

Einstieg in ABAP®

Für
Ausbildung
und Beruf

- › Ihre Einführung in die moderne ABAP-Entwicklung
- › Alle Grundlagen der Cloud- und On-Premise-Entwicklung
- › Praxisnahe Beispiele für die direkte Anwendung

 Mit Programmierbeispielen zum Download

Felix Roth

Kapitel 4

ABAP Dictionary

Das ABAP Dictionary ist die zentrale Stelle für alle im SAP-System vorhandenen Datendefinitionen – von einfachen Datenelementen über komplexe Strukturen bis hin zu vollständigen Datenbanktabellen. Damit bildet es die Grundlage für nahezu jede ABAP-Entwicklung.

In Kapitel 2, »Erste Schritte«, habe ich Ihnen gezeigt, wie Sie erste Felder, Konstanten, Strukturen und lokale Typen im ABAP-Quelltext auf Basis von eingebauten Datentypen wie `i` oder `string` definieren. In der Praxis wird jedoch häufig auch auf globale Datentypen, Strukturen und Datenbanktabellen zurückgegriffen, die im ABAP Dictionary definiert sind. In diesem Kapitel möchte ich Ihnen erläutern, was das ABAP Dictionary ist und wie Sie damit arbeiten.

Die Hauptaufgabe des ABAP Dictionary ist die Verwaltung der an das SAP-System angebotenen Datenbank *SAP HANA*, die unter dem System liegt. Damit ist das Dictionary eine Sammlung von Werkzeugen, um auf Tabellen auf dieser Datenbank zuzugreifen, sie zu erzeugen und zu verwalten. Dazu kann mit ABAP und ABAP SQL über das ABAP Dictionary auf die Datenbanktabellen zugegriffen werden, ohne diesen Zugriff (z. B. den Aufbau und Abbau der Verbindung) explizit orchestrieren zu müssen. Listing 4.1 zeigt einen solchen Zugriff auf die Datenbanktabelle `/DMO/FLIGHT` mit einer `SELECT`-Anweisung.

```
SELECT FROM /dmo/flight
  FIELDS *
  INTO TABLE @DATA(lt_ergebnis).
```

Listing 4.1: Selektion auf eine Datenbanktabelle

In diesem Beispiel werden alle Spalten und Zeilen dieser Datenbanktabelle in der internen Tabelle `LT_ERGEBNIS` geladen, mit der nun auf dem Applikationsserver mit ABAP weitergearbeitet werden kann. Die Arbeit mit internen Tabellen erläuterte ich Ihnen in Kapitel 6, »Mit internen Tabellen arbeiten«. Der Zugriff auf Datenbanken mit ABAP SQL wird in Kapitel 7, »Zugriff auf die Datenbank«, erläutert.

In diesem Kapitel konzentrieren wir uns zunächst auf die grundlegenden Objekte des ABAP Dictionary, auf denen u. a. alle Datenbanktabellen basieren. Zentrale Grundlage ist dabei das *zweistufige Domänenprinzip*, das im gleichnamigen Abschnitt 4.1 erläutert wird und zeigt, wie die Datenbanktabelle aus Listing 4.1 aufgebaut ist.

Anschließend lernen Sie die Anlage der wichtigsten Entwicklungsobjekte kennen:

- Domänen (siehe Abschnitt 4.2)
- Datenelemente (siehe Abschnitt 4.3)
- Datenbanktabellen (siehe Abschnitt 4.4)
- Strukturen (siehe Abschnitt 4.5)

Diese Entwicklungsobjekte zu verstehen und anlegen zu können, ist elementar, zum einen, um Quelltext von SAP zu analysieren, zum anderen, um auch größere kundeneigene Entwicklungen umsetzen zu können, die auf eigenen Datenbanktabellen basieren.

Zusammenfassend lässt sich sagen, dass das ABAP Dictionary ein zentraler Bestandteil der Entwicklungsumgebung von SAP ist. Es dient dazu, Datenstrukturen einheitlich zu definieren, zu verwalten und systemweit zur Verfügung zu stellen. Das ABAP Dictionary ist daher nicht nur ein Werkzeug, sondern besteht aus einer Sammlung von mehreren Werkzeugen, die jeweils ein Entwicklungsobjekt bearbeiten.

Das ABAP Dictionary sorgt für:

- zentrale Datenbeschreibung
- Konsistenz im gesamten System
- Wiederverwendbarkeit von Definitionen
- automatische Datenbankanpassung

4.1 Das zweistufige Domänenprinzip

Das ABAP Dictionary basiert auf einem zweistufigen Domänenprinzip, das technische und semantische Domänen vorsieht:

- Die technischen Domänen beschreiben den Wertebereich eines Felds, der durch die Angabe eines eingebauten Datentyps, der Ausgabelänge und eventueller Festwerte festgelegt wird. Im ABAP-Umfeld werden diese technischen Domänen nur *Domänen* genannt.
- Die semantischen Domänen weisen den technischen Domänen durch die Vergabe von Texten einen bestimmten Zusammenhang zu. Im ABAP-Umfeld werden diese semantischen Domänen *Datenelement* genannt.

Wie in Abbildung 4.1 dargestellt, verweist ein Feld einer Struktur oder Tabelle auf ein Datenelement. Dieses Datenelement referenziert wiederum eine Domäne, die die technischen Eigenschaften festlegt.

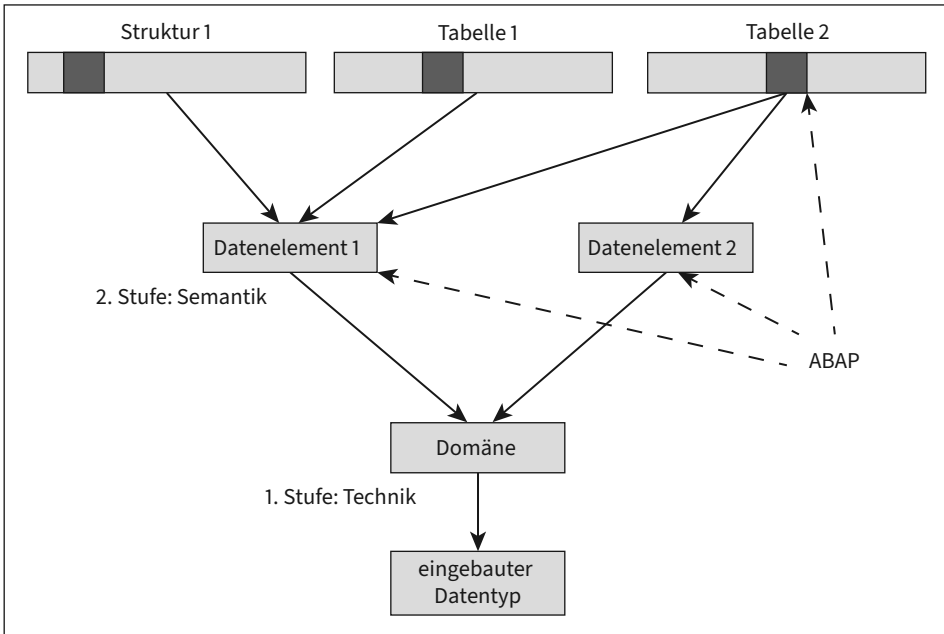


Abbildung 4.1: Das zweistufige Domänenprinzip

Eine Domäne kann demzufolge in mehreren Datenelementen verwendet werden, und ein Datenelement kann in vielen Feldern von Tabellen und Strukturen verwendet werden. Die Domäne fasst also technische Informationen über mehrere Tabellen hinweg zusammen und kann in Form von Datenelementen verschiedene Ausprägungen haben. Darüber hinaus können Sie auf die Datenelemente auch aus ABAP heraus zugreifen und z. B. eine Variable von diesem Typ mit der Anweisung `DATA` und dem Zusatz `TYPE` definieren. Es geht bei den Domänen und Datenelementen also primär um das Thema der Wiederverwendbarkeit, aber auch darum, die eingebauten Datentypen mit Zusatzinformationen anzureichern, wie z. B.:

- Suchhilfen
- verschiedene Texten
- Wertebereiche
- Konvertierungsbausteine

Abbildung 4.2 zeigt beispielhaft die Datenbanktabelle `/DMO/FLIGHT`. Die Tabelle besteht aus neun Feldern. Während für das Feld `client` der eingebaute Datentyp `clnt` verwendet worden ist, basieren die anderen acht Felder jeweils auf einem Datenelement.

```

1 @EndUserText.label : 'Flight Reference Scenario: Flight'
2 @AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
3 @AbapCatalog.tableCategory : #TRANSPARENT
4 @AbapCatalog.deliveryClass : #A
5 @AbapCatalog.dataMaintenance : #RESTRICTED
6 define table /dmo/flight {
7
8   key client      : abap.clnt not null;
9   key carrier_id  : /dmo/carrier_id not null;|
10  key connection_id : /dmo/connection_id not null;
11  key flight_date  : /dmo/flight_date not null;
12  @Semantics.amount.currencyCode : '/dmo/flight.currency_code'
13  price           : /dmo/flight_price;
14  currency_code   : /dmo/currency_code;
15  plane_type_id   : /dmo/plane_type_id;
16  seats_max       : /dmo/plane_seats_max;
17  seats_occupied  : /dmo/plane_seats_occupied;
18
19 }

```

Abbildung 4.2: Definition der Datenbanktabelle /DMO/FLIGHT

Für das Feld CARRIER_ID aus Zeile 9 ist beispielsweise das Datenelement /DMO/CARRIER_ID angegeben. Wenn Sie den Cursor auf den Namen des Datenelements (/DMO/CARRIER_ID) setzen und die Taste **F3** drücken (alternativ **Strg** + Mausclick oder den Eintrag **Navigate to** über das Kontextmenü wählen), wird Ihnen das Datenelement als eigene Maske wie in Abbildung 4.3 dargestellt.

Abbildung 4.3: Definition des Datenelements /DMO/CARRIER_ID

In dieser Ansicht sind alle Eigenschaften des Datenelements dargestellt. Unter anderem erkennen Sie, dass dieses Datenelement auf der gleichnamigen Domäne /DMO/CARRIER_ID aufbaut. Über einen Klick auf das Feld **Type Name** gelangen Sie zur Definition der Domäne (siehe Abbildung 4.4).

The screenshot shows the SAP domain definition interface for the domain /DMO/CARRIER_ID. The interface is divided into several sections:

- Format:** Contains fields for 'Data Type' (set to CHAR), 'Length' (set to 3), and a 'Browse...' button.
- Output Characteristics:** Contains fields for 'Output Length' (set to 3), 'Conversion Routine', and a checkbox for 'Case-sensitive' which is unchecked.
- Fixed Values:** A section titled 'Fixed Values' with the instruction 'Define the single fixed values or interval of values'. It includes icons for adding, deleting, and moving values, and an 'Intervals' checkbox.
- Value Table:** A section with a 'Value Table' field and a note: 'Specify a value table as the proposed value for checking the foreign key'.

Fixed Value	Description
<Enter new va...	

Abbildung 4.4: Definition der Domäne /DMO/CARRIER_ID

In der Definition ist letztlich der eigentliche eingebaute Datentyp CHAR (Feld **Data Type**) mit der Länge 3 (Feld **Length**) hinterlegt. Weitere Informationen, wie eine Konvertierungsroutine (Feld **Conversion Routine**), ein Wertebereich (Bereich **Fixed Values**) oder eine Wertetabelle (Feld **Value Table**) sind in diesem Beispiel nicht hinterlegt. Auch die Groß- und Kleinschreibung (Feld **Case-sensitive**) ist für diese Domäne nicht aktiviert, sodass die Inhalte nur in Großschreibung gespeichert werden.

