ABAP-Entwickler möchten über das SAP HANA Studio angelegte Views und Datenbankprozeduren in ABAP nutzen. Außerdem sind sie ein leistungsfähiges Transportwesen gewöhnt und erwarten einen konsistenten Transport nativer SAP-HANA-Entwicklungsobjekte über das Change and Transport System.

5 Einbindung nativer SAP-HANA-Entwicklungsobjekte in ABAP

In Kapitel 4, »Native Datenbankentwicklung mit SAP HANA«, haben wir Ihnen gezeigt, wie Sie analytische Modelle (Views) und Datenbankprozeduren über das SAP HANA Studio anlegen können. Nun möchten wir Ihnen erklären, wie Sie diese nativen HANA-Objekte aus ABAP heraus aufrufen können.

Außerdem möchten wir Ihnen erläutern, wie Sie ABAP-Programme, die native HANA-Objekte nutzen, konsistent in Ihrer Systemlandschaft transportieren können.

5.1 Einbindung von analytischen Views

Sie haben in den vorangegangenen Abschnitten gelernt, wie Sie die unterschiedlichen Arten von Views im SAP HANA Studio modellieren können und wie Sie mit dem Data Preview oder in Microsoft Excel auf die Ergebnisse des Views zugreifen können. Wir haben Ihnen in Abschnitt 4.4.4, »Laufzeitobjekte und SQL-Zugriff«, auch bereits gezeigt, wie Sie über SQL die generierten Column Views adressieren können.

Zugriff vor und mit ABAP-Release 7.4

In diesem Abschnitt gehen wir auf den Zugriff aus ABAP ein. Wir müssen dabei zwischen ABAP-Release 7.4 und älteren Versionen unterscheiden. Vor ABAP 7.4 ist die einzige Zugriffsmöglichkeit die über natives SQL, was wir Ihnen in Abschnitt 5.1.1, »Zugriff über natives SQL«, kurz vorstellen werden. Ab ABAP 7.4 können Sie die

Views aus dem SAP HANA Repository in das ABAP Dictionary importieren und danach auf diese über Open SQL zugreifen. Darauf gehen wir in den Abschnitten 5.1.2, »Externe Views im ABAP Dictionary«, und 5.1.3, »Zugriffsmöglichkeiten auf externe Views«, im Detail ein. Im letzten Abschnitt geben wir Ihnen einige Empfehlungen, Tipps und Tricks für die SAP-HANA-View-Modellierung.

5.1.1 Zugriff über natives SQL

Bei allen vorgestellten View-Typen in SAP HANA entsteht bei der Aktivierung ein Column View im Datenbankkatalog im Schema `_SYS_BIC` mit einem öffentlichen Synonym (*Public Synonym*), z. B. `'test.a4h.book.chapter04::AT_FLIGHT'`.

Über diesen Bezeichner können Sie aus ABAP auf diesen View zugreifen. Listing 5.1 zeigt den Zugriff auf den Attribute View `AT_FLIGHT`, den wir in Abschnitt 4.4.1, »Attribute Views«, angelegt haben über ADBC.

```
" Definition der Resultatstruktur
TYPES: BEGIN OF ty_data,
         carrid   TYPE s_carr_id,
         connid   TYPE s_conn_id,
         fldate   TYPE s_date,
         route    TYPE string,
       END OF ty_data.

CONSTANTS:  gc_view TYPE string VALUE
                'test.a4h.book.chapter04::AT_FLIGHT'.
DATA: lt_data TYPE TABLE OF ty_data.

" Zugriff auf Attribute View
DATA(lv_statement) =
   | SELECT carrid, connid, fldate, route |
&& |  FROM "{ gc_view }"|
&& |  WHERE mandt = '{ sy-mandt }' ORDER BY fldate|.

TRY.
   " SQL-Verbindung und Statement vorbereiten
   DATA(lo_result_set) =
      cl_sql_connection=>get_connection(
         )->create_statement(
              tab_name_for_trace = conv #( gc_view )
         )->execute_query( lv_statement ).
```



```
   " Resultat abholen
   lo_result_set->set_param_table( REF #( lt_data ) ).
   lo_result_set->next_package( ).
   lo_result_set->close( ).
  CATCH cx_sql_exception INTO DATA(lo_ex).
    " Fehlerbehandlung
&& |  WHERE mandt = '{ sy-mandt }' ORDER BY fldate|.
    WRITE: | { lo_ex->get_text( ) } |.
ENDTRY.

LOOP AT lt_data ASSIGNING FIELD-SYMBOL(<l>).
  WRITE: / <l>-carrid , <l>-connid, <l>-fldate,
           <l>-route .
ENDLOOP.
```

Listing 5.1 Zugriff auf einen Attribute View über ADBC

Wie Sie sehen, handelt es sich um einen herkömmlichen nativen SQL-Zugriff. Falls es bei der Ausführung zu einem Fehler kommt, erhalten Sie im Text der SQL-Exception Hinweise auf die Ursache. Neben den SQL-Code-Fehlern, die Sie auch beim Zugriff über die SQL-Konsole sehen, können auch Fehler bei der Abbildung in die ABAP-Zielstruktur auftreten. Wir kommen darauf in den Empfehlungen in Abschnitt 5.1.4 zurück.

5.1.2 Externe Views im ABAP Dictionary

In ABAP 7.4 gibt es im ABAP Dictionary einen neuen View-Typ, einen sogenannten *externen View*, der es erlaubt, im SAP HANA Repository definierte Views in das ABAP Dictionary zu importieren. Der View wird als extern bezeichnet, weil er nicht vollständig im ABAP Dictionary definiert ist, sondern als eine Art *Proxy* (also Stellvertreter) dient, über den ein Zugriff aus ABAP auf den zugehörigen Column View im Schema `_SYS_BIC` möglich ist.

Externen View in Eclipse anlegen

Einen externen View können Sie ausschließlich über die ABAP Development Tools in Eclipse definieren. Dazu legen Sie ein neues Entwicklungsobjekt vom Typ Dictionary View an. Abbildung 5.1 zeigt den Anlagedialog anhand des Attribute Views `AT_FLIGHT`.

Prüfung der Importierbarkeit

Während der Anlage wird geprüft, ob sich der View in das ABAP Dictionary importieren lässt. Sie müssen dabei beachten, dass nicht alle HANA-Datentypen in ABAP unterstützt werden. Wenn Sie also berechnete Attribute definieren oder in Views auf Tabellen zugreifen,

die nicht über das ABAP Dictionary angelegt wurden, können potenziell solche nicht unterstützen Datentypen auftauchen. In diesem Fall erhalten Sie einen Fehler beim Anlegen des externen Views und können diesen View nicht importieren. Die Liste der unterstützten Datentypen entnehmen Tabelle 3.1 in Abschnitt 3.1.3, »Datentypen«.

