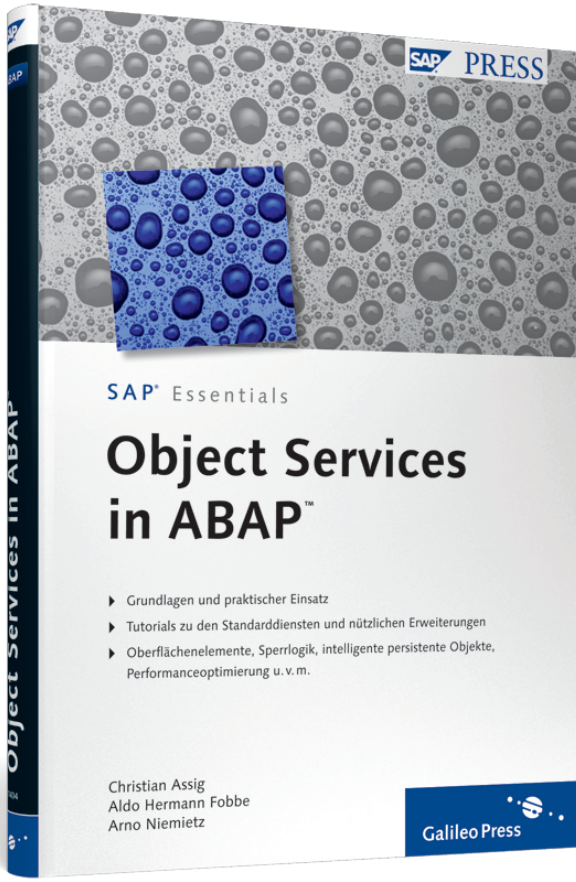
Christian Assig, Aldo Hermann Fobbe, Arno Niemietz

Object Services in ABAP™



Galileo Press

Bonn • Boston

Inhalt

Vorwort 9

1 Einleitung 11

2 Lesen von persistenten Objekten 19

2.1 Anlegen einer persistenten Klasse 20
2.1.1 Auswahl einer Datenbanktabelle zur Persistenzabbildung 21
2.1.2 Zuordnung der einzelnen Felder zu Attributen der Klasse 22
2.1.3 Vollständigkeit der Persistenzabbildung 26
2.1.4 Instanz-GUIDs und Business-Keys 27
2.1.5 Mehr-Tabellen-Mapping 29
2.1.6 Transiente Attribute 30
2.1.7 Aktivierung der persistenten Klasse 30
2.2 Instanzierung von persistenten Objekten 31
2.2.1 IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_OID 32
2.2.2 IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_KEY 34
2.2.3 GET_PERSISTENT 35
2.2.4 Ausnahmebehandlung 36
2.2.5 Verhalten bei bereits geladenen Objekten 37
2.3 Lesen von Attributwerten 38
2.4 Persistente Referenzen 40
2.4.1 Definition einer persistenten Referenz in der Persistenzabbildung 41
2.4.2 Laufzeitverhalten 43
2.4.3 Ausnahmebehandlung 43
2.4.4 Grenzen beim Einsatz persistenter Referenzen 44
2.5 Vererbung 45
2.5.1 Vertikales Mapping 45
2.5.2 Horizontales Mapping 46
2.5.3 Gegenüberstellung der beiden Mapping-Varianten ... 48
2.5.4 Typidentifikator 48
2.5.5 Verwendungsnachweis bei Vererbungsbeziehungen ... 49
2.6 Zusammenfassung 50

3 Anlegen und Ändern von persistenten Objekten 51

3.1 Anlegen von persistenten Objekten 51
3.1.1 IF_OS_FACTORY~CREATE_PERSISTENT 52
3.1.2 IF_OS_FACTORY~CREATE_PERSISTENT_BY_KEY 53
3.1.3 CREATE_PERSISTENT 54
3.1.4 Ausnahmebehandlung 55
3.1.5 Anlegen von transienten Objekten 55
3.2 Änderung von Attributwerten 57
3.3 Transaktionsdienst 58
3.3.1 Transaktionen in SAP-Systemen 58
3.3.2 Objektorientierte Transaktionen 60
3.3.3 Verkettung von Transaktionen 62
3.3.4 Subtransaktionen 63
3.3.5 Transaktionsmodi 65
3.3.6 Verbuchungsmodi 71
3.3.7 Check Agents 76
3.3.8 Undo-Mechanismus für persistente Objekte 78
3.3.9 Transaktionsstatus 80
3.4 Verwaltungszustände persistenter Objekte 81
3.4.1 Zustandsübergänge bei persistenten Objekten, die noch nicht in der Datenbank existieren 82
3.4.2 Zustandsübergänge bei persistenten Objekten, die bereits in der Datenbank existieren 84
3.4.3 Zustandsübergänge beim Beenden von Transaktionen und SAP Logical Units of Work 88
3.5 Zusammenfassung 90

4 Selektion von persistenten Objekten 91

4.1 Ermittlung von Schlüsseln persistenter Objekte mit Open SQL 91
4.2 Masseninstanzierung 93
4.3 Einfache Selektionen mit dem Query-Dienst 95
4.4 Komplexere Selektionen mit dem Query-Dienst 99
4.4.1 Festlegen der Abfrageparameter 99
4.4.2 Festlegen der Filterbedingung 101
4.4.3 Festlegen der Sortierbedingung 109
4.4.4 Übergabe von konkreten Werten für die Abfrageparameter 112
4.5 Vergleich von Query-Dienst und Open SQL 115
4.6 Umgang mit neu erstellten und geänderten Objekten 117
4.7 Zusammenfassung 120

5 Interne Funktionsweise der Object Services 121

5.1 Persistenzdienst 121
5.1.1 Klassenakteur und Base Agent 121
5.1.2 Instanzmanager und Persistenzmanager 126
5.1.3 Datenbanktabellen mit Informationen zur Persistenzabbildung 128
5.1.4 Automatische Quelltext-Generierung 128
5.1.5 Garbage Collection in ABAP Objects 131
5.2 Transaktionsdienst 134
5.2.1 Implementierung des Transaktionsmanagers und der Transaktionen 134
5.2.2 Undo-Management 136
5.3 CL_OS_SYSTEM 139
5.4 Zusammenfassung 141

6 Nützliche Erweiterungen für den praktischen Einsatz 143

6.1 Objekte neu von der Datenbank laden 143
6.2 Freigabe nicht mehr benötigter Objekte 146
6.3 Konvertierung zwischen Objekt und Struktur 147
6.3.1 Lesen der Attributwerte aus einem persistenten Objekt 148
6.3.2 Schreiben der Attributwerte in ein persistentes Objekt 150
6.3.3 Strukturen im Zusammenhang mit persistenten Klassen 152
6.4 Verwendung persistenter Objekte in Benutzeroberflächen 156
6.4.1 SAP Control Framework 157
6.4.2 Verwendung von persistenten Objekten in Web-Dynpro-Kontexten 160
6.4.3 Setzen des Transaktionsmodus in einer Web-Dynpro-Anwendung 165
6.5 Zusammenfassung 166

7 Intelligente persistente Objekte 167

7.1 Plausibilitätsprüfungen 168
7.1.1 Variante 1: Eigene Zugriffsmethoden mit anderem Namen in der persistenten Klasse 169
7.1.2 Variante 2: Delegation mit Zugriffsmethoden in einer gewöhnlichen Klasse 171

7.1.3 Variante 3: Persistente Subklasse ... 173
7.1.4 Variante 4: Enhancement Framework ... 175
7.1.5 Vergleich der verschiedenen Varianten ... 181
7.2 Lazy Loading ... 183
7.3 Zusammenfassung ... 186

8 Integration von SAP-Sperrkonzept und Object Services 187

8.1 SAP-Sperrkonzept ... 188
8.2 Pessimistisches und optimistisches Sperrverfahren ... 192
8.2.1 Sperrmodus ... 193
8.2.2 Pessimistisches Sperrverfahren ... 194
8.2.3 Optimistisches Sperrverfahren ... 196
8.2.4 Einsatz beider Sperrverfahren in einem SAP-System ... 198
8.3 Integration des optimistischen Sperrverfahrens ... 199
8.3.1 Setzen der optimistischen Sperre ... 200
8.3.2 Registrierung des Check Agents ... 204
8.3.3 Wandlung der optimistischen Sperre in eine exklusive ... 207
8.4 Integration des pessimistischen Sperrverfahrens ... 212
8.5 Integration beider Sperrverfahren ... 213
8.6 Zusammenfassung ... 215

9 Fazit ... 217

Die Autoren ... 221

Index ... 223

Ihnen stehen verschiedene Möglichkeiten zur Verfügung, persistente Objekte zu identifizieren, die bestimmte Kriterien erfüllen. Diese Möglichkeiten, zu denen klassisches Open SQL und der Query-Dienst der Object Services gehören, werden in diesem Kapitel vorgestellt.

4 Selektion von persistenten Objekten

In den vorangegangenen Kapiteln wurde beschrieben, wie Sie Objekte instanzieren können, deren Schlüssel – das heißt entweder deren Business-Keys oder deren Instanz-GUIDs – Ihnen bereits bekannt sind. In der Praxis benötigen Sie jedoch häufig die Option, mehrere Objekte zu instanzieren, deren Schlüssel Sie noch nicht kennen. In diesem Kapitel wird gezeigt, wie Sie alle Objekte instanzieren, die bestimmten Kriterien entsprechen. Als Kriterien kommen dabei beispielsweise ein Teil eines komponierten Schlüssels, bestimmte Werte oder Wertebereiche von Attributen oder die Klasse infrage, zu der die Objekte gehören.

4.1 Ermittlung von Schlüsseln persistenter Objekte mit Open SQL

Mit den in ABAP integrierten, unabhängig vom Datenbanksystem einsetzbaren Open-SQL-Anweisungen können Sie die Schlüssel der Objekte ermitteln, die Sie instanzieren wollen. Ohne auf weitere Bestandteile der Object Services als die bereits vorgestellten zurückgreifen zu müssen, können Sie so alle Objekte instanzieren, die bestimmten Kriterien entsprechen.

Diesen Ansatz verfolgt Listing 4.1. Eine `SELECT`-Abfrage in Open SQL ermittelt zunächst die Business-Keys aller Flüge aus dem SAP-Flugdatenmodell, die am aktuellen Tag stattfinden, und speichert sie in einer internen Tabelle zwischen. In einer `LOOP`-Schleife über die interne Tabelle der Business-Keys instanziert der Klassenakteur danach mit der Methode `IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_KEY` in jedem Schleifendurchlauf zu einem Business-Key das zugehörige persistente Objekt. Eine weitere interne Tabel-

le speichert schließlich die Referenzen auf die instanzierten persistenten Objekte.

```
DATA:
  rf_ca_sflight            TYPE REF TO /iot/ca_sflight,
  rf_flight                TYPE REF TO /iot/cl_sflight,
  ta_business_keys_flights TYPE STANDARD TABLE OF
                               scol_flight_key,
  ta_rf_flights            TYPE STANDARD TABLE OF
                               REF TO /iot/cl_sflight,
  wa_business_key_flight   TYPE scol_flights_key.

rf_ca_sflight = /iot/ca_sflight=>agent.

* Business-Keys aller Flüge am heutigen Datum mit einer
* Open-SQL-Abfrage ermitteln
SELECT carrid connid fldate
       INTO CORRESPONDING FIELDS OF TABLE
            ta_business_keys_flights
       FROM sflight
       WHERE fldate = sy-datum.

* Zu jedem Business-Key einzeln das zugehörige persistente
* Objekt instanzieren
LOOP AT ta_business_keys_flights INTO wa_business_key_flight.
  rf_flight ?=
    rf_ca_sflight->if_os_ca_persistency~get_persistent_by_key(
      wa_business_key_flight ).
  APPEND rf_flight TO ta_rf_flights.
ENDLOOP.
```

Listing 4.1 Instanzieren aller Flüge am aktuellen Datum mit Open SQL und der Methode GET_PERSISTENT_BY_KEY

Vor Release 7.0 des AS ABAP war die Kombination aus einer Open-SQL-Anweisung und der einzeln für jedes Objekt durchgeführten Instanzierung die einzige Möglichkeit, um alle persistenten Objekte zu instanzieren, die bestimmten Kriterien entsprechen. Seit Release 7.0 stellt der AS ABAP zwei neue Alternativen zur Verfügung, um mehrere persistente Objekte zu instanzieren.