Diese Domäne kann in beliebig vielen anderen Datenelementen wiederverwendet werden. Die Datenelemente können dann wiederum in weiteren Strukturen, Datenbanktabellen oder in Datendefinitionen mit ABAP verwendet werden.

Im nächsten Abschnitt zeige ich Ihnen, wie Sie Datenelemente und Domänen anlegen und diese in Strukturen und Datenbanktabellen verwenden.

4.2 Domänen anlegen

Um eine Domäne anzulegen, gehen Sie im Menü der ABAP Development Tools für Eclipse auf **File • New • ABAP Repository Object** und wählen im sich öffnenden Pop-up-Fenster aus Abbildung 4.5 im Ordner **Dictionary** den Eintrag **Domain** aus. Zusätzlich können Sie im Eingabefeld **type filter text** den Begriff »Domain« eingeben, um die Liste zu filtern. Klicken Sie anschließend auf **Next >**.

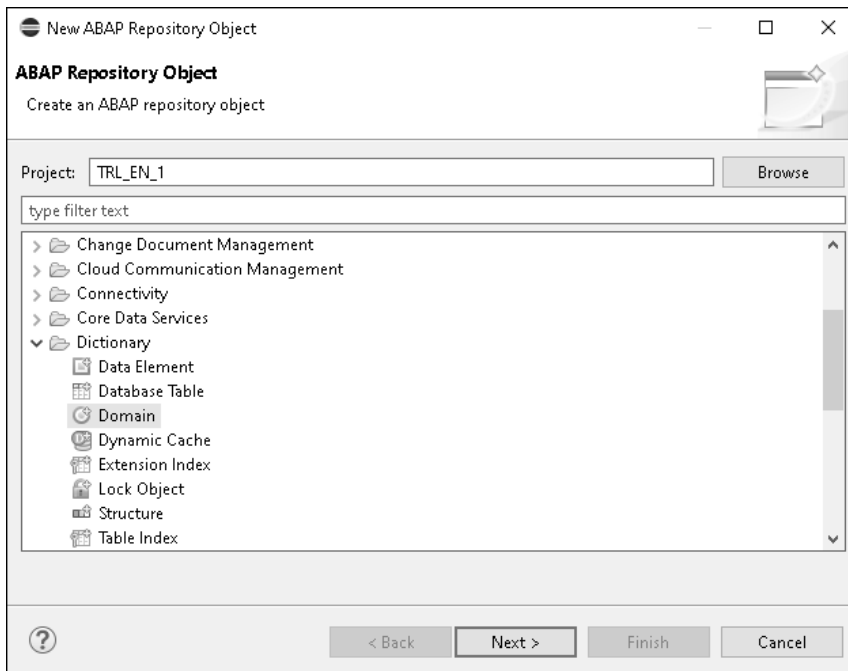


Abbildung 4.5: Dictionary-Entwicklungsobjekt anlegen

Im nächsten Schritt geben Sie im Feld **Package** Ihr Paket aus Abschnitt 2.2.1, »Paket anlegen«, ein (siehe Abbildung 4.6). Vergeben Sie zusätzlich im Feld **Name** einen Namen (hier »ZD_EMP_NAME«), gegebenenfalls mit Ihrem Namenskürzel, und im Feld **Description** eine Beschreibung (hier »Mitarbeitername«). Klicken Sie anschließend auf **Next >**.

Nach dem Bestätigen öffnet sich die Eigenschaftsseite der Domäne aus Abbildung 4.7. Hier können Sie nun in den Feldern **Data Type** und **Length** den Datentyp und die gewünschte Länge eingeben. In unserem Beispiel soll die Domäne vom Typ CHAR sein und die Länge 60 haben. Zusätzlich wird über die Checkbox **Case-sensitive** festgelegt, ob der Inhalt in Groß- und Kleinschreibung sein kann.

Domain
Create Domain

Project: *

Package: *

Add to favorite packages

Name: *

Description: *

Original Language:

Abbildung 4.6: Namen und Beschreibung einer Domäne angeben

Domain: ZD_EMP_NAME

Format **Output Characteristics**

Data Type: *

Length: *

Output Length:

Conversion Routine:

Case-sensitive

Fixed Values
Define the single fixed values or interval of values

Intervals

Fixed Value	Description
<Enter new>	

Specify a value table as the proposed value for checking the foreign key

Value Table:

Abbildung 4.7: Eigenschaften einer Domäne

Neben diesen grundlegenden Eigenschaften können Sie im Feld **Conversion Routine** eine Konvertierungsroutine angeben.

The screenshot shows the configuration interface for the data element 'ZE_EMP_NAME'. It includes fields for 'Category' (Domain), 'Type Name' (ZD_EMP_NAME), 'Data Type' (CHAR), and 'Length' (60). The 'Field Labels' section allows defining labels for different display lengths: Short (MitarbName, length 10), Medium (Mitarbeitername, length 20), Long (Mitarbeitername, length 40), and Heading (Mitarbeitername, length 55). The 'Additional Properties' section includes search help fields (Name, Parameter, Parameter ID, Component Name), checkboxes for 'Change Document Logging' and 'Input History', and 'Bidirectional Options' (Basic Direction: Right to Left, BIDI Filtering checked).

Abbildung 4.8: Eigenschaften eines Datenelements

Bei der Anlage des Datenelements haben Sie darüber hinaus aber noch die Möglichkeit, im Bereich **Field Labels** (hier »Mitarbeitername« in verschiedenen Längen) die Texte zum Datenelement und im Bereich **Additional Properties** beispielsweise eine Suchhilfe oder eine Parameter-ID anzugeben.


Nachdem Sie nun eine Domäne und ein darauf aufbauendes Datenelement angelegt haben, wollen wir im nächsten Schritt eine Datenbanktabelle anlegen, die dieses Datenelement verwendet.

4.4 Datenbanktabellen

Eine der Hauptfunktionen des ABAP Dictionary ist die Verwaltung der zentralen *Datenbanktabellen* des SAP-Systems. Eine Datenbanktabelle beinhaltet eine Menge von persistenten Daten. Das heißt, die Daten existieren nicht nur zur Laufzeit, sondern werden dauerhaft auf einer Festplatte des Datenbankservers vorgehalten. Die Struktur einer Datenbank lässt sich gut mit einem Excel-Sheet vergleichen: Daten sind in *Zeilen* und *Spalten* strukturiert. Die Spaltennamen definieren, welche Art von Daten in jeder Spalte stehen sollen, während die einzelnen Zeilen einen konkreten Datensatz repräsentieren. Über das ABAP Dictionary als Schnittstelle zur an das SAP-System angebotenen Datenbank können Sie sowohl die Strukturen als auch die Daten aller Datenbanktabellen anzeigen lassen. Zusätzlich können Sie auch eigene Datenbanktabellen anlegen, um Daten für Ihre Anwendung dauerhaft zu speichern.

Im gleichnamigen Abschnitt 4.4.1 lernen Sie, wie Sie bestehende Datenbanktabellen anzeigen, während ich in Abschnitt 4.4.2 die Anlage einer Datenbanktabelle zeige. Abschließend erfahren Sie in Abschnitt 4.4.3, wie Sie eine Datenbanktabelle umsetzen können. Dies wird insbesondere dann relevant, wenn sich Eigenschaften wie Datentyp oder Länge einer Domäne ändern und dadurch bereits gespeicherte Daten auf der Datenbanktabelle in »Gefahr« sind.

4.4.1 Datenbanktabellen anzeigen

Um eine Datenbanktabelle anzuzeigen, klicken Sie in der Funktionsleiste der ABAP Development Tools auf das Icon  (**Open ABAP Development Object**), oder Sie nutzen die Tastenkombination `Strg` + `⇧` + `A`). Geben Sie im sich öffnenden Pop-up-Fenster aus Abbildung 4.9 den Namen der gewünschten Tabelle ein. Wie in Abbildung 4.9 dargestellt, empfiehlt es sich, die Suche mithilfe des Präfixes »type:dtab« gefolgt von einem Leerzeichen einzuschränken. Dadurch wird gezielt nach Datenbanktabellen gesucht, während andere Entwicklungsobjekte ausgeblendet werden.

In Abbildung 4.9 wird beispielsweise die Datenbanktabelle `/DMO/FLIGHT` des Flugdatenmodells geöffnet, da diese auf jedem SAP-System vorhanden ist und wir so die umgangsweise mit Eclipse ADT üben können. Natürlich können Sie auf diese Art und Weise jede in Ihrem SAP-System vorhandene Datenbanktabelle öffnen und so nachvollziehen, wie diese Datenbanktabelle aufgebaut ist und welche Daten sie enthält.

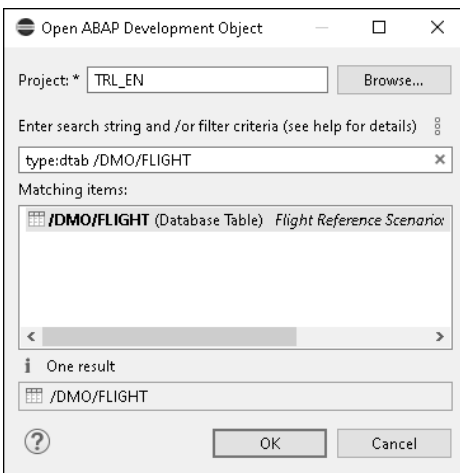


Abbildung 4.9: Nach einer Datenbanktabelle suchen

Öffnen Sie die gewünschte Tabelle mit einem Doppelklick. Dadurch wird Ihnen die Definition der Datenbanktabelle wie in Abbildung 4.10 angezeigt. Wenn Sie nun den Inhalt der Datenbanktabelle sehen möchten, können Sie mit einem Rechtsklick das Kontextmenü öffnen und

über die beiden Einträge **Open with • Data Preview** (ohne Feldauswahl) und **Open with • Data Preview...** (mit Feldauswahl) die Datenvorschau aufrufen.

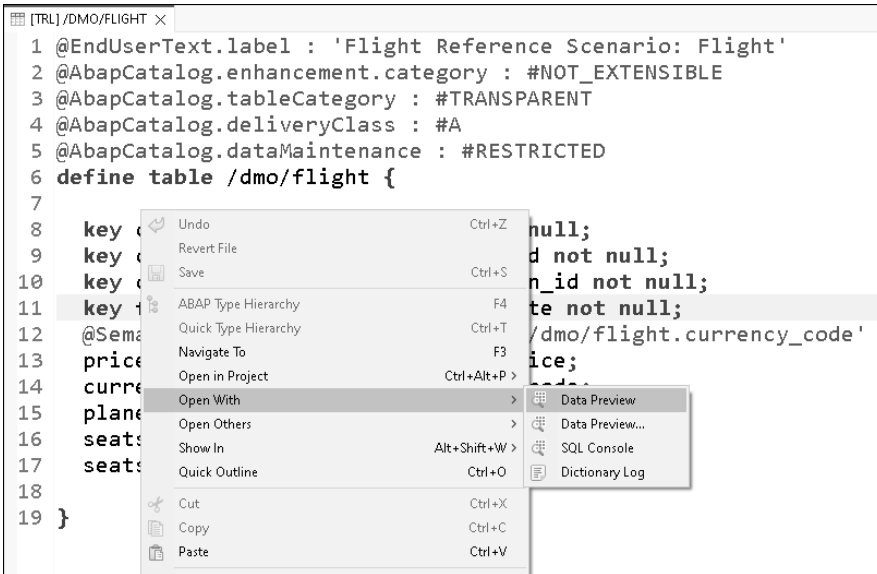


Abbildung 4.10: Datenvorschau für eine Datenbanktabelle aufrufen

In der sich öffnenden Datenvorschau (siehe Abbildung 4.11) sehen Sie nun (falls vorhanden) die ersten 100 Einträge der Datenbanktabelle. Hier haben Sie die Möglichkeit, nach Einträgen zu suchen, Filter hinzuzufügen oder über den Button **SQL Console** gezielt die SQL-Anweisungen zu modifizieren.