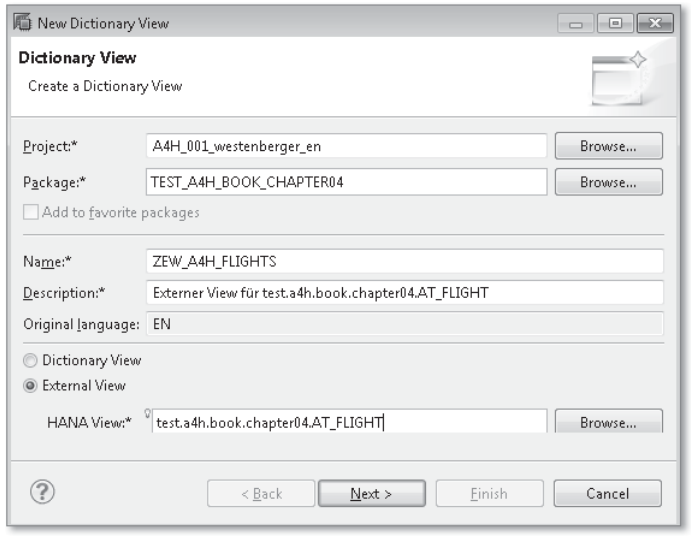

Abbildung 5.1 Anlegen eines externen Views im ABAP Dictionary

View-Struktur und Synchronisierung

Nach erfolgreichem Import des SAP HANA Views in das ABAP Dictionary zeigt der Editor die Struktur des Views mit Abbildung der Datentypen (siehe Abbildung 5.2). Zusätzlich gibt es in dem Dialog einen Button zum Synchronisieren (SYNCHRONIZE) des Views nach einer Änderung der Struktur des zugehörigen Views im SAP HANA Studio. Falls Sie also Attribute in die Ausgabestruktur aufnehmen, Attribute löschen oder Datentypen ändern, müssen Sie den externen View abgleichen, da es ansonsten zu Laufzeitfehlern kommt. Empfehlungen zur Synchronisierung von Entwicklungen in einem Entwicklungsteam geben wir Ihnen in Kapitel 14, »Praxistipps«.

Abbildung von Datentypen

Wie Sie in Abschnitt 3.1.3, »Datentypen«, erfahren haben, ist die Abbildung von SQL-Datentypen auf Dictionary-Typen nicht eindeutig. Vom Datentyp hängt aber die richtige Behandlung von Operationen ab (z. B. die Berechnung von Differenzen bei einem Datum). Aus diesem Grund müssen Sie als Entwickler die Zuordnung des richtigen ABAP-Datentyps manuell vornehmen.

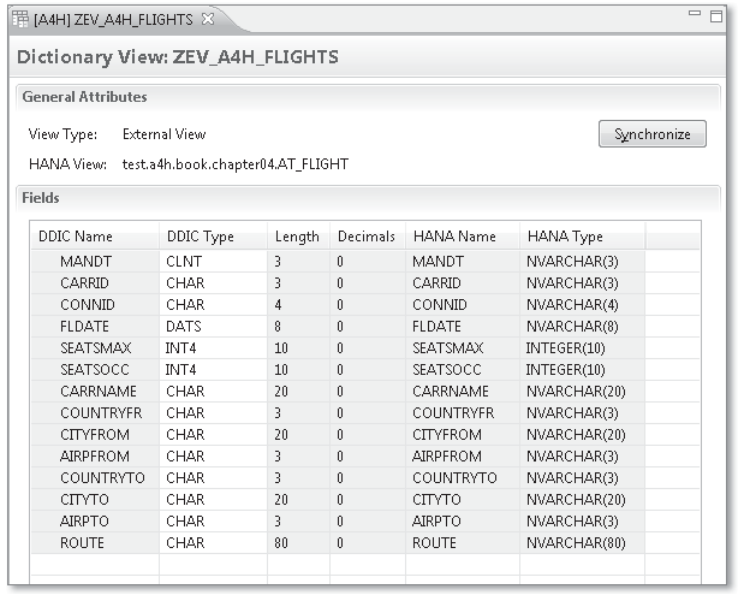

Abbildung 5.2 Externer ABAP Dictionary View, basierend auf einem Attribute View

Tabelle 5.1 zeigt anhand des Beispiel-Views `AT_FLIGHT` die möglichen Datentypzuordnungen für einige Spalten.

Spalte	SQL-Datentyp	Mögliche Dictionary-Typen
`CARRID`	`NVARCHAR(3)`	`CHAR(3)`, `NUMC(3)`, `SSTR`, `CLNT`, `UNIT`, `CUKY`
`FLDATE`	`NVARCHAR(8)`	`CHAR(8)`, `NUMC(8)`, `SSTR`, `DATS`
`CARRNAME`	`NVARCHAR(20)`	`CHAR(20)`, `NUMC(20)`, `SSTR`

Tabelle 5.1 Beispiel für mögliche Typzuordnungen

Bei dem externen View in Abbildung 5.2 haben wir daher manuell der Spalte `FLDATE` den ABAP-Datentyp `DATS` zugeordnet. Dies mag auf den ersten Blick merkwürdig erscheinen, da diese Information ja in der zugrunde liegenden Dictionary-Tabelle bereits vorliegt, die Spalten von Column Views in SAP HANA haben jedoch keine für das ABAP Dictionary ersichtliche Referenz auf Spalten existierender Tabellen. Zum Beispiel könnte die Spalte `FLDATE` auch ein berechnetes Attribut sein.

Voraussetzungen

Die Definition eines externen Views, der auf einem Analytic View oder Calculation View basiert, funktioniert wie auf der Basis eines

Attribute Views. Das ABAP Dictionary verfügt bei einem externen View aktuell über kein Wissen darüber, um welchen speziellen HANA-View-Typ es sich handelt. Voraussetzung für die Verwendung externer Views ist lediglich, dass der View im SAP HANA Repository definiert ist. Column Views, die lediglich im Datenbankkatalog vorhanden sind (z. B. durch eine Generierung), können Sie nicht in das ABAP Dictionary importieren.

Auf den Transport von externen Views (und anderen HANA-spezifischen Entwicklungen) gehen wir in Abschnitt 5.3, »Transport nativer Entwicklungsobjekte«, ein.