Seitdem enthalten die Klassenakteure persistenter Klassen zwei neue Methoden zur Masseninstanzierung. Über diese Methoden können Sie mit einem Methodenaufruf zu mehreren bereits bekannten Schlüsseln die persistenten Objekte instanzieren.

Noch umfangreicher sind die Neuerungen, die sich hinter einer dritten neu eingeführten Methode im Klassenakteur verbergen. Dieser Methode übergeben Sie ein Objekt, das Bedingungen enthält, eine sogenannte *Abfrage* oder *Query*. Die Methode instanziert daraufhin alle persistenten Objekte, die den Bedingungen der Query entsprechen. Sie müssen dabei in Ihrer Anwendung keine Open-SQL-Anweisungen verwenden. Die Funktionalitäten zum Formulieren und Ausführen von Abfragen werden als Query-Dienst bezeichnet. Der Query-Dienst bildet mit dem Persistenzdienst und dem Transaktionsdienst die drei aktuellen Bestandteile der Object Services.

4.2 Masseninstanzierung

Um die persistenten Objekte zu mehreren bereits bekannten Schlüsseln zu instanzieren, enthält jeder Klassenakteur jeweils eine Methode, der Sie eine interne Tabelle mit Business-Keys bzw. eine interne Tabelle mit Instanz-GUIDs übergeben können. Listing 4.2 instanziert wie Listing 4.1 alle Flüge am aktuellen Datum. Die Business-Keys der persistenten Objekte ermittelt erneut eine Open-SQL-Anweisung. Darauf folgt ein Aufruf der Methode `GET_PERSISTENT_BY_KEY_TAB` aus dem Interface `IF_OS_CA_PERSISTENCY`. Diese Methode versucht, zu jedem übergebenen Business-Key das zugehörige persistente Objekt aus der Datenbank zu laden. Als Rückgabewert liefert die Methode eine interne Tabelle mit Referenzen auf die instanzierten persistenten Objekte zurück. Die Referenz auf ein persistentes Objekt befindet sich in der Ergebnistabelle jeweils in derselben Zeile, an der sich der zugehörige Schlüssel in der übergebenen Tabelle befand.

```
DATA: rf_ca_sflight            TYPE REF TO /iot/ca_sflight,
      ta_business_keys_flights TYPE STANDARD TABLE OF
                                    scol_flights_key,
      ta_ro_flights            TYPE osreftab.

rf_ca_sflight = /iot/ca_sflight=>agent.

* Business-Keys aller Flüge am heutigen Datum mit einer
* Open-SQL-Abfrage ermitteln
SELECT carrid connid fldate
       INTO CORRESPONDING FIELDS OF TABLE
            ta_business_keys_flights
       FROM sflight
       WHERE fldate = sy-datum.
```

```
* Zu jedem Business-Key das zugehörige persistente Objekt
* instanzieren
ta_ro_flights =
  rf_ca_sflight->if_os_ca_persistency~get_persistent_by_key_tab(
    ta_business_keys_flights ).
```

Listing 4.2 Instanzieren aller Flüge am aktuellen Datum mit Open SQL und der Methode GET_PERSISTENT_BY_KEY_TAB

Die Methode `GET_PERSISTENT_BY_KEY_TAB` steht Ihnen in den Klassenakteuren aller persistenten Klassen zur Verfügung, zu denen Sie in der Persistenzabbildung einen Business-Key definiert haben. Die übergebene interne Tabelle kann eine Standardtabelle oder eine sortierte Tabelle sein. Der Zeilentyp der Tabelle muss der Business-Key-Struktur der persistenten Klasse entsprechen. Auch hier ist zu beachten, dass Sie eine Struktur verwenden müssen, in der die einzelnen Komponenten nach den Attributnamen sortiert sind (siehe Warnung in Abschnitt 2.2.2).

Der Rückgabewert der Methode ist vom Typ `OSREFTAB`. Dahinter verbirgt sich eine Standardtabelle, in der jede Zeile eine Referenz vom Typ `OBJECT` enthält. Bei späteren Zugriffen auf die persistenten Objekte ist daher in der Regel ein Downcast erforderlich.

Analog zur Methode `GET_PERSISTENT_BY_KEY_TAB` enthält der Klassenakteur die Methode `GET_PERSISTENT_BY_OID_TAB`, mit der Sie mehrere persistente Objekte jeweils über ihren Instanz-GUID instanzieren können. Die Methode können Sie im Klassenakteur von persistenten Klassen nutzen, in deren Persistenzabbildung ein Instanz-GUID definiert ist. Sie übergeben der Methode eine Standardtabelle oder eine sortierte Tabelle mit dem Zeilentyp `OS_GUID` und erhalten auch von dieser Methode einen Rückgabewert vom Typ `OSREFTAB`.

Die Verbesserung der Geschwindigkeit, die Sie erreichen, wenn Sie statt der mehrfach aufgerufenen Instanzierung jeweils eines persistenten Objektes wie in Listing 4.1 die Masseninstanzierung verwenden, fällt abhängig vom System und von der Art der verwendeten Objekte unterschiedlich aus. Beobachtet wurden sowohl deutliche Geschwindigkeitsverbesserungen um mehr als 50% als auch ein Laufzeitverhalten, das sich in beiden Varianten kaum unterschied. Grundsätzlich ist jedoch ab Support Package 13 für den AS ABAP 7.0 die Geschwindigkeit der Masseninstanzierung in jedem Fall mindestens so hoch wie die der einzeln aufgerufenen Instanzierung. In älteren Releases konnte dagegen die Masseninstanzierung ab etwa 500 auf einmal instanzierten persistenten Objekten aufgrund eines mittlerweile beho-

benen Fehlers auch langsamer ablaufen als die mehrfach nacheinander aufgerufene Instanzierung einzelner Objekte.

Die Methoden zur Masseninstanzierung lösen nur dann eine Ausnahme der Klasse `CX_OS_OBJECT_NOT_FOUND` aus, wenn sie zu einem übergebenen Schlüssel auf ein Objekt stoßen, das sich im Verwaltungszustand `DELETED` oder `TRANSIENT` befindet. Das Ausnahmeobjekt enthält in einem Attribut den Business-Key bzw. den Instanz-GUID des Objektes, bei dem die Ausnahme aufgetreten ist. Tritt eine solche Ausnahme auf, bricht die jeweilige Methode auch die Instanzierung weiterer persistenter Objekte zu den übergebenen Schlüsseln ab.

Übergeben Sie einer Methode zur Masseninstanzierung einen Schlüssel, zu dem in der Datenbank kein Objekt existiert, führt dies im Unterschied zur Instanzierung eines einzelnen Objektes nicht zu einer Ausnahme. Auch die Instanzierung weiterer Objekte wird in diesem Fall nicht abgebrochen. Stattdessen enthält die Ergebnistabelle in der entsprechenden Zeile eine Nullreferenz. Wenn Sie beispielsweise der Methode in der dritten Zeile der Tabelle einen Schlüssel übergeben, zu dem die Methode weder in der Verwaltung des Klassenakteurs noch in der Datenbank ein persistentes Objekt findet, enthält die Ergebnistabelle in der dritten Zeile statt einer Referenz auf ein persistentes Objekt eine Nullreferenz. Jede Zeile der Ergebnistabelle, zu der Sie einen Schlüssel eines existierenden persistenten Objektes übergeben haben, enthält eine gültige Referenz.

4.3 Einfache Selektionen mit dem Query-Dienst

Neben der Masseninstanzierung, mit der Sie mehrere persistente Objekte zu bereits bekannten Schlüsseln instanzieren können, bieten die Object Services auch die Möglichkeit, alle persistenten Objekte zu instanzieren, deren persistente Attribute bestimmte Bedingungen erfüllen. Mithilfe des Query-Dienstes können Sie dazu eine Abfrage (Query) formulieren, in der Sie in Form einer Filterbedingung definieren, welche Bedingungen die persistenten Attribute der persistenten Objekte erfüllen sollen. Außerdem können Sie über die Sortierbedingung einer Abfrage angeben, nach welchen Kriterien das System die persistenten Objekte sortieren soll.

Zur Formulierung der Filterbedingung und der Sortierbedingung haben Sie zwei Möglichkeiten: Sie können eine Syntax verwenden, die der `WHERE`- und bzw. der `ORDER BY`-Klausel einer SQL-Abfrage ähnelt. Daneben ist es möglich,

Filter- und Sortierbedingungen mithilfe von Objekten zu definieren. Diese alternative Art, Bedingungen zu formulieren, wird in Abschnitt 4.4, »Komplexere Selektionen mit dem Query-Dienst«, erläutert.

Listing 4.3 gibt ein weiteres Beispiel dafür, wie Sie die Ermittlung aller Flüge am aktuellen Tag implementieren können. In diesem Fall erfolgen die Ermittlung und die Instanzierung der Flüge mithilfe des Query-Dienstes.

```
DATA: rf_ca_sflight    TYPE REF TO /iot/ca_sflight,
      ri_query_manager TYPE REF TO if_os_query_manager,
      ri_query         TYPE REF TO if_os_query,
      ta_ro_flights    TYPE osreftab,
      v_filter         TYPE string.

rf_ca_sflight = /iot/ca_sflight=>agent.
ri_query_manager = cl_os_system=>get_query_manager( ).

* Filterbedingung formulieren und Abfrage anlegen
CONCATENATE 'FLDATE = ''' sy-datum '''' INTO v_filter.
ri_query =
  ri_query_manager->create_query(
    i_filter = v_filter ).

* Alle persistenten Objekte instanzieren, die den Bedingungen
* der Query entsprechen
ta_ro_flights =
  rf_ca_sflight->if_os_ca_persistency~get_persistent_by_query(
    ri_query ).
```

Listing 4.3 Instanzieren aller Flüge am aktuellen Datum mit dem Query-Dienst

Eine Abfrage des Query-Dienstes legen Sie auf ähnliche Art an wie eine Transaktion des Transaktionsdienstes: Über eine statische Methode der Klasse `CL_OS_SYSTEM`, in diesem Fall `GET_QUERY_MANAGER`, erhalten Sie eine Referenz auf den Query-Manager. Der Query-Manager stellt eine Methode namens `CREATE_QUERY` zur Verfügung, die eine neue Abfrage anlegt und eine Referenz auf diese Abfrage zurückgibt. Die Filter- und Sortierbedingungen können Sie jeweils beim Anlegen der Abfrage an die Methode `CREATE_QUERY` übergeben oder sie nach dem Anlegen über verschiedene Methoden des Abfrageobjektes setzen (siehe Abschnitt 4.4, »Komplexere Selektionen mit dem Query-Dienst«).

Bei der Formulierung der Filter- und Sortierbedingungen müssen Sie sich jeweils auf die Namen der persistenten Attribute beziehen. Solange Sie in der Persistenzabbildung den persistenten Attributen jeweils denselben Namen

gegeben haben wie dem Feld in der Datenbanktabelle, stellt dies keine Fehlerquelle dar. Haben Sie in der Persistenzabbildung Attributnamen definiert, die von den Feldnamen der Datenbanktabelle abweichen, müssen Sie bei der Formulierung der Filter- und Sortierkriterien jeweils die Attributnamen verwenden, nicht die Feldnamen aus der zugrunde liegenden Datenbanktabelle. Außerdem dürfen Sie in Filter- und Sortierbedingungen von Abfragen ausschließlich persistente Attribute verwenden, für die Sie in der Persistenzabbildung die Sichtbarkeit PUBLIC gewählt haben.

Nachdem Sie die gewünschten Filter- und Sortierbedingungen für die Abfrage festgelegt haben, rufen Sie die Methode `IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_QUERY` des Klassenakteurs der Klasse auf, zu der Sie die persistenten Objekte instanzieren möchten. Dieser Methode müssen Sie das Anfrageobjekt übergeben, in dem Sie die Filter- und Sortierbedingungen definiert haben.

Das Verhalten der Methode `IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_QUERY` können Sie über die Parameter `I_UPTO` und `I_SUBCLASSES` beeinflussen:

- Mit dem Parameter `I_UPTO` können Sie angeben, wie viele persistente Objekte die Methode maximal zurückgeben soll. Mit einer solchen Beschränkung können Sie die Anzahl der instanzierten Objekte und damit die Ladezeit sowie den Speicherverbrauch der Anwendung begrenzen. Dies ist insbesondere dann sinnvoll, wenn der Anwender die Filterbedingung vorgibt und zur persistenten Klasse eine sehr große Anzahl von persistenten Objekten existiert.