The screenshot shows the SAP Data Preview window for the table /dmo/flight. The table contains 40 rows of data. The columns are CLIENT, CARRIER_ID, CONNECTION_ID, FLIGHT_DATE, PRICE, CURRENCY_CODE, and PLAN. The data is as follows:

CLIENT	CARRIER_ID	CONNECTION_ID	FLIGHT_DATE	PRICE	CURRENCY_CODE	PLAN
100	SQ	0001	2026-09-16	10818.00	SGD	767-200
100	SQ	0001	2025-11-20	5950.00	SGD	A340-600
100	SQ	0002	2026-09-17	11765.00	SGD	747-400
100	SQ	0002	2025-11-21	10953.00	SGD	747-400
100	SQ	0011	2026-09-17	2359.00	SGD	767-200
100	SQ	0011	2025-11-21	4880.00	SGD	A340-600
100	SQ	0012	2026-09-19	4665.00	SGD	767-200
100	SQ	0012	2025-11-23	2574.00	SGD	747-400
100	UA	0058	2026-09-14	6629.00	USD	767-200
100	UA	0058	2025-11-18	4996.00	USD	747-400
100	UA	0059	2026-09-15	4131.00	USD	A340-600
100	UA	0059	2025-11-19	6053.00	USD	A340-600
100	UA	1537	2026-09-18	893.00	USD	A321-200
100	UA	1537	2025-11-22	805.00	USD	737-800
100	AA	0322	2026-09-20	1103.00	USD	A320-200
100	AA	0322	2025-11-24	1611.00	USD	A320-200
100	AA	0317	2026-09-16	462.00	USD	A321-200

Abbildung 4.11: Datenvorschau für eine Datenbanktabelle

4.4.2 Datenbanktabellen anlegen

Nun legen wir eine neue Datenbanktabelle an: Gehen Sie hierzu im Menü auf **File • New • ABAP Repository Object**, und wählen Sie im sich öffnenden Pop-up-Fenster aus Abbildung 4.5 im Ordner **Dictionary** den Eintrag **Database Table** aus. Zusätzlich können Sie im Eingabefeld darüber bei **type filter text** den Begriff »Database Table« (oder einen Teil davon) eingeben, damit die Ergebnisliste danach gefiltert wird. Bestätigen Sie anschließend mit einem Klick auf **Next >**.

Geben Sie im nächsten Schritt, wie Sie es bereits von den Domänen und den Datenelementen kennen, im Feld **Package** Ihr Paket aus Abschnitt 2.2.1, »Paket anlegen«, ein. Vergeben Sie im Feld **Name** einen Namen (hier »ZFR_EMPLOYEES«), gegebenenfalls mit Namenskürzel, und im Feld **Description** eine Beschreibung (hier »Mitarbeitername«). Klicken Sie anschließend auf **Next >**. Dadurch öffnet sich der Texteditor, in dem der grundlegende Quellcode der Datenbanktabelle dargestellt wird.

Wie in Listing 4.2 dargestellt, werden die Felder der Tabelle innerhalb der geschweiften Klammern `{ }` definiert. Jedes Feld erhält einen Namen und wird über einen Doppelpunkt `(:)` einem Datenelement oder einem eingebaute Datentypen zugeordnet.

Wenn Ihre Tabelle mandantenabhängig sein soll (das ist in der Regel der Fall), müssen Sie als erstes Feld das Schlüsselfeld `CLIENT` vom Typ `abap.clnt` hinzufügen. Dieses Feld wird dann automatisch bei jeder ABAP-SQL-Anweisung mit dem aktuellen Mandanten gefüllt bzw. beim lesenden Zugriff entsprechend verarbeitet.



Mandantenabhängigkeit

Mandantenabhängigkeit bedeutet, dass Daten für verschiedene Systeme auf einer Datenbanktabelle liegen können, die sich nur anhand der Mandantennummer voneinander unterscheiden. Für den fiktiven Mandanten 100 könnten 50 Mitarbeitende existieren, für einen anderen Mandanten 10. Je nachdem, auf welchem Mandanten sich die Anwenderinnen und Anwender einloggen, können Sie nur den jeweiligen Datenbereich sehen. Nur wenn Sie wollen, dass beide (oder mehr) Mandanten sich die Daten teilen, können Sie diese Spalte weglassen.


Wählen Sie als Tabellenkategorie hier immer `TRANSPARENT` aus. Die weiteren möglichen Annotationen sind im weiteren Verlauf dieses Abschnitts beschrieben.

Ergänzen Sie Ihren Quellcode mit den Feldern aus Listing 4.2.

```
@EndUserText.label : 'Mitarbeiter'
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #C
```

```
@AbapCatalog.dataMaintenance : #RESTRICTED
define table zfr_employees {
  key client : abap.clnt not null;
  key emp_id : abap.numc(5);
  emp_name   : ze_emp_name;
  mgr_id     : abap.numc(5);
}
```

Listing 4.2: Definition einer eigenen Datenbanktabelle

Nach der Definition können Sie Ihre Datenbanktabelle erneut über das Icon  (**Activate**) aktivieren. Diese Tabelle werden wir zu einem späteren Zeitpunkt, in Abschnitt 7.2, »Ändernde ABAP-SQL-Anweisungen«, mit Inhalt befüllen.

Erweiterungskategorie

Sie haben die Möglichkeit, in der Tabellendefinition eine *Erweiterungskategorie* anzugeben. Diese bestimmt, ob und wie die Datenbanktabelle erweitert werden darf, und wird am Kopf mit der Annotation `@AbapCatalog.enhancement.category` angegeben. Die folgenden Optionen stehen Ihnen zur Verfügung:

- `#NOT_EXTENSIBLE`: Tabelle ist nicht erweiterbar.
- `#EXTENSIBLE_CHARACTER`: Die Struktur darf mit zeichenartigen flachen Komponenten (c, n, d oder t) erweitert werden.
- `#EXTENSIBLE_CHARACTER_NUMERIC`: Die Tabelle ist erweiterbar und zeichenartig oder numerisch: Die Struktur darf erweitert werden, die Erweiterung darf aber keine tiefen Datentypen enthalten.
- `#EXTENSIBLE_ANY`: Die Struktur darf mit beliebigen Komponenten erweitert werden.

Für eigene Entwicklungen spielt dies meist keine Rolle, da sie üblicherweise nicht an Drittanbieter ausgeliefert werden. Wichtig wird dies vor allem bei den Standard-Datenbanktabellen von SAP, da diese Annotation darüber entscheidet, ob Sie neue Z-Felder über eine sogenannte Append-Struktur hinzufügen können.

Auslieferungsklasse

Ein weiterer Teil der Tabellendefinition ist die *Auslieferungsklasse*. Sie wird am Kopf mit der Annotation `@AbapCatalog.deliveryClass` angegeben und bestimmt das Verhalten der Datenbanktabelle bei der Installation, beim Upgrade, bei einer Mandantenkopie und beim Transport zwischen Kundensystemen. Für Sie sind die Werte `A` für Anwendungstabelle (Stamm- und Bewegungsdaten) als auch `C` für Customer Table (dt. Kundentabelle) am wichtigsten. Eine Kundentabelle bedeutet in diesem Fall, dass die Daten ausschließlich von uns als Kunden gepflegt werden.

Datenpflege

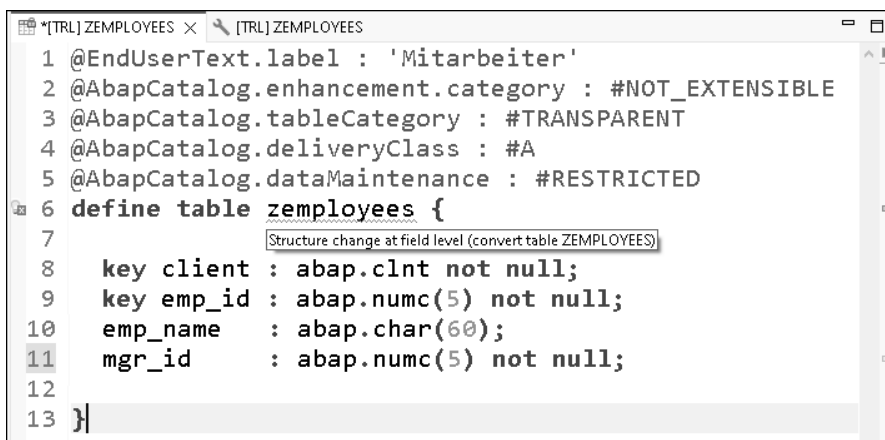
Die Berechtigung zur *Datenpflege* wird am Kopf mit der Annotation `@AbapCatalog.dataMaintenance` angegeben und bestimmt, wie auf die Datenbanktabelle über Pflegedialoge oder in der Datenvorschau zugegriffen werden kann.

- `#DISPLAY`: Datenbanktabelle kann nur angezeigt werden.
- `#ALLOWED`: Datenbanktabelle kann in der Datenanzeige angezeigt und bearbeitet werden. Pflegedialoge können erstellt werden.
- `#NOT_ALLOWED`: Datenbanktabelle kann in der Datenanzeige angezeigt und bearbeitet werden. Es können keine Pflegedialoge für die Datenbanktabelle erstellt werden.
- `#RESTRICTED`: Datenbanktabelle kann in der Datenanzeige angezeigt, aber nicht bearbeitet werden.

4.4.3 Datenbanktabelle umsetzen

Eine Datenbanktabelle müssen Sie immer dann umsetzen, wenn die Datenbanktabelle bereits Daten beinhaltet und Sie Änderungen auf Feldebene vornehmen, durch die sich diese Daten ändern würden. »Umsetzen« bedeutet in diesem Kontext, dass die Daten der Datenbanktabelle vom alten Format in das neue Format umgewandelt, also »transformiert«, werden. Das Ziel dieser Umsetzung ist immer, die Daten zu erhalten.

Eine solche Situation ist in Abbildung 4.12 dargestellt. Die in Abschnitt 4.4.2, »Datenbanktabellen anlegen«, angelegte Datenbanktabelle wurde dahingehend geändert, dass das Feld `EMP_NAME` von 60 auf 50 Zeichen verkürzt wurde. Dazu wurde das vorher angelegte Datenelement mit der direkten Angabe des eingebauten Typs `char` ersetzt. Das Aktivieren schlägt daher fehl, und es wird die Fehlermeldung **Structure change at field level (convert table ...)** angezeigt.



```

1 @EndUserText.label : 'Mitarbeiter'
2 @AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
3 @AbapCatalog.tableCategory : #TRANSPARENT
4 @AbapCatalog.deliveryClass : #A
5 @AbapCatalog.dataMaintenance : #RESTRICTED
6 define table zemployees {
7     Structure change at field level (convert table ZEMPLOYEES)
8     key client : abap.clnt not null;
9     key emp_id : abap.numc(5) not null;
10    emp_name   : abap.char(60);
11    mgr_id    : abap.numc(5) not null;
12
13 }

```

Abbildung 4.12: Strukturänderung einer Tabelle auf Feldebene

Um diesen Fehler zu beheben, rufen Sie die Quick-Fix-Funktion auf, indem Sie auf den Namen der Datenbanktabelle in Zeile 6 klicken und die Tastenkombination `Strg` + `1` drücken. Dadurch wird Ihnen das Menü aus Abbildung 4.13 angezeigt.