5.1.3 Zugriffsmöglichkeiten auf externe Views

Vorteile

Der wesentliche Vorteil externer Views besteht darin, dass ein Zugriff auf SAP HANA Views auch über Open SQL möglich ist. Sie können daher insbesondere von folgenden Qualitäten in Open SQL profitieren:

- Syntaxprüfung durch den ABAP Compiler und Vorschlagswerte bei der Entwicklung (*Code Completion*)
- automatische Mandantenbehandlung
- Iterieren über eine Ergebnismenge innerhalb einer `SELECT`-Schleife
- Verwendung des Ausdrucks `INTO CORRESPONDING FIELDS` zur passenden Selektion in eine Zielstruktur unabhängig von der Reihenfolge in der Projektionsliste
- Verwendung von `IN` bei der `WHERE`-Bedingung zur Übertragung von Selektionsoptionen

Zugriff per Open SQL

Listing 5.2 setzt den Zugriff auf den externen View aus Abbildung 5.2 um und entspricht funktional der ADBC-Zugriffsvariante aus Listing 5.1. Wie Sie sehen, hat sich der für den Zugriff notwendige ABAP-Code deutlich reduziert und entspricht einem Zugriff auf einen Standard-Dictionary-View.

```
REPORT ZR_A4H_CHAPTER4_VIEW_OPEN.

DATA: wa TYPE zev_a4h_flights.
" Daten von externem View lesen
SELECT carrid connid fldate route
       FROM zev_a4h_flights
```



```
       INTO CORRESPONDING FIELDS OF wa.
  WRITE: / wa-carrid, wa-connid, wa-fldate, wa-route.
ENDSELECT.
```

Listing 5.2 Zugriff auf externen View über Open SQL

Mögliche Laufzeitfehler beim Zugriff auf externe Views [!]

Auch bei dem Open-SQL-basierten Zugriff auf einen externen View wird letztlich eine SQL-Abfrage auf den zugehörigen Column View in SAP HANA ausgeführt. Dabei gelten die gleichen Regeln und Restriktionen wie bei einem Zugriff mit nativem SQL.

Wie wir in Abschnitt 4.4.4, »Laufzeitobjekte und SQL-Zugriff«, erklärt haben, müssen Sie etwa beim Zugriff auf Analytic Views über SQL einige Einschränkungen beachten. Eine nicht unterstützte Anfrage über Open SQL führt zu einem Laufzeitfehler. Dies erfordert von Ihnen als ABAP-Entwickler eine gewisse Vorsicht, da beim normalen Open-SQL-basierten Zugriff auf ABAP-Tabellen solche Fehler eher selten vorkommen. Auf die Werkzeuge zur Fehleranalyse und die möglichen Laufzeitfehler bei einem SQL-Zugriff kommen wir in Abschnitt 7.2, »Fehleranalyse«, zurück.

Nativer Zugriff über ADBC

Neben dem Open-SQL-basierten Zugriff können Sie externe Views auch über natives SQL ansprechen. Diese auf den ersten Blick etwas merkwürdig erscheinende Variante ist dennoch sinnvoll, falls Sie etwa über eine SQL-Abfrage auf einen SAP HANA View zugreifen wollen, die über Open SQL nicht möglich ist. Ein Beispiel ist eine *Fuzzy-Suche* in einem Attribute View (siehe Abschnitt 10.4, »Einsatz der Textsuche in ABAP«). Im Vergleich zu einem Zugriff über natives SQL auf den generierten Column View im Schema `_SYS_BIC` hat der externe View den Vorteil, dass es dabei im ABAP Dictionary bereits eine geeignete Zielstruktur für eine Selektion über ADBC gibt.

5.1.4 Empfehlungen

Zum Abschluss dieses Abschnitts wollen wir Ihnen einige Empfehlungen für die Verwendung von SAP HANA Views geben. Wir beschränken uns dabei auf funktionale Empfehlungen. Für Werkzeuge und Empfehlungen zur Performanceanalyse verweisen wir auf Kapitel 7, »Laufzeit- und Fehleranalyse auf SAP HANA«, und Kapitel 14, »Praxistipps«, wo wir auch Designaspekte wie Namenskonventionen aufgreifen werden.

Verwendung der View-Typen

Falls der Funktionsumfang von Standard-ABAP-Dictionary-Views für Sie ausreicht und Sie diese in der Vergangenheit verwendet haben,

gibt es keine Notwendigkeit, Ihre Anwendung auf einen nativen SAP HANA View umzubauen. Im nächsten Kapitel werden Sie über CDS-Views eine Möglichkeit kennenlernen, wie Sie auch direkt in ABAP komplexe Sichten mit berechneten Feldern definieren können.

Für spezielle analytische Szenarien bieten allerdings die modellierten SAP HANA Views einen einfachen Zugang. Welcher der drei vorgestellten View-Typen in SAP HANA für Ihr Szenario der richtige ist, können Sie über den folgenden Fragenkatalog bestimmen:

- Geht es um eine Sicht auf Stammdaten, die eventuell durch berechnete Attribute erweitert werden? Hier sollte der Attribute View der Startpunkt sein.
- Handelt es sich um eine Datenanalyse von Bewegungsdaten, basierend auf einem Sternschema? In diesem Fall sollte der Analytic View der erste Anlaufpunkt sein, wobei Sie die Dimensionen als Attribute Views realisieren.
- Müssen Sie Ergebnisse aus verschiedenen Tabellen und HANA-Views zusammenbringen oder adaptieren? In diesem Fall bietet sich der modellierte Calculation View an. Falls die modellierte Variante für einen Teil Ihres Szenarios nicht ausreicht, können Sie für diesen Teil auf eine SQLScript-basierte Implementierung ausweichen.

Mandantenbehandlung

Bei der Modellierung von Views sollten Sie auf die richtige Behandlung des Mandanten achten. Wir empfehlen Ihnen auch, stets das Mandantenfeld als erstes Feld des Views aufzunehmen und sicherzustellen, dass Sie den Mandanten in die Join-Modellierung aufgenommen haben. In den meisten Fällen sollte der Konfigurationswert SESSION CLIENT für Views, die auf ABAP-Tabellen aus dem gleichen System basieren, die richtige Einstellung sein. Falls Tabellen durch eine Replikation aus einem anderen System entstanden sind, kann auch ein fester Wert für den Mandanten sinnvoll sein. Nur in wenigen Fällen macht hingegen ein Zugriff über Mandantengrenzen hinweg Sinn.

Schemabehandlung

Sie sollten für Analytic und Calculation Views stets das richtige Standardschema auswählen. In diesem Schema wird insbesondere das für Konvertierungen relevante Customizing gesucht, wenn keine spezielle Einstellung am Attribut vorgenommen wurde. Noch wichtiger ist die Einstellung des Standardschemas bei der implementierten Variante von Calculation Views.

