ELSEIF`-Strukturen oder sie durchlaufen den Anweisungsblock für die bis dahin nicht zutreffenden Bedingungen. `ELSE` ist gewissermaßen der »Sammelschacht«, für all die Fälle, auf die bislang in der Struktur noch keine Bedingungen zutrafen.

Sammelschacht

Natürlich können `IF`-Strukturen auch geschachtelt werden. Dabei ist aber darauf zu achten, dass Strukturanfang und Strukturende richtig zueinander passen. Wie in der Mathematik die Klammerebenen, müssen auch hier die Schachtelungsebenen sauber getrennt werden. Es ist eine Frage des Arbeitsstils, der gültigen Programmierkonventionen und der Transparenz, inwieweit Sie geschachtelte `IF`-Strukturen durch Verknüpfung mehrerer logischer Ausdrücke ersetzen. Dazu zwei Beispiele:

`IF`-Strukturen
schachteln

Beispiel 1: Geschachtelte `IF`-Strukturen

```
IF Bedingung_1
  IF Bedingung_2
    Anweisungsblock_1
  ENDIF
ENDIF
```

Beispiel 2: Verknüpfte logische Ausdrücke

```
IF Bedingung_1 und Bedingung_2
  Anweisungsblock_1
ENDIF
```

**Logische
Ausdrücke**

Beide Beispiele sind inhaltlich gleich: Anweisungsblock_1 wird nur dann ausgeführt, wenn beide Bedingungen erfüllt sind.

Grundsätzlich geht Transparenz aber vor Genialität. Wenn durch Verknüpfen von logischen Ausdrücken Konstrukte entstehen, die Bildschirmseiten füllen, sollten Sie bitte an Dritte denken, die diese Konstrukte später verstehen und ändern müssen. In solchen Fällen sollte geprüft werden, ob durch Schachtelung vielleicht eine bessere Lesbarkeit, Transparenz und Änderbarkeit erzielt werden könnte. Wenn dies der Fall ist, ist die Schachtelung vorzuziehen.

10.3.2 CASE-Struktur

CASE ... ENDCASE

Starten wir für die CASE-Struktur zunächst wieder mit einer Analogie aus dem täglichen Leben: Eine Fußgängerampel ist entweder rot oder grün. Im Fall rot bleiben wir stehen, im Fall grün gehen wir. Wir blicken nur auf die Ampel, weitere Bedingungen interessieren uns nicht.¹

Ähnlich wie der sture Fußgänger »blickt« die CASE-Struktur nur auf ein Objekt. Beim Fußgänger ist es die Ampel, bei der CASE-Struktur ein Feld, ein Datenobjekt. Dabei wird eine Fallunterscheidung getroffen und der Inhalt des Felds wird abgefragt. Stimmen Bedingung und Feldinhalt überein, wird der zugehörige Anweisungsblock ausgeführt, sonst nicht. Auch in einer CASE-Struktur wird immer nur ein Anweisungsblock abgearbeitet. Nach dem Anweisungsblock verzweigt das System zu ENDCASE, ohne weitere Feldinhalte zu prüfen, geht danach zum ersten Kommando nach der CASE-Struktur und setzt die Verarbeitung dort fort.

**Unterschiede
zwischen IF- und
CASE-Struktur**

Während also die IF-Struktur komplexe logische Bedingungen prüfen kann, trifft die CASE-Struktur Fallunterscheidungen je nach Inhalt nur eines einzigen Feldes. Vereinfacht ausgedrückt: Bei der IF-Struktur kann es den Fall geben, dass es regnet und die Sonne scheint. Bei der CASE-Struktur gibt es diesen Fall nicht, entweder es scheint die Sonne oder es regnet.

**Eindeutige
Abstimmung**

Wie so oft kommt es also wieder einmal auf die Aufgabenstellung an. Die CASE-Struktur ist immer dann im Vorteil, wenn es gelingt, eine Bedingung eindeutig mit einem Feldinhalt abzustimmen.

In unserem Beispiel, in dem die Teilnehmer eines Kurses gezählt werden, könnte man auch sehr gut mit der CASE-Struktur arbeiten: Sie prüfen den

¹ Die Konsequenzen einer derart eingeschränkten Sichtweise können im wahren Leben zwar fatal sein, aber lassen wir das hier einmal aus dem Spiel.

Inhalt des Tabellenfeldes ZTEILNEHMERO2-ZZKURSTITEL, je nach Feldinhalt wird in den entsprechenden Anweisungsblock verzweigt.

Danach geht das System zu ENDCASE und setzt die Verarbeitung mit dem ersten Kommando nach der CASE-Struktur fort. Die Strukturanweisung WHEN OTHERS übernimmt dabei dieselbe Funktion wie die Anweisung ELSE bei der IF-Struktur. Vergleichen Sie dazu den folgenden Beispieldcode:

```
CASE zteilnehmer02-zzkurstitel.  
  WHEN 'Netzwerktechnik'.  
    tz_netzwerktechnik = tz_netzwerktechnik + 1.  
  WHEN 'PC-Grundlagen'.  
    tz_pc_grundlagen = tz_pc_grundlagen + 1.  
  WHEN OTHERS.  
    tz_sonst = tz_sonst + 1.  
ENDCASE.
```

Ein typischer Anwendungsfall für die der CASE-Struktur ist die Auswertung des Returncodes. Es ist eindeutig, welche Anweisung einen Returncode zurückgibt, wie die Werte für den Returncode sein können und welche Fehlerfälle damit gemeint sind. Je nach Erfolg oder Misserfolg einer Anweisung hat SY-SUBRC einen eindeutigen Inhalt, dieser Inhalt wird über die CASE-Struktur abgefragt.

**Anwendungsfall:
Returncode**

Damit können für den Erfolgsfall und alle dokumentierten Fehlerfälle eigene Anweisungsblöcke geschrieben werden. Bei jeder ABAP-Anweisung ist hinterlegt, ob ein Returncode generiert wird, welcher Returncode zurückgeliefert werden kann und welche Fehlerfälle damit definiert werden. Entsprechend kann man dann die Anweisungsblöcke gestalten. Ziel dieser Bemühungen ist es, das Programm trotz eines Fehlers zu einem »ordnungsgemäßen« Ende zu führen, bzw. sinnvolle Alternativprozesse einzuleiten.

**Fehlerfälle
differenzieren**

Suchen Sie beispielsweise mit dem SELECT-Kommando in einer Tabelle, liefert das Kommando den Returncode 0, wenn es mindestens einen Satz in der Tabelle lesen konnte. Konnte kein einziger passender Satz gelesen werden, liefert die Anweisung den Returncode 4 zurück. Unter Returncode 0 steht also der Anweisungsblock für den günstigen Okay-Fall. Ist der Returncode 4, wird die Fehlerbehandlungsroutine durchlaufen. Vergleichen Sie dazu folgenden Codeausschnitt:

```
CASE sy-subrc.  
  WHEN 0.
```

```

* Okay-Fall
  WRITE: / tz_netzwerktechnik,
         'Teilnehmer für den Kurs Netzwerktechnik
         gefunden'.

  WHEN 4.
* Fehlerfall
  WRITE: / 'keine Teilnehmer für den Kurs
         Netzwerktechnik gefunden'.

  WHEN OTHERS.
* Kann nicht sein, anderer sy-subrc wird von der
  Anweisung nicht geliefert
ENDCASE.

```

Es gibt durchaus Anweisungen, die mehrere Fehler differenzieren können. Auch Funktionsbausteine liefern häufig differenzierte Fehlermeldungen zurück. CASE-Strukturen mit sechs oder acht verschiedenen Fehlerbehandlungsblöcken sind in der Praxis keine Seltenheit.

CASE-Strukturen können genau wie IF-Strukturen geschachtelt werden. Dabei müssen Sie ebenfalls auf eine saubere Trennung der Ebenen achten. Die Kombination von IF- und CASE-Strukturen ist ebenfalls möglich.

10.4 Schleifen

Schleifen, auch Wiederholungen genannt, führen Anweisungsblöcke mehrfach aus. Dabei werden Schleifen ohne Bedingung und mit Bedingung unterschieden.

10.4.1 SELECT-Schleife

SELECT ...
ENDSELECT

Mit der SELECT-Anweisung kennen Sie bereits die erste Schleife, der es die Möglichkeit gibt, eine Bedingung mit der Anweisung zu verknüpfen. Wenn Sie in einer Tabelle Sätze lesen, können Sie über eine Selektionsbedingung filtern und nur die Sätze lesen, die diese Bedingung erfüllen. Möchten Sie in der Teilnehmertabelle nur die Sätze lesen, bei denen als Kurstitel »Netzwerktechnik« steht, schreiben Sie:

```

SELECT * FROM zteilnehmer02
        WHERE zzkurstitel = 'Netzwerktechnik'.
        tz_netzwerktechnik = tz_netzwerktechnik + 1.
ENDSELECT.

```

Der Anweisungsblock, der hier nur aus einer Zeile besteht – hinter der Bedingung, die die Teilnehmeranzahl erhöht –, wird nur für die Treffersätze ausgeführt, sonst nicht.

10.4.2 DO-Schleife

Die Struktur dieser Schleife beginnt mit der Anweisung `DO` und endet mit der Anweisung `ENDDO`. Dazwischen steht der zu wiederholende Anweisungsblock.

DO ... ENDDO

```
DO
  Anweisungsblock
ENDDO
```

Bereits in diesem Moment sollten bei Ihnen die Alarmglocken schrillen, denn es in dieser Anweisung nicht angegeben, wie oft der Anweisungsblock ausgeführt werden soll, ob die Wiederholung ewig laufen oder wann sie abgebrochen werden soll. Doch keine Panik, im Folgenden werden einige Möglichkeiten vorgestellt, wie Sie die Zahl der Schleifendurchläufe begrenzen und Schleifen abbrechen.

**Vorsicht vor
Endlosschleifen!**

Bei der `DO`-Schleife ist zunächst wichtig, dass der Anweisungsblock in der Struktur das erste Mal angelaufen wird, ohne dass das System vorher eine Bedingung prüft. Selbst wenn man ein Kriterium zum Verlassen der Schleife einbaut, werden die Kommandos vor dem Abbruchkriterium dennoch mindestens einmal ausgeführt. Der Ort der Abbruchbedingung im Anweisungsblock ist deshalb für die Programmlogik sehr bedeutsam.