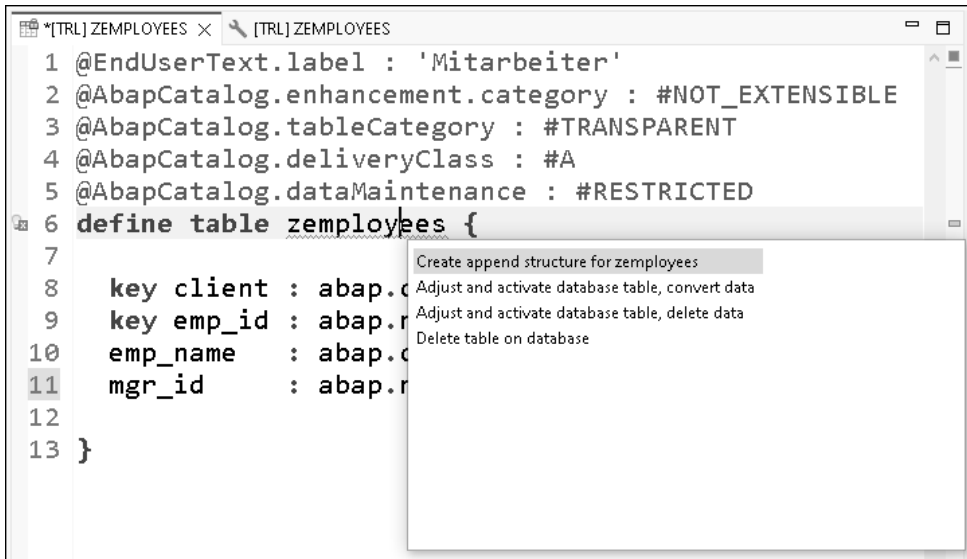


Abbildung 4.13: Tabelle über die Quick-Fix-Funktion umsetzen

Zum Umsetzen der Datenbanktabelle können Sie nun zwischen den beiden folgenden Funktionen wählen:

- **Adjust and activate database table, convert data:** Ändert die Datenbanktabelle auf der Datenbank und konvertiert die Daten in die neue Form.
- **Adjust and activate database table, delete data:** Ändert die Datenbanktabelle auf der Datenbank und löscht alle Daten.

Das Löschen der Daten ist immer dann notwendig, wenn die Daten nicht erhalten werden können. Dies ist beispielsweise der Fall, wenn Sie ein Feld verkürzen. Änderungen sind also mit Vorsicht zu genießen, weil diese direkte Auswirkungen auf gespeicherte Produktivdaten haben können.

Wenn die Verkürzung des Mitarbeiternamens stattdessen auf Domänenebene stattfindet, würde das Aktivieren der Domäne fehlschlagen, da dies indirekt zu Datenverlust in darauf aufbauenden Datenbanktabellen führen würde. Abbildung 4.14 stellt diese Situation dar.

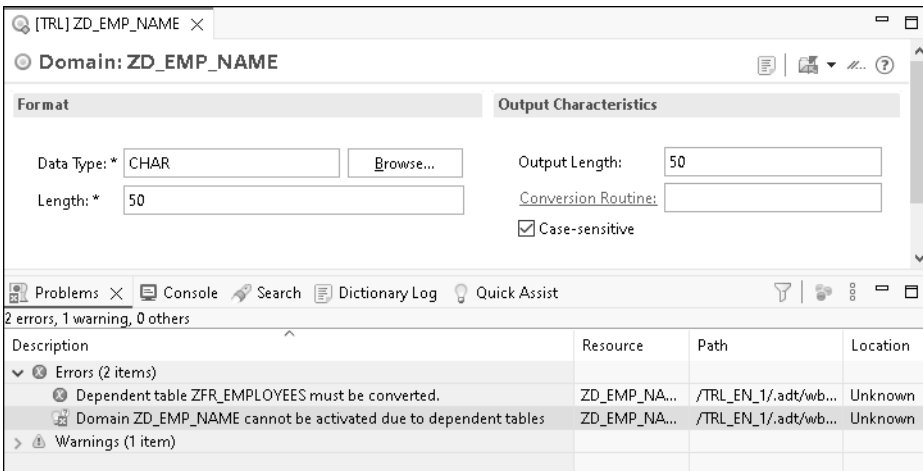


Abbildung 4.14: Fehlermeldung beim Aktivieren der geänderten Domäne

Markieren Sie in diesem Fall die Fehlermeldung in der Ansicht **Problems** und rufen Sie hier ebenfalls die Quick-Fix-Funktion über die Tastenkombination **Strg** + **1** auf. Dadurch öffnet sich das Pop-up-Fenster aus Abbildung 4.15.

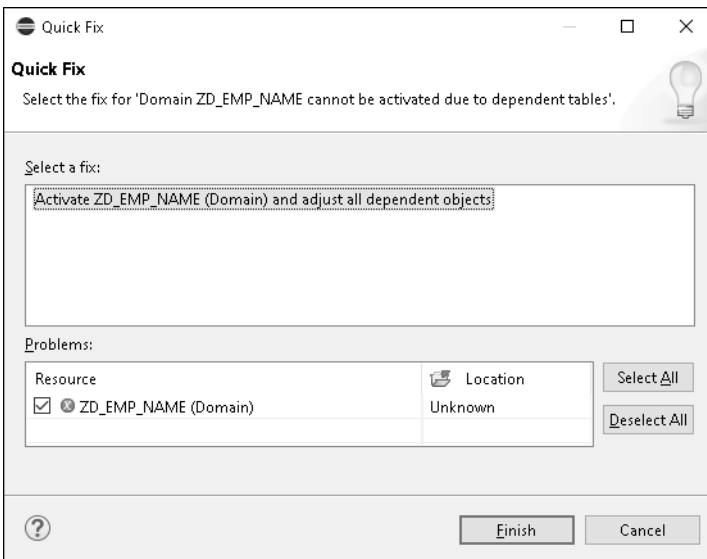


Abbildung 4.15: Quick Fix bei einer indirekten Änderung

Selektieren Sie hier die die Lösung **Activate ... (Domain) and adjust all dependent objects** und bestätigen Sie Ihre Auswahl mit **Finish**.

4.5 Strukturen

In Abschnitt 2.7.5, »Strukturen«, hatte ich Ihnen bereits erläutert, was Strukturen sind und wie Sie eine Variable vom Typ einer Struktur definieren können. Hier möchte ich Ihnen zeigen, wie Sie selbst eine solche Feldleiste als Typ im ABAP Dictionary definieren können. Nach der Definition steht ein solcher Strukturtyp global im System zur Verfügung und kann aus jedem ABAP-Quellcode heraus zur Definition einer Struktur referenziert werden.

4.5.1 Struktur anlegen

Um eine neue Struktur anzulegen, gehen Sie im Menü auf **File • New • ABAP Repository Object** und wählen im sich öffnenden Pop-up-Fenster aus Abbildung 4.5 im Ordner **Dictionary** den Eintrag **Structure** aus. Zusätzlich können Sie im Eingabefeld darüber bei **type filter text** den Begriff »Structure« eingeben, damit die Ergebnisliste danach gefiltert wird. Klicken Sie anschließend auf **Next >**.

Folgen Sie auch hier dem bekannten Schema und geben Sie im Feld **Package** Ihr Paket aus Abschnitt 2.2.1, »Paket anlegen«, ein und vergeben Sie zusätzlich im Feld **Name** einen Namen (hier »ZST_FR_EMPLOYEES«), gegebenenfalls mit Namenskürzel, und im Feld **Description** eine Beschreibung (hier »Mitarbeiter«). Klicken Sie anschließend auf **Next >**. Dadurch öffnet sich der Texteditor, in dem der grundlegende Quellcode der Struktur dargestellt wird.

Analog zur Definition von Datenbanktabellen werden die Felder innerhalb der geschweiften Klammern {} angegeben (siehe Listing 4.4). Jedes Feld erhält einen Namen und wird über einen Doppelpunkt (:) einem Datentyp oder Datenelement zugeordnet.

```
@EndUserText.label : 'Mitarbeiter'
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
define structure zst_fr_employees {
    emp_id      : abap.numc(5);
    emp_name    : ze_emp_name;
    mgr_id      : abap.numc(5);
}
```

Listing 4.3: Definition einer eigenen Struktur

Die Annotation `@AbapCatalog.enhancement.category` steuert, wie bei den Datenbanktabellen, die Erweiterbarkeit der Struktur. Die möglichen Ausprägungen habe ich bereits in Abschnitt 4.4.2, »Datenbanktabellen anlegen«, beschrieben.

Nach dem Aktivieren können Sie diese Strukturdefinition mit ABAP verwenden, in dem Sie sich beispielsweise eine Struktur damit definieren:

```
DATA: ls_mitarbeiter TYPE zst_employees.
```

Dies wäre natürlich auch mit der in Abschnitt 4.4.2 beschriebenen Datenbanktabelle möglich gewesen. Manchmal möchten Sie jedoch noch zusätzliche Spalten zur Struktur hinzufügen, die in der Datenbank nicht vorhanden sind. Dazu gehören insbesondere, aber nicht ausschließlich, Textfelder, die die technischen Begriffe lesbarer für Anwenderinnen und Anwender machen. So könnte beispielsweise zu einer Spalte mit Materialnummer eine zweite Spalte hinzugefügt werden, die den Text zum Material in der angemeldeten Sprache anzeigt.

4.5.2 Verwendung von Währungsfeldern

Wenn Sie in Ihrer Struktur *Währungsfelder* verwenden, werden Sie beim Aktivieren die folgende Fehlermeldung erhalten: **Annotation with reference to currency code for field PRICE is missing.**

Jedes Währungsfeld muss mit einem Einheitenfeld verknüpft sein. Diese Information wird bei der Anzeige benötigt, um die Währung entsprechend formatieren zu können. Die japanische Währung Yen, die beispielsweise keine Nachkommastellen hat, muss anders dargestellt werden als die europäische Währung Euro.

In Listing 4.4 sehen Sie eine definierte Struktur, in der die beiden Felder zu Betrag und Währung korrekt miteinander verknüpft wurden. Die zusätzliche Annotation `@Semantics.amount.currencyCode`, die sich oberhalb des Betragsfelds befindet, stellt die Beziehung zwischen den Feldern her. Dadurch kann das System Beträge korrekt formatieren und anzeigen.

```
@EndUserText.label : 'Beispielstruktur für Währung'  
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE  
define structure zst_curr {  
    @Semantics.amount.currencyCode: 'zst_curr.currency_code'  
    price           : /dmo/flight_price;  
    currency_code   : /dmo/currency_code;  
}
```

Listing 4.4: Definition einer Struktur mit einem Währungsfeld

4.6 Checkliste

Die Checkliste in Tabelle 4.1 prüft, ob Sie die Grundlagen des ABAP Dictionary und die Anlage der wichtigsten Dictionary-Objekte verstanden haben, und dient Ihrer Selbstkontrolle.