Definition von externen Views

Sie sollten externe Views nur für solche SAP HANA Views definieren, über die Sie aus ABAP zugreifen wollen, da Sie diese bei Änderungen manuell synchronisieren müssen. Ebenso ist es nicht empfehlenswert, mehrere externe Views für einen SAP HANA View zu definieren.

Fehleranalyse

Falls Sie bei der Aktivierung von SAP HANA Views Fehlermeldungen erhalten, finden Sie in der Regel im Fehlertext Hinweise auf die Ursache. Die richtige Interpretation der Fehlermeldung erfordert jedoch in manchen Fällen ein wenig Erfahrung. Aus diesem Grund ist eine gewisse Heuristik bei der Fehleranalyse sinnvoll. Zunächst sollten Sie sicherstellen, dass Sie bei Attribute Views mindestens ein Feld als Schlüsselfeld ausgezeichnet sowie bei Analytic Views mindestens eine Kennzahl definiert haben. Falls es berechnete Attribute in Ihrem View gibt, sollten Sie prüfen, ob Sie eventuell bei dem zugehörigen Ausdruck einen Fehler gemacht haben.

Falls Sie bei der Fehleranalyse einmal nicht weiterkommen, können Sie das zugehörige Attribut (z. B. in einer Kopie des Views) probeweise einmal entfernen. Falls Sie beim Aufruf des Data Previews eine Fehlermeldung oder unerwartete Daten bekommen, deutet das in vielen Fällen auf einen Fehler bei der Join-Modellierung hin. Im Fall von Währungskonvertierungen kann es durch einen fehlenden Mandantenkontext zu Fehlern kommen.

Beim Zugriff auf einen SAP HANA View aus ABAP über natives SQL sollten Sie den Namen des Views mitgeben (über den Parameter `tab_name_for_trace` wie in Listing 5.1 oder über die Methode `SET_TABLE_NAME_FOR_TRACE`), der in Support-Szenarien eine einfachere Fehleranalyse erlaubt.

5.2 Einbettung von nativen Prozeduren in ABAP

In Kapitel 4, »Native Datenbankentwicklung mit SAP HANA«, haben Sie gelernt, was SQLScript ist und wie Sie es zur Implementierung von Datenbankprozeduren nutzen können. Nun möchten wir Ihnen erklären, wie Sie Datenbankprozeduren aus ABAP aufrufen. Dabei unterscheiden wir zwei Möglichkeiten:

- Zugriff über *natives SQL* und *ABAP Database Connectivity* (*ADBC*, siehe auch Kapitel 3, »Datenbankprogrammierung mit dem SAP NetWeaver AS ABAP«)
- Nutzung von sogenannten *Database Procedure Proxies*

Voraussetzungen

Der Aufruf von Datenbankprozeduren in SAP HANA über ADBC ist ab dem ABAP-Release 7.0 und SAP-Kernel 7.20 möglich. Die Database Procedure Proxies stehen ab dem Release 7.4 zur Verfügung. Sie setzen voraus, dass SAP HANA als Primärdatenbank genutzt wird. Außerdem unterstützen Database Procedure Proxies ausschließlich das – eigentlich veraltete – XML-Dateiformat (*.procedure*).

5.2.1 Zugriff über natives SQL

Wie in Abschnitt 4.3, »Datenbankprozeduren«, beschrieben, erzeugt das System bei der Aktivierung einer Datenbankprozedur verschiedene Laufzeitobjekte im Datenbankkatalog (z. B. im Schema `_SYS_BIC`) sowie ein öffentliches Synonym. Darüber können Sie über natives SQL aus ABAP auf die Datenbankprozedur zugreifen.

Nachteile von nativem SQL

Die Verwendung von nativem SQL zum Aufruf einer Datenbankprozedur ist allerdings verhältnismäßig umständlich und fehleranfällig. Tabellarische Eingabe- und Ausgabeparameter können Sie, wie Sie im weiteren Verlauf dieses Abschnitts sehen werden, nur über temporäre Tabellen mit der Datenbankprozedur austauschen. Außerdem erkennt der SAP NetWeaver AS ABAP Syntaxfehler in nativen SQL-Anweisungen erst zur Laufzeit. Für mehr Details verweisen wir Sie auf die Erläuterungen in Kapitel 3, »Datenbankprogrammierung mit dem SAP NetWeaver AS ABAP«.

Beispiele

Wir beschreiben Ihnen die Verwendung von nativem SQL zum Zugriff auf Datenbankprozeduren anhand mehrerer Beispiele im Detail. Zunächst betrachten wir dazu eine Datenbankprozedur, die auf Basis der Kurzbezeichnung den Namen einer Fluggesellschaft ermittelt. Für die anschließend folgenden Beispiele greifen wir auf bereits bekannte Datenbankprozeduren aus Kapitel 4, »Native Datenbankentwicklung mit SAP HANA«, zurück.

Beispiel 1: Aufruf einer Datenbankprozedur

Für den Aufruf einer Datenbankprozedur über ADBC stellt die Klasse `CL_SQL_STATEMENT` die Methode `EXECUTE_PROCEDURE` zur Verfügung.

Diese können Sie nutzen, solange eine Datenbankprozedur keine tabellarischen Eingabe-/Ausgabeparameter hat.

Beispiel für Aufruf

Das Programm `ZR_A4H_CHAPTER5_CARRNAME_ADBC` zeigt die Verwendung der Methode `EXECUTE_PROCEDURE` beispielhaft (siehe Listing 5.3). Es ruft die Datenbankprozedur `DETERMINE_CARRNAME` auf. Diese hat folgende Eingabe- und Ausgabeparameter:

- `IV_MANDT` (Mandant)
- `IV_CARRID` (Kurzbezeichnung einer Fluggesellschaft)
- `EV_CARRNAME` (Name einer Fluggesellschaft)

```
PARAMETERS: p_carrid TYPE s_carr_id.

DATA: lo_sql_statement  TYPE REF TO cl_sql_statement,
      lv_carrname TYPE s_carrname.

TRY.
  " create statement
  lo_sql_statement =
  cl_sql_connection=>get_connection(
  )->create_statement( ).

  " bind parameters
  lo_sql_statement->set_param( data_ref =
  REF #( sy-mandt )
  inout = cl_sql_statement=>c_param_in ).

  lo_sql_statement->set_param( data_ref =
  REF #( p_carrid )
  inout = cl_sql_statement=>c_param_in ).

  lo_sql_statement->set_param( data_ref =
  REF #( lv_carrname )
  inout = cl_sql_statement=>c_param_out ).

  " call procedure
  lo_sql_statement->execute_procedure(
 '"test.a4h.book.chapter04::DETERMINE_CARRNAME"' ).

CATCH cx_sql_exception INTO DATA(lo_ex).
  " error handling
  WRITE: | { lo_ex->get_text( ) } |.
ENDTRY.

WRITE: / lv_carrname.
```

Listing 5.3 Aufruf einer Datenbankprozedur über natives SQL

Erläuterung des Programms

Das Programm erzeugt zunächst eine Instanz der Klasse `CL_SQL_STATEMENT`. Danach belegt es die Eingabe- und Ausgabeparameter der Datenbankprozedur durch Aufruf der Methode `SET_PARAM` mit den Aktualparametern. Anschließend ruft es die Methode `EXECUTE_PROCEDURE` auf.