  Ist die Anzahl der persistenten Objekte, die der Filterbedingung entsprechen, größer als der Wert von `I_UPTO`, liefert die Methode eine Teilmenge der persistenten Objekte, die der Filterbedingung entsprechen. Haben Sie dabei eine Sortierbedingung angegeben, liefert die Methode die ersten persistenten Objekte gemäß der Sortierung. Ohne Angabe einer Sortierbedingung sollten Sie keine Annahmen darüber treffen, wie der Query-Dienst die zurückgegebene Teilmenge von Objekten auswählt. Wenn Sie den Parameter `I_UPTO` nicht verwenden oder den Wert `0` übergeben, erfolgt keine Begrenzung der Anzahl zurückgelieferter Objekte. Negative Zahlen dürfen Sie dem Parameter nicht übergeben.
- Der Parameter `I_SUBCLASSES` ist nur relevant, wenn Subklassen zur persistenten Klasse existieren und Sie in der Persistenzabbildung einen Typidentifikator definiert haben. Wenn Sie diesem Parameter den Wert `abap_`

`true` übergeben, instanziert die Methode auch die Objekte der persistenten Klasse, die gleichzeitig zu einer Subklasse der persistenten Klasse gehören. Die Methode verhält sich bei vorhandenem Typidentifikator polymorph, das heißt sie liefert gegebenenfalls automatisch Objekte der Subklassen. Attribute, die in Subklassen definiert sind, sind in diesem Fall ebenfalls gefüllt, auch wenn Sie die Methode im Klassenakteur der Superklasse aufgerufen haben. Mit der Standardeinstellung `abap_false` gibt die Methode bei vorhandenem Typidentifikator nur persistente Objekte zurück, die zu der jeweiligen persistenten Klasse gehören, aber nicht zu einer Subklasse.

Ist kein Typidentifikator definiert, hat die Methode keine effiziente Möglichkeit, zu entscheiden, ob ein Objekt zu einer Subklasse gehört. In diesem Fall liefert sie auch dann eine Referenz auf ein Objekt der vom Klassenakteur verwalteten persistenten Klasse, wenn das Objekt ebenfalls zu einer Subklasse gehört.

Ein großer Unterschied zwischen den drei unterschiedlichen Verfahren, mit denen Sie über die Object Services mehrere persistente Objekte instanzieren können, besteht bei der Anzahl der notwendigen Datenbankzugriffe:

- Bei der Einzelinstanzierung (siehe Listing 4.1) erfolgt ein Datenbankzugriff, mit dem Sie die Schlüssel der persistenten Objekte ermitteln. Darauf folgt bei der Instanzierung jedes persistenten Objektes ein weiterer Datenbankzugriff.
- Bei der Masseninstanzierung (siehe Listing 4.2) sind im Idealfall nur zwei Datenbankzugriffe notwendig, um eine Vielzahl von persistenten Objekten zu instanzieren: Nachdem Sie mit dem ersten Datenbankzugriff die Schlüssel der persistenten Objekte ermittelt haben, kann der Persistenzdienst häufig mit nur einem weiteren Datenbankzugriff alle angeforderten persistenten Objekte aus der Datenbank laden. Bei einer großen Anzahl von Schlüsseln kann es vorkommen, dass die ABAP-Laufzeitumgebung mehrere Datenbankzugriffe durchführt und dabei jeweils eine Teilmenge der angeforderten persistenten Objekte aus der Datenbank lädt.
- Die geringste mögliche Anzahl von Datenbankzugriffen erreichen Sie durch die Verwendung des Query-Dienstes (siehe Listing 4.3). Im Rahmen nur eines Datenbankzugriffs übergibt der Query-Dienst die von Ihnen definierten Filter- und die Sortierbedingungen an das Datenbanksystem. Als Ergebnis dieser Abfrage liefert das Datenbanksystem die persistenten Attribute zu allen Objekten, die der Abfrage entsprechen. Daher sind keine weiteren Datenbankzugriffe notwendig.

Durch den reduzierten Kommunikationsbedarf zwischen Applikationsserver und Datenbanksystem können Sie die Instanzierung von persistenten Objekten mit dem Query-Dienst gegenüber der Masseninstanzierung in vielen Fällen noch weiter beschleunigen. Beispielsweise kann die Instanzierung von 100.000 persistenten Objekten mit dem Query-Dienst durchaus doppelt so schnell ablaufen wie mit der Masseninstanzierung. Um welchen Faktor sich die Geschwindigkeit unterscheidet, hängt auch hier vom System und der persistenten Klasse ab.

4.4 Komplexere Selektionen mit dem Query-Dienst

Das einfache Beispiel aus Abschnitt 4.3 hat gezeigt, wie Sie mithilfe des Query-Dienstes eine Abfrage mit einer einfachen Filterbedingung erstellen. In diesem Abschnitt wird nun ausgeführt, wie Sie aus mehreren Einzelkriterien eine komplexere Filterbedingung zusammensetzen und die Sortierbedingung festlegen können.

In einer Filterbedingung können Sie außerdem mit *Abfrageparametern* arbeiten. Mithilfe von Abfrageparametern haben Sie die Möglichkeit, einmalig ein Abfrageobjekt mit einer Filterbedingung anzulegen und diese Abfrage danach mit verschiedenen Werten in den einzelnen Kriterien der Filterbedingung auszuführen. So können Sie beispielsweise ein Abfrageobjekt anlegen, in dessen Filterbedingung Sie angeben, dass Sie Flugpläne mit einem bestimmten Start- und einem bestimmten Zielort instanzieren möchten. In der Filterbedingung geben Sie die konkreten Flughäfen noch nicht an, sondern beziehen sich zunächst auf Abfrageparameter. Die so angelegte Abfrage können Sie dann wiederholt ausführen und dabei jeweils über die Abfrageparameter angeben, welcher konkrete Startort und welcher Zielort bei dieser Ausführung der Abfrage gewünscht sind.

4.4.1 Festlegen der Abfrageparameter

Ohne dass Sie weitere Angaben machen müssen, stehen in jeder Abfrage drei Abfrageparameter zur Verfügung, die Sie bei der Formulierung einer Filterbedingung verwenden können. Sie haben die Namen `PAR1`, `PAR2` und `PAR3`. Möchten Sie in der Filterbedingung mehr als drei Abfrageparameter verwenden oder den Abfrageparametern sprechende Namen geben, haben Sie zwei Möglichkeiten:

- Sie übergeben beim Anlegen der Abfrage der Methode CREATE_QUERY den Importing-Parameter I_PARAMETERS. Dazu verwenden Sie einen String, in dem die einzelnen Namen der Abfrageparameter durch ein Leerzeichen getrennt sind. Listing 4.4 zeigt die Definition von zwei Abfrageparametern mit den Namen PAR_AIRPFROM und PAR_AIRPTO. Die Groß-/Kleinschreibung hat bei dieser Art, Abfrageparameter zu definieren, keine Auswirkungen. Unabhängig davon, ob Sie die Namen der Parameter groß oder klein schreiben, legt das System Abfrageparameter mit Namen in Großbuchstaben an.

```
DATA: ri_query_manager   TYPE REF TO if_os_query_manager,
      ri_query           TYPE REF TO if_os_query.

ri_query_manager = cl_os_system=>get_query_manager( ).

* Neue Abfrage anlegen und dabei die Namen der Parameter
* festlegen
ri_query =
  ri_query_manager->create_query(
    i_parameters = 'PAR_AIRPFROM PAR_AIRPTO' ).
```

Listing 4.4 Setzen der Parameternamen beim Anlegen einer Abfrage

- Alternativ besteht die Möglichkeit, die Namen der Abfrageparameter nach dem Anlegen der Abfrage über einen sogenannten *Parameterausdruck* zu definieren. Dazu können Sie sich vom Abfrageobjekt mit der Methode GET_EXPR_FACTORY eine Referenz auf eine Ausdrucks-Factory geben lassen. Die Ausdrucks-Factory bietet eine Methode namens CREATE_PARAMETERS_EXPR an, die einen Parameterausdruck anlegt und eine Referenz auf das Objekt zurückliefert. Der Methode APPEND des Parameterausdrucks können Sie dann nacheinander bei jedem Aufruf jeweils den Namen eines Abfrageparameters übergeben. Den fertigen Parameterausdruck mit den Namen aller Abfrageparameter übergeben Sie schließlich der Methode SET_PARAMETERS_EXPR der Abfrage.

  Bei der Erstellung eines Parameterausdrucks ist die Groß-/Kleinschreibung der Namen der Abfrageparameter relevant. Sie sollten alle Parameternamen ausschließlich in Großbuchstaben schreiben, um Schwierigkeiten bei der Ausführung der Abfrage zu verhindern.

  Listing 4.5 zeigt den beschriebenen Ablauf wie Listing 4.4 anhand von zwei Parametern mit den Namen PAR_AIRPFROM und PAR_AIRPTO.

```
DATA: ri_query_manager   TYPE REF TO if_os_query_manager,
      ri_query           TYPE REF TO if_os_query,
```

```
      ri_expr_factory    TYPE REF TO if_os_query_expr_factory,
      ri_parameters_expr TYPE REF TO if_os_query_parameters_expr.

* Neue Abfrage anlegen
ri_query_manager = cl_os_system=>get_query_manager( ).
ri_query = ri_query_manager->create_query( ).

* Referenz auf Ausdrucks-Factory ermitteln
ri_expr_factory = ri_query->get_expr_factory( ).

* Neuen Parameterausdruck anlegen
ri_parameters_expr = ri_expr_factory->create_parameters_expr( ).

* Namen der Parameter im Parameterausdruck festlegen
ri_parameters_expr->append( 'PAR_AIRPFROM' ).
ri_parameters_expr->append( 'PAR_AIRPTO' ).

* Parameterausdruck der Abfrage zuordnen
ri_query->set_parameters_expr( ri_parameters_expr ).
```

Listing 4.5 Setzen der Parameternamen über einen Parameterausdruck

Die Definition der Parameternamen ist über einen Parameterausdruck deutlich aufwendiger zu implementieren als die Definition beim Anlegen der Abfrage. Da sich die beiden Varianten in ihrer Ausdrucksmächtigkeit nicht unterscheiden, wird die Definition der Parameternamen beim Anlegen der Abfrage empfohlen.

Der Name eines Abfrageparameters darf aus den Buchstaben von A bis Z, den Ziffern 0 bis 9 und dem Unterstrich bestehen. Der Name muss mit einem Buchstaben beginnen. Um Namenskonflikte zu vermeiden, darf der Name eines Parameters nicht identisch mit dem Namen eines Attributes der persistenten Klasse sein, zu der Sie persistente Objekte instanzieren wollen.

Die definierten Parameter können Sie bei der Formulierung der Filterbedingung verwenden. Wenn Sie die Abfrage ausführen, müssen Sie jeweils einen Wert für jeden Parameter angeben, den Sie in der Filterbedingung verwendet haben.

4.4.2 Festlegen der Filterbedingung

Mit der Filterbedingung haben Sie die Möglichkeit, einzuschränken, welche Objekte eine Abfrage mit dem Query-Dienst zurückliefern soll. Dazu können Sie, bezogen auf die persistenten Attribute der persistenten Objekte, Kriterien angeben, die jedes von der Abfrage zurückgegebene Objekt erfüllen soll.

Sie können eine Abfrage mit dem Query-Dienst auch durchführen, ohne eine Filterbedingung anzugeben. In diesem Fall liefert die Abfrage Referenzen auf alle persistenten Objekte der persistenten Klasse zurück.

Ähnlich wie die Parameternamen können Sie auch die Filterbedingung auf zwei Wegen festlegen:

- Sie geben die Filterbedingung in einer SQL-ähnlichen Syntax beim Anlegen der Abfrage an.
- Sie definieren nach dem Anlegen der Abfrage einen sogenannten Filterausdruck.

Definition der Filterbedingung beim Anlegen der Abfrage

Beim Anlegen einer Abfrage mit der Methode `CREATE_QUERY` können Sie über den Parameter `I_FILTER` einen String übergeben, der die Filterbedingung enthält. Zur Formulierung der Filterbedingung können Sie eine Teilmenge der grundlegenden Sprachelemente verwenden, mit denen Sie auch in Open SQL eine `WHERE`-Klausel gestalten können. Außerdem steht ein zusätzlicher Operator zur Verfügung, mit dem Sie Referenzen auf andere persistente Objekte in Bedingungen verwenden können.