**Bedingung in der
DO-Schleife**

Um die Zahl der Schleifendurchläufe zu begrenzen, können Sie eine Obergrenze von Durchläufen festlegen. Wollen Sie, dass eine Schleife maximal zwölfmal durchlaufen wird, schreiben Sie:

TIMES

```
DO 12 TIMES.
```

Statt die Anzahl von Durchläufen fix als Zahl anzugeben, können Sie diesen Wert auch in einer Variablen hinterlegen. Weisen Sie beispielsweise der Variablen `MONATE` vom Typ `i` den Wert `6` zu, lautet die Anweisung:

```
DO monate TIMES.
```

Die Schleife wird dann sechsmal durchlaufen, danach wird das Programm mit der ersten Anweisung nach `ENDDO` fortgesetzt.

Während der Anweisungsblock in der Schleife wiederholt wird, führt das System automatisch einen Schleifenzähler mit. Im Feld `SY-INDEX` steht zu jedem Schleifendurchlauf der aktuelle Wert, beim ersten Durchlauf also

SY-INDEX

die 1, beim zweiten Durchlauf die 2 usw. Dieser Schleifenzähler kann sowohl im Debugging-Modus zur Fehlersuche als auch als Abbruchkriterium sehr nützlich sein.

DO-Schleifen sind schachtelbar und SY-INDEX wird im Falle einer Schachtelung vom System getrennt für jede Ebene extra geführt. Doch auch hier sollten Sie besondere Vorsicht walten lassen: Durch das Schachteln von Schleifen kann sehr schnell eine erhebliche Systembelastung entstehen. Zum Beispiel mit folgendem Konstrukt:

```
DO 50 TIMES.  
  Anweisungsblock_1  
  DO 100 TIMES.  
    Anweisungsblock_2  
  ENDDO.  
ENDDO.
```

Geschachtelte DO-Schleifen In diesem Beispiel gibt es eine innere und eine äußere Schleife. Die äußere Schleife wird 50-mal ausgeführt. Zu jedem Durchlauf des Anweisungsblocks 1 wird die innere Schleife 100-mal durchlaufen, sodass Anweisungsblock_2 somit 100 mal 50, also 5.000-mal ausgeführt.

Ablauflogik der Uhr Zum Verständnis, wie schnell die Zahl der Schleifendurchläufe ausufern kann, hilft ein Blick auf die Uhr – und zwar auf ihre Ablauflogik. Ein Tag hat 24 Stunden, eine Stunde hat 60 Minuten und eine Minute 60 Sekunden. Bildet man diese »Programmlogik« in ABAP ab, ergeben sich drei geschachtelte Schleifen. Im Kern steht ein Anweisungsblock, der die berechnete Zeit zur Anzeige übermittelt, und in diesem Beispiel nur aus einer Zeile besteht, der WRITE-Anweisung als Stellvertreter für die Anzeigevorbereitung:

```
DO 24 TIMES.  
  min = 0.  
  DO 60 TIMES.  
    sek = 0.  
    DO 60 TIMES.  
      WRITE: / std, min, sek.  
      sek = sek + 1.  
    ENDDO.  
    min = min + 1.  
  ENDDO.  
  std = std + 1.  
ENDDO.
```

Der Verarbeitungsblock im Kern zur Anzeige der Zeit wird an einem Tag 60 mal 60 mal 24, also 86.400-mal durchlaufen. Dabei würden 86.400 Zeilen in einer Liste ausgegeben. Unterstellt man 60 Zeilen pro Seite, wäre das eine Liste mit 1.440 Seiten! Damit es erst gar nicht so weit kommt, sollten Sie also wohl überlegt vorgehen, Schleifen gründlich planen, diese mit dem Debugger verfolgen und Abbruchbedingungen einbauen.

1.440 Seiten

10.4.3 WHILE-Schleife

Im Unterschied zur DO-Schleife prüft die WHILE-Schleife eine Bedingung, bevor der Anweisungsblock das erste Mal ausgeführt wird. Gilt die Bedingung, wird der Anweisungsblock ausgeführt. Danach prüft das System wieder, ob die Bedingung gültig ist. Falls ja, wird der Anweisungsblock noch einmal ausgeführt usw. Das Spiel wiederholt sich endlos, es sei denn, die Bedingung gilt nicht mehr, weil sie beispielsweise im Anweisungsblock verändert oder ein anderes Abbruchkriterium erreicht wurde.

WHILE ...
ENDWHILE

Die Syntax der WHILE-Schleife ist relativ einfach. Die Struktur beginnt mit der Anweisung WHILE und einer Bedingung, anschließend folgt der Anweisungsblock, der ausgeführt wird, falls die Bedingung erfüllt ist. Danach steht für das Strukturende die Anweisung ENDWHILE.

Bedingung vor der
WHILE-Schleife

```
WHILE Bedingung
  Anweisungsblock
ENDWHILE
```

Im folgenden Beispiel läuft eine Stoppuhr rückwärts und startet mit einer Vorlaufzeit in Sekunden. Ist der Wert für die Sekunden auf null heruntergezählt, soll der Startschuss erfolgen. Lässt man einmal das Problem beiseite, wie die Anzahl von Quarzschwingungen gezählt wird, damit eine Sekunde auch genau die richtige Zeiteinheit im Programm abbildet, und betrachtet nur den Anteil der Programmierlogik, dann gilt: Solange die Sekunden größer sind als null, soll eine weitere Sekunde abgezogen werden. Die Anweisungen für die Schleife lauten:

Stoppuhr

```
WHILE sek > 0.
  WRITE: / sek.
  sek = sek - 1.
ENDWHILE.
```

Natürlich muss vor der Schleife der Wert für Sekunden gesetzt sein und es wird geprüft, ob der Feldinhalt größer ist als null. Falls ja, wird der Wert ausgegeben und um 1 vermindert. Dann wird erneut geprüft, ob

der Wert größer ist als null usw. Hat das Feld SEK den Wert 0 erreicht, wird die Anweisung nicht mehr ausgeführt und zu `ENDWHILE` verzweigt. Anschließend setzt das System die Programmausführung mit der ersten Anweisung nach `ENDWHILE` fort.

SY-INDEX Die Anzahl der Schleifendurchläufe wird genau wie bei der `DO`-Schleife im Feld `SY-INDEX` protokolliert. `SY-INDEX` kann im Anweisungsblock gelesen und für die Programmsteuerung ausgewertet werden.

Natürlich kann man `WHILE`-Schleifen ebenso wie `DO`-Schleifen auch schachteln. Die Hinweise, die bei den `DO`-Schleifen hierzu gegeben wurden, gelten analog auch für die `WHILE`-Schleifen.

Schleifenkonstruktion Welche Schleifenstruktur im konkreten Fall besser geeignet ist, ist wieder von Aufgabenstellung abhängig. Wichtig zu merken ist für die Schleifenkonstruktion der prinzipielle Unterschied zwischen beiden Schleifen: Bei der `WHILE`-Schleife wird zuerst eine Bedingung geprüft. Ist sie erfüllt, führt das System die erste Anweisung in der Schleife aus, sonst nicht. Bei der `DO`-Schleife wird die erste Anweisung in der Schleife ausgeführt, ohne dass vorher eine Bedingung geprüft wurde.

10.4.4 Abbrucharweisungen für Schleifen

Zum Abbruch von Schleifen kennt ABAP mehrere verschiedene Abbrucharweisungen.

CONTINUE Die erste dieser Anweisungen, `CONTINUE`, ist ausschließlich zum Abbruch eines Anweisungsblocks innerhalb einer Schleife gedacht. Steht die `CONTINUE`-Anweisung mitten in einem Anweisungsblock, wird der Rest der Anweisungen dieses Blocks nicht mehr ausgeführt und ein neuer Schleifendurchlauf gestartet, falls die Bedingungen für einen neuen Durchlauf noch gelten. Da das Kommando selbst keine Bedingung prüft, packt man es oft in eine Verzweigung, über deren Bedingung der Programmablauf gesteuert wird. Falls die Bedingung erfüllt ist, wird das `CONTINUE`-Kommando ausgeführt, der Anweisungsblock beendet, ein neuer Durchlauf geprüft und eventuell gestartet. Die Struktur der `CONTINUE`-Anweisung ist relativ einfach:

```
IF Bedingung_1
  CONTINUE
ENDIF
```

Um in der Teilnehmertabelle die Sätze der Kursteilnehmer zu verarbeiten, die nach dem 01.01.1970 geboren sind, schreiben Sie beispielsweise:

```

SELECT * FROM zteilnehmer02.
  IF zteilnehmer02-tgeburtsdatum LT '19700101'.
    CONTINUE.
  ENDIF.
* Anweisungsblock_1
ENDSELECT.

```

Die Bedingung nach der IF-Anweisung ist ein logischer Ausdruck. LT ist dabei der Operand für »kleiner als«, mit dem die beiden Werte verglichen werden. Liest das System einen Satz mit einem Geburtsdatum vor dem 01.01.1970, wird der – hier nicht weiter ausgeführte – Anweisungsblock_1 nicht ausgeführt, sondern sofort der nächste Satz gelesen. Nur die Sätze, bei denen das Geburtsdatum am oder nach dem 01.01.1970 liegt, gelangen in die Verarbeitung des Anweisungsblocks.

Die zweite Abbruchanweisung CHECK ist mächtiger: Falls die Anweisung in einer Schleife ausgeführt wird, wird die Verarbeitung des Anweisungsblocks beendet und der nächste Schleifendurchlauf gestartet, sofern die Bedingungen dafür noch gelten. In diesem Sinne könnte man die Anweisung CHECK als Kombination aus den Anweisungen IF und CONTINUE bezeichnen. Sie schreiben einfach:

CHECK Bedingung

Ist die Bedingung erfüllt, werden alle Anweisungen des Anweisungsblocks nach dem CHECK-Kommando ausgeführt. Ist die Bedingung dagegen nicht erfüllt, werden alle nachstehenden Kommandos des Anweisungsblocks übersprungen und nach Prüfung der Bedingungen wird ein neuer Schleifendurchlauf gestartet.

Interessiert Sie beispielsweise eine Verteilung der Kursteilnehmer nach Geschlecht, könnten Sie die Sätze entsprechend dem Geschlechtsschlüssel selektieren:

```

SELECT * FROM zteilnehmer02.
  CHECK zteilnehmer02-tgeschlecht = 'W'.
* Anweisungsblock
ENDSELECT.

```

In der Schleife wird geprüft, ob die Bedingung des Geschlechtsschlüssels erfüllt ist. Falls ja, wird der Anweisungsblock ausgeführt; falls nein, wird der nächste Satz gelesen und geprüft.