Beispiel 2: Tabellarische Ausgabeparameter

Alternativ können Sie eine Datenbankprozedur auch über die Methode `EXECUTE_QUERY` (zusammen mit dem Zusatz `WITH OVERVIEW`) ausführen. Dies funktioniert auch bei Datenbankprozeduren mit tabellarischen Eingabe- und Ausgabeparametern.

Beispiel für Ausgabeparameter

Das Programm `ZR_A4H_CHAPTER5_TOP_ADBC` in Listing 5.4 zeigt die Verwendung der Methode `EXECUTE_QUERY` beispielhaft, indem es die Datenbankprozedur `DETERMINE_TOP_CONNECTIONS` aufruft. Diese ermittelt die wichtigsten Verbindungen einer Fluggesellschaft und hat folgende Eingabe- und Ausgabeparameter:

- `IV_MANDT` (Mandant)
- `IV_CARRID` (Kurzbezeichnung einer Fluggesellschaft)
- `IV_ALGORITHM` (steuert, wie die wichtigsten Verbindungen ermittelt werden sollen)
- `ET_CONNECTIONS` (Tabellenparameter; enthält die Kurzbezeichnung `CARRID` der Fluggesellschaft sowie den Verbindungscode `CONNID`)

```
PARAMETERS: p_carrid TYPE s_carr_id.

" Definition der Resultatstruktur
TYPES: BEGIN OF ty_connections,
         carrid TYPE s_carr_id,
         connid TYPE s_conn_id,
       END OF ty_connections.

DATA: lt_connections TYPE TABLE OF ty_connections,
      lv_statement TYPE string,
      lo_result_set TYPE REF TO cl_sql_result_set,
      lo_connections TYPE REF TO data.

TRY.
    " lokale temporäre Tabelle löschen
    lv_statement = | DROP TABLE #ET_CONNECTIONS |.
    cl_sql_connection=>get_connection(
  )->create_statement( )->execute_ddl( lv_statement ).
```



```
  CATCH cx_sql_exception.
    " unter Umständen existiert die lokale temporäre
    " Tabelle nicht, diesen Fehler ignorieren wir
ENDTRY.

TRY.
    " lokale temporäre Tabelle anlegen
    lv_statement = | CREATE LOCAL TEMPORARY ROW|
    && | TABLE #ET_CONNECTIONS LIKE "_SYS_BIC".|
    && |"test.a4h.book.chapter04::GlobalTypes.t|
    && |t_connections" |.
    cl_sql_connection=>get_connection(
    )->create_statement( )->execute_ddl( lv_statement ).

    " Datenbankprozedur aufrufen
    lv_statement = | CALL "test.a4h.bo|
    && |ok.chapter04::DETERMINE_TOP_CONNECTIONS|
    && |"( '{ sy-mandt }' , '{ p_carrid }', 'P'|
    && |, #ET_CONNECTIONS ) WITH OVERVIEW |.
    lo_result_set = cl_sql_connection=>get_connection(
      )->create_statement( )->execute_query(
      lv_statement ).
    lo_result_set->close( ).

    " lokale temporäre Tabelle auslesen
    lv_statement = | SELECT * FROM #ET_CONNECTIONS |.
    lo_result_set = cl_sql_connection=>get_connection(
      )->create_statement( )->execute_query(
      lv_statement ).

    " Resultat auslesen
    GET REFERENCE OF lt_connections INTO
      lo_connections.
    lo_result_set->set_param_table( lo_connections ).
    lo_result_set->next_package( ).
    lo_result_set->close( ).
  CATCH cx_sql_exception INTO DATA(lo_ex).

    " Fehlerbehandlung
    WRITE: | { lo_ex->get_text( ) } |.
ENDTRY.

LOOP AT lt_connections ASSIGNING
  FIELD-SYMBOL(<ls_connections>).
  WRITE: / <ls_connections>-carrid ,
           <ls_connections>-connid.
ENDLOOP.
```

Listing 5.4 Behandlung von tabellarischen Ausgabeparametern

Temporäre Tabellen

Wir möchten Ihnen anhand des Programms vor allem die Übergabe tabellarischer Eingabe- und Ausgabeparameter an eine Datenbankprozedur erläutern: Das Programm `ZR_A4H_CHAPTER5_TOP_ADBC` verwendet zur Übergabe des Tabellenparameters `ET_CONNECTIONS` die *temporäre Tabelle* `#ET_CONNECTIONS`.

[»] Temporäre Tabellen

Viele Datenbanken, so auch die HANA-Datenbank, bieten Ihnen die Möglichkeit, Zwischen- und Endergebnisse von Kalkulationen *vorübergehend* in sogenannten *temporären Tabellen* zu speichern. Temporäre Tabellen haben, verglichen mit herkömmlichen Tabellen, verschiedene Vorteile für diesen Anwendungsfall:

- Tabellendefinition und Tabelleninhalt werden von der Datenbank automatisch gelöscht, wenn sie nicht mehr benötigt werden.
- Die Datenbank isoliert die Daten paralleler Sitzungen (*Sessions*) automatisch voneinander. Sperren auf temporären Tabellen sind weder notwendig noch möglich.
- Für temporäre Tabellen schreibt die Datenbank kein Transaktionslog.
- Die Verwendung temporärer Tabellen ist in der Regel performanter als die Verwendung herkömmlicher Tabellen.

SAP HANA unterstützt globale und lokale temporäre Tabellen:

- Die Tabellendefinition *globaler* temporärer Tabellen ist sitzungsübergreifend nutzbar. Der Tabelleninhalt ist nur für die aktuelle Session sichtbar, er wird zum Sitzungsende von der Datenbank automatisch gelöscht.
- Bei *lokalen temporären Tabellen* sind sowohl Tabellendefinition als auch Tabelleninhalt nur für die aktuelle Session sichtbar, d. h., beide werden am Ende der Sitzung von der Datenbank automatisch gelöscht.