Mit den Vergleichsoperatoren = (gleich), <> (ungleich), < (kleiner), <= (kleiner oder gleich), > (größer) und >= (größer oder gleich) können Sie festlegen, welche Werte ein persistentes Attribut annehmen darf, damit das persistente Objekt zum Ergebnis der Abfrage gehört. Während auf der linken Seite des Vergleichsoperators immer der Name eines persistenten Attributes stehen muss, kann auf der rechten Seite der Name eines persistenten Attributes, der Name eines Abfrageparameters oder ein Wert in Form eines Literals stehen. Literale mit allen Datentypen, auch ganze Zahlen, müssen immer umschlossen von einfachen Hochkommata (') im String mit der Filterbedingung stehen.

Listing 4.6 zeigt drei mögliche Filterbedingungen mit Vergleichsoperatoren:

- `v_filter_1` selektiert alle Flugpläne, in denen eine Flugzeit von maximal 120 Minuten angegeben ist. Die maximale Flugzeit ist dabei als Literal eingeschlossen in Hochkommata angegeben. Bei der Angabe der Filterbedingung in Form eines String-Literals im ABAP-Quelltext müssen Sie dazu jeweils zwei einfache Hochkommata angeben, damit der String an der entsprechenden Stelle ein Hochkomma enthält.

- Mit `v_filter_2` können Sie alle Flüge instanzieren, die an einem bestimmten Datum stattfinden. Dazu müssen Sie für den standardmäßig definierten Abfrageparameter `PAR1` noch beim Ausführen der Abfrage einen konkreten Datumswert übergeben.
- In `v_filter_3` findet ein Vergleich von zwei persistenten Attributen statt. Mit diesen Filterbedingungen selektieren Sie alle Flüge, bei denen in der Business Class und in der First Class gleich viele Plätze belegt sind.

```
DATA: v_filter_1 TYPE string,
      v_filter_2 TYPE string,
      v_filter_3 TYPE string.

v_filter_1 = 'FLTIME <= ''120'''.
v_filter_2 = 'FLDATE = PAR1'.
v_filter_3 = 'SEATSOCC_B = SEATSOCC_F'.
```

Listing 4.6 Filterbedingungen mit Vergleichsoperatoren

Mit dem Operator `LIKE` können Sie den Wert eines persistenten Attributes gegen ein Muster prüfen. In dem Muster können Sie den Unterstrich (_) verwenden, wenn an der entsprechenden Stelle genau ein beliebiges Zeichen stehen darf. Dürfen an einer Stelle beliebig viele beliebige Zeichen stehen, sollten Sie das Prozentzeichen (%) verwenden. Mit dem Zusatz `ESCAPE` können Sie ein sogenanntes Fluchtsymbol definieren. Steht das gewählte Fluchtsymbol im Muster vor dem Unterstrich oder dem Prozentzeichen, verlieren diese Zeichen ihre besondere Bedeutung. Auf diese Art können Sie auch mit dem Operator `LIKE` ausdrücken, dass sich an einer bestimmten Stelle tatsächlich ein Unterstrich bzw. ein Prozentzeichen befinden muss.

Beispiele für die Verwendung des Operators `LIKE` enthält Listing 4.7:

- `v_filter_4` durchsucht alle Flugkunden nach E-Mail-Adressen, die die Zeichen »@sap.« enthalten. Vor diesen Zeichen dürfen beliebig viele weitere Zeichen stehen, dahinter genau zwei beliebige Zeichen.
- `v_filter_5` sucht nach allen Kunden mit einer E-Mail-Adresse, die den Namen Benjamin gefolgt von einem Unterstrich enthält. Da in diesem Fall wirklich das Unterstrichzeichen gesucht und nicht die Bedeutung des Unterstrichzeichens als Platzhalter für ein beliebiges Zeichen gewünscht ist, wird die Raute (#) als Fluchtsymbol definiert. Hintereinander geschrieben, bewirken das Fluchtsymbol Raute und der Unterstrich, dass die Abfrage in der E-Mail-Adresse nach dem Unterstrich sucht und den Unterstrich nicht als Platzhalter für ein beliebiges Zeichen interpretiert.

```
DATA: v_filter_4 TYPE string,
      v_filter_5 TYPE string.

v_filter_4 = 'EMAIL LIKE ''%@sap.__'''.
v_filter_5 = 'EMAIL LIKE ''%Benjamin#_%'' ESCAPE ''#'''.
```

Listing 4.7 Filterbedingungen mit dem Operator LIKE

Nullwerte, das heißt explizit als leer gekennzeichnete Felder einer Datenbanktabelle, kommen normalerweise nicht vor, wenn Sie mit den Object Services auf einer Datenbanktabelle arbeiten. Nur wenn Sie zu einer Datenbanktabelle, die bereits Datensätze enthielt, neue Felder angelegt haben, oder wenn Sie in die Datenbanktabelle auch ohne die Object Services direkt mit Open SQL schreiben, kann es vorkommen, dass in der Datenbanktabelle statt des jeweiligen Initialwertes des Datentyps ein Nullwert steht. Um eine solche Konstellation zu identifizieren, können Sie den Operator `IS NULL` verwenden.

Zusätzlich zu den geschilderten Operatoren, die auch in Open SQL vorhanden sind, können Sie in der Filterbedingung einer Abfrage mit dem Query-Dienst auch den Operator `EQUALSREF` verwenden. Mit dem Operator `EQUALSREF` können Sie prüfen, ob eine persistente Referenz auf dasselbe persistente Objekt verweist wie eine gewöhnliche Referenz auf ein persistentes Objekt in der laufenden Anwendung. Vor dem Operator `EQUALSREF` müssen Sie immer den Namen eines persistenten Attributes angeben, das in der Persistenzabbildung als persistente Referenz definiert ist. Hinter dem Operator muss immer der Name eines Parameters stehen.

Mit den logischen Operatoren `AND` und `OR` können Sie jeweils zwei Ausdrücke zu einem neuen Ausdruck verknüpfen. Mit dem logischen Operator `NOT` können Sie das Ergebnis eines Ausdrucks umkehren. Um festzulegen, in welcher Reihenfolge das System die einzelnen logischen Operatoren auswerten soll, können Sie runde Klammern verwenden. Dabei muss jede Klammer durch mindestens ein Leerzeichen von allen anderen Bestandteilen der Filterbedingung getrennt sein.

Die Abfrage, die Sie mit der Filterbedingung aus Listing 4.8 ausführen können, instanziert alle Flugpläne, in denen ein Abflughafen in Deutschland und ein Zielflughafen in den USA oder in Japan angegeben sind.

```
DATA: v_filter_6 TYPE string.

v_filter_6 =
  'COUNTRYFROM = ''DE'' AND ' &
  '( COUNTRYTO = ''US'' OR COUNTRYTO = ''JP'' )'.
```

Listing 4.8 Filterbedingung mit logischen Operatoren

Definieren Sie die Filterbedingung beim Anlegen der Abfrage, müssen Sie nicht auf die Groß-/Kleinschreibung achten. Sowohl die Namen von Parametern und Attributen als auch die logischen Operatoren konvertiert das System beim Ausführen der Abfrage automatisch in Großbuchstaben. Lediglich bei Literalen, gegen die Sie die Werte von Attributen prüfen wollen, müssen Sie die Groß-/Kleinschreibung beachten.

Definition der Filterbedingung über einen Filterausdruck

Ähnlich wie bei der Festlegung der Namen der verwendeten Abfrageparameter können Sie auch zur Definition der Filterbedingung die Ausdrucks-Factory verwenden. Mithilfe der Ausdrucks-Factory erzeugen Sie in diesem Fall einen Filterausdruck, den Sie schließlich der Abfrage zuordnen.

Die Grundidee dieser Art der Definition einer Filterbedingung ist, dass Sie einen Baum von Objekten erstellen, in dem jedes Objekt einen Teil der Filterbedingung repräsentiert. Dabei steht jedes Blatt eines Baums für eine Einzelbedingung mit Bezug auf ein persistentes Attribut. Die inneren Knoten dienen der Verknüpfung von Teilbedingungen mithilfe von logischen Operatoren.

Als Blätter des Baums kommen vier Arten von Ausdrucksobjekten infrage:

- *Operator-Ausdrücke* ermöglichen, ein persistentes Attribut unter Anwendung eines Vergleichsoperators mit einem anderen persistenten Attribut, mit einem Parameter oder mit einem übergebenen Wert zu vergleichen. Hierbei können Sie dieselben Vergleichsoperatoren verwenden wie bei der Definition der Filterbedingung beim Anlegen der Abfrage.
- Über einen *LIKE-Ausdruck* können Sie wie mit dem Operator `LIKE` ein persistentes Attribut gegen ein Muster prüfen.
- Prüfungen auf den Nullwert führen Sie mit einem *IS NULL-Ausdruck* durch. Eine solche Prüfung entspricht der Verwendung des Operators `IS NULL`.

- Ein *Referenz-Ausdruck* stellt wie der Operator `EQUALSREF` fest, ob eine persistente Referenz auf ein von Ihnen angegebenes persistentes Objekt verweist.

Alle Ausdrücke legen Sie über eine entsprechende Methode der Ausdrucks-Factory an. Jede Methode bietet individuelle Parameter an, mit denen Sie Angaben wie das zu vergleichende persistente Attribut, den Vergleichsoperator oder den Vergleichswert übergeben können. In den meisten Methoden ist ein Importing-Parameter mit dem Namen `I_IDX` definiert. Falls Sie sich in einem Ausdruck auf einen Parameter der Abfrage beziehen möchten, geben Sie mit diesem Importing-Parameter den Index des Abfrageparameters an. Der Index eines Abfrageparameters ergibt sich aus der Reihenfolge, in der Sie die Namen der Abfrageparameter definiert haben. Der erste Abfrageparameter erhält den Index 1, die weiteren Parameter nummeriert der Query-Dienst aufsteigend.

Außerdem können Sie bei mehreren Methoden wahlweise entweder einen Wert übergeben, den der Query-Dienst für die Datenbankabfrage automatisch in Hochkommata stellt, oder einen Wert, der die benötigten Hochkommata bereits enthält. Die Namen der Importing-Parameter, bei denen Sie Werte inklusive der Hochkommata übergeben müssen, enden auf dem Suffix `_W_QUOTES` (with quotes). Im Zweifelsfall sollten Sie immer den alternativen Importing-Parameter ohne das Suffix `_W_QUOTES` bevorzugen.

Mithilfe von `AND`- und `OR`-Ausdrücken können Sie jeweils zwei Ausdrücke mit einem logischen Und bzw. mit einem logischen Oder miteinander verknüpfen. Da es sich bei den verknüpften Ausdrücken sowohl um einzelne Ausdrücke als auch um bereits erstellte Verknüpfungen mehrerer Ausdrücke in Baumform handeln kann, können Sie mithilfe dieser Ausdrücke Bäume von beliebiger Größe aufbauen. Beim Anlegen eines `NOT`-Ausdrucks übergeben Sie nur einen Ausdruck. Der `NOT`-Ausdruck negiert diesen Ausdruck.

Abbildung 4.1 und Listing 4.9 zeigen dieselbe Filterbedingung, die bereits in Listing 4.8 formuliert wurde. Das UML-Objektdiagramm in Abbildung 4.1 zeigt die Objekte, aus denen sich diese Filterbedingung zusammensetzt. Für jeden Teilausdruck, aus dem sich die Filterbedingung zusammensetzt, existiert jeweils ein Objekt. Der gesamte Baum zusammengenommen beschreibt die Filterbedingung, dass der Abflughafen in Deutschland und der Zielflughafen entweder in den USA oder in Japan liegen sollen.