Ich habe...	Abschnitt
... verstanden, dass das ABAP Dictionary die zentrale Stelle für alle Datendefinitionen im SAP-System ist und für zentrale Datenbeschreibung, Konsistenz, Wiederverwendbarkeit und automatische Datenbankanpassung sorgt.	Einleitung
... verstanden, dass das ABAP Dictionary die Schnittstelle zur SAP-HANA-Datenbank bildet und ABAP-SQL-Zugriffe wie SELECT auf Datenbanktabellen ermöglicht.	Einleitung
... das zweistufige Domänenprinzip mit technischen Domänen (Domänen) und semantischen Domänen (Datenelementen) verstanden.	Abschnitt 4.1
... nachvollzogen, wie ein Feld einer Tabelle oder Struktur auf ein Datenelement und dieses wiederum auf eine Domäne verweist.	Abschnitt 4.1
... eine eigene Domäne mit Datentyp, Länge und ggf. weiteren Eigenschaften (z. B. Konvertierungsroutine, Festwerte, Wertetabelle) angelegt und aktiviert.	Abschnitt 4.2
... verstanden, wozu Konvertierungsroutinen dienen und wie sie zwischen interner und externer Darstellung von Daten umwandeln.	Abschnitt 4.2
... ein Datenelement angelegt, einer Domäne zugeordnet und mit Field Labels sowie ggf. zusätzlichen Eigenschaften (z. B. Suchhilfe) versehen.	Abschnitt 4.3
... verstanden, wann es sinnvoll ist, eine Domäne zu verwenden, statt direkt einen eingebauten Datentyp im Datenelement zu hinterlegen.	Abschnitt 4.3
... bestehende Datenbanktabellen über das Icon  (Open ABAP Development Object) mit dem Filter »type:dtab« gefunden und geöffnet.	Abschnitt 4.4.1
... die Datenvorschau für eine Datenbanktabelle aufgerufen und dort Filter sowie die SQL Console verwendet.	Abschnitt 4.4.1
... eine eigene Datenbanktabelle mit Schlüsselfeldern, weiteren Feldern und dem Mandantenfeld CLIENT (abap.clnt) angelegt.	Abschnitt 4.4.2
... verstanden, was Mandantenabhängigkeit bedeutet und wann eine Tabelle mandantenabhängig sein sollte.	Abschnitt 4.4.2
... die wichtigsten Annotationen einer Tabellendefinition (Erweiterungskategorie, Tabellenkategorie, Auslieferungsklasse, Datenpflege) kennengelernt und sinnvoll gesetzt.	Abschnitt 4.4.2

Tabelle 4.1: Checkliste zum ABAP Dictionary

Ich habe...	Abschnitt
... gelernt, wann eine Datenbanktabelle umgesetzt werden muss und welche Auswirkungen Strukturänderungen auf vorhandene Daten haben.	Abschnitt 4.4.3
... die Quick-Fix-Funktion (<code>STRQ</code> + <code>1</code>) zum Umsetzen einer Datenbanktabelle bzw. zum Aktivieren abhängiger Objekte bei Domänenänderungen angewendet.	Abschnitt 4.4.3
... den Unterschied zwischen den Optionen Adjust and activate database table, convert data und Adjust and activate database table, delete data verstanden.	Abschnitt 4.4.3
... eine eigene Struktur als Dictionary-Objekt angelegt und mit ABAP über die Anweisung <code>DATA ... TYPE</code> referenziert.	Abschnitt 4.5.1
... verstanden, warum es sinnvoll sein kann, eine Struktur statt einer Datenbanktabelle als Typ zu verwenden, etwa um zusätzliche (nicht persistierte) Felder zu ergänzen.	Abschnitt 4.5.1
... gelernt, dass Währungsfelder über die Annotation <code>@Semantics.amount.currencyCode</code> mit einem Einheitenfeld verknüpft werden müssen, damit sie korrekt formatiert werden können.	Abschnitt 4.5.2

Tabelle 4.1: Checkliste zum ABAP Dictionary (Forts.)

Kapitel 5

ABAP-Grundbefehle

Dieses Kapitel bietet Ihnen eine kompakte und praxisorientierte Einführung in die wichtigsten ABAP-Grundbefehle. Sie lernen die zentralen Sprachelemente kennen, die Sie in nahezu jedem ABAP-Programm benötigen.

In Kapitel 2, »Erste Schritte«, habe ich Ihnen neben einfachen Syntaxregeln und der Möglichkeit des Kommentierens von Quellcode bereits gezeigt, wie Sie einfache Variablen mit der Anweisung `DATA` definieren, befüllen und wieder initialisieren können. Ziel dieses Kapitels ist es, Ihnen einen Überblick über die ABAP-Grundbefehle zu geben, also die grundlegenden Bestandteile der Syntax, die Sie vielleicht schon aus anderen Programmiersprachen kennen:

- Steueranweisungen (siehe Abschnitt 5.1)
- Arbeiten mit Zeichenketten (siehe Abschnitt 5.2)
- Rechenoperationen (siehe Abschnitt 5.3)
- Typdefinitionen (siehe Abschnitt 5.4)

Dieses Kapitel ist so konzipiert, dass Neulinge schrittweise in die ABAP-Welt eingeführt werden. Es erhebt keinen Anspruch auf Vollständigkeit. Stattdessen konzentriere ich mich bewusst auf eine Auswahl der wichtigsten und am häufigsten verwendeten Anweisungen. Wenn Sie diese sicher beherrschen, können Sie den Großteil der Problemstellungen des ABAP-Alltags bestreiten und sind bestens auf die darauf aufbauenden Funktionen vorbereitet, die in den weiteren Kapiteln des Buchs behandelt werden.

Ich erläutere jede ABAP-Anweisung anhand eines kurzen Beispiels, das sich auf die wesentliche Funktion der Anweisung konzentriert. Sie sollten so ein Verständnis dafür gewinnen, was der Nutzen einer Anweisung ist und wie Sie diese verwenden.

5.1 Steueranweisungen

ABAP stellt Ihnen Steueranweisungen zur Verfügung, um *Verzweigungen* und *Schleifen* in Ihren Programmen zu realisieren. Die wichtigsten Steueranweisungen in ABAP sind:

- bedingte Anweisungen mit `IF` (siehe Abschnitt 5.1.1)
- verwendbare logische Ausdrücke (siehe Abschnitt 5.1.2)

- bedingte Verzweigungen mit `CASE` (siehe Abschnitt 5.1.3)
- nicht bedingte Schleifenverarbeitung mit `DO` (siehe Abschnitt 5.1.5)
- bedingte Schleifenverarbeitung mit `WHILE` (siehe Abschnitt 5.1.6)
- bedingte Durchführung von Schleifen mit `CHECK` (siehe Abschnitt 5.1.7)
- Verlassen des aktuellen Anweisungsblocks mit `EXIT` (siehe Abschnitt 5.1.8)
- Überspringen von Durchläufen mit `CONTINUE` (siehe Abschnitt 5.1.9)

Innerhalb dieser Konstrukte können Sie beliebige weitere Anweisungen verschachteln, beispielsweise Methodenaufrufe, Schleifen oder weitere Bedingungen. Achten Sie dabei stets darauf, dass alle Blöcke korrekt abgeschlossen werden, und ebenso darauf, dass die verwendeten Bedingungen und Schleifen innerhalb eines Ereignisses abgeschlossen werden.

5.1.1 IF-Abfragen

Mit der `IF`-Abfrage steuern Sie den Programmablauf abhängig von einer Bedingung. In der einfachsten Form der Abfrage einer Bedingung führt das System alle Anweisungen `anweisungen1` zwischen den Schlüsselwörtern `IF` und `ENDIF` durch, sobald die Bedingung `bedingung1` der `IF`-Anweisung erfüllt ist. Dabei kann der Anweisungsteil wiederum neue Abfragen oder Schleifen enthalten.

Verwenden Sie `ELSE` in der `IF`-Anweisung, führt das System die Anweisungen `anweisungen2` nach `ELSE` aus, wenn die Bedingung `bedingung1` der `IF`-Anweisung nicht erfüllt ist.

Verwenden Sie `ELSEIF` in der `IF`-Anweisung, wird eine neue Bedingung `bedingung2` abgefragt, wenn die Bedingung `bedingung1` der ersten `IF`-Anweisung nicht erfüllt ist.

Die wichtigsten möglichen logischen Ausdrücke, die Sie hierfür verwenden können, finden Sie in Abschnitt 5.1.2, »Logische Ausdrücke«.

Syntax

In Listing 5.1 sehen Sie die Syntax einer `IF`-Anweisung.

```
IF bedingung1.  
    [anweisungen1]  
[ELSEIF bedingung2.  
    [anweisungen2]]  
...  
[ELSE.  
    [anweisungen3]]  
ENDIF.
```

Listing 5.1: Syntax der IF-Anweisung

Beispiele

Die folgenden Beispiele zeigen typische Einsatzszenarien der IF-Anweisung in unterschiedlichen Ausprägungen. Eine einfache Bedingung sieht wie folgt aus:

```
IF sy-subrc <> 0.
  "Variable ist ungleich 0
ENDIF.
```

Eine Bedingung mit einer Alternative sehen Sie in Listing 5.2:

```
IF sy-subrc <> 0.
  "Variable ist ungleich 0
ELSE.
  "Die obere Bedingung wurde nicht erfüllt
ENDIF.
```

Listing 5.2: IF-Anweisung mit einem ELSE-Zweig

Eine komplexe Bedingung mit Alternative könnte hingegen aussehen wie in Listing 5.3.

```
IF sy-subrc <> 0.
  "Variable ist ungleich 0
ELSEIF sy-abcnt = 1.
  "Variable ist gleich 1
ELSE.
  "Beide oberen Bedingungen wurden nicht erfüllt
ENDIF.
```

Listing 5.3: Komplexe IF-Bedingung mit mehreren Prüfungen

5.1.2 Logische Ausdrücke

Logische Ausdrücke werden in Vergleichen z. B. zur Fallunterscheidung gebraucht. ABAP stellt Ihnen die in Tabelle 5.1 gezeigten logischen Ausdrücke zur Verfügung. Aus historischen Gründen gibt es für einen logischen Ausdruck jeweils mehrere Schreibweisen.

Bedeutung	Mögliche Schreibweisen
gleich	A = B . oder A EQ B .
größer als	A > B . oder A GT B .
kleiner als	A < B . oder A LT B .

Tabelle 5.1: Logische Ausdrücke

Bedeutung	Mögliche Schreibweisen
ungleich	A <> B., A >< B., oder A NE B.
größer gleich	A >= B. oder A GE B.
kleiner gleich	A <= B. oder A LE B.
zwischen	A BETWEEN B AND C BT (für Ranges-Tabellen)
initiale Belegung	A IS INITIAL

Tabelle 5.1: Logische Ausdrücke (Forts.)

Bedingungen können mit Klammern gruppiert werden und über die Anweisungen NOT (nicht), OR (oder) und AND (und) verknüpft werden. Die Auswertung erfolgt immer von links nach rechts. Sobald das Ergebnis gefunden wurde, wird die nächste Anweisung verarbeitet.

5.1.3 CASE-Anweisungen

Die CASE-Anweisung nutzen Sie, um abhängig vom Inhalt eines bestimmten Datenfelds verschiedene Anweisungsblöcke zu verarbeiten. Dabei vergleicht das System den Wert der CASE-Anweisung `operand` mit dem Wert der WHEN-Anweisungen (`operand1`, `operand2` ... `operandn`). Sobald eine Übereinstimmung gefunden wird, führt das System den zugehörigen Anweisungsteil der WHEN-Anweisung aus und setzt die Verarbeitung nach ENDCASE fort. Wird kein passender Wert gefunden, wird, sofern vorhanden, die optionale Anweisung WHEN OTHERS durchlaufen.

Die Anweisung CASE hat gegenüber der IF-Anweisung einen Performancevorteil, kann aber immer nur den Wert *eines* Felds abfragen.