Nutzung über den AS ABAP

Hinsichtlich der Nutzung temporärer Tabellen für die Datenübergabe zwischen dem SAP NetWeaver AS ABAP und einer Datenbankprozedur sollten Sie folgende Dinge beachten:

- Wenn Sie mit globalen temporären Tabellen arbeiten, können Sie diese (da sie sitzungsübergreifend nutzbar sind) einmalig anlegen, müssen allerdings organisatorisch sicherstellen, dass der Tabellenname nicht für verschiedene Anwendungsfälle (die eine unterschiedliche Tabellenstruktur voraussetzen) verwendet wird.
- Die Anlage globaler temporärer Tabellen kann bereits zur *Designtime* erfolgen. Dann müssen Sie dafür sorgen, dass die Tabellen nach einem Transport auch in Test- und Produktivsystemen zur Verfügung stehen.

- Falls Sie sich zur Anlage globaler temporärer Tabellen zur *Laufzeit (Runtime)* entscheiden, müssen Sie vor jedem Aufruf einer Datenbankprozedur sicherstellen, dass die Tabellenstruktur zur Schnittstelle der aufgerufenen Datenbankprozedur passt (denn diese könnte sich ja zwischenzeitlich geändert haben).
- Lokale temporäre Tabellen müssen Sie zumindest einmalig pro Session anlegen (beachten Sie dazu auch die nachfolgenden Erläuterungen zum Verhältnis von ABAP-Workprozess und Datenbankverbindung). Daher können Sie lokale temporäre Tabellen nur zur Run Time eines ABAP-Programms anlegen.
- Da jeder ABAP-Workprozess über eine stehende Verbindung mit der Datenbank verbunden ist, stellen mehrere nacheinander durch den gleichen Workprozess ausgeführte ABAP-Programme aus Datenbanksicht eine Session dar. Nach Beendigung eines ABAP-Programms werden daher weder Definition noch Inhalt lokaler (und globaler) temporärer Tabellen automatisch gelöscht.
- Sowohl für globale als auch für lokale temporäre Tabellen sollten Sie den Inhalt (der aktuellen Sitzung) vor Aufruf der Datenbankprozedur löschen.

Erläuterung des Programms

Das Programm `ZR_A4H_CHAPTER5_TOP_ADBC` in Listing 5.4 arbeitet mit einer lokalen temporären Tabelle. Es löscht zunächst über `DROP TABLE #ET_CONNECTIONS` die eventuell bereits vorhandene lokale temporäre Tabelle `#ET_CONNECTIONS`. Anschließend legt es über die Anweisung `CREATE LOCAL TEMPORARY ROW TABLE` eine (neue) lokale temporäre Tabelle mit dem Namen `#ET_CONNECTIONS` an. Dabei bezieht sich das Programm auf den Tabellentyp, der beim Aktivieren der Datenbankprozedur vom System automatisch für den Ausgabeparameter `ET_CONNECTIONS` angelegt wurde. Durch dieses Vorgehen stellt das Programm vor dem Aufruf der Datenbankprozedur sicher, dass die temporäre Tabelle leer ist und zur aktuellen Struktur des Ausgabeparameters `ET_CONNECTIONS` passt.

Jetzt ruft das Programm die Datenbankprozedur über die Methode `EXECUTE_QUERY` auf. Dabei übergibt es `SY-MANDT`, `P_CARRID` und `'P'` an die Eingabeparameter und die temporäre Tabelle `#ET_CONNECTIONS` an den Ausgabeparameter der Datenbankprozedur.

Nach dem Aufruf der Datenbankprozedur liest das Programm den Inhalt der temporären Tabelle `#ET_CONNECTIONS` aus. Der Inhalt ent-

spricht den wichtigsten Verbindungen der übergebenen Fluggesellschaft.

Beispiel 3: Tabellarische Eingabeparameter

Wenn eine Datenbankprozedur über tabellarische Eingabeparameter verfügt, können Sie analog zu tabellarischen Ausgabeparametern vorgehen. Das Programm `ZR_A4H_CHAPTER5_KPIS_ADBC` in Listing 5.5 zeigt, wie Sie die Datenbankprozedur `GET_KPIS_FOR_CONNECTIONS` für eine Menge von Flugverbindungen aufrufen können. Die Datenbankprozedur ermittelt pro übergebener Verbindung einige Kennzahlen.

Beispiel für Eingabeparameter

Sie hat folgende Eingabe- und Ausgabeparameter:

- `IV_MANDT` (Mandant)
- `IT_CONNECTIONS` (Tabellenparameter; enthält die Kurzbezeichnung `CARRID` der Fluggesellschaft sowie den Verbindungscode `CONNID`)
- `ET_KPIS` (Tabellenparameter; enthält die Kennzahlen der Verbindungen)

```
...
LOOP AT lt_connections INTO ls_connections.
  lv_statement = | INSERT INTO #IT_CONNECTIONS VALUES
    ( '{ ls_connections-carrid }', '{ ls_connections-
    connid }' )|.
  cl_sql_connection=>get_connection(
    )->create_statement(
      )->execute_update( lv_statement ).
ENDLOOP.
" Datenbankprozedur aufrufen
lv_statement = | CALL "test.a4h.bo|
&& |ok.chapter04::GET_KPIS_FOR_CONNECTIONS|
&& |"( '{ sy-mandt }' , #IT_CONNECTIONS, #ET_KPIS )
  WITH OVERVIEW |.
lo_result_set = cl_sql_connection=>get_connection(
)->create_statement( )->execute_query( lv_statement ).
lo_result_set->close( ).
...
```

Listing 5.5 Behandlung von tabellarischen Eingabeparametern

Erläuterung des Programms

Vor dem Aufruf der Datenbankprozedur füllt das Programm die lokale temporäre Tabelle `#IT_CONNECTIONS` mit den gewünschten Flugverbindungen. Der Aufruf der Datenbankprozedur erfolgt über `EXECUTE_QUERY`.

5.2.2 Definition von Database Procedure Proxies

Ab dem ABAP-Release 7.4 können Sie einen sogenannten *Database Procedure Proxy* definieren, um aus ABAP nativ auf Datenbankprozeduren zuzugreifen, die im SAP HANA Repository der Primärdatenbank definiert worden sind. Beachten Sie, dass nur das XML-Dateiformat (*.procedure*) unterstützt wird (siehe Abschnitt 4.3, »Datenbankprozeduren«).

Bei einem Database Procedure Proxy handelt es sich (wie der Name bereits vermuten lässt) um ein *Proxy-Objekt*. Dieses repräsentiert eine Datenbankprozedur im ABAP Dictionary.