Wundern Sie sich nicht, falls Sie später einmal die Anweisung `CHECK` außerhalb einer Schleife im Quellcode finden. `CHECK` kann nicht nur Schleifendurchläufe beenden, sondern auch andere Verarbeitungsblöcke wie Unterprogramme oder Ereignisse. Die `CHECK`-Anweisung wirkt immer auf die Struktur und auf diejenige Ebene, in der sie ausgeführt wird. Im Okay-Fall werden die nachstehenden Anweisungen ausgeführt, sonst wird der jeweilige Verarbeitungsblock beendet.

EXIT Die dritte Anweisung `EXIT` ist ebenfalls mächtig: Sie beendet nicht nur den Anweisungsblock, sondern die Schleifenverarbeitung an sich. Das System verzweigt zum Ende der Struktur und setzt die Programmausführung mit der ersten Anweisung nach dem Strukturende fort. Dabei bezieht sich das Kommando immer auf die Ebene, in der es ausgeführt wird: Es beendet die Verarbeitung auf dieser Ebene und bewirkt die Programmfortsetzung auf der nächsthöheren, äußeren Ebene.

Steht die ausgeführte `EXIT`-Anweisung bei zwei geschachtelten Schleifen beispielsweise in der inneren Schleife, wird diese beendet und das Programm mit der Anweisung nach dem Strukturende der inneren Schleife fortgesetzt, also eine Ebene höher in der äußeren Schleife. Steht die `EXIT`-Anweisung hingegen in der äußeren Schleife und wird dort ausgeführt, wird die äußere Schleife beendet und mit der ersten Anweisung nach ihrem Strukturende fortgesetzt.

Absichtlicher Zeitabbruch Zur Verdeutlichung soll wieder das Zeitmesserbeispiel dienen: Dieses Mal lassen Sie ihn aber nicht durchlaufen, sondern Sie brechen die Schleifen mit der `EXIT`-Anweisung ab, um die Wirkung zu beobachten. In der innersten der drei Schleifen steht ein Anweisungsblock. Hat `SY-INDEX` für die innerste Schleife den Wert 3, wird die innerste Schleife beendet und zur mittleren Schleife verzweigt. Der Anweisungsblock wurde aber zuvor noch ausgeführt. Hat in der mittleren Schleife `SY-INDEX` den Wert 3, werden die nachfolgenden Anweisungen der mittleren Schleife nicht mehr ausgeführt, die dritte Schleife und ihr Anweisungsblock werden also übersprungen. Das System geht zur äußeren Schleife und startet den nächsten Schleifendurchlauf auf dieser Ebene. Hat auch der `SY-INDEX` der äußeren Schleife den Wert 3, wird auch diese Schleife beendet.

```
DO 24 TIMES.  
  IF sy-index = 3.  
    EXIT.  
  ENDIF.  
  min = 0.  
DO 60 TIMES.
```

```

IF sy-index = 3.
    EXIT.
ENDIF.
sek = 0.
DO 60 TIMES.
*   Anweisungsblock
    sek = sek + 1.
    IF sy-index = 3.
        EXIT.
    ENDIF.
ENDDO.
min = min + 1.
ENDDO.
std = std + 1.
ENDDO.

```

Wie die Anweisung CHECK ist die Wirkung der Anweisung EXIT abhängig von der Struktur, also der Ebene, in der sie ausgeführt wird. Wird die Anweisung in einer Schleife ausgeführt, wird die Schleife abgebrochen. Wird die Anweisung in einem Unterprogramm ausgeführt, wird das Unterprogramm abgebrochen und in der übergeordneten Struktur weitergearbeitet. Dasselbe gilt für Ereignisse: Ein ausgeführtes EXIT-Kommando in einem Ereignis beendet das Ereignis und das System setzt mit der Verarbeitung des nächsten Ereignisses fort. Sind keine weiteren Ereignisse deklariert und Anweisungen dafür vorgesehen, wird das Programm beendet. Im Zusammenhang mit dem Beenden von Ereignissen gibt es noch weitere Möglichkeiten, auf die hier aber nicht weiter eingegangen werden soll.

Abbruchbedingungen sind kontextsensitiv

Die beschriebenen Abbruchbedingungen funktionieren bei allen Schleifenarten gleich. Aus diesem Grund wurde nur jeweils ein Beispiel für eine Abbruchbedingung beschrieben.

10.5 Logische Ausdrücke

Auch im täglichen Leben müssen wir uns permanent entscheiden. Allerdings laufen menschliche Entscheidungen nicht immer ganz logisch ab. Solange Computer aber nicht in der Lage sind, zu verstehen, was wir meinen, müssen in einem Programmablauf eindeutig messbare und nachvollziehbare Bedingungen formuliert werden.

Dies ... oder das?

Zugegebenermaßen ist es schwierig, die Komplexität von Prozessen des Alltags- und des Geschäftslebens auf eine logische, programmtechnische

Ebene umzusetzen. Sie müssen auf die Inhalte und die Vergleichbarkeit von Datentypen und Feldinhalten achten sowie die Reaktionen des Systems antizipieren.

10.5.1 Einfache logische Ausdrücke

Logische Operatoren

Betrachten Sie zunächst einmal die so genannten *logischen Operatoren* in Tabelle 10.1: Mit ihnen können Sie Bedingungen formulieren, die zwei Felder vergleichen, zum Beispiel:

```
IF ZAHL01 > ZAHL02.  
    Anweisungsblock  
ENDIF
```

Sind beide Felder hinsichtlich ihrer Attribute kompatibel oder konvertierbar, ist die Bedingung dann erfüllt, wenn der Inhalt von ZAHL01 größer ist als der Inhalt von Feld ZAHL02, und der Anweisungsblock wird ausgeführt. Im Quellcode können für die logischen Ausdrücke wahlweise logische Operatoren oder ihre Symbole verwendet werden.

Logischer Operator	Symbol	Bedeutung
EQ	=	gleich
NE	<>	ungleich
LT	<	kleiner als
LE	<=	kleiner gleich
GT	>	größer als
GE	>=	größer gleich

Tabelle 10.1 Logische Operatoren für Datenobjekte

Automatische Typkonvertierung

Je weiter die Attribute der zu vergleichenden Felder jedoch auseinander driften, desto mehr muss man die systemimmanenten Konvertierungsregeln kennen. Zeichenketten vom Typ `c` werden beispielsweise bei unterschiedlicher Länge von links nach rechts gemäß der relevanten Codetabelle verglichen. Klingt ganz einfach, aber wissen Sie immer genau, wo ein Blank oder ein Sonderzeichen in der Codetabelle steht – vor oder nach den Kleinbuchstaben –, und ist die lexikalische Sortierung zur Codetabelle identisch? Man kann sich im Detail hier ganz schön ärgern und täuschen.

Sind die Datentypen unterschiedlich, wird zuerst eine automatische Typkonvertierung durchgeführt und dann verglichen. Wenn beispielsweise ein Feld vom Typ `c` mit einem Feld vom Typ `n` verglichen werden soll, werden vor dem Vergleich beide Felder in den Typ `p` konvertiert. Auch bei dieser Sorte Aufgabenstellung kann einem das System Überraschungen bereiten: Formal arbeitet es natürlich einwandfrei, es ist der Mensch, der sich nicht an die Regeln hält – oder sie nicht kennt – und staunt. Als Konsequenz daraus sollten Sie versuchen, die zu vergleichenden Felder hinsichtlich ihrer Attribute aufeinander abzustimmen.

Für Zeichenketten gibt es spezielle logische Operatoren. Mit ihnen formuliert man Bedingungen, um herauszufinden, ob eine Zeichenkette bestimmte Zeichen enthält usw. Für andere spezielle Aufgaben existieren noch einige weitere Operatoren. Für den Einstieg in ABAP soll hier aber nicht weiter darauf eingegangen werden, sondern nur noch die Prüfung auf den Initialwert eines Feldes und die Prüfung auf ein Intervall erwähnt werden.

Spezielle logische Operatoren

Die Bedingung, dass eine Teilaufgabe nur dann ausgeführt werden soll, wenn der Feldwert der Initialwert ist, kommt relativ häufig vor. Dabei hängt es nur von der Aufgabenstellung ab, ob der Initialwert von Anfang an unverändert bleibt oder – wie in unseren vorherigen Beispielen – während der Programmausführung per Anweisung gesetzt wird. In beiden Fällen soll der Anweisungsblock nur unter dieser Bedingung ausgeführt werden.

IS INITIAL

```
IF zahl01 IS INITIAL.  
    Anweisungsblock  
ENDIF.
```

Der Anweisungsblock wird nur durchlaufen, wenn der Feldinhalt von ZAHLO1 der Initialwert ist. Hat das Feld schon einen Inhalt, ist die Bedingung nicht erfüllt und der Anweisungsblock wird dann nicht ausgeführt.

Bei der Prüfung auf ein Intervall wird der Feldinhalt hinsichtlich einer Untergrenze und einer Obergrenze geprüft. Liegt er dazwischen oder genau auf einer von beiden Grenzen, ist die Bedingung gültig und der Anweisungsblock wird ausgeführt. Die Ober- und Untergrenze des Intervalls können entweder wieder als Literal oder als Variable angegeben werden; werden die Grenzen als Variable angegeben, müssen die Grenzwerte, also die Feldinhalte, natürlich vorher sinnvoll gesetzt sein.

BETWEEN

Soll in unserem Beispiel für die Teilnehmer des Jahrganges 1969 eine besondere Verarbeitung starten, müssen Sie prüfen, ob das Geburts-

datum zwischen dem 01.01.1969 und dem 31.12.1969 liegt. Ist die Bedingung erfüllt, soll der Anweisungsblock ausgeführt werden.

```
IF zteilnehmer02-tgeburtsdatum
    BETWEEN '19690101' AND '19691231'.
*   Anweisungsblock
ENDIF.
```

Hat jemand genau am 01.01.1969 oder am 31.12.1969 Geburtstag, ist für diese Person ebenfalls die Bedingung erfüllt und der Satz durchläuft den Anweisungsblock.

10.5.2 Verknüpfte logische Ausdrücke

Richtig spannend wird die Arbeit, wenn man komplexe Bedingungen durch Verknüpfen von logischen Ausdrücken abbilden will oder muss. Es ist eine schwierige Gratwanderung, einerseits die Komplexität abzubilden, und andererseits die Transparenz, Lesbarkeit und Änderbarkeit des Quellcodes zu erhalten. Dazu sollten Sie zunächst die Regeln für das Verknüpfen von logischen Ausdrücken beachten.