Syntax

In Listing 5.4 sehen Sie die Syntax der CASE-Anweisung.

```
CASE operand.
  [WHEN operand1 [OR operand2 [OR operandn [...]]].
    [anweisungen1]]
  ...
  [WHEN OTHERS.
    [anweisungen0]]
ENDCASE.
```

Listing 5.4: Syntax der CASE-Anweisung

Beispiel

Listing 5.5 zeigt die Definition einer CASE-Anwendung.

```
CASE sy-subrc.
  WHEN 0.
    "Satz gefunden
  WHEN 4.
    "Satz nicht gefunden
  WHEN OTHERS.
    "Unbekannter Fehler
ENDCASE.
```

Listing 5.5: CASE-Anweisung mit mehreren Prüfungen

5.1.4 Weitere Fallunterscheidungen

Neben den klassischen Anweisungen IF und CASE stellt ABAP mit den Konstruktorausdrücken COND und SWITCH zwei weitere Anweisungen zur Verfügung, mit denen Sie Fallunterscheidungen abbilden können. Ein wesentlicher Vorteil dieser Ausdrücke besteht darin, dass Sie direkt an lesenden Operandenpositionen angegeben werden können und nicht in eine separate Zeile geschrieben werden müssen.

Operandenpositionen

Eine ABAP-Anweisung besteht aus einem Operator (z. B. = oder +) und Operanden (Variablen oder Konstanten). Die Position des Operanden bestimmt, ob dieser geschrieben oder gelesen wird. Am Beispiel vom Operator = wird der Operand davor geschrieben und danach gelesen.



Durch die Verwendung solcher Ausdrücke an lesenden Operandenpositionen wird der Quellcode kürzer, sodass Zuweisungen und gegebenenfalls auch die Adressierung von Speicherbereichen eingespart werden.

Mit der Anweisung COND können Sie eine IF-Abfrage an lesenden Operandenpositionen durchführen. Für einfache Fallunterscheidungen zur Überprüfung des Werts einer Variablen steht Ihnen die Anweisung SWITCH zur Verfügung.

COND: Bedingte Ausdrücke

Während bei einer klassischen IF-Anweisung nur eine Fallunterscheidung stattfindet, erzeugt die Anweisung COND als Ergebnis einen Wert mit dem angegebenen Datentyp datentyp. Das heißt, es wird immer ein Wert zurückgegeben. Ist der Typ vollständig erkennbar, kann auch die Raute # anstelle des Typs angegeben werden.

Inhalt

Einleitung	17
1 Technische Vorbereitungen für ein Testsystem	23
1.1 Die ABAP Development Tools herunterladen und installieren	23
1.1.1 Installation von Eclipse	24
1.1.2 ABAP Development Tools installieren	26
1.2 Einen SAP-BTP-Testaccount erstellen	31
1.2.1 Eine ABAP-Umgebung erstellen	33
1.2.2 Einen Service Key generieren	36
1.3 Checkliste	38
2 Erste Schritte	41
2.1 Eine Systemverbindung herstellen	41
2.1.1 Ein ABAP-Cloud-Projekt anlegen (Cloud-Systeme)	43
2.1.2 Ein ABAP-Projekt anlegen (On-Premise-Systeme)	46
2.2 Eine Hello-World-Anwendung erstellen	48
2.2.1 Paket anlegen	48
2.2.2 Rumpf-Klasse anlegen	51
2.2.3 Ausführen der Rumpf-Klasse	56
2.3 Entwicklungsobjekte öffnen	57
2.3.1 Über das Paket unter »Favorite Packages«	57
2.3.2 Über die Funktion »Open ABAP Development Object«	58
2.4 Mit Kommentaren arbeiten	59
2.5 SAP-Hilfe nutzen	60
2.6 Syntaxregeln von ABAP	62
2.7 Deklaration von Variablen	63
2.7.1 Felder	63
2.7.2 Konstanten	67
2.7.3 Feldsymbole	67

2.7.4	Inline-Deklarationen	69
2.7.5	Strukturen	70
2.7.6	Unterschied zwischen TYPE und LIKE	70
2.8	Initialisierung von Variablen	71
2.8.1	Felder initialisieren	71
2.8.2	Speicherbereich freigeben	72
2.9	Checkliste	72
3	Debugging	75
3.1	Debugger aufrufen und beenden	75
3.2	Variablen-Anzeige	78
3.3	Steuerung des Debuggers	79
3.4	Aufrufstack	80
3.5	Watchpoints	82
3.6	Checkliste	83
4	ABAP Dictionary	85
4.1	Das zweistufige Domänenprinzip	86
4.2	Domänen anlegen	90
4.3	Datenelemente anlegen	92
4.4	Datenbanktabellen	93
4.4.1	Datenbanktabellen anzeigen	94
4.4.2	Datenbanktabellen anlegen	96
4.4.3	Datenbanktabelle umsetzen	98
4.5	Strukturen	101
4.5.1	Struktur anlegen	101
4.5.2	Verwendung von Währungsfeldern	102
4.6	Checkliste	102

5	ABAP-Grundbefehle	105
5.1	Steueranweisungen	105
5.1.1	IF-Abfragen	106
5.1.2	Logische Ausdrücke	107
5.1.3	CASE-Anweisungen	108
5.1.4	Weitere Fallunterscheidungen	109
5.1.5	DO-Schleifen	112
5.1.6	WHILE-Schleifen	113
5.1.7	CHECK-Anweisungen	114
5.1.8	EXIT-Anweisungen	114
5.1.9	CONTINUE-Anweisungen	115
5.2	Mit Zeichenketten arbeiten	116
5.2.1	Vergleich von Zeichenketten	116
5.2.2	Verkettungsoperatoren	117
5.2.3	Teilfeldzugriffe	119
5.2.4	Teilzeichenketten finden	121
5.2.5	Teilzeichenketten ersetzen	123
5.2.6	Eingebaute Funktionen für Zeichenketten	125
5.2.7	Zeichenketten-Templates	126
5.3	Rechenoperationen	130
5.3.1	Mathematische Funktionen	131
5.3.2	Berechnungszuweisungen	132
5.3.3	Mit Datums- und Zeitwerten rechnen	133
5.4	Typdefinitionen	136
5.4.1	Felder	137
5.4.2	Strukturen	138
5.5	Checkliste	140
6	Mit internen Tabellen arbeiten	143
6.1	Tabellenarten	145
6.1.1	Standardtabellen	146
6.1.2	Sortierte Tabellen	146
6.1.3	Hash-Tabellen	146

6.2	Interne Tabellen definieren	147
6.3	Zeilen hinzufügen	149
6.3.1	Daten mit SELECT hinzufügen	149
6.3.2	Zeilen mit APPEND anhängen	150
6.3.3	Zeilen mit INSERT hinzufügen	151
6.3.4	Werte mit VALUE hinzufügen	153
6.3.5	Der Zusatz FOR	155
6.3.6	Der Zusatz LINES OF	157
6.4	Inhalt auslesen	158
6.4.1	Tabellen mit READ TABLE auslesen	159
6.4.2	Tabellenausdrücke	161
6.4.3	Tabellen mit LOOP AT auslesen	165
6.5	Inhalt ändern	167
6.5.1	Tabelle mit READ TABLE ändern	167
6.5.2	Tabelle mit MODIFY ändern	168
6.5.3	Tabelle mit LOOP AT ändern	169
6.5.4	Tabelle mit Tabellenausdrücken ändern	170
6.6	Einträge löschen	171
6.7	Interne Tabellen kopieren	171
6.7.1	Strukturgleiche interne Tabellen kopieren	172
6.7.2	Strukturfremde interne Tabellen kopieren	173
6.7.3	CORRESPONDING: Mapping von Strukturen und internen Tabellen ...	173
6.8	Interne Tabellen aufbereiten	176
6.8.1	Sortieren mit SORT	176
6.8.2	Zusammenfassung mit COLLECT	177
6.8.3	Reduzierungen mit REDUCE	178
6.8.4	Filterungen mit FILTER	179
6.9	Gruppieren mit dem Zusatz GROUP BY	180
6.9.1	Repräsentantenbindung	181
6.9.2	Gruppenschlüsselbindung	183
6.9.3	Zusatz WITHOUT MEMBERS	184
6.9.4	Zusätze GROUP SIZE und GROUP INDEX	184
6.10	Eingebaute Funktionen für interne Tabellen	185
6.10.1	Funktion »lines()«	185
6.10.2	Funktion »concat_lines_of()«	186

6.10.3	Funktion »line_index()«	186
6.10.4	Funktion »line_exists()«	187
6.11	Checkliste	187
7	Zugriff auf die Datenbank	191
7.1	ABAP-SQL-Anweisung SELECT	193
7.1.1	Einträge lesen	193
7.1.2	Gelesene Spalten einschränken	197
7.1.3	Zielspalten angeben	197
7.1.4	Inline-Deklaration	199
7.1.5	WHERE-Klauseln	201
7.1.6	Ranges-Tabellen	203
7.1.7	FOR ALL ENTRIES IN: Einschränkung durch interne Tabellen	204
7.1.8	Gruppierung und Sortierung der Ergebnisse	205
7.1.9	Host-Ausdrücke	207
7.1.10	OFFSET: Begrenzung der Ergebnismenge	209
7.1.11	JOIN: Verknüpfung	210
7.1.12	WITH: Allgemeine Tabellenausdrücke	215
7.1.13	UNION: Vereinigung	217
7.1.14	INTERSECT: Schnittmenge	218
7.1.15	EXCEPT: Ausschließen	218
7.1.16	Unterabfragen	219
7.1.17	SELECT auf interne Tabellen	221
7.2	Ändernde ABAP-SQL-Anweisungen	222
7.2.1	Sperrkonzept	223
7.2.2	Datenkonsistenz	225
7.2.3	INSERT: Einträge einfügen	226
7.2.4	MODIFY: Einfügen oder ändern	228
7.2.5	UPDATE: Einträge ändern	230
7.2.6	DELETE: Löschen von Einträgen	232
7.3	ABAP-SQL-Ausdrücke	234
7.3.1	CASE-Anweisungen	235
7.3.2	Verknüpfungen von Zeichenketten mit »&&«	236
7.3.3	Arithmetische Ausdrücke	236

7.3.4	Typumwandlungen mit CAST	237
7.3.5	Elementare Werte	238
7.4	ABAP-SQL-Funktionen	239
7.4.1	Aggregatfunktionen	239
7.4.2	Zeichenkettenfunktionen	242
7.4.3	Numerische Funktionen	243
7.4.4	Datums- und Zeitfunktionen	244
7.4.5	Coalesce-Funktion: Nullwerte ersetzen	246
7.4.6	UUID-Funktion: Erzeugung eindeutiger Schlüssel	247
7.5	Vier goldene Regeln für SAP HANA	247
7.6	Checkliste	249
8	Grundlagen der Objektorientierung	253
8.1	Einleitung: Klassen und Objekte	255
8.2	Grundaufbau einer ABAP-Klasse	258
8.3	Klassen instanziiieren	259
8.3.1	Instanziierung mit CREATE OBJECT	261
8.3.2	Instanziierung mit NEW	262
8.4	Klassen anlegen	262
8.4.1	Globale Klasse anlegen	262
8.4.2	Lokale Klasse anlegen	263
8.5	Statische Klassen	265
8.6	Datenkapselung	266
8.7	Datentypen und Attribute	270
8.7.1	Zugriff auf Attribute	271
8.7.2	Selbstreferenz	271
8.8	Methoden und Parameter	272
8.8.1	Methoden implementieren	274
8.8.2	Methoden aufrufen	275
8.8.3	Autovervollständigung	279
8.9	Konstruktoren	280
8.9.1	Instanzkonstruktor	281
8.9.2	Statischer Konstruktor	282