Mehrere Proxy-Objekte für eine Datenbankprozedur **[!]**

Technisch ist es möglich, für eine Datenbankprozedur mehrere Database Procedure Proxies anzulegen. Wir empfehlen dies jedoch *nicht*. Legen Sie zu einer Datenbankprozedur immer maximal ein Proxy-Objekt im ABAP Dictionary an.

Interface des Proxy-Objekts

Für jeden Database Procedure Proxy legt das System automatisch auch ein Interface an. Über dieses Interface können Sie Parameternamen und Datentypen, die beim Aufruf der Datenbankprozedur mit ABAP verwendet werden, beeinflussen:

- Die Namen von Eingabe- und Ausgabeparametern können Sie in SAP HANA ändern, sobald diese 30 Zeichen überschreiten. In diesem Fall verkürzt das System die Parameternamen zunächst. Sie können die verkürzten Namen bei Bedarf überschreiben.
- Komponentennamen von Tabellenparametern können Sie immer überschreiben.
- Jedem Parameter können Sie den zu verwendenden Datentyp zuordnen. Dies ist wichtig, da die Abbildung von SQL-Datentypen auf ABAP-Datentypen und Dictionary-Datentypen nicht eindeutig ist. Daher kann das System bei der Anlage eines Proxy-Objekts nicht (immer) den richtigen ABAP-Datentyp bzw. Dictionary-Typ ableiten.

Anlage eines Database Procedure Proxys

Wir werden Ihnen nun erläutern, wie Sie für die Datenbankprozedur `DETERMINE_TOP_CONNECTIONS_XML` ein Proxy-Objekt anlegen. Dazu müssen Sie sich in den ABAP Development Tools in Eclipse befinden. Dort wählen Sie den Menüpunkt File • New • Other... aus. Anschließend selektieren Sie den Eintrag Database Procedure Proxy und kli-

cken auf Next. Abbildung 5.3 zeigt das Fenster, das daraufhin erscheint.

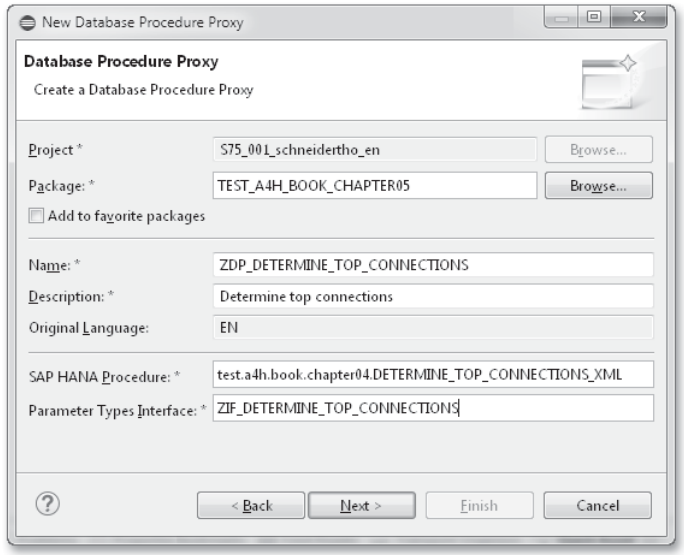

Abbildung 5.3 Anlegen eines Database Procedure Proxys

Parameter für die Anlage

In diesem Fenster erfassen Sie für den Database Procedure Proxy folgende Daten:

- Name: Über den Namen des Database Procedure Proxys können Sie die Datenbankprozedur (später) nativ in ABAP aufrufen.
- Description: Bei der Beschreibung handelt es sich um einen erläuternden Text.
- SAP HANA Procedure: Name der (bereits existierenden) Datenbankprozedur im SAP HANA Repository
- Parameter Types Interface: Name des Interface, das beim Anlegen des Proxy-Objekts automatisch angelegt wird (siehe Listing 5.6)

Nach nochmaligem Klick auf Next und anschließendem Klicken des Buttons Finish legt das System den Database Procedure Proxy und das entsprechende Interface an.

Im Project Explorer finden Sie den Database Procedure Proxy im entsprechenden Paket unterhalb des Knotens Dictionary • DB Procedure Proxies. Das Interface liegt (ebenso wie andere Interfaces) im entsprechenden Paket unterhalb des Knotens Source Library.

Anpassung des Interface

Abbildung 5.4 zeigt den Database Procedure Proxy für die Datenbankprozedur `DETERMINE_TOP_CONNECTIONS_XML`. Wenn Sie Parameternamen oder Datentypen anpassen möchten, können Sie dies in den Spalten ABAP Name, ABAP Type und DDIC Type Override tun. Zum Beispiel können Sie die Spalte `CONNID` des tabellarischen Ausgabeparameters `ET_CONNECTIONS` auf das Datenelement `S_CONN_ID` (und damit auf den ABAP-Datentyp `N length 4`) abbilden.

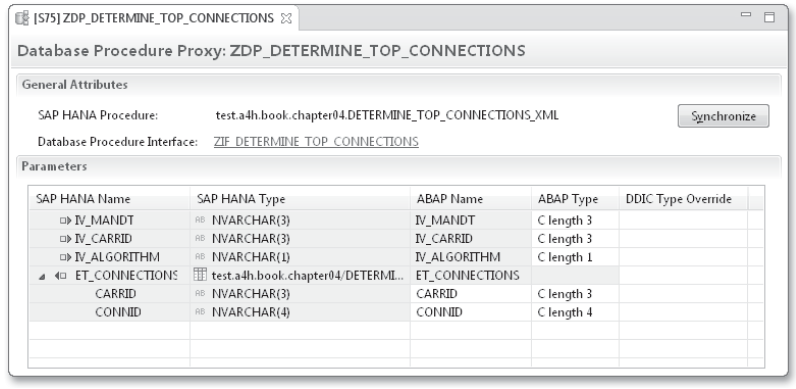

Abbildung 5.4 Database Procedure Proxy und Interface

Listing 5.6 zeigt das automatisch angelegte Interface nach Anpassung der Datentypen.

```
interface ZIF_DETERMINE_TOP_CONNECTIONS public.
types: iv_mandt type mandt.
types: iv_carrid type s_carr_id.
types: iv_algorithm type c length 1.
types: begin of et_connections,
         carrid type s_carr_id,
         connid type s_conn_id,
       end of et_connections.
endinterface.
```