- OR** Arbeiten Sie mit zwei Bedingungen und dem logischen Oder als Verknüpfungsoperator, muss eine von beiden Bedingungen oder müssen beide erfüllt sein, damit der Anweisungsblock durchlaufen wird. Stellen Sie sich vor, Sie möchten ein Buch ausleihen und lesen. Sie kennen zwei Personen und fragen beide. Sie können das Buch dann ausleihen und lesen, wenn Ihnen eine von beiden Personen das Buch leiht – oder beide.

In ABAP wird das logische Oder durch den Operand **OR** abgebildet. Die allgemeine Grundstruktur ist dieselbe wie beim einfachen **IF**: Es gibt die Verzweigungsstruktur **IF . . . ENDIF** sowie eine Bedingung **b1**.

```
IF b1.
*   Anweisungsblock
ENDIF.
```

Der Unterschied ist jetzt nur, dass sich **b1** wiederum aus Bedingungen, wie **b2**, **b3** usw. zusammensetzt, die mit einem logischen Operanden verknüpft sind, beispielsweise:

```
b1 = b2 OR b3
```

- Klammerebenen** Dieses Auflösen von Bedingungen kann beliebig fortgesetzt werden, kombiniert mit anderen logischen Operanden sowie mehreren Klammerebenen. Sie können auf diese Weise wirklich komplexe Bedingungen

abbilden. Aber auch hier wird den meisten Menschen schneller schwindelig als einer Maschine: Wenn solche Bedingungen ganze Bildschirmseiten füllen, können Sie leicht den Überblick verlieren. Kleinste Änderungen an logischen Ausdrücken können riesige Auswirkungen auf den Programmablauf haben. Bitte vergessen Sie also die Grundsätze der Lesbarkeit, Transparenz und Änderbarkeit von Quellcode nicht.

Sollen die Sätze aller Teilnehmer der Kurse »SAP-Finanzwesen« und »SAP-Grundlagen« durch einen eigenen Anweisungsblock laufen, schreiben Sie die logische Oder-Bedingung so:

```
IF zteilnehmer02-zzkurstitel = 'SAP-Finanzwesen' OR
   zteilnehmer02-zzkurstitel = 'SAP-Grundlagen'.
*   Anweisungsblock
ENDIF.
```

Das Feld für den Kurstitel muss entweder den Inhalt »SAP-Finanzwesen« oder den Inhalt »SAP-Grundlagen« haben, damit die Bedingung erfüllt ist. So wie die Bedingung formuliert ist, kann das Feld natürlich nicht gleichzeitig beide Bedingungen erfüllen. Dazu bräuchte man zwei verschiedene Felder.

Strenger als das logische Oder bindet das logische Und. Bei zwei Bedingungen bedeutet das, dass beide Bedingungen gleichzeitig erfüllt sein müssen, damit der Anweisungsblock durchlaufen wird. Wenn Sie in den Urlaub fahren möchten, brauchen Sie Zeit und Geld – nur eines von beidem zu haben, genügt nicht. **AND**

In ABAP wird das logische Und durch den Operanden AND symbolisiert. Um wiederum ein einfaches Beispiel zu wählen, nehmen wir an, es sollen die Sätze aller männlichen Teilnehmer, die den Kurs »PC-Grundlagen« besuchen, durch einen eigenen Anweisungsblock laufen. Der ABAP-Quellcode lautet sinngemäß:

```
IF zteilnehmer02-tgeschlecht = 'M' AND
   zteilnehmer02-zzkurstitel = 'PC-Grundlagen'.
*   Anweisungsblock
ENDIF.
```

Es gibt zwei Felder für Geschlecht und Kurstitel, und beide müssen einen bestimmten Inhalt haben, damit die Bedingung erfüllt ist und der Anweisungsblock durchlaufen wird.

Am strengsten bindet der dritte Verknüpfungsoperator NOT: die Negation. Wenn Sie Abfragen wie »Möchten Sie diese Änderung nicht spei- **NOT**

chern? – Ja!/Nein!« kennen, haben Sie das Problem der Negation bereits erkannt. Für viele Menschen ist es ziemlich schwer, mit negierten Abfragen und Bedingungen zu arbeiten, besonders in komplexen Situationen und unter Druck. Deshalb bereits an dieser Stelle die Bitte, dass Sie Negationen in Ihrer praktischen Arbeit so weit wie möglich vermeiden – formulieren Sie positiv, wo immer es geht.

Nicht immer
vermeidbar

Manchmal lässt sich aber das Arbeiten mit Negationen nicht vermeiden, beispielsweise im Umfeld von Abfragen auf Initialwerte von Feldern. Abzufragen, ob ein Feld den Initialwert hat, ist nicht schwer, wie Sie gesehen haben. Alle möglichen Feldinhalte dagegen abzuprüfen, ist fast unmöglich. Manchmal genügt es auch zu wissen, dass ein Feld einen anderen als den Initialwert, also eben nicht den Initialwert hat, um einen Anweisungsblock auszuführen.

Formal zu beachten ist, dass die Negation NOT grundsätzlich vor der Bedingung stehen muss. Soll ein Anweisungsblock nur dann ausgeführt werden, wenn die Bedingung b1 nicht erfüllt ist, wäre die Schreibweise sinngemäß:

```
IF NOT b1.
```

Im folgenden Beispiel sollen die Sätze aller Teilnehmer, die nicht an SAP-Kursen teilnehmen, gesondert verarbeitet werden. Sie schreiben in ABAP:

```
IF NOT zteilnehmer02-zzkurstitel(3) = 'SAP'.  
* Anweisungsblock  
ENDIF.
```

Die Bedingung lautet, dass die ersten drei Zeichen der Kursbezeichnung nicht »SAP« sein dürfen, damit der Anweisungsblock durchlaufen wird.

Zum Abschluss dieses Abschnitts kombinieren wir unser neues Wissen über Verknüpfungen logischer Ausdrücke in einem kleinen Beispiel.

366 Tage – aber
wann?

Beispiel

Ermitteln Sie, ob ein Jahr ein Schaltjahr ist.

In unserem Kalender ist ein Jahr immer dann ein Schaltjahr, wenn es durch vier ganzzahlig ohne Rest teilbar ist. Immer dann, wenn es durch 100 ganzzahlig ohne Rest teilbar ist, ist es aber kein Schaltjahr. Es ist aber trotzdem ein Schaltjahr, wenn es durch 400 ganzzahlig ohne Rest teilbar ist.

Für die Umsetzung in ABAP müssen zunächst drei Bedingungen formuliert werden:

- ▶ b1: das Jahr ist durch 400 ganzzahlig teilbar
- ▶ b2: das Jahr ist durch 100 ganzzahlig teilbar
- ▶ b3: das Jahr ist durch 4 ganzzahlig teilbar

Ein Jahr ist dann ein Schaltjahr, wenn b1 oder wenn b3 erfüllt ist, dann darf aber b2 nicht gelten. In kürzerer und eindeutiger Schreibweise kann die Bedingung so formuliert werden:

```
b1 OR b3 AND NOT b2
```

Geklammert werden muss nicht, weil durch die hierarchische Ordnung der Operanden die Negation am engsten bindet, das logische Und als zweitstärkstes und das logische Oder am schwächsten. Wenn Sie die Lesbarkeit erhöhen möchten, können Sie natürlich auch Klammern setzen. Die Bedingung würde dann lauten:

```
b1 OR ( b3 AND NOT b2 )
```

Bitte beachten Sie aber, dass anders gesetzte Klammern die Logik der Bedingung völlig verändern können.

Für die endgültige Umsetzung könnten Sie in ABAP die Ergebnisse der Prüfungen, ob ein Jahr ganzzahlig teilbar ist, in entsprechenden Feldern hinterlegen und dann die Bedingung beispielsweise so schreiben:

```
IF jahr400 = 0 OR jahr004 = 0 AND NOT jahr100 = 0.  
* Anweisungsblock  
ENDIF.
```

Im Codebeispiel zu WHILE und logischen Ausdrücken (siehe Abschnitt 10.9) finden Sie nähere Erläuterungen.

10.6 Codebeispiel zu IF

```
1 * &-----*  
2 * & Report Z_TEILNEHMERLISTE10_if *  
3 * & *  
4 * &-----*  
5 * & *  
6 * & *  
7 * &-----*  
8
```

```

9  REPORT  z_teilnehmerliste10_if.
10
11 * Tabellen deklarieren
12 TABLES zteilnehmer02.
13
14 * Zeichenketten deklarieren
15 DATA: tz_pc_grundlagen TYPE i,
16         tz_netzwerktechnik TYPE i,
17         tz_sap_grundlagen TYPE i,
18         tz_sonst TYPE i,
19         tz_total TYPE i.
20
21 * vollständigen Tabelleninhalt ausgeben
22 SELECT * FROM zteilnehmer02.
23     WRITE: / zteilnehmer02-tnummer,
24             zteilnehmer02-tname,
25             zteilnehmer02-tgeburtsdatum,
26             zteilnehmer02-tgeschlecht,
27             zteilnehmer02-tkurspreis,
28             zteilnehmer02-twaehrung,
29             zteilnehmer02-zzkfztyp,
30             zteilnehmer02-zzkurstitel.
31 ENDSELECT.
32 SKIP.
33
34 * Tabelleninhalt selektieren mit Bedingung
35 SELECT * FROM zteilnehmer02
36         WHERE zzkurstitel = 'Netzwerktechnik'.
37     WRITE: / zteilnehmer02-tnummer,
38             zteilnehmer02-tname,
39             zteilnehmer02-tgeburtsdatum,
40             zteilnehmer02-tgeschlecht,
41             zteilnehmer02-zzkurstitel.
42 ENDSELECT.
43
44 * alle Sätze aus der Tabelle lesen,
   aber nur bestimmte Sätze verarbeiten
45 SELECT * FROM zteilnehmer02.
46     IF zteilnehmer02-zzkurstitel =
       'PC-Grundlagen'.