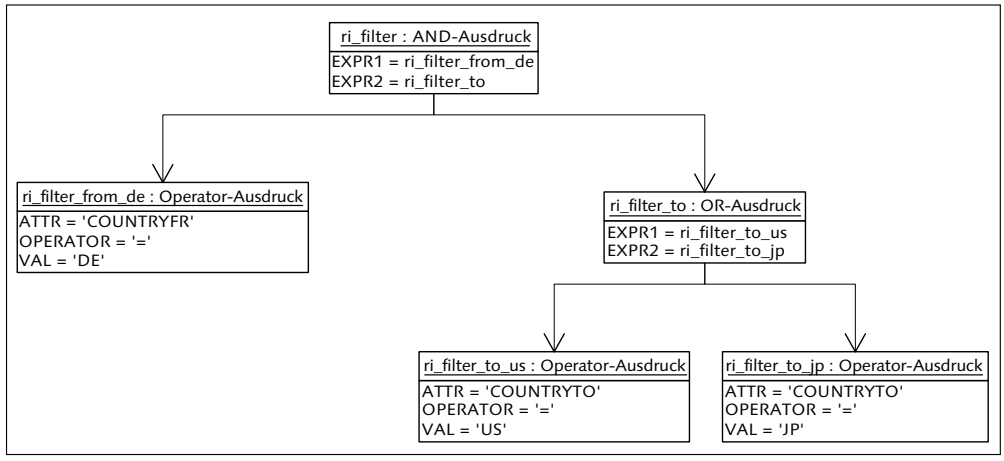


Abbildung 4.1 Objektdiagramm eines Filterausdrucks

```
DATA: rf_ca_spfli       TYPE REF TO /iot/ca_spfli,
      ri_query_manager  TYPE REF TO if_os_query_manager,
      ri_query          TYPE REF TO if_os_query,
      ri_expr_factory   TYPE REF TO if_os_query_expr_factory,
      ri_filter_from_de TYPE REF TO if_os_query_filter_expr,
      ri_filter_to_us   TYPE REF TO if_os_query_filter_expr,
      ri_filter_to_jp   TYPE REF TO if_os_query_filter_expr,
      ri_filter_to      TYPE REF TO if_os_query_filter_expr,
      ri_filter         TYPE REF TO if_os_query_filter_expr,
      ta_ro_spfli       TYPE osreftab.

ri_query_manager = cl_os_system=>get_query_manager( ).
rf_ca_spfli = /iot/ca_spfli=>agent.

* Abfrage anlegen
ri_query = ri_query_manager->create_query( ).

* Referenz auf Ausdrucks-Factory ermitteln
ri_expr_factory = ri_query->get_expr_factory( ).

* Filterausdruck anlegen: Abflugland Deutschland
ri_filter_from_de =
  ri_expr_factory->create_operator_expr(
    i_attr1    = 'COUNTRYFR'
    i_operator = '='
    i_val      = 'DE' ).

* Filterausdruck anlegen: Zielland USA
ri_filter_to_us =
  ri_expr_factory->create_operator_expr(
```

```
    i_attr1    = 'COUNTRYTO'
    i_operator = '='
    i_val      = 'US' ).

* Filterausdruck anlegen: Zielland Japan
ri_filter_to_jp =
  ri_expr_factory->create_operator_expr(
    i_attr1    = 'COUNTRYTO'
    i_operator = '='
    i_val      = 'JP' ).

* Filterausdruck anlegen: Oder-Verknüpfung der Zielländer
ri_filter_to =
  ri_expr_factory->create_or_expr(
    i_expr1 = ri_filter_to_us
    i_expr2 = ri_filter_to_jp ).

* Filterausdruck anlegen: Und-Verknüpfung des Abfluglandes
* und der beiden alternativen Zielländer
ri_filter =
  ri_expr_factory->create_and_expr(
    i_expr1 = ri_filter_from_de
    i_expr2 = ri_filter_to ).

* Gesamten Filterausdruck der Abfrage zuordnen
ri_query->set_filter_expr( ri_filter ).

* Abfrage ausführen
ta_ro_spfli =
  rf_ca_spfli->if_os_ca_persistency~get_persistent_by_query(
    ri_query ).
```

Listing 4.9 Definition und Verwendung eines Filterausdrucks

Um die einzelnen Objekte anzulegen, die den Baum mit der Filterbedingung bilden sollen, benötigen Sie zunächst wieder eine Referenz auf die Ausdrucks-Factory. Auf der Ausdrucks-Factory rufen Sie dann nacheinander jeweils eine Methode auf, mit der Sie ein Ausdrucksobjekt einer bestimmten Art anlegen.

Bei komplexeren Filterbedingungen, die wie in Listing 4.9 aus mehreren Objekten bestehen, müssen Sie den Baum nach der Bottom-up-Vorgehensweise aufbauen: Sie erzeugen zunächst die Blätter des Baums, das heißt die Bedingungen, die sich direkt auf ein persistentes Attribut beziehen. Dabei müssen Sie die Namen der Attribute ausschließlich in Großbuchstaben übergeben. Erst danach legen Sie die inneren Knoten an, die die bereits existierenden Objekte mit den logischen Operatoren `AND`, `OR` und `NOT` verknüpfen.

Die Wurzel des Baums, in Abbildung 4.1 das oberste dargestellte Objekt, weisen Sie schließlich der Abfrage über ihre Methode `SET_FILTER_EXPR` als Filterbedingung zu. Da nur die Wurzel über Referenzen mit allen Bestandteilen des Baums verbunden ist, ist sie das einzige Objekt, aus dem der Query-Dienst die gesamte Filterbedingung ableiten kann.

Gegenüberstellung der beiden Varianten zur Definition der Filterbedingung

Die beiden geschilderten Varianten zur Definition der Filterbedingung sind in ihrer Ausdrucksmächtigkeit identisch, das heißt jede Filterbedingung, die Sie mit einer der beiden Varianten erstellen können, könnten Sie auch mit der anderen Variante formulieren. Es wird empfohlen, Filterbedingungen beim Anlegen der Abfrage zu definieren, da dies verglichen mit der Verwendung eines Filterausdrucks in der Regel deutlich komfortabler zu implementieren ist und zu besser lesbarem Quelltext führt. Lediglich wenn Sie die Filterbedingung zur Laufzeit dynamisch zusammenstellen wollen, kann es sinnvoll sein, den Filterausdruck zu verwenden.

4.4.3 Festlegen der Sortierbedingung

Mit der Sortierbedingung einer Abfrage können Sie festlegen, in welcher Reihenfolge die persistenten Objekte nach der Abfrage in der Ergebnistabelle sortiert sind. Dazu können Sie sich auf ein oder mehrere persistente Attribute beziehen und zu jedem Attribut angeben, ob die Sortierung aufsteigend oder absteigend erfolgen soll.

Wenn Sie keine Sortierbedingung angeben, geben die Object Services die persistenten Objekte in der Reihenfolge zurück, in der das Datenbanksystem die entsprechenden Datensätze zurückgeliefert hat. In vielen Fällen sind die Datensätze dabei aufsteigend nach dem Primärschlüssel der Datenbanktabelle sortiert. Da jedoch nicht garantiert ist, dass sich jedes System so verhält, sollten Sie die Sortierbedingung immer explizit angeben, wenn es für Ihre Anwendung relevant ist, in welcher Reihenfolge die persistenten Objekte sortiert sind.

Die Sortierbedingung können Sie auf zwei Arten festlegen:

- Sie übergeben beim Anlegen der Abfrage der Methode `CREATE_QUERY` den Parameter `I_ORDERING`.
- Sie definieren über die Ausdrucks-Factory einen Sortierausdruck und übergeben diesen der Methode `SET_ORDERING_EXPR` der Abfrage.

Definition der Sortierbedingung beim Anlegen der Abfrage

Dem Parameter `I_ORDERING` der Methode `CREATE_QUERY` müssen Sie einen String übergeben, in dem jeweils der Name eines persistenten Attributes gefolgt von dem Schlüsselwort `ASCENDING` (aufsteigend sortiert) oder von dem Schlüsselwort `DESCENDING` (absteigend sortiert) steht. Die persistenten Attribute und die Schlüsselwörter zur Sortierreihenfolge müssen jeweils durch ein Leerzeichen voneinander getrennt sein. Die Groß-/Kleinschreibung ist bei dieser Variante irrelevant.

Die Reihenfolge, in der Sie mehrere persistente Attribute angeben, ist ausschlaggebend dafür, in welcher Reihenfolge das System die einzelnen Attribute bei der Sortierung berücksichtigt: Das System sortiert die persistenten Objekte zunächst nach dem ersten angegebenen persistenten Attribut. Falls notwendig und angegeben, werden danach die weiteren persistenten Attribute zur Sortierung herangezogen.

Listing 4.10 zeigt ein Beispiel für eine Sortierbedingung mit zwei persistenten Attributen. Mit der im Listing angelegten Abfrage sollen Flüge aus dem Flugdatenmodell instanziert werden. Die Object Services sollen die Flüge zunächst aufsteigend nach dem Flugdatum sortieren. Frühere Flüge sollen demnach in der Ergebnistabelle weiter vorn stehen als spätere Flüge. Alle Flüge am selben Datum sortiert die Abfrage absteigend nach dem Attribut `SEATSMAX`. Sie gibt daher zuerst die Flüge mit den meisten Sitzplätzen in der Economy Class zurück und zuletzt die Flüge mit der geringsten Kapazität.

```
ri_query =
  ri_query_manager->create_query(
    i_ordering = 'FLDATE ASCENDING SEATSMAX DESCENDING' ).
```

Listing 4.10 Definition der Sortierbedingung beim Anlegen der Abfrage

Definition der Sortierbedingung über einen Sortierausdruck

Genau wie bei den Parameternamen und der Filterbedingung besteht auch bei der Sortierbedingung die Möglichkeit, zunächst eine Abfrage anzulegen, in der keine Sortierbedingung definiert ist, und dann über die Ausdrucks-Factory einen *Sortierausdruck* anzulegen.

Auf einem Sortierausdruck können Sie zwei Methoden aufrufen: eine Methode zum Hinzufügen eines persistenten Attributes, nach dem aufsteigend sortiert werden soll (`APPEND_ASCENDING`), und eine Methode zum Hinzufügen eines persistenten Attributes, nach dem absteigend sortiert werden soll (`APPEND_DESCENDING`). Den Namen des persistenten Attributes müssen

Sie jeweils ausschließlich in Großbuchstaben übergeben. Auch hier entscheidet die Reihenfolge, in der Sie die persistenten Attribute hinzufügen, darüber, in welcher Reihenfolge die persistenten Attribute für die Sortierung herangezogen werden.

Listing 4.11 definiert über einen Sortierausdruck dieselbe Sortierbedingung wie Listing 4.10. Dazu erfolgt ein Aufruf der Methode `CREATE_ORDERING_EXPR` der Ausdrucks-Factory, die einen Sortierausdruck anlegt. Daraufhin werden dem Sortierausdruck über die jeweiligen Methoden für aufsteigende und absteigende Sortierung nacheinander die Namen der persistenten Attribute übergeben, nach denen die Object Services das Ergebnis der Abfrage sortieren sollen. Der Sortierausdruck wird schließlich der Abfrage zugewiesen.

```
DATA: ri_query_manager TYPE REF TO if_os_query_manager,
      ri_query         TYPE REF TO if_os_query,
      ri_expr_factory  TYPE REF TO if_os_query_expr_factory,
      ri_ordering_expr TYPE REF TO if_os_query_ordering_expr.

ri_query_manager = cl_os_system=>get_query_manager( ).

* Abfrage anlegen
ri_query = ri_query_manager->create_query( ).

* Referenz auf Ausdrucks-Factory ermitteln
ri_expr_factory = ri_query->get_expr_factory( ).

* Sortierausdruck anlegen
ri_ordering_expr = ri_expr_factory->create_ordering_expr( ).

* Persistente Attribute festlegen, nach denen sortiert wird:
* - Aufsteigend nach dem Flugdatum
* - Absteigend nach der maximalen Belegung in der
*   Economy Class
ri_ordering_expr->append_ascending( 'FLDATE' ).
ri_ordering_expr->append_descending( 'SEATSMAX' ).

* Sortierausdruck der Abfrage zuordnen
ri_query->set_ordering_expr( ri_ordering_expr ).
```

Listing 4.11 Definition der Sortierbedingung über einen Sortierausdruck

Gegenüberstellung der beiden Varianten zur Definition der Sortierbedingung

Auch bei der Sortierbedingung besteht kein Unterschied in der Ausdrucksmächtigkeit der beiden Varianten, mit denen Sie die Sortierbedingung for-

mulieren können. Sie können jegliche Sortierbedingung grundsätzlich immer mit beiden Varianten formulieren. Die Variante beim Anlegen der Abfrage ist in der Regel komfortabler und übersichtlicher, besonders bei konstanten Sortierbedingungen. Die Variante der Definition über einen Sortierausdruck kann eventuell geeigneter sein, wenn Sie die Sortierbedingung erst zur Laufzeit dynamisch aufbauen wollen.