8.10	Ereignisse	283
8.10.1	Ereignisse definieren	285
8.10.2	Ereignisse auslösen	286
8.10.3	Ereignisbehandler definieren	286
8.10.4	Ereignisbehandler registrieren	286
8.10.5	Beispiel: Definition, Auslösen und Behandlung eines Ereignisses	287
8.11	Vererbung	289
8.11.1	Redefinition	291
8.11.2	Klassenhierarchien	295
8.12	Weitere Klassenarten	297
8.12.1	Abstrakte Klassen	297
8.12.2	Finale Klassen	297
8.12.3	Ausnahmeklassen	298
8.13	Freunde	298
8.14	Interfaces	300
8.14.1	Interface anlegen	303
8.14.2	Interface implementieren	304
8.14.3	Interface verwenden	305
8.15	Ausnahmen für Methoden	307
8.15.1	Klassenbasierte Ausnahmen	307
8.15.2	Lokale Ausnahmen	310
8.16	Casting	312
8.16.1	Casting mit dem Zuweisungsoperator	313
8.16.2	Casting mit dem Casting-Operator	313
8.16.3	Casting mit der Anweisung CAST	313
8.17	Checkliste	313
9	ABAP Core Data Services	315
9.1	CDS View anlegen	316
9.2	CDS Views mit Parametern	322
9.3	CDS-Annotationen	325
9.4	CDS-Sprachelemente	326
9.4.1	Klassische Sprachelemente	327

9.4.2	Vergleichsoperatoren	328
9.4.3	Aggregatfunktionen	328
9.4.4	Zeichenkettenfunktionen	328
9.4.5	Fallunterscheidungen	329
9.4.6	Numerische Funktionen	330
9.4.7	Datumsfunktionen	330
9.4.8	Typwandlung mit CAST	331
9.4.9	Umwandlung von Einheiten	332
9.4.10	Umwandlung von Währungen	332
9.5	CDS-Zugriffskontrollen	333
9.6	CDS-Assoziationen	335
9.7	CDS-Hierarchien	337
9.8	Checkliste	342
10	ABAP RESTful Application Programming Model	345
10.1	Grundlagen der RAP-Services	345
10.2	Eine RAP-Anwendung erstellen	352
10.2.1	Datenbanktabelle kopieren	352
10.2.2	Generator ausführen	354
10.2.3	Vorschau aufrufen	358
10.2.4	Oberfläche anpassen	361
10.2.5	Aktion ausprogrammieren	363
10.3	Checkliste	369
11	Zugriff auf Geschäftsobjekte mit der Entity Manipulation Language	371
11.1	Lesen	372
11.1.1	Lesen bestimmter Felder	375
11.1.2	Zusatz FROM	376
11.2	Anlegen	377
11.3	Aktualisieren	379
11.4	Löschen	380

11.5	Aktion aufrufen	382
11.6	Checkliste	383
Anhang		385
A	Das SAP-Flugdatenmodell	387
B	Übersicht der ABAP-Anweisungen	389
C	Eingebaute Datentypen	395
D	Wichtige Systemfelder	397
E	Namenskonventionen für die Programmierung	399
Der Autor		403
Index		405

CDS View (Forts.)	
<i>Datumsfunktionen</i>	330
<i>Einheiten</i>	332
<i>Hierarchien</i>	337
<i>klassische Sprachelemente</i>	327
<i>mit Parametern</i>	322
<i>numerische Funktionen</i>	330
<i>Projektion</i>	361
<i>Rolle</i>	333
<i>Typwandlung</i>	331
<i>Vergleichsoperatoren</i>	328
<i>Währungen</i>	332
<i>Zugriffskontrolle</i>	333
CDS-Annotation	325
CDS-Rolle	333
CHECK	114
CHECK (Anweisung)	167
CLASS	
<i>DEFINITION (Zusatz)</i>	259
<i>IMPLEMENTATION (Zusatz)</i>	259
<i>INHERITING FROM (Zusatz)</i>	290
<i>REDEFINITION (Zusatz)</i>	294
CLASS ... ENDCLASS (ABAP-Anweisung)	259
CLASS ... IMPLEMENTATION	
(ABAP-Anweisung)	274
CLEAR	71
CLEAR (ABAP-Anweisung)	71
Code-to-Data	191
COLLECT (Anweisung)	177
COMMIT ENTITIES	377, 379, 381-382
concat_lines_of() (eingebaute Funktion)	186
CONCATENATE (ABAP-Anweisung)	117
<i>INTO (Zusatz)</i>	118
<i>SEPARATED BY (Zusatz)</i>	118
COND (ABAP-Anweisung)	109
<i>LET (Zusatz)</i>	110
CONSTANTS	67
CONTINUE (ABAP-Anweisung)	115, 167
CORRESPONDING (ABAP-Anweisung)	173
<i>BASE (Zusatz)</i>	174
<i>DEEP (Zusatz)</i>	174
<i>EXCEPT (Zusatz)</i>	175
<i>MAPPING (Zusatz)</i>	175
CREATE OBJECT	
<i>EXPORTING (Zusatz)</i>	261
CREATE OBJECT (ABAP-Anweisung)	261
CREATE PUBLIC PROTECTED PRIVATE	
(Zusatz)	269
CRUD-Operation	350, 371
<i>Aktualisieren</i>	379
CRUD-Operation (Forts.)	
<i>Anlegen</i>	377
<i>Lesen</i>	372
<i>Löschen</i>	380
CURRENCY_CONVERSION	332
D	
DATA (ABAP-Anweisung)	64
<i>ALIAS (Zusatz)</i>	148
<i>COMPONENTS (Zusatz)</i>	148
<i>DEFAULT KEY (Zusatz)</i>	148
<i>LIKE LINE OF (Zusatz)</i>	148
<i>READ-ONLY (Zusatz)</i>	270
<i>TYPE (Zusatz)</i>	147
<i>TYPE HASHED TABLE (Zusatz)</i>	145, 147
<i>TYPE SORTED TABLE (Zusatz)</i>	145, 147
<i>TYPE STANDARD TABLE (Zusatz)</i>	147
<i>TYPE TABLE OF (Zusatz)</i>	147
<i>WITH NON-UNIQUE (Zusatz)</i>	148
<i>WITH UNIQUE (Zusatz)</i>	148
Data Definition Language	315
Datenbank	85
Datenbanktabelle	85, 87, 93, 378
<i>aktivieren</i>	353
<i>anlegen</i>	96, 352
<i>anzeigen</i>	93-94
<i>Erweiterungskategorie</i>	97
<i>kopieren</i>	352
<i>umsetzen</i>	98
Datenbankzugriff	191
Datenelement	86
<i>anlegen</i>	92
<i>Eigenschaften</i>	92
Datenkapselung	254, 266
Datenmanagement	346
Datenobjekt	63, 255
Datenpflege	98
Datentyp	257, 270
<i>definieren</i>	136
<i>eigener</i>	136
Debugger	
<i>aufrufen</i>	75
<i>beenden</i>	78
Debugger-Perspektive	76
Debugger-Steuerung	79
<i>Jump to line</i>	80
<i>Resume</i>	80
<i>Run to line</i>	80

Debugger-Steuerung (Forts.)		Feld	63
<i>Step into</i>	80	<i>initialisieren</i>	71
<i>Step over</i>	80	Feldleiste → Struktur	
<i>Step return</i>	80	Feldsymbol	67
Debugging	75	FIELDS (Zusatz)	375
Definitionsteil	258	FILTER (ABAP-Anweisung)	179
DELETE (ABAP-Anweisung)	171	<i>EXCEPT (Zusatz)</i>	179
Destruktor	281	<i>IN (Zusatz)</i>	179
Division	130	<i>USING KEY (Zusatz)</i>	179
DO (ABAP-Anweisung)	112	<i>WHERE (Zusatz)</i>	179
<i>TIMES (Zusatz)</i>	112	FIND (ABAP-Anweisung)	121
Domäne		<i>ALL OCCURRENCES (Zusatz)</i>	121
<i>anlegen</i>	90	<i>FIRST OCCURENCE (Zusatz)</i>	121
<i>Definition</i>	89	<i>IGNORING CASE (Zusatz)</i>	121
<i>semantische</i>	86	<i>RESPECTING CASE (Zusatz)</i>	121
<i>technische</i>	86	find() (Suchfunktion)	122
Down Cast	313	FREE (ABAP-Anweisung)	71
		Freund	298–299
E		FRIENDS (ABAP-Anweisung)	299
		FROM (Zusatz)	376
Eclipse	24, 41	Funktion	
<i>installieren</i>	23	<i>eingebaute</i>	125, 185
<i>Workspace</i>	25	<i>mathematische</i>	131
eingebaute Funktion		<i>Zeichenketten</i>	125
<i>interne Tabellen</i>	185	G	
ELSE	106		
ELSEIF	106	Ganzzahldivision	130
EML → Entity Manipulation Language		Garbage Collector	259
ENDLOOP (ABAP-Anweisung)	165	Generalisierung	289
ENDMETHOD (ABAP-Anweisung)	274	Get-Methode	268
Entity Manipulation Language (EML)	371	Gruppenschlüsselbindung	183
Ereignis	283		
<i>auslösen</i>	286	H	
<i>definieren</i>	285		
Ereignisbehandler	284–285	Hash-Tabelle	146, 150
<i>definieren</i>	286	Hello-World-Anwendung erstellen	48
<i>registrieren</i>	286		
Erweiterungskategorie	97	I	
EXCEPTIONS (Zusatz)	272		
EXIT (Anweisung)	114, 167	IF (ABAP-Anweisung)	106, 109
Expression Language	315	Implementierungsteil	258
		IMPORTING (Zusatz)	272
F		IN LOCAL MODE (Zusatz)	375
		Inline-Deklaration	69
Fallunterscheidung	107, 329		
Fehler	298		
Fehlertext	298		

INSERT (Anweisung)	151	Klasse (Forts.)	
INDEX (Zusatz)	151	Implementierungsteil	258
LINES OF (Zusatz)	152	instancieren	259
Instanz	256	Klassenergebnis	284
Instanzieren	256	Komponente	269
Interface	300	Komponentensichtbarkeit	269
IF_MESSAGE	309	Konstruktor	280, 282
implementieren	304	Oberklasse	290
Verwendung	305	statische	265
INTERFACE (ABAP-Anweisung)		Unterklasse	290
ABSTRACT METHODS (Zusatz)	304	Vererbung	290
ALL METHODS (Zusatz)	304	Klassenkomponentenselektor	271
DATA VALUES (Zusatz)	304	Klassifikation	255
FINAL METHODS (Zusatz)	304	Kommentar	59
INTERFACES (ABAP-Anweisung)	304	Konstante	63, 67
interne Tabelle	143	Konstruktor	280–281
ändern	167	Instanz	280–281
Arten	145	Klassen	280, 282
aufbereiten	176	Redefinition	292
auslesen	158	statischer	282
eingebaute Funktion	185	Konstruktorausdruck	262
füllen	149	Kontrollstruktur	377
kopieren	171		
löschen	171	L	
sortieren	176	LIKE	70
strukturfremde	172	line_exists() (eingebaute Funktion)	187
strukturgleiche	171	line_index() (eingebaute Funktion)	186
Summen bilden	177	LINES OF (Zusatz für REDUCE, NEW und VALUE)	157
Verkettung	186	lines() (eingebaute Funktion)	185
Zuweisung	173	Literal	127
ipow() (eingebaute Funktion)	156	logischer Ausdruck	107
Iteration, bedingte	155	LOOP AT (ABAP-Anweisung)	165
Iterationsausdruck	155	FROM (Zusatz)	165
Iterationsvariable	155	GROUP BY (Zusatz)	180
		GROUP INDEX (Zusatz)	184
K		GROUP SIZE (Zusatz)	184
Kindklasse → Unterklasse		TO (Zusatz)	165
Klasse	255, 347	WITHOUT MEMBERS (Zusatz)	183
abstrakte	297		
Art	297	M	
Attribut	266, 270	Mandantenabhängigkeit	96
Aufbau	258	MAPPED (Zusatz)	377
Ausnahmeklasse	298	Mapping-Vorschrift	175
Datentypen	270	mathematische Funktion	131
Definitionsteil	258		
finale	297		
Freund	299		
Hierarchie	295		