```

```

47     tz_pc_grundlagen = tz_pc_grundlagen + 1.
48     ENDIF.
49 ENDSELECT.
50 SKIP.
51 WRITE: / tz_pc_grundlagen, 'Personen besuchen
        den Kurs PC-Grundlagen'.
52
53 * alle Sätze der Tabelle lesen,
        Anzahl der Kurszeilnehmer grob aufteilen
54 SELECT * FROM zteilnehmer02.
55     IF zteilnehmer02-zzkurstitel =
        'SAP-Grundlagen'.
56         tz_sap_grundlagen = tz_sap_grundlagen + 1.
57     ELSE.
58         tz_sonst = tz_sonst + 1.
59     ENDIF.
60 ENDSELECT.
61 tz_total = tz_sap_grundlagen + tz_sonst.
62 SKIP.
63 WRITE: / 'von den',
64         / tz_total, 'Teilnehmern besuchten',
65         / tz_sap_grundlagen, 'den Kurs
        SAP-Grundlagen, und',
66         / tz_sonst, 'sonstige Kurse'.
67
68 * wieder benötigte Zähler initialisieren
69 CLEAR: tz_netzwerktechnik,
70         tz_pc_grundlagen,
71         tz_sap_grundlagen,
72         tz_sonst.
73
74 * alle Sätze der Tabelle lesen,
        Anzahl der Kurszeilnehmer fein aufteilen
75 SELECT * FROM zteilnehmer02.
76     IF zteilnehmer02-zzkurstitel =
        'Netzwerktechnik'.
77         tz_netzwerktechnik = tz_netzwerktechnik + 1.
78     ELSEIF zteilnehmer02-zzkurstitel =
        'PC-Grundlagen'.
79         tz_pc_grundlagen = tz_pc_grundlagen + 1.

```

```

80     ELSEIF zteilnehmer02-zzkurstitel =
           'SAP-Grundlagen'.
81         tz_sap_grundlagen = tz_sap_grundlagen + 1.
82     ELSE.
83         tz_sonst = tz_sonst + 1.
84     ENDIF.
85 ENDSELECT.
86
87 SKIP.
88 WRITE: / 'Verteilung der Personen auf die
           Kurse:',
89         / tz_netzwerktechnik, 'Netzwerktechnik',
90         / tz_pc_grundlagen,   'PC-Grundlagen',
91         / tz_sap_grundlagen,  'SAP-Grundlagen',
92         / tz_sonst,           'sonstige Kurse'.

```

Listing 10.1 Report Z_TEILNEHMERLISTE10_if

10.6.1 Anmerkungen zum Quellcode

Zeile 35 bis 42

Nur die Sätze, die die Bedingung erfüllen, dass im Feld für die Kursbezeichnung »Netzwerktechnik« steht, gelangen in die Verarbeitung. Die Verarbeitung wird durch Ausgabe in einer Liste simuliert.

Vorteil: Vorselektion

Diese Vorselektion ist gleichzeitig der große Vorteil dieser Arbeitsweise, sie ist für bestimmte Fälle effizienter. Man kann sich gut vorstellen, dass es umso sinnvoller ist, sich gezielt die »richtigen« Sätze aus einer Tabelle heraus zu picken, je größer die Tabelle ist. Der Nachteil dieser Selektion mit Bedingung könnte sein, dass die anderen Sätze, die vielleicht auch verarbeitet werden sollen, aber eben anders, in diesem Beispiel gar nicht gelesen werden und für eine Verarbeitung nicht zur Verfügung stehen. Es kommt also wie immer auf die Aufgabenstellung an.

Zeile 45 bis 49

Es werden alle Sätze der Tabelle gelesen. Zwischen Zeile 45 und 46 könnte man sich einen Verarbeitungsblock vorstellen, der für alle Sätze gleich ist. Zusätzlich findet für die Sätze, die die Bedingung »PC-Grundlagen« erfüllen, eine gesonderte Verarbeitung statt: Sie durchlaufen einen eigenen Anweisungsblock und werden gezählt.

Zeile 51

Hier sehen Sie ein wichtiges Prinzip, vielleicht auch erst auf den zweiten Blick. In Zeile 45 bis 49 läuft eine Schleife. In der Schleife wird in Zeile 46 bis 48 die Zahl der Teilnehmer für den Kurs »PC-Grundlagen« berechnet. Die Ausgabe des Ergebnisses, also die Summe der Anzahl der Teilnehmer, darf erst nach der Schleife erfolgen, die Ausgabe in der Schleife wäre nur ein Zwischenergebnis. Das könnte zwar bei anderen Aufgabenstellungen sinnvoll sein, hier aber nicht. Also denken Sie bitte immer daran:

Wichtiges Prinzip

- ▶ Summen *ermitteln*: immer *in* der Schleife
- ▶ Summen *ausgeben*: immer *nach* der Schleife

Zeile 54 bis 60

Wieder werden alle Sätze der Tabelle gelesen, dabei werden zwei Summen ermittelt. Wenn das Feld ZTEILNEHMERO2-ZZKURSTITEL den Wert »SAP-Grundlagen« enthält, wird die Summe für diese Kursteilnehmer erhöht. Sonst steigt die Summe für die Teilnehmer an den restlichen Kursen.

Zeile 61 bis 66

Aus beiden Gruppensummen wird die Gesamtsumme gebildet. In der Liste erscheinen die Gesamtanzahl von Kursteilnehmern, die Anzahl der Teilnehmer am Kurs »SAP-Grundlagen« und die Anzahl der Teilnehmer, die sonstige Kurse besuchen.

Zeile 69 bis 72

Die Summenfelder werden für die nachfolgenden Beispiele wieder benötigt und deshalb initialisiert.

Zeile 76 bis 85

Alle Sätze aus der Datenbanktabelle werden selektiert. Für jeden Satz werden nacheinander die Bedingungen geprüft. Erfüllt er die erste Bedingung, wird der erste Anweisungsblock ausgeführt und zu `ENDIF` verzweigt. Erfüllt er die erste Bedingung nicht, wird geprüft, ob er die zweite Bedingung erfüllt usw. Es werden mehrere Fälle unterschieden und Gruppensummen für die Kurse »Netzwerktechnik«, »PC-Grundlagen« und »SAP-Grundlagen« gebildet. Die restlichen Kursteilnehmer fallen wieder unter eine Summe für die sonstigen Kurse.

Zeile 88 bis 92

Nach der SELECT-Schleife werden die Gruppensummen mit entsprechenden Texten in die Liste geschrieben.

10.6.2 Ausgabe des Quellcodes

Lesbarkeit durch
SKIP erhöhen

Zur Kontrolle werden zunächst alle Sätze der Tabelle ausgelesen, hier insgesamt acht Sätze (siehe Abbildung 10.6). Dann erfolgt die Ausgabe der Treffersätze über die SELECT-Anweisung mit Bedingung, daran schließen sich die Ausgaben der Gruppensummen aus den einzelnen Beispielen an. Zur besseren Lesbarkeit wurde mit dem SKIP-Kommando jeweils eine Leerzeile eingeschoben, auf weitere optische Aufbereitungsoptionen wurde bewusst verzichtet.

Teilnehmerliste						
07.05.2005			Teilnehmerliste			
00001	Schmidt	08.08.1998	M	999,00	EUR	KOMBI Netzwerktechnik
00002	Schmidt	09.09.1999	W	999,00	EUR	KOMBI Netzwerktechnik
00003	Schmidt	08.08.1998	M	999,00	EUR	KOMBI PC-Grundlagen
00004	Schmidt	09.09.1999	W	999,00	EUR	KOMBI PC-Grundlagen
00005	Schmidt	08.08.1998	M	999,00	EUR	KOMBI SAP-Grundlagen
00006	Schmidt	09.09.1999	W	999,00	EUR	KOMBI SAP-Grundlagen
00007	Schmidt	08.08.1998	M	999,00	EUR	KOMBI SAP-Finanzwesen
00008	Schmidt	09.09.1999	W	999,00	EUR	KOMBI SAP-Finanzwesen
00001	Schmidt	08.08.1998	M	Netzwerktechnik		
00002	Schmidt	09.09.1999	W	Netzwerktechnik		
2 Personen besuchen den Kurs PC-Grundlagen						
von den						
8 Teilnehmern besuchten						
2 den Kurs SAP-Grundlagen, und						
6 sonstige Kurse						
Verteilung der Personen auf die Kurse:						
2 Netzwerktechnik						
2 PC-Grundlagen						
2 SAP-Grundlagen						
2 sonstige Kurse						

Abbildung 10.6 Listbild zum IF-Beispiel

10.7 Codebeispiel zu CASE

```
1 * & ----- *
2 * & Report Z_TEILNEHMERLISTE10_case *
3 * & *
4 * & ----- *
5 * & *
6 * & *
7 * & ----- *
8
```