4.4.4 Übergabe von konkreten Werten für die Abfrageparameter

Zu allen Abfrageparametern, die Sie definiert und in der Filterbedingung auch tatsächlich verwendet haben, müssen Sie beim Aufruf der Abfrage den konkreten Wert übergeben, mit dem Sie die Abfrage ausführen möchten. Nur wenn Sie in der Filterbedingung keinen Bezug auf Abfrageparameter genommen haben, können Sie eine Abfrage ausführen, ohne dabei Werte für Abfrageparameter angeben zu müssen.

Für die Übergabe von Parameterwerten existieren eine vereinfachte Möglichkeit mit begrenztem Funktionsumfang und eine Möglichkeit für komplexere Abfragen:

- Wenn Sie maximal drei Parameter verwenden und keine persistenten Referenzen in der Abfrage auswerten, können Sie der Methode `IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_QUERY` über die Importing-Parameter `I_PAR1` bis `I_PAR3` die Werte der Abfrageparameter übergeben.
- Eine interne Tabelle mit Datenreferenzen können Sie der Methode `IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_QUERY` in jedem Fall übergeben, das heißt auch in Verbindung mit persistenten Referenzen und mit beliebig vielen Parametern.

Übergabe einzelner Parameter: I_PAR1 bis I_PAR3

Die Parameterschnittstelle der Methode `IF_OS_CA_PERSISTENCY~GET_PERSISTENT_BY_QUERY` enthält drei Importing-Parameter mit den Namen `I_PAR1`, `I_PAR2` und `I_PAR3`, über die Sie konkrete Werte für die verwendeten Parameter übergeben können. Da die Importing-Parameter mit dem Typ `ANY` definiert sind, können Sie ihnen beliebige Werte elementaren Typs übergeben. Die Übergabe einer Referenz auf ein persistentes Objekt ist dagegen nicht möglich.

Listing 4.12 instanziert alle Flüge, die im Oktober 2009 stattfinden. Dazu bezieht sich die Filterbedingung auf die standardmäßig definierten Parame-

ter PAR1 und PAR2. Die konkreten Werte, in diesem Fall der 1. und der 31. Oktober 2009, werden erst beim Ausführen der Abfrage übergeben. Ohne ein neues Abfrageobjekt anlegen zu müssen, können Sie dieselbe Abfrage danach beliebig oft erneut ausführen und dabei jeweils die Flüge aus einem anderen Datumsintervall anfordern.

```
DATA: ri_query_manager TYPE REF TO if_os_query_manager,
      ri_query         TYPE REF TO if_os_query,
      ta_ro_flights    TYPE osreftab,
      rf_ca_sflight    TYPE REF TO /iot/ca_sflight.

rf_ca_sflight = /iot/ca_sflight=>agent.
ri_query_manager = cl_os_system=>get_query_manager( ).

* Abfrage anlegen
ri_query =
  ri_query_manager->create_query(
    i_filter = 'FLDATE >= PAR1 AND FLDATE <= PAR2' ).

* Abfrage ausführen
ta_ro_flights =
  rf_ca_sflight->if_os_ca_persistency~get_persistent_by_query(
    i_query = ri_query
    i_par1  = '20091001'
    i_par2  = '20091031' ).
```

Listing 4.12 Abfrage mit Übergabe einzelner Parameter

Um bei der Ausführung der Abfrage die Importing-Parameter I_PAR1 bis I_PAR3 verwenden zu können, müssen Sie nicht unbedingt mit Abfrageparametern mit den Namen PAR1 bis PAR3 arbeiten. Die Importing-Parameter I_PAR1 bis I_PAR3 beziehen sich auf jeweils den ersten, zweiten und dritten definierten Abfrageparameter, unabhängig von seinem Namen. Haben Sie demnach zwei Abfrageparameter mit selbst festgelegten Namen definiert, können Sie diesen beim Ausführen der Abfrage über die Importing-Parameter I_PAR1 und I_PAR2 die konkreten Werte zuordnen.

Übergabe einer internen Tabelle mit Abfrageparametern: I_PARAMETER_TAB

Die Alternative zur Übergabe von Werten für Abfrageparameter steht Ihnen mit dem Importing-Parameter I_PARAMETER_TAB zur Verfügung. Über diesen Importing-Parameter können Sie eine interne Tabelle übergeben, die in jeder Zeile eine Datenreferenz auf den Wert enthält, den Sie dem Abfragepa-

rameter übergeben wollen. Mit einer Datenreferenz können Sie sowohl auf eine Variable, eine Konstante oder ein Literal elementaren Typs verweisen als auch auf eine Referenz auf ein Objekt.

Auch bei dieser Art der Übergabe von Abfrageparametern erfolgt kein direkter Bezug zu den Namen der Abfrageparameter. Entscheidend ist auch hier die Reihenfolge, in der Sie die Namen der Abfrageparameter definiert haben, und die Reihenfolge der Datenreferenzen in der internen Tabelle: Die Datenreferenz auf den Wert für den zuerst definierten Abfrageparameter muss in der ersten Zeile stehen, die Datenreferenz auf den Wert für den danach definierten Parameter in der zweiten Zeile und so weiter.

Wie Sie eine interne Tabelle mit Datenreferenzen befüllen und mit dieser Tabelle eine Abfrage ausführen, zeigt Ihnen Listing 4.13.

```
DATA: dr_rf_airport    TYPE REF TO data,
      rf_ca_airport    TYPE REF TO /iot/ca_sairport,
      rf_ca_counter    TYPE REF TO /iot/ca_scounter,
      rf_airport       TYPE REF TO /iot/cl_sairport,
      ri_query_manager TYPE REF TO if_os_query_manager,
      ri_query         TYPE REF TO if_os_query,
      ta_parameters    TYPE osdreftab,
      ta_ro_counters   TYPE osreftab.

rf_ca_airport = /iot/ca_sairport=>agent.
rf_ca_counter = /iot/ca_scounter=>agent.
ri_query_manager = cl_os_system=>get_query_manager( ).

* Abfrage anlegen
ri_query =
  ri_query_manager->create_query(
    i_filter = 'RF_AIRPORT EQUALSREF PAR1').

* Flughafen Frankfurt instanzieren
rf_airport = rf_ca_airport->get_persistent( i_id = 'FRA' ).

* Datenreferenz auf Flughafenobjekt an Parametertabelle
* hängen
GET REFERENCE OF rf_airport INTO dr_rf_airport.
APPEND dr_rf_airport TO ta_parameters.

* Abfrage ausführen
ta_ro_counters =
  rf_ca_counter->if_os_ca_persistency~get_persistent_by_query(
    i_query         = ri_query
    i_parameter_tab = ta_parameters ).
```

Listing 4.13 Abfrage mit Übergabe einer internen Tabelle mit Parametern

Die Filterbedingung der Abfrage enthält den Operator `EQUALSREF`, das heißt eine Bedingung mit Bezug auf eine persistente Referenz. In diesem Fall soll die Abfrage alle Verkaufsschalter von Fluggesellschaften an einem bestimmten Flughafen ermitteln. Dazu nimmt die Filterbedingung Bezug auf das Attribut `RF_AIRPORT`, das zu jedem Schalter eine persistente Referenz auf den Flughafen enthält, an dem sich der Schalter befindet.

Um eine Bedingung mit einer persistenten Referenz in einer Abfrage verwenden zu können, benötigen Sie eine Referenz auf ein bereits instanziertes persistentes Objekt. Für die Ermittlung der Schalter am Flughafen Frankfurt wird daher das entsprechende persistente Objekt instanziert.

Die Anweisung `GET REFERENCE OF` erzeugt eine Datenreferenz, die Sie für die interne Tabelle mit den Werten der Abfrageparameter benötigen. Angewendet auf die Referenz auf ein persistentes Objekt, erhalten Sie eine zweistufige Referenz, hier eine Datenreferenz auf eine Referenz auf ein persistentes Objekt. Diese Datenreferenz müssen Sie noch an die interne Tabelle mit den Werten der Abfrageparameter anhängen, bevor Sie die Tabelle bei der Ausführung der Abfrage dem Importing-Parameter `I_PARAMETER_TAB` übergeben.

Gegenüberstellung der beiden Varianten zur Übergabe von konkreten Werten für die Abfrageparameter

Die Übergabe der einzelnen Parameter `I_PAR1` bis `I_PAR3` ist deutlich einfacher zu nutzen als die interne Tabelle mit Parametern `I_PARAMETER_TAB`. Die interne Tabelle sollten Sie daher nur verwenden, wenn die Übergabe der einzelnen Parameter nicht infrage kommt, das heißt bei mehr als drei Parametern oder bei einem Bezug auf eine persistente Referenz.

4.5 Vergleich von Query-Dienst und Open SQL

Sie können von zwei Vorteilen profitieren, wenn Sie zur Instanzierung von persistenten Objekten eine einfache Abfrage mit dem Query-Dienst verwenden, anstatt mit einer `SELECT`-Abfrage in Open SQL die Schlüssel von persistenten Objekten zu ermitteln und diese danach zu laden: Der von Ihnen zu implementierende ABAP-Quelltext ist deutlich kompakter, und der gesamte Vorgang kommt mit einem einzigen Datenbankzugriff aus. Letzteres reduziert die Belastung von Applikationsserver und Datenbanksystem sowie die Kommunikation zwischen beiden, und führt damit gegebenenfalls zu einer deutlich schneller ablaufenden Anwendung.

Da bei der Verwendung des Query-Dienstes die Schlüssel der Objekte auf dem Applikationsserver nicht bekannt sind, bevor die Objekte instanziert sind, haben Sie keine Möglichkeit, vor der Instanzierung mit den Schlüsseln der Objekte zu arbeiten. Wie in Kapitel 8, »Integration des SAP-Sperrkonzeptes und der Object Services«, noch näher erklärt wird, ist dies insbesondere im Zusammenspiel mit dem SAP-Sperrkonzept ein Nachteil.

In der Ausdrucksmächtigkeit unterscheiden sich die Abfragen mit dem Query-Dienst und die `SELECT`-Abfragen in Open SQL deutlich. Der Query-Dienst instanziert immer vollständige persistente Objekte. Eine `SELECT`-Klausel zur Angabe der zu übertragenden Felder existiert daher ebenso wenig wie Äquivalente zu den `GROUP BY`- und `HAVING`-Klauseln oder Aggregatfunktionen. Auch Subquerys oder Joins über beliebige Tabellen gehören nicht zum Funktionsumfang des Query-Dienstes.

Eine `FROM`-Klausel benötigen Sie bei der Verwendung des Query-Dienstes nicht, da der Query-Dienst auf die Informationen aus der Persistenzabbildung zurückgreift, in der die zugrunde liegenden Datenbanktabellen hinterlegt sind. Auch in Bezug auf persistente Referenzen nutzt der Query-Dienst im System bereits vorhandene Informationen und reduziert so den Entwicklungsaufwand im Vergleich zu einer äquivalenten Abfrage in Open SQL.

Open SQL ist ein integraler Bestandteil der Programmiersprache ABAP. Daher kann das System schon zur Entwicklungszeit die Syntax von Abfragen in Open SQL prüfen, wenn diese keine dynamischen Bestandteile enthalten. Die syntaktische Korrektheit einer Filterbedingung einer Abfrage mit dem Query-Dienst kann das System dagegen immer erst zur Laufzeit prüfen.

Sowohl die Selektion mit dem Query-Dienst als auch die Selektion mit Open SQL basieren ausschließlich auf den Daten in der Datenbank. Wenn Sie im laufenden Programm persistente Objekte angelegt, geändert oder gelöscht und diese Änderungen noch nicht in die Datenbank geschrieben haben, basiert das Ergebnis der Selektion noch auf dem ursprünglichen Zustand ohne Einbeziehung der Änderungen. Im folgenden Abschnitt wird beschrieben, wie Sie mithilfe der Funktionalität der Klassenakteure bei der Selektion auch die Änderungen berücksichtigen können, die Sie zunächst nur an persistenten Objekten im Speicher der laufenden Anwendung durchgeführt haben.