Listing 5.6 Interface des Proxy-Objekts

5.2.3 Aufruf von Database Procedure Proxies

Nach der Aktivierung des Database Procedure Proxys können Sie das Proxy-Objekt zum Aufruf der Datenbankprozedur verwenden. Das Programm `ZR_A4H_CHAPTER5_TOP_PROXY` in Listing 5.7 zeigt die Verwendung beispielhaft.

```
PARAMETERS: p_carrid TYPE s_carr_id.

DATA: lt_connections TYPE TABLE OF
      zif_determine_top_connections=>et_connections.

TRY.
    CALL DATABASE PROCEDURE
      zdp_determine_top_connections
      EXPORTING
        iv_mandt = sy-mandt
        iv_carrid = p_carrid
        iv_algorithm = 'P'
      IMPORTING
       et_connections = lt_connections.

  CATCH cx_sy_db_procedure_sql_error
    cx_sy_db_procedure_call INTO DATA(lo_ex).
    " Fehlerbehandlung
        iv_algorithm = 'P'
    WRITE: | { lo_ex->get_text( ) } |.
ENDTRY.

LOOP AT lt_connections ASSIGNING
  FIELD-SYMBOL(<ls_connections>).
  WRITE: / <ls_connections>-carrid ,
           <ls_connections>-connid.
ENDLOOP.
```

Listing 5.7 Aufruf eines Database Procedure Proxys

Erläuterung des Programms

Das Programm verwendet die Anweisung `CALL DATABASE PROCEDURE`, um über den Proxy `ZDP_DETERMINE_TOP_CONNECTIONS` die Datenbankprozedur `DETERMINE_TOP_CONNECTIONS_XML` aufzurufen. Bei der Definition der internen Tabelle `LT_CONNECTIONS` bezieht sich das Programm auf das Interface `ZIF_DETERMINE_TOP_CONNECTIONS`. Beim Aufruf der Datenbankprozedur eventuell auftretende Probleme (Ausnahmen vom Typ `CX_SY_DB_PROCEDURE_SQL_ERROR` sowie `CX_SY_DB_PROCEDURE_CALL`) fängt das Programm ab.

5.2.4 Anpassung von Database Procedure Proxies

Wenn Sie eine Datenbankprozedur – genauer gesagt die Schnittstelle einer Datenbankprozedur – im SAP HANA Studio ändern, nachdem Sie einen Database Procedure Proxy angelegt haben, müssen Sie das Proxy-Objekt mit dem SAP HANA Repository synchronisieren. Dazu

steht Ihnen der Button Synchronize (siehe Abbildung 5.4) zur Verfügung.

Während des Synchronisationsvorgangs können Sie entscheiden, ob Sie am Proxy-Objekt vorgenommene Anpassungen (Komponentennamen oder Datentypen) beibehalten oder überschreiben wollen.

In Kapitel 6, »Erweiterte Datenbankprogrammierung mit ABAP 7.4«, lernen Sie ABAP-Datenbankprozeduren (*ABAP Managed Database Procedures*) kennen. Diese haben – bei der Verwendung im Rahmen von ABAP – im Vergleich zur Verwendung von Prozeduren, die Sie über das SAP HANA Studio angelegt haben, mehrere Vorteile. Daher empfehlen wir grundsätzlich die Nutzung von ABAP-Datenbankprozeduren, wenn Sie SQLScript innerhalb von ABAP nutzen möchten.

5.3 Transport nativer Entwicklungsobjekte

In diesem Abschnitt möchten wir Ihnen erklären, wie Sie ABAP-Programme, die native HANA-Objekte nutzen, konsistent in Ihrer Systemlandschaft transportieren können. Wir beschäftigen uns dazu mit dem sogenannten *HANA-Transportcontainer*. Auf das *erweiterte Change and Transport System (CTS+)*, das Ihnen ebenfalls Möglichkeiten bietet, gehen wir nicht ein.

Wir nehmen für unsere Ausführungen an, dass Sie sich mit der Entwicklungsorganisation und dem Transport im SAP NetWeaver AS ABAP bereits auskennen.

5.3.1 Exkurs: Entwicklungsorganisation und Transport in SAP HANA

Damit Sie die Funktionsweise des HANA-Transportcontainers (besser) verstehen, geben wir Ihnen in diesem Abschnitt einige Hintergrundinformationen zur Entwicklungsorganisation und zum Transport in SAP HANA.

Entwicklungsorganisation

Die Entwicklungsorganisation in SAP HANA ähnelt in vielerlei Hinsicht der im SAP NetWeaver AS ABAP. Sie unterscheidet sich aber auch in einigen wesentlichen Aspekten von ihr. Wie in Kapitel 2,

»Einführung in die Entwicklungsumgebung«, beschrieben, ist das SAP HANA Repository die zentrale Ablage von Entwicklungsobjekten der HANA-Datenbank.

Namensraum für Kunden

Innerhalb des Repositorys liefert SAP Content unterhalb des Wurzelpakets `sap` aus. Unterhalb dieses Pakets dürfen daher keine Kundenentwicklungen angelegt werden, da diese andernfalls versehentlich überschrieben werden könnten. Bauen Sie stattdessen eine parallele Pakethierarchie für Kundenentwicklungen auf. Verwenden Sie als Wurzelpaket z. B. Ihren Domänennamen.

Lokale Entwicklungen

Einen Sonderfall stellt das Paket `system-local` dar. Es ähnelt von der Idee her den lokalen Paketen des SAP NetWeaver AS ABAP. Verwenden Sie es für Entwicklungsobjekte, die nicht transportiert werden sollen.

Transport

Delivery Units

Ein Transport erfolgt in SAP HANA in der Regel auf Basis einer *Delivery Unit* (übersetzt heißt das so viel wie *Liefereinheit*). Eine Delivery Unit fasst Pakete zusammen, die gemeinsam transportiert bzw. ausgeliefert werden sollen. Sie entspricht konzeptionell weitestgehend einer Softwarekomponente im Sinn des AS ABAP. Während Sie im AS ABAP allerdings in der Regel mit der Softwarekomponente `HOME` arbeiten, müssen Sie in SAP HANA immer eigene Delivery Units für Kundenentwicklungen anlegen. Voraussetzung dafür ist, dass Sie bzw. ein Administrator vorher den Systemparameter `content_vendor` in der Datei *indexserver.ini* über die Administration Console des SAP HANA Studios gepflegt haben.

Zuweisung Delivery Unit

Betrachten wir die Zuweisung einer Delivery Unit und den anschließenden Transport anhand eines Attribute Views `AT_CUSTOMER`. Bei der Anlage des Attribute Views `AT_CUSTOMER` ordnen Sie diesem ein Paket zu. In den Eigenschaften des Pakets können Sie eine Delivery Unit pflegen