Index

\$TMP 37, 133

& 296

* 65

. 66

/ 70

_ 76

A

ABAP 15

ABAP Debugger 115, 118, 127, 354, 367

Aufruf 115

Ausführungsebene 119

beenden 128

Systemfelder 121

Tabellenbereich 122

ABAP Dictionary 31, 33, 35, 132, 154,

166, 178, 194, 198

ABAP Editor 31, 55, 58, 72

dynamische Breakpoints 128

Icons 60

Steuerung 60

Symbolleiste 60

ABAP Memory 403, 404

ABAP Objects 23, 326, 383

ABAP Workbench 29, 68, 373

ABAP-Anweisungen 66

ABAP-Befehle 66

ABAP-Kommandos 66

ABAP-Laufzeitumgebung 23

ABAP-Programme

modularer Aufbau 24

Struktur 23

ABAP-Prozessor 22

ABAP-Report

anlegen 55

ausführen 67

ABAP-Schlüsselwortdokumentation

66

Abbrucharweisungen 235

Wirkung 236

Abbruchbedingungen 237, 257

Abbruchkriterium 231

Ablaufsteuerung 426

ADD 89

Addition 88

ADJACENT DUPLICATES 343

Advanced Business Application

Programming -> s. ABAP

Aktivierungsreihenfolge 136

Allgemeiner Berichts-Aufbereitungs-

Prozessor -> s. ABAP

alphanumerische Zeichen 99

AND 241

AND RETURN 387

Anfangsposition 107

Anmeldebildschirm 26

Anmeldemandant 51

Anmeldung 26

Antwortzeit 321

Anweisungen 66

Anweisungsblock 221, 226, 228, 297

beenden 234

anwendungsübergreifende Kompo-

nenten 28

APPEND 327, 333, 366, 368

Append-Struktur 131, 148

aktivieren 150

anlegen 148

Komponenten 149, 150

Name 149

pflegen 148

zuordnen 148

APPLo 49

Applikationsschicht 16

Applikationsserver 17

Arbeitsbereich 324, 328, 335

Arbeitsspeicherbereiche 387, 402

Arbeitsteilung 194

arithmetischen Operationen 174

Array-Fetch 330, 352

Artikelnummern 100

AS CHECKBOX 273

AS TEXT 341

AT 106

AT SELECTION-SCREEN 267, 268, 295

AT SELECTION-SCREEN ON 270

ausführbares Programm 56

Auslieferungsklasse 35

Ausnahmebehandlung 402, 425

Ausschneiden 156

Auswahlknöpfe -> s. Radiobuttons

AUTHORITY-CHECK 195

- automatische Mandantenverwaltung
 - 51
- automatische Typkonvertierung 239, 329
- B**
- Backend-Modus 58, 59
 - Vorteile 59
- Batch-Job 56
- Bedingungen 233
 - auflösen 240
 - komplexe 240
- Befehle 66
 - rückgängig machen 64
- BEGIN OF 322
- BEGIN OF BLOCK 298
- BEGIN OF LINE 296
- Benutzerstammsatz 194
- Berechnung 79
 - Alter bestimmen 172
 - Dimension von Zeitspannen 173
 - Fälligkeitsdatum bestimmen 168
 - Fristen ermitteln 167
 - Mehrwertsteuer 179
 - mit Datumsfeldern 167
 - mit Mengenfeldern 178
 - mit Währungsfeldern 178
 - mit Zeitfeldern 174
 - Monatsanfang ermitteln 168, 169
 - Rechnungsbetrag ermitteln 179
 - Rechnungsdatum bestimmen 168
 - Stichtag ermitteln 168
 - Stichtage ermitteln 167
 - Ultimo ermitteln 170
 - verbleibende Tage 172
 - verbleibende Tage ermitteln 171
 - vergangene Tage ermitteln 171
 - Zeit bis Mitternacht 175
 - Zeitspanne in Jahren 173
 - Zeitspanne in Minuten 174
 - Zeitspanne in Sekunden 174
 - Zeitspanne in Stunden 174
 - Zeitspanne in Tagen 173
 - Zeitspanne mit Datumswechsel 176
- Berechtigung
 - Programmausführung 195
 - Transaktionen 195
- Berechtigungsgruppe 57
- Berechtigungskonzept 194
- Berechtigungsobjekt 194
- Berechtigungsprüfung im Quellcode 195
- BETWEEN 239
- Bewegungsdaten 36
- Bindestrich 69
- Block
 - verdoppeln 64
 - verschieben 64
- Block/Ablage 81
- Blockname 299
- blockweises Übertragen 330
- BREAK-POINT 117, 129
- Breakpoints 61, 62, 115, 124
 - aktive 127
 - dynamische 117, 128
 - entfernen 128
 - löschen 127
 - setzen 117, 126
 - sichern 127
 - statische 129
 - verwalten 128
- Breakpoints-Modus 126
- Buchungskreis 27
- Büro 28
- BY 340
- C**
- c, Datentyp 99, 102, 107, 167, 178, 200, 201, 238
- CALL FUNCTION 396, 402
- CASE 228, 230, 251
- CASE-Struktur 228, 402
- CHAR, Datentyp 136, 138
- CHECK 235, 379
- Checkbox 386
- CLEAR 200, 201, 202, 344, 345, 354, 358, 367, 369, 382
- Client/Server-Architektur 16, 193
- Cluster 404
- Cluster-Name 403
- Clustertabellen 34
- Codetabelle 238
- COMMENT 297
- COMPUTE 91
- CONCATENATE 105, 113
 - SEPARATED BY 105

CONDENSE 104, 113
 NO-GAPS 104
 CONSTANTS 87
 CONTINUE 234
 CUKY, Datentyp 139, 140, 178
 CURK, Datentyp 167
 CURR
 Datentyp 167
 CURR, Datentyp 139, 140, 167, 178
 Customizing 18

D

d, Datentyp 165, 200
 DATA 84, 86, 99, 165, 186, 200, 201,
 322, 354, 365, 378
 BEGIN OF 322
 DECIMALS 84
 END OF 322
 LIKE 85, 100, 165
 OCCURS 323
 TYPE 84
 VALUE 86, 100, 166
 Data Browser 36, 135, 155
 Data Dictionary 33
 Daten, temporäre 276
 Datenart 49
 Datenbank, anpassen 131, 158
 Datenbankschicht 16
 Datenbankschnittstelle 22
 Datenbankserver 17
 Datenbanksperre 195
 Datenbanktabellen 402
 ausgeben 69
 lesen 69
 Modifikation 131
 transparente 131
 Datenbank-Utility 158, 160
 Datenblock 329
 Datendeklarationen 375
 Datenelement 38
 aktivieren 48
 anlegen 39
 prüfen 47
 Datenkonsistenz 387
 Datenobjekte 83, 323, 332, 378
 kompatible 89
 konvertible 89

Datensatz 318
 Datensätze
 anzeigen 53
 eingeben 51
 erfassen 51
 löschen 203
 prüfen 134
 Datenschnittstelle 391, 392
 Datentyp 40, 43, 84, 90, 323, 378
 elementare 84, 323
 generischer 99, 100
 komplexe 324
 NUMC 43
 Unterschiede 102
 zeichenartige 165
 Datentyp c 99, 102, 107, 167, 178, 200,
 201, 238
 Datentyp CHAR 136, 138
 Datentyp CUKY 139, 140, 178
 Datentyp CURK 167
 Datentyp CURR 139, 140, 167, 178
 Datentyp d 165, 200
 Datentyp DEC 139, 178
 Datentyp f 269
 Datentyp i 84, 90, 173
 Datentyp n 100, 102, 107
 Datentyp NUMC 138
 Datentyp p 84, 87, 90, 167, 173, 178
 Datentyp QUAN 139, 178
 Datentyp string 425
 Datentyp t 166
 Datentyp UNIT 139, 178
 Datenübergabe 403
 Datenverlust 157, 160
 Datumsfelder 165
 deklarieren 165
 füllen 166, 168
 Rechnen mit 167
 Verarbeitung von 167
 Datumsformat 78
 Debugging 115
 Debugging-Modus 115, 117
 DEC, Datentyp 139, 178
 DECIMALS 84
 DEFAULT 271, 280
 Deklaration 85
 Deklarationsteil 23

- DELETE 198, 203, 204, 218, 221, 339, 343, 356, 367
 - ADJACENT DUPLICATES 343
 - DESCENDING 341
 - DESCRIBE 336
 - LINES 337
 - Dezimaltrennzeichen 86
 - Dezimalzahlen 84
 - Dialogmodule 24
 - Dictionary-Elemente 324
 - Dictionary-Objekte 34, 38
 - digitale Signatur 321
 - Dispatcher 21
 - DIV 92, 173
 - DIVIDE 89
 - Division 88, 91, 173
 - ganzzahlige 92, 263
 - verschiedene Fälle 91
 - DO 231
 - Dokumentation 42
 - Domäne 38, 39, 86
 - aktivieren 45
 - anlegen 43
 - prüfen 45
 - selbst erstellen 40
 - DO-Schleifen 232
 - drucken 265
 - Druckposition 96
 - Dump-Aufbereitung 68
 - dynamische Namensvergabe 201, 204
 - Dynamisches Programm -> s. Dynpro
 - Dynpro 21, 265
 - Dynpro-Prozessor 22
 - Dynpro-Prüfung 145
- E**
- Editor-Modus 60
 - Editorsperre 57
 - Eigenschaften 56
 - semantische 145
 - Eingabeblocke 296
 - Eingabefelder positionieren 296
 - Eingabehilfen 147
 - Eingabeprüfung 138, 141, 143, 145, 268, 270, 313
 - Einsetzen 156
 - Einstellungen, technische 49
 - Einzelschritt 119, 355, 367
 - Einzelwerte 44, 277
 - elementare Datentypen 84
 - Elemente
 - kopieren 80
 - selbst erstellen 40
 - ELSE 226, 227
 - ELSEIF 226, 227
 - END OF 322
 - END OF BLOCK 298
 - END OF LINE 296
 - ENDCASE 228
 - ENDDO 231
 - ENDFORM 377, 380, 381
 - ENDIF 222, 225
 - ENDLOOP 335
 - Endlosschleifen 231
 - ENDSELECT 70
 - ENDWHILE 233
 - Entwicklerschlüssel 29
 - Entwicklungssystem 18
 - Ereignisblöcke 24
 - Ereignisschlüsselwort 266, 312
 - Ereignisse 24, 266, 312
 - beenden 237
 - Beispiele 268
 - Ereignissteuerung 312
 - Ersatzzeichen 389
 - EXIT 236, 237, 257
 - Exponentialzahlen 90
 - EXPORT 403
 - EXPORT TO MEMORY 403
 - Exportparameter 397, 401
 - externer Report aufrufen 385
- F**
- f, Datentyp 269
 - F1 52, 66, 272, 297
 - F3 68, 292
 - F4 43, 52, 137, 146, 272, 285
 - F5 119
 - F6 119
 - F7 119
 - F8 68, 120, 313
 - Fallunterscheidung 221, 228
 - Fehleranalyse 68
 - Fehlerausgang 402, 425
 - Fehlerbehandlung 207, 269, 294, 295
 - Fehlerfall 229, 251

- Fehlernachricht 145
- Fehlerprotokoll 157
- Fehlersuche 115
- Feld 37
- Feldbezeichner 42, 282
- Felddeklarationen 165
- Felder 79, 83, 115, 150
 - deklarieren 83
 - hinzufügen 131
 - Länge 84
 - namensgleiche 323, 328, 329, 333
 - Namensgleichheit 334
 - Sternchen 70
 - verarbeiten 83
- Felder-Modus 118
- Feldinhalte
 - ändern 121
 - modifizieren 121
 - prüfen 120
 - überwachen 126
 - verschieben 102
- Feldlänge 99, 100
- Feldliste 70, 139, 325, 379, 403
- Feldname 37
- Feldtyp 324
- Festpunktarithmetik 57
- Festwerte 88, 136
 - Einzelwerte 137
 - Intervalle 137
 - pflügen 137
 - testen 137
- FOR FIELD 297
- FORM 377, 378, 381, 382, 384
- FREE 345, 346, 354
- freie Texte 296
- Fremdschlüssel 45, 141, 144, 196
 - pflügen 143, 144
 - testen 146
- Fremdschlüsselbeziehung 159
- Fremdschlüsselfeld, Art 145
- Fremdschlüsseltabelle 144
- FROM 344
- Frontend-Modus 58, 59
- Function Builder 31, 388, 390
 - Testumgebung 394
- Funktionsbausteine 24, 31, 61, 196, 372, 387
 - Ablaufart 391