4.6 Umgang mit neu erstellten und geänderten Objekten

Jeder Klassenakteur einer persistenten Klasse enthält Methoden, mit denen Sie jeweils die persistenten Objekte der Klasse ermitteln können, die sich in einem bestimmten Verwaltungszustand befinden. Auf diese Art können Sie unter anderem alle persistenten Objekte der Klasse eruieren, die Sie in der laufenden Top-Level-Transaktion angelegt, geändert oder gelöscht haben.

Die Methoden sind im Interface `IF_OS_CA_INSTANCE` definiert. Ihre Namen beginnen jeweils mit dem Präfix `GET_`, und bis auf eine Ausnahme folgt darauf der Name des jeweiligen Verwaltungszustands. Diese Ausnahme bildet die Methode `GET_CREATED`, die die persistenten Objekte im Verwaltungszustand `NEW` ermittelt.

Die Methoden liefern jeweils nur Objekte, die direkt zur jeweiligen persistenten Klasse gehören, die der Klassenakteur verwaltet. Objekte, die sich in der Verwaltung eines Klassenakteurs einer Sub- oder Superklasse befinden, liefern die Methoden nicht zurück.

Listing 4.14 zeigt, wie Sie das Ergebnis einer Abfrage so anpassen, dass es auch neu angelegte Objekte berücksichtigt. Die Abfrage ermittelt alle Flüge, die am aktuellen Datum stattfinden. Nach der Ausführung der Abfrage werden mit der Methode `GET_CREATED` alle Referenzen auf neu angelegte Objekte ermittelt. In einer `LOOP`-Schleife wird dann manuell jeder neu angelegte Flug darauf hin geprüft, ob er der Filterbedingung entspricht. In diesem Fall wird dazu mit der Zugriffsmethode `GET_FLDATE` das Flugdatum ausgelesen und mit dem aktuellen Datum verglichen. Entspricht das Flugdatum dem aktuellen Datum, wird der neu angelegte Flug der internen Tabelle mit allen zuvor ermittelten Flügen am aktuellen Tag hinzugefügt.

```
DATA: rf_ca_sflight    TYPE REF TO /iot/ca_sflight,
      rf_sflight       TYPE REF TO /iot/cl_sflight,
      ri_query_manager TYPE REF TO if_os_query_manager,
      ri_query         TYPE REF TO if_os_query,
      ro_sflight       TYPE REF TO object,
      ta_ro_sflights   TYPE osreftab,
      ta_ro_created    TYPE osreftab.

rf_ca_sflight = /iot/ca_sflight=>agent.
ri_query_manager = cl_os_system=>get_query_manager( ).
```

```
* Abfrage anlegen
ri_query =
  ri_query_manager->create_query(
    i_filter = 'FLDATE = PAR1' ).

* Abfrage ausführen
ta_ro_sflights =
  rf_ca_sflight->if_os_ca_persistency~get_persistent_by_query(
    i_query = ri_query
    i_par1  = sy-datum ).

* Neu angelegte persistente Objekte ermitteln
ta_ro_created = rf_ca_sflight->if_os_ca_instance~get_created( ).

* Jedes neu angelegte persistente Objekt gegen die Filter-
* bedingung prüfen
LOOP AT ta_ro_created INTO ro_sflight.
* Typecast: OBJECT -> /IOT/CL_SFLIGHT
  rf_sflight ?= ro_sflight.

  IF rf_sflight->get_fldate( ) = sy-datum.
* Objekt entspricht der Filterbedingung
* -> Zur Ergebnistabelle hinzufügen
    APPEND rf_sflight TO ta_ro_sflights.
  ENDIF.
ENDLOOP.
```

Listing 4.14 Manuelle Berücksichtigung neu angelegter Objekte

Um auch geänderte Objekte bei einer Selektion zu berücksichtigen, müssen Sie zwei Konstellationen betrachten:

- Ein persistentes Objekt kann durch die Änderung der Filterbedingung entsprechen, während der bisherige Stand in der Datenbank der Filterbedingung noch nicht entspricht. Diesen Fall berücksichtigen Sie analog zu Listing 4.14 und ermitteln dabei die geänderten Objekte über die Methode `GET_CHANGED`. Zusätzlich müssen Sie dabei beachten, dass ein geändertes Objekt auch schon zuvor der Filterbedingung entsprochen haben kann. Sie müssen daher sicherstellen, dass Sie der Ergebnistabelle nicht ein bereits enthaltenes Objekt erneut hinzufügen.
- Im umgekehrten Fall entspricht zwar der bisherige Stand in der Datenbank der Filterbedingung, das geänderte Objekt im Speicher der laufenden Anwendung jedoch nicht mehr. Um solche Objekte aus dem Ergebnis der Abfrage zu entfernen, sollten Sie zunächst den Verwaltungszustand jedes gefundenen Objektes prüfen (Methode `IF_OS_CA_INSTANCE~GET_`

STATUS, siehe Abschnitt 3.4, »Verwaltungszustände persistenter Objekte«). Alle Objekte im Verwaltungszustand CHANGED müssen Sie über Zugriffsmethoden manuell gegen die Filterbedingung prüfen. Entspricht ein Objekt nicht mehr der Filterbedingung, entfernen Sie es aus dem Ergebnis.

Im laufenden Programm gelöschte Objekte berücksichtigen die Object Services bis zu einem gewissen Grad automatisch. Versuchen Sie über die Masseninstanzierung oder über eine Abfrage mit dem Query-Dienst, ein Objekt zu instanzieren, das in der Datenbank noch existiert, das im laufenden Programm aber bereits gelöscht wurde, tritt eine Ausnahme der Klasse CX_OS_OBJECT_NOT_FOUND auf. Das gelöschte Objekt wird daher automatisch nicht instanziert. Mit dem Auftreten der Ausnahme bricht jedoch auch die Instanzierung weiterer Objekte ab.

Sie erhalten demnach in einem solchen Fall kein Ergebnis, das um gelöschte Objekte bereinigt ist, sondern gar kein Ergebnis. Um trotz vorhandener gelöschter Objekte alle Objekte zu instanzieren, die nicht gelöscht sind, können Sie auf einen Ansatz wie in Listing 4.1 zurückgreifen: Mit einer Abfrage in Open SQL bestimmen Sie die Schlüssel der Objekte und instanzieren diese dann einzeln. Dabei übernehmen Sie dann nur die persistenten Objekte ins Ergebnis, bei deren Instanzierung keine Ausnahme der Klasse CX_OS_OBJECT_NOT_FOUND auftritt.

Ab dem Enhancement Package 2 für das Release 7.0 des AS ABAP ist es außerdem möglich, das Verhalten des Query-Dienstes in Bezug auf gelöschte Objekte zu steuern. Übergeben Sie bei der Ausführung einer Abfrage die Option IGNORE_DELETED, löst der Query-Dienst keine Ausnahme aus, wenn er auf gelöschte Objekte trifft. Stattdessen liefert die Abfrage ein Ergebnis, das bereits um die gelöschten Objekte bereinigt ist.

Neu erstellte und geänderte Objekte müssen Sie bei der Selektion von persistenten Objekten typischerweise in komplexeren Anwendungen berücksichtigen, die im Rahmen einer Top-Level-Transaktion viele persistente Objekte verändern. Ist es in Ihrer Anwendung möglich, dass Sie Abfragen durchführen, nachdem Sie in der laufenden Top-Level-Transaktion bereits persistente Objekte geändert haben, liefern die Abfragen ohne die Berücksichtigung von neu erstellten und geänderten Objekten kein inhaltlich korrektes Ergebnis. Ist Ihre Anwendung dagegen so strukturiert, dass Sie in jeder Top-Level-Transaktion zunächst alle beteiligten Objekte laden und erst danach damit beginnen, Änderungen umzusetzen, müssen Sie bei der Selektion keine Sonderbehandlung für neu angelegte und geänderte Objekte durchführen.

4.7 Zusammenfassung

In diesem Kapitel haben Sie erfahren, wie Sie persistente Objekte nach beliebigen Kriterien selektieren können. In einigen Fällen ist es möglich, dazu den Query-Dienst der Object Services zu verwenden. Sie haben dabei gelernt, in welchen Fällen es notwendig und sinnvoll ist, statt des Query-Dienstes auf die Open-SQL-Anweisungen zurückzugreifen, die schon im klassischen ABAP verfügbar waren.

Wir haben außerdem bereits einen ersten Vorschlag für eine Erweiterung der Funktionalität der Object Services erläutert: die Berücksichtigung von Objekten im Speicher bei der Selektion. Im nächsten Kapitel lernen Sie mit der internen Funktionsweise der Object Services die Grundlagen für die Beschreibung weiterer Erweiterungen kennen, die Thema der anschließenden Kapitel 6 bis 8 sind.

Index

A

ABAP Debugger 179
Abbildung, objektrelationale 11, 20, 167, 171, 173, 182, 202, 217
Abfrage 93, 95, 96, 99
Abfrageparameter 99, 102, 105, 112, 113
abstrakte Klasse 46, 122, 190
ACTIVE_CLASS_AGENT 140, 146
AGENT 32, 33, 69, 123, 129
Aktivierung 26, 30, 129, 178
aktuelle Transaktion 64, 134
ALV Grid Control 157
AND 106
Änderbarkeit 24, 57, 152
APPEND_ASCENDING 110
APPEND_DESCENDING 110
Append-Struktur 26
ASCENDING 110
asynchrone Verbuchung 72, 74, 75
Attribut
  persistentes 30
  transientes 30, 56, 86, 129, 153, 159, 161, 167, 184
  Typ 26, 42, 44
Ausdrucks-Factory 100, 105, 109, 110

B

Base Agent 121, 129, 155, 201
Beziehung, Ist-ein 45
Business-Key 24, 27, 30, 32, 34, 52, 53, 54, 55, 91, 93, 125, 152, 155, 201

C

CALL FUNCTION IN UPDATE TASK 73
Casting-Operator 33, 34, 35, 52
CHANGED 84, 89, 119, 144, 160, 206, 208
Check Agent 76, 79, 81, 131, 135, 199, 203, 204, 207, 212
CL_ABAP_WEAK_REFERENCE 133
CL_OS_CA_COMMON 122
CL_OS_STATE 137
  SET_OBJECT_FROM_STATE 138
  SET_STATE_FROM_OBJECT 138
CL_OS_SYSTEM 61, 68, 74, 96, 121, 126, 139, 146
  ACTIVE_CLASS_AGENT 140
  EXTERNAL_COMMIT 141
  GET_CLASS_AGENT 139
  GET_CLASS_AGENT_INFO 140
  GET_QUERY_MANAGER 96
  GET_TRANSACTION_MANAGER 61
  INIT_AND_SET_MODES 68, 74
  INIT_STATE 141
CL_OS_TRANSACTION 134, 207
  CHAINED 135
  CHECK_AGENTS 135
  DATA_SAVE_STATE 135
  FLAG_UPDT_MODE_CHANGED 136
  SUBTRANSACTION 135
  SUPER_TRANSACTION 135
  TRANSACTION_STATE 136
  UNDO_RELEVANT 136
  UPDATE_MODE 136
CL_OS_TRANSACTION_MANAGER 134
  CURRENT_TRANSACTION 134
  TOP_TRANSACTION 134
CL_SYSTEM_UUID 27
Class Agent 31
Class Builder 20, 21, 177, 179, 182
COMMIT WORK 59, 65, 73, 75, 79, 88, 135, 192
CREATE_PERSISTENT 54, 55, 82, 85
CREATE_STRUCTURE 154
CREATE_TRANSIENT 56, 82
CREATED 117
CX_DYNAMIC_CHECK 36, 55, 65
CX_NO_CHECK 37, 68, 75
CX_OS_CHECK_AGENT_FAILED 77, 200, 212
CX_OS_CLASS_NOT_FOUND 37
CX_OS_DB 75
CX_OS_NO_IMPLEMENTATION 37, 55
CX_OS_OBJECT_EXISTING 55
CX_OS_OBJECT_NOT_FOUND 36, 38, 43, 57, 95, 119

CX_OS_SYSTEM 68
CX_OS_TRANSACTION 65, 75
CX_SY_DYN_CALL_ILLEGAL_METHOD 149
CX_SY_MOVE_CAST_ERROR 44
CX_SY_NO_HANDLER 44

D

Daten
persistente 13
transiente 12
Datenbank-LUW 59, 189
Datenbankzustand 124
Datenkapselung 39
DELETE_PERSISTENT 83, 85
DELETED 84, 89, 95, 119, 144, 208
DESCENDING 110
direkte Verbuchung 71, 74, 75, 192, 195
Downcast 35, 94, 185, 206

E

Enhancement Framework 168, 175, 178, 182, 184
Entwurfsmuster, Singleton 31, 70, 123
EQUALSREF 104, 115
Erweiterung 176, 182, 185
Implementierung 176, 177
Option 175, 185
ESCAPE 103
exklusive Sperre 193, 194, 196, 199, 202, 204, 207, 212, 213
EXTERNAL_COMMIT 141

F

Filter
Ausdruck 102, 105
Bedingung 95, 99, 101, 112, 115, 117, 185, 204, 218
Flugdatenmodell 17, 32, 35, 40, 48, 91, 110, 124, 156, 158, 187
Funktionsbaustein
SYSTEM_UUID_CREATE 27

G

Garbage Collection 121, 123, 131, 147
Generatoreinstellung 86, 128, 129
GET_PERSISTENT 35, 37, 85
gewöhnliche Referenz 132
Globally Unique Identifier 27
GUID 27

H

HANDLER_CHANGED 205, 208
horizontales Mapping 45, 46, 48
Hülle 38, 43, 57, 63, 83, 85, 87, 89

I

IF_OS_CA_INSTANCE 82, 117, 126, 210
GET_STATUS 82, 210
IF_OS_CA_PERSISTENCY 32, 34, 37, 91, 93, 97, 112, 125, 155
GET_PERSISTENT_BY_KEY 34, 37, 91, 125, 155
GET_PERSISTENT_BY_KEY_TAB 93, 125
GET_PERSISTENT_BY_OID 32, 37
GET_PERSISTENT_BY_OID_TAB 94
GET_PERSISTENT_BY_QUERY 97, 112
IF_OS_CHECK 76, 131, 206, 207
IS_CONSISTENT 76, 207
IF_OS_FACTORY 51, 52, 53, 55, 83, 85, 125, 144, 146, 155, 185, 204
CREATE_PERSISTENT 52
CREATE_PERSISTENT_BY_KEY 53, 55, 125, 155
DELETE_PERSISTENT 83, 85
REFRESH_PERSISTENT 85, 144, 185, 204
RELEASE 83, 85, 146
IF_OS_FACTORY~CREATE_TRANSIENT 56
IF_OS_FACTORY~CREATE_TRANSIENT_BY_KEY 56
IF_OS_INSTANCE_MANAGER 126
IF_OS_PERSISTENCY_MANAGER
GET_PERSISTENT_BY_QUERY 127

IF_OS_STATE 128, 137, 186
GET 137
INIT 130
INVALIDATE 130, 186
SET 137
IF_OS_TRANSACTION 134
END 62, 64, 66, 77, 79, 80, 88, 192
END_AND_CHAIN 62, 81, 89, 135
GET_STATUS 80
REGISTER_CHECK_AGENT 77
START 61, 80
UNDO 62, 64, 66, 77, 79, 80, 88, 136, 192, 207
UNDO_AND_CHAIN 62, 81, 89, 135, 136
IF_OS_TRANSACTION_MANAGER 134
CREATE_TRANSACTION 61, 80
GET_CURRENT_TRANSACTION 64
GET_TOP_TRANSACTION 65
IGNORE_DELETED 119
INIT_STATE 141
Initialisierungsblock, statischer 70
Instanz-GUID 27, 30, 32, 34, 36, 39, 40, 41, 44, 52, 53, 54, 55, 91, 93, 123, 128, 153, 202
Instanzierung 20, 31
IS NULL 105
Ist-ein-Beziehung 45

K

Klasse
abstrakte 46, 122, 190
persistente 13, 19, 20, 21, 23, 26, 28, 30, 31, 38
Klassenakteur 31, 32, 35, 37, 39, 48, 51, 53, 54, 55, 58, 67, 82, 83, 85, 91, 93, 97, 116, 117, 121, 126, 129, 132, 136, 139, 143, 146, 155, 171, 176, 182, 185, 194, 196, 199, 200, 205, 210, 212, 214, 218
Klassen-GUID 40, 41, 44, 48, 127, 140
Klassenidentifikator 25, 41
Klasseninitialisierer 70
Klassenkonstruktor 69, 70, 123, 129
Kompatibilitätsmodus 65, 74, 75, 78, 82, 89, 135, 141, 192, 206
Konstruktor, statischer 70

L

Lazy Loading 38, 43, 167, 183
Lesen 24, 152
LIKE 103, 105
LOADED 84, 86, 88, 126, 132, 144, 147, 201
Logical Unit of Work 59
lokale Verbuchung 71, 75
LUW 59

M

Mapping
horizontales 45, 46, 48
objektrelationales 11
vertikales 45, 48
Mapping Assistant 21
Mehr-Tabellen-Mapping 29, 46
MOVE-CORRESPONDING 148, 150

N

NEW 80, 83, 89, 117, 144, 208
NOT 106
NOT LOADED 83, 84, 88, 123, 130, 132, 144, 147, 201, 213
NULL 95, 104, 105, 133, 135
Nummernkreisobjekt 28, 55

O

Object Navigator 20
OBJECT_INFO 123, 132
OBJECT_TO_STRUCTURE 148
Objekt
persistentes 13, 19, 27, 31, 38, 51
transientes 36, 55, 83, 88, 126
objektorientierte Transaktion 60, 88
objektorientierter Transaktionsmodus 65, 70, 89, 135, 192, 206
Objektreferenz 25, 41
objektrelationale Abbildung 11, 20, 167, 171, 173, 182, 202, 217
objektrelationales Mapping 11
OO-Transaktion 70

OO-Transaktionsmodell 70
Open SQL 91, 93, 102, 115, 119, 123, 159, 191, 202, 218
Operator-Ausdruck 105
optimistische Sperre 193, 196, 199, 200, 203, 204, 207, 213
optimistisches Sperrverfahren 192, 196, 199, 200, 203, 205, 208, 212, 214
OR 106
OS_GUID 25, 27, 40, 48, 94, 153
OSCON 74, 80, 82, 210
OSREFTAB 94
OSTYP_CA_INFO 140

P

Parameterausdruck 100
persistente Daten 13
persistente Klasse 13, 19, 20, 21, 23, 26, 28, 30, 31, 38
persistente Referenz 26, 28, 40
persistentes Attribut 30
persistentes Objekt 13, 19, 27, 31, 38, 51
Persistenzabbildung 21, 22, 26, 27, 29, 30, 32, 34, 35, 37, 39, 40, 41, 45, 48, 51, 53, 55, 57, 86, 94, 96, 104, 116, 122, 128, 152, 155, 169, 173, 201
Persistenzdienst 13, 19, 23, 27, 29, 30, 31, 35, 36, 37, 38, 40, 42, 43, 44, 48, 51, 52, 53, 54, 55, 57, 58, 60, 63, 64, 67, 74, 81, 85, 86, 88, 93, 98, 121, 122, 128, 132, 143, 153, 155, 169, 171, 175, 183, 201, 204
pessimistisches Sperrverfahren 192, 194, 198, 203, 212, 213
Plausibilitätsprüfung 160, 167, 168, 183
Polymorphie 49, 50
Präfix
  /IOT/ 17
  *DEQUEUE_* 190
  *ENQUEUE_* 190
  *GET_* 23, 38, 117, 149, 150, 169, 181
  *OSCON_DMODE_* 74
  *OSCON_OSTATUS_* 82
  *OSCON_SSTATUS_* 135
  *OSCON_TSTATUS_* 80
  *SET_* 57, 151, 169, 181
Private 24
Protected 24
Public 24, 97

Q

Query 93, 95
Query-Dienst 13, 93, 95, 99, 115, 119, 127, 158, 164, 185, 203, 218
Query-Manager 96, 126

R

Referenz
  Ausdruck 105
  gewöhnliche 132
  persistente 26, 28, 40
  schwache 132
  starke 132
REFRESH_ALL_OBJECTS 145
REFRESH_OBJECTS_BY_AGENT 145
REFRESH_PERSISTENT 85, 144, 185, 204
RELEASE 83, 85, 146
ROLLBACK WORK 60, 62, 65, 88, 192
RTTC 149, 153, 155
RTTI 149
RTTS 149
Run Time Type Creation 149, 153, 155
Run Time Type Identification 149
Run Time Type Services 149

S

SAP Control Framework 157
SAP Help Portal 21
SAP LUW 58, 59, 62, 65, 73, 79, 82, 85, 88, 191, 194
SAP NetWeaver Application Server ABAP 11, 18, 19, 60, 130, 147, 182, 187, 189, 218
  Release 6.10 19, 36, 181
  Release 6.40 153
  Release 6.40 SP16 86
  Release 7.0 92, 175, 181
  Release 7.0 EhP2 27, 29, 34, 87, 119

Release 7.0 SP13 94
Release 7.0 SP6 86
Schreibsperre 193
schwache Referenz 132
_SCOPE 191, 192
SE24 20
SE80 20
SE93 70
SEOCLSKEY 127
SET UPDATE TASK LOCAL 72
SET_MODE_UNDO_RELEVANT 79
Sichtbarkeit 24, 58, 97
Private 24, 39, 169
Protected 24, 39, 123, 155
Public 24, 149, 155, 169
Singleton 31, 70, 123
Sortierausdruck 109, 110
Sortierbedingung 95, 99, 109, 110, 164
SPECIAL_BKEY_TAB 125
SPECIAL_OBJECT_INFO 125
SPECIAL_OID_TAB 125
Sperre
exklusive 193, 194, 196, 199, 202, 204, 207, 212, 213
optimistische 193, 196, 199, 200, 203, 204, 207, 213
Wandlung 193, 196, 199, 202, 204, 207, 212, 214
Sperrkonzept 14, 116, 187
Sperrmodus 192, 193, 202
Sperrverfahren 192
optimistisches 192, 196, 199, 200, 203, 205, 208, 212, 214
pessimistisches 192, 194, 198, 203, 212, 213
starke Referenz 132
STATE_WRITE_ACCESS 87
statischer Initialisierungsblock 70
statischer Konstruktor 70
STRUCTURE_TO_OBJECT 150
Subtransaktion 63, 67, 79, 88, 134, 206, 207
Suffix
_W_QUOTES 106
synchrone Verbuchung 72, 74, 75
SYSTEM_UUID_CREATE 27
Systemzustand 141

T

Top-Level-Transaktion 62, 64, 66, 75, 78, 82, 85, 86, 88, 117, 130, 134, 165, 191, 194, 196, 201, 205, 207, 213
Transaktion 58
aktuelle 64, 134
objektorientierte 60, 88
übergeordnete 64, 135
untergeordnete 135
Verkettung 62, 135
Transaktionscode 59, 70, 74
SE24 20
SE80 20
SE93 70
Transaktionsdienst 13, 51, 52, 58, 134
Transaktionsmanager 61, 64, 68, 80, 126, 134, 165, 206
Transaktionsmodell 70
Transaktionsmodus 65, 69, 70, 141, 165
Kompatibilitätsmodus 65
objektorientierter 65, 70, 89, 135, 192, 206
Transaktionsstatus 80, 136
TRANSIENT 84, 88, 147
transiente Daten 12
transientes Attribut 30, 56, 86, 129, 153, 159, 161, 167, 184
transientes Objekt 36, 55, 83, 88, 126
Typ, OS_GUID 25, 27
TYP_BUSINESS_KEY 125, 155
Typgruppe 74
Typidentifikator 25, 48, 97, 153
Typobjekt 149, 151, 154

U

übergeordnete Transaktion 64, 135
Undo-Mechanismus 78, 81, 88, 136
Universally Unique Identifier 27
untergeordnete Transaktion 135
UUID 27

V

Verbuchung
asynchrone 72, 74, 75
direkte 71, 74, 75, 192, 195
lokale 71, 75
synchrone 72, 74, 75
Verbuchungsmodus 71, 74, 75, 136
Verkettungsmethode 62, 67, 79, 81, 89
vertikales Mapping 45, 48
Verwaltungszustand 36, 51, 81, 117, 124, 132, 137
Verwendungsnachweis 49

W

Web Dynpro ABAP 160
Wertattribut 24, 54, 152
WHERE-Klausel 95, 102
WRITE_ACCESS 139

Z

Zuordnungstyp 24, 41, 48, 128