MESSAGE RAISING (ABAP-Anweisung)	311
Metadata Extension	361
METHOD (ABAP-Anweisung)	274
Methode	256
<i>Ausnahmen</i>	307
<i>definieren</i>	272
<i>implementieren</i>	274
<i>polymorphe</i>	291
<i>Redefinition</i>	294
<i>Verkettung</i>	277
METHODS	
<i>EXCEPTIONS (Zusatz)</i>	310
<i>FOR EVENT (Zusatz)</i>	286
<i>RAISING (Zusatz)</i>	272, 307
<i>RESUMABLE (Zusatz)</i>	307
<i>RETURNING (Zusatz)</i>	272
METHODS (ABAP-Anweisung)	272
Microsoft Visual C++ Runtime	24
Modifikator	268
MODIFY (ABAP-Anweisung)	168
<i>INDEX (Zusatz)</i>	168
<i>TRANSPORTING (Zusatz)</i>	168
<i>USING KEY (Zusatz)</i>	168
<i>WHERE (Zusatz)</i>	168
MODIFY ENTITIES OF (Anweisung)	379
MOVE-CORRESPONDING (Anweisung)	173
<i>KEEPING TARGET LINES (Zusatz)</i>	172
Multiplikation	130

N

Namenskonvention	399
NEW (ABAP-Anweisung)	262

O

Oberklasse	289, 294
Objekt	255
<i>Klassifikation</i>	255
Objektidentität	260
Objektorientierung	253
Objektreferenz	63
OData	347
OData-Service	347
Offset	119
On-Premise-System	24
Open SQL	191

P

PACKAGE SECTION (Sichtbarkeitsbereich)	270
Paging	209
Paket anlegen	48
Parameter	257
Performance	247
Polymorphie	254, 291
Potenzieren	130
Prädikatfunktion	187
PRIVATE SECTION (Sichtbarkeitsbereich)	270
Projektion	197
PROTECTED SECTION (Sichtbarkeitsbereich)	270
PUBLIC SECTION (Sichtbarkeitsbereich)	270

Q

Query Language	315
Query-Funktion	373
Quick-Fix-Funktion	99

R

RAISE (ABAP-Anweisung)	311
RAISE EVENT (ABAP-Anweisung)	286
Ranges-Tabelle	145
RAP-Geschäftsobjekt	371–372
READ ENTITIES (Anweisung)	372
READ TABLE (ABAP-Anweisung)	159, 167
<i>TRANSPORTING NO-FIELDS (Zusatz)</i>	161
Rechenausdruck	130
Rechenoperation	130
Redefinition	291–292
REDUCE (ABAP-Anweisung)	178
<i>INIT (Zusatz)</i>	178
<i>NEXT (Zusatz)</i>	178
Referenzvariable	260, 262
REPLACE (ABAP-Anweisung)	123
replace() (eingebaute Funktion)	124
Repräsentant	181
Repräsentantenbindung	181
Rest (Rechenoperation)	130
Return-Code	397
Robustheit	267
ROLLBACK ENTITIES (ABAP-Anweisung)	377
Rumpf-Klasse anlegen	51

S

SAP BTP Cockpit	32
SAP BTP → SAP Business Technology Platform (SAP BTP)	
SAP Business Technology Platform (SAP BTP)	23
<i>Subaccount</i>	32
<i>Testaccount</i>	31
SAP GUI	17
SAP HANA	85
SAP-Hilfe	60
SAPUI5	347
Schleife	105
Schlüssel, interne Tabelle	148
Schlüsselwortdokumentation	60
SELECT (ABAP-Anweisung)	149
<i>INTO TABLE (Zusatz)</i>	149
SELECT-Anweisung	85
Selektion	201
Selektor	268
Service Binding	351
Service Key erstellen	36
SET HANDLER	
<i>ACTIVATION (Zusatz)</i>	286
<i>ALL INSTANCES (Zusatz)</i>	286
<i>FOR (Zusatz)</i>	286
Set-Methode	268
Sichtbarkeit	267
<i>geschützt</i>	267
<i>öffentlich</i>	267
<i>Paketzugriff</i>	267
<i>privat</i>	267
Soft Breakpoint	76
SORT (ABAP-Anweisung)	176
<i>ASCENDING (Zusatz)</i>	176
<i>BY (Zusatz)</i>	176
<i>DESCENDING (Zusatz)</i>	176
sortierte Tabelle	146, 150
Spalte	93
Speicherbereich freigeben	72
Sperrobjekt	224, 346
Spezialisierung	289
Sprache	237
SQL	
@	195
@DATA	199
&&	236
ABAP SQL	192
abs()	244
SQL (Forts.)	
ADD_DAYS()	245
ADD_MONTHS()	245
Addition (+)	236
Aggregat-Funktionen	239
AVG()	239
CASE	235
CAST	237
ceil()	244
coalesce()	246
Coalesce-Funktion	239
Common Table Expression	215
CONCAT_WITH_SPACE()	242
CONCAT()	242
COUNT()	239
CROSS JOIN	212
DATS_ADD_DAYS()	245
DATS_DAYS_BETWEEN()	245
DATS_IS_VALID()	245
Datums- und Zeitfunktionen	239
DAYNAME()	245
DAYS_BETWEEN()	245
DELETE	232
DISTINCT	240
div()	244
Division (/)	236
division()	244
elementare Werte	238
EXCEPT	218
EXISTS	219
EXTRACT_DAY()	245
EXTRACT_MONTH()	245
EXTRACT_YEAR()	245
floor()	244
FOR ALL ENTRIES IN	204
Funktionen	239
GROUP BY	206
HAVING	207
Host-Ausdrücke	207
IN	203, 219
INNER JOIN	212
INSERT	226
INSTR()	243
INTERSECT	218
INTO CORRESPONDING FIELDS OF	197
IS_VALID()	245
JOIN	210
LEFT OUTER JOIN	212
LEFT()	243
LENGTH()	242

SQL (Forts.)	
LIKE %_	202
LOWER()	243
LPAD()	242
LTRIM()	242
MAX()	239
MIN()	239
mod()	244
MODIFY	228
MONTHNAME()	245
Multiplikation (*)	236
numerische Funktionen	239
OFFSET	209
Open SQL	191
ORDER BY	205
Performance	247
PRODUCT()	239
REPLACE()	242
RIGHT OUTER JOIN	212
RIGHT()	242
round()	244
RPAD()	242
RTRIM()	242
SELECT	193
SELECT *	197
SELECT SINGLE	193
STRING_AGG()	239
Subselects	219
SUBSTRING()	243
Subtraktion (-)	236
UNION	217
Unterabfragen	219
UPDATE	230
UPPER()	243
UUID-Funktion	239
WEEKDAY()	245
WHERE	201
WITH	215
Zeichenkettenfunktionen	239
Standardtabelle	146
statischer Konstruktor → Klassenkonstruktor	
Steueranweisung	105
String-Template	126
strlen() (Zeichenkettenfunktion)	120
Struktur	70, 101
anlegen	101
definieren	138
Währungsfeld	102
Struktur (Forts.)	
zu interner Tabelle	148
Zuweisung	173
Subklasse → Unterklasse	
substring() (Teilfeldfunktion)	120
Subtraktion	130
SUPER (Referenz)	294
Superklasse → Oberklasse	
SWITCH	111
LET (Zusatz)	111
Syntaxregel	62
Systemfeld	
sy-index	112
sy-subrc	397
Übersicht	397
T	
Tabelle	
Datenbank	93
interne	143
mandantenabhängige	96
sortierte	146
Tabellenausdruck	161
DEFAULT (Zusatz)	162
OPTIONAL (Zusatz)	162
VALUE (Zusatz)	164
Tabellen-Comprehension	156
Tabellenfunktion	185
Tabelleniteration	155
Teilfeldfunktion	120
Teilfeldzugriff	119, 123
Top-down-Ansatz	315
TRY ... ENDTRY (ABAP-Anweisung)	309
TYPE (Zusatz)	
TYPES	70, 137
TYPES	137
INCLUDE STRUCTURE (Zusatz)	139
Typkonvertierung	66
U	
UNIT_CONVERSION	332
Unterklasse	289, 294
UNTIL (Zusatz)	156
Up Cast	313

V

VALUE (ABAP-Anweisung)	153
Variable	
<i>deklarieren</i>	63
<i>initialisieren</i>	71
<i>zuweisen</i>	66
Variablen-Anzeige	78
Vererbung	254, 289–290
Vergleich	107
Verhaltensdefinition	351, 372
Verhaltensprojektion	351, 366
Verkettungsfunktion	186
Verkettungsoperator	117
Verzweigung	105

W

Währungsfeld	102
Watchpoint	82
Wertparameter	285
WHILE (Zusatz)	113, 156
Workspace	25

X

xsdbool() (Funktion)	117
----------------------------	-----

Z

Zeichenkette	116
<i>ersetzen</i>	123
<i>finden</i>	121
<i>Operator</i>	116
<i>Vergleich</i>	116
<i>Verkettung</i>	117
Zeichenketten-Template	126
Zeiger	63
Zeigervariable	259
Zeile	93
Zuweisung	260
Zuweisungsoperator	313
zweistufiges Domänenprinzip	86

Ihr Sprungbrett in die Welt der SAP-Entwicklung

Lernen Sie ABAP von Grund auf und erzielen Sie schnell erste Erfolge mit eigenem Code. Von der Entwicklungsumgebung über Syntaxgrundlagen bis hin zu Datenbanken und objektorientierter Programmierung führt Sie dieses Buch Schritt für Schritt zum Ziel. Die Beispiele decken klassische und moderne Entwicklungsansätze ab – in On-Premise- ebenso wie in Cloud-Umgebungen. Mit CDS Views und dem ABAP RESTful Application Programming Model steigen Sie tiefer in die moderne ABAP-Entwicklung ein. Der ideale Begleiter – auch zur SAP-Schulung S4D400!

Mit diesem Buch lernen Sie:

- ABAP Development Tools für Eclipse einrichten
- Datentypen deklarieren
- Debugging durchführen
- ABAP Dictionary verwenden
- Steueranweisungen und Tabellen einsetzen
- ABAP SQL für den Datenbankzugriff nutzen
- Objektorientierte Programmierung anwenden
- CDS Views erstellen
- ABAP RESTful Application Programming Model einsetzen
- Auf Geschäftsobjekte mit EML zugreifen



Der Autor

Felix Roth ist selbstständiger ABAP-Trainer, -Entwickler und -Berater (LOOP AT Consulting). Er berät Kunden in Entwicklungsprojekten und beschäftigt sich mit den neuesten SAP-Technologien. Seit 2014 hält er regelmäßig SAP-Schulungen, u. a. beim Rheinwerk Verlag.



Programmierbeispiele stehen zum Download bereit