- Aufruf 396
- aufrufen 398
- Ausnahmen 393
- Dokumentation 397
- Eigenschaften 391
- Export 392
- Exportparameter 392
- Import 391
- Importparameter 392
- Name 396
- suchen 388, 389
- testen 388, 394, 395
- Wertübergabe 391
- Funktionsbibliothek 388
- Funktionscodefeld 30
- Funktionsgruppen 388, 389

G

- ganze Zahlen 84
- Ganzzahldivision mit Rest 92
- ganzzahlige Division 92
- generischer Datentyp 99, 100
- generischer Schlüssel 145
- gepackte Zahlen 90
- GET PARAMETER 403
- GET PARAMETER ID 402
- Gleitpunktzahl 269
- globale Variablen 377
- globales SAP Memory 402
- Grenzen 319
- Groß- und Kleinschreibung 100, 204, 256, 270, 336, 384
- Größenkategorie 50
- Grundrechenarten 83, 88
- Grundsätze der ABAP-Programmierung 25

H

- Haltepunkte 117
- Hash-Algorithmus 321, 339
- Hash-Tabelle 321, 339
- Hierarchiestufen 100
- HIGH 276
- Hilfe 61
- Hintergrundverarbeitung 288
- Hochkommas 74
- Hut 354, 369, 381

I

- i, Datentyp 84, 90, 173
- IF 222, 225, 227, 235, 240
- IF-Struktur 228, 275, 278, 402
 - schachteln 227
- IMPORT 404
- IMPORT FROM MEMORY 403
- Importparameter 397
- IN PROGRAM 384
- INCLUDE 330, 334, 373
- Include
 - expandieren 154
 - komprimieren 154
- INCLUDE STRUCTURE 330
- Include-Report 373
- Include-Struktur 151
 - aktivieren 154
 - anlegen 152
 - einfügen 152
 - einpflegen 151
 - Positionierung 151
 - Voraussetzungen 151
- INDEX 336, 342
- Index 320, 336, 338, 339
- Indextabellen 321, 336, 339, 343
- Industrieminuten 189
- Infosysteme 28
- INITIAL SIZE 325
- INITIALIZATION 267, 268, 312
- Initialwert 38, 86, 200
- Inkonsistenzen prüfen 146
- Inline-Dokumentation 65
- INSERT 198, 199, 202, 337, 342, 366
- Instanz 18, 20
- Integer 84
- interne Tabellen 317, 380, 381, 392, 404
 - deklarieren 319, 322, 325
 - satzweise füllen 326
 - Struktur 318
 - verarbeiten 335
 - Vorteile 318
- Intervall 44, 239, 277, 279
- INTO 342, 343
- INTO CORRESPONDING FIELDS 331
- IS INITIAL 239, 274

K

- Kardinalität 146
- Kernbereiche 28
- Kettensatz 71, 77, 95, 322, 323, 324
- Key 37
- Klammerebenen 240
- Klammern 243
- Kommandos 66
- Kommentare 65
 - Anführungszeichen 65
 - im Bildschirm 297
 - Sternchen 65
- Kommentarzeichen 400
- Kommentarzeilen 81, 223
 - umwandeln von Zeilenblöcken in 81
- kompatible Datenobjekte 89
- komplexe Bedingung 225
- Komponenten, Append-Struktur 149, 150
- Konsolidierungssystem 18, 19
- Konstanten 83
- kontextsensitiv 295
- Kontrollstruktur 221, 225
- konvertible Datenobjekte 89
- Konvertierungsregeln 89, 90
- Kopfzeile 318, 319, 325, 326, 328, 335, 380, 381
- Kopfzeile initialisieren 344
- Kopieren 156
- Korrekturwesen 19
- Kryptologie 321
- Kundennamensraum 149, 152
- Kurzbeschreibung 40
- Kurzdokumentation 73

L

- Länge 107
- Langtext 397
- Laufzeiten 158
- Laufzeitobjekt 134
- Laufzeitumgebung 56, 266
- Leerzeichen 88
 - entfernen 104
- Leerzeile erzeugen 71
- Lesbarkeit 25, 84
- Lesesperre 197
- LIKE 85, 100, 165, 272, 328
- LINES 337

LINE-SIZE 76
Linie erzeugen 71
Listaufbereitung 70
Listbild 77
Listbildschirm 265
Liste anzeigen 120
Listen aufbereiten 70
Listprozessor 265, 267
Listüberschriften 281
Literal 66, 87, 100, 204, 218, 336, 356, 396, 404
Literale 186, 256, 291
LOAD-OF-PROGRAM 24, 267
logische Ausdrücke 221, 237
 verknüpfen 240
 verknüpfte 227
logische Datenbank 57
logische Operatoren 238
logisches Oder 240
logisches Und 241
Logistik 28
lokale Namen 382
lokale Objekte 20
lokale Variable 378, 380
 explizit deklarieren 378
 implizit deklarieren 378
lokales Objekt 37, 80, 133
lokales SAP Memory 403
LOOP 335, 336, 339, 342
LOW 276
LOWER CASE 272, 273, 280
LT 235

M

Mandant 27
Mandantenbehandlung, automatische 198
Mandantenummer 198
Mandantenunabhängigkeit 198
Mandantenverwaltung, automatische 51
Markiermodus 63, 64, 81
Mehrfachselektion 277
Mehrfachverwendbarkeit 152
Meldung in der Statuszeile 269
Memory-ID 403
Mengeneinheit 139
Mengenfelder 138, 139

Menu Painter 32
MESSAGE 293, 295, 313
MESSAGE-ID 293
MOD 92, 173
MODIFY 198, 202, 336, 342
 FROM 342
Modularisierung 16, 324, 371
Module 378
Modulpool 266
Modus 68
MOVE-CORRESPONDING 328
Multi-Byte-Codierung 57
Multiplikation 88
MULTIPLY 89
Muster 62, 222
 einfügen 222, 224

N

n, Datentyp 100, 102, 107
Nachkommastellen 84, 86
Nachrichten 291
 pflegen 294
 sichern 294
Nachrichtenklasse 293, 295
 anlegen 293
Nachrichtenummer 295
Nachrichtenpflege 293
Nachrichtenpool 293
Nachrichtentyp 295, 313
Nachrichtentyp E 295
Namensgleichheit 333
Namenskonventionen 35, 55, 149
Namensräume 35
Namensvergabe
 dynamisch 204
 für Variablen 83
 statisch 202, 203
Native SQL 22, 35
Navigationsfenster 61
Negation 241
Netzlaufwerke 395
Netzwerkpfade 395
NEW-PAGE PRINT ON 265
Nicht-Schlüsselfelder 155
 ergänzen 136
NO-EXTENSION 280
NO-GAPS 104
NOT 241

NUMC, Datentyp 138
numerische Textfelder 100
numerische Zeichen 100

O

Object Navigator 32
Objekt
 aktivieren 136
 lokale 20
 lokales 37, 80, 133
Objektklassen 194
Objektliste 61
Objektorientierung 322, 323
Objektschlüssel 148
obligatorische Parameter 397
OBLIGATORY 271, 280
OCCURS 323
Open SQL 22, 31, 35, 193
 Anweisungen 198
Operationen, arithmetische 174
OPTION 276
OR 240
Originalsprache 56

P

Paket 36
Parameter
 obligatorische 397, 400
 optionale 400
Parameter-ID 402
 anlegen 403
 initialisieren 414
 lesen 414
 schreiben 424
Parameterliste, prüfen 383
PARAMETERS 269
PERFORM 376, 377, 380, 381, 382,
 384, 414
 IN PROGRAM 384
 USING 378
Personal 28
Pflegebildschirm 35, 153
Pflegedialog 135
physikalischer Bereich 49
Platzhalter 295
Pooltabellen 34
POSITION 298
Position 96

 angeben 96
Positionierung 107
 direkte 107
Positionsnummern 100
Postleitzahlen 86
Präsentationsschicht 16
Präsentationsserver 16
Pretty Printer 62
Primärschlüssel 202, 203
Produktivsystem 18, 19
Profile 194
Programm
 aktivieren 61, 67
 prüfen 67
 sichern 67
 starten 68
Programmabbruch 68
Programmablaufsteuerung 221
Programmausführung 68
Programmlogik prüfen 115
Programmnamen 55
 dynamische 384
Programmstatus 57, 60
Prozeduren 24, 372, 375, 387
Prüfsummenverfahren 321
Prüftabelle 144, 159
Pufferung 50, 193

Q

Qualitätssicherungssystem 19
QUAN, Datentyp 139, 178
Quellcode
 editieren 63, 72
 modularisieren 371
 schreiben 72
 umschalten 60
Quellfeld 90, 105
Quelltablelle 132

R

RADIOBUTTON GROUP 275
Radiobuttons 274
Rahmen 296
Rahmenprogramm 376
READ 337, 338, 343, 353, 355
 INDEX 342
 INTO 342
Rechengenauigkeit 91, 92, 178

Rechenzeichen 89
 - 89
 * 89
 + 88
 / 89, 91
 Rechnungswesen 28
 Redundanzen 157
 Referenzfeld 140, 159, 167
 REFRESH 344, 345, 354, 367
 Registerblatt, Felder 150
 Reorganisationsarbeiten 132
 REPLACE 103, 113
 REPORT 66, 80, 116, 293
 Report 21
 Datenübergabe 402
 kopieren 79
 mit Variante starten 290
 über Variante starten 385
 REPORT, LINE-SIZE 76
 Repository-Infosystem 388
 reservierter Speicherbereich frei-
 geben 345
 Returncode 121, 198, 200, 202, 203,
 229, 251, 356, 402, 425
 Rolle 194
 Rückgabewert 121
 Rumpfeile 381, 404
 löschen 343, 344, 345
 verändern 336

S

Sammelprofile 194
 Sammelschacht 227
 Sandkasten 19
 SAP GUI 17, 26
 SAP Logon 26
 SAP Memory 414
 lesen 402
 schreiben 402
 SAP R/3 15
 SAPScript-Editor 42
 SAP-System, Architektur 16
 Satz
 ändern 202
 anlegen 202
 Schachteln 75
 Schachtelung 230, 232, 234, 324, 375
 Schachtelungsebenen 227
 Schaltjahr 242
 Schleife 70, 74, 75, 191, 230, 318
 Abbruchanweisungen 234
 äußere 232
 Endlosschleifen 231
 innere 232
 mit Bedingung 230
 ohne Bedingung 230
 Schleifendurchläufe begrenzen 231
 Schleifenverarbeitung 75
 Schlüssel 325, 336, 338, 343
 benutzerdefinierter 320
 eindeutige 320
 generischer 145
 nicht eindeutige 320, 325
 Schlüsselfelder 37, 151, 156, 338, 343
 löschen 157, 159
 Manipulation 131
 Veränderung 155
 Schlüsselwortdokumentation 66
 Schlüsselwörter 61
 Schnittstellen 324
 Schrägstich 70
 Schreibsperrung 197
 Schreibweise 322
 moderne 383
 traditionelle 383
 Screen Painter 32
 SE11 33, 132
 SE38 55
 SE91 293
 SELECT 70, 191, 198, 229, 230, 327, 331
 SELECTION-SCREEN 296
 BEGIN OF BLOCK 298
 BEGIN OF LINE 296
 COMMENT 297
 END OF BLOCK 298
 END OF LINE 296
 FOR FIELD 297
 POSITION 298
 TITLE 299
 WITH FRAME 299
 SELECT-OPTIONS 276, 281
 Selektionen 269, 275
 Selektionsbildschirm 57, 265, 272, 284,
 289, 385, 386
 speichern 284
 übergehen 386

Selektionsbildverarbeitung 295
 Selektionstabelle 276
 Selektionstexte 281
 aktivieren 283
 anlegen 281
 pflegen 282
 verwenden 281
 Selektionsvariante anlegen 285
 semantische Eigenschaften 145
 SEPERATED BY 105
 SET PARAMETER 403
 SET PARAMETER ID 402
 SHIFT 102
 SIGN 276
 Sizing 18
 SKIP 71, 248
 SORT 340, 358
 BY 340
 SORTED TABLE 325
 Sortierkriterien 341
 Sortierreihenfolge 341
 sortierte Tabelle 321, 337, 339
 Sortierung
 binär 341
 sprachspezifisch 341
 SPACE 112
 Spaltenaufteilung 96
 Speicherbereich 323
 Speicherplatz, initialer 325
 Sperre
 aufheben 197
 logische 196
 Zeitpunktproblematik 197
 Sperrkonzept 194, 195
 physikalische Datenbank 195
 SAP R/3 196
 Sperrobject 196
 entsperren 196
 Sperre setzen 196
 Sperrproblematik 197
 Sperrtabelle 196
 SPLIT 106, 113
 AT 106
 Sprachenschlüssel 293
 Stammdaten 36
 STANDARD TABLE OF 325
 Standardschlüssel 320
 Standard-Selektionsbild 266
 Standardtabelle 149, 155, 320, 325, 337
 erweitern 148
 START-OF-SELECTION 24, 267, 314,
 376
 Startwert 86
 statische Namensvergabe 202, 203
 Status 56
 Stellungsooperanden 105, 295, 378,
 380, 384
 Stoppschild 355
 Stoppuhr 233
 Streuwertfunktion 321
 string, Datentyp 425
 String-Operation 187
 String-Operationen 101, 108, 174
 Struktur 69, 318
 einbinden 330
 Komponenten 150
 Strukturanfang 222
 Strukturen 150
 Strukturende 222, 266
 Strukturpflegebildschirm 153
 SUBMIT 385, 386, 387, 403
 AND RETURN 387
 VIA SELECTION-SCREEN 385
 WITH 386
 SUBTRACT 89
 Subtraktion 88
 Suchbegriff 103
 Suche, generische 389
 Suchen 65
 Suffix 395
 Summen
 ausgeben 247
 ermitteln 247
 Summenfelder, initialisieren 247
 SY-DATUM 165
 SY-INDEX 234, 236
 Symbolleisten 60
 Syntax 321
 prüfen 61
 Syntaxfehler 61
 Syntaxprüfung 61, 375
 Systemdatum 165
 Systemmandant 39
 Systemtabelle 121, 293
 SY-SUBRC 121, 198, 200, 229, 393
 SY-TABIX 339

T

- t, Datentyp 166
- T100 293, 313
- Tabelle 61, 324
 - aktivieren 50
 - anlegen 35
 - deklarieren 73
 - Fehlermeldung 140
 - interne 317
 - kopieren 132
 - löschen 160, 162
 - modifizieren 131
 - ohne Kopfzeile 326
 - Pflegedialog 51
 - pflegen 35, 135
 - physikalische Umsetzung 157
 - prüfen 49
 - sichern 36
 - temporäre 317
 - transparente 34
 - Umsetzung 159
 - vervollständigen 48
- Tabelle T100 313
- Tabelle TCURC 141
- Tabellenarbeitsbereich 69
 - als Puffer 200
 - löschen 200
- Tabellenbereich 69, 73
 - prüfen 122
- Tabellendeklaration 321
- Tabelleneinträge 135
- Tabellenfeld
 - löschen 159
 - pflegen 37
- Tabellenindex 339
- Tabelleninhalt
 - anzeigen 52
 - modifizieren 198
- Tabellenkopf 319
- Tabellen-Modus 118, 122
- Tabellenrumpf 318, 319, 326, 335
- Tabellenstatus 134
- Tabellenstruktur 76
 - physikalische 155
- Tabellentyp 324, 325, 332
- Tabellenverarbeitung 321
 - mehrdimensional 319
 - n-dimensional 319
- Tabellenzeile 134, 318
- TABLE 344
- TABLES 69, 199, 380, 382
- Tablespace 49
- TCURC 142, 144
 - Tabelle 141
- technische Einstellungen 49
- Teilstring 107
- temporäre Daten 276
- temporäre Tabellen 317
- Testdatensätze eingeben 135
- Testmodus 61
- Testsystem 18
- Texte länderspezifisch pflegen 56
- Textelemente 186, 281
 - Pflegedialog 291
- Textfelder, numerische 100
- Textobjekte, ergänzende 291
- Textpool 281
 - aktivieren 292
- Textsymbole 281, 291, 297
 - aktivieren 292
 - pflegen 291
- Textsymbolnummer 292
- TIMES 231
- TITLE 299
- Transaktion SE91 293
- Transaktion, Laufzeit 403
- Transaktionscode 30, 195
- TRANSLATE 273
- transparente Tabellen 34
- Transparenz 25
- Transport 36
- Transportwesen 19
- Trennzeichenkette 106
- Trick 35 169, 171
- Typdeklaration 69, 324, 325, 334
- TYPE 84
- TYPES 324
- Typkonvertierung 113, 165
 - automatische 239

U

- Übersetzbarkeit 291
- ULINE 71
- Umsetzung im Dialog 158
- Unabhängigkeit 15
- UNDER 96

- Unicodeprüfung 57
- UNIT, Datentyp 139, 178
- Unterprogramm 24, 372, 375, 379
 - aufrufen 376, 383
 - externe 383
 - interne 383
- Unterprogrammfelder 379
- Unterstrich 76
- Unterstrukturen 151
- UPDATE 198, 201, 202
- Upgrade 148
- USING 378, 382

V

- VALUE 86, 100, 166, 354
- VALUE CHECK 272
- Variablen 83, 404
- Variablenamen 83
- Variante 58, 285
 - anlegen 285
 - Eigenschaften 288
 - schützen 288, 289
 - Start über 58
- Variantenpflege 287
- Verarbeitung
 - lesende 318
 - schreibende 318
- Verarbeitungsart
 - Direkt 158
 - Hintergrund 158
- Verarbeitungsblöcke 24, 266, 371
- Verbuchungsaufgaben 391
- Verprobung 138, 146
- Verschieben von Feldinhalten 102
- Verwendungsnachweis 61, 160, 161
- Verzweigung 225
- VIA SELECTION-SCREEN 385, 386
- Vorbelegung 86
- Vorselektion 246
- Vorwärtsnavigation 40, 69, 150, 153, 154, 373, 376, 403

W

- WAERS, Domäne 141, 143
- Währungsbetrag 139
- Währungsbezeichnung 167
- Währungseinheit 139, 143, 215
- Währungsfelder 138, 139, 166
 - deklarieren 166
- Währungsschlüssel 167
- Wartbarkeit 25
- Watchpoints 115, 124
 - anlegen 124
 - sichern 127
- Watchpoints-Modus 124
- Wechselkursumrechnung 178
- Weitersuchen 65
- Werte
 - gültige 141
 - negativer 86
 - übergeben 385, 386
- Wertebereich 44
- Werteliste 141
- Wertetabelle 44, 141, 143, 144
- Werthilfe 136, 146
- Werthilfeliste 137, 146
- WHEN OTHERS 229
- WHERE 204, 230, 340, 353, 367
- WHILE 233
- WHILE-Schleife 233
- Wiederholung 221
- Wiederverwendbarkeit 324
- WITH 295, 386
- WITH FRAME 299
- WITH HEADER LINE 325, 365
- WITH KEY 338, 343
- WITH UNIQUE KEY 325
- Workarea 200, 202, 318, 319, 323, 341, 343, 366
 - deklarieren 200, 325
 - initialisieren 208, 345
- Workprozess 21, 196
- WRITE 66, 70, 71, 96
 - UNDER 96

X

- X-Ablage 62

Y

- Y-Ablage 62

Z

- Z-Ablage 62
- Zahlen
 - ganze 84
 - gepackte 90

- Zeichen
 - alphanumerische 99
 - ausschneiden 101
 - numerische 100
- Zeichenkette 74, 99, 165
 - ergänzen 101
 - ersetzen 101, 103
 - modifizieren 99
 - suchen 101
 - verdichten 104
 - verschieben 101
 - zerlegen 106
 - zusammenziehen 105
- Zeichenkettenoperationen 101
- Zeile
 - aktualisieren 203
 - ändern 336
 - anlegen 202
 - ausschneiden 63
 - Doppelungen 321
 - einfügen 156, 337
 - einsetzen 63
 - frei gestalten 296
 - kopieren 63
 - löschen 64, 156, 203, 339
 - markieren 59
 - neue Zeile einfügen 64
 - redundante 343
 - verdoppeln 64
 - verketteten 64
 - verschieben 64
- Zeilenblock 298
 - löschen 81
 - markieren 81
- Zeilenstruktur 325
- Zeilentyp 320, 323, 324, 331
 - deklarieren 326
- Zeilenumbruch 70, 76
- Zeitfelder 166
 - deklarieren 166
 - füllen 166
- Zeitpunktproblematik 197
- Zeitspanne bei Datumswerten 172
- zentrale Steuerungskomponente 426
- Zielfeld 90, 105
- Zieltabelle 132
- Zugriffsaufwand 338
- Zugriffsschlüssel 320
- Zugriffszeit 320, 321
- Zusätze 66
- Zwischenablagen 59, 62
- Zwischenpuffer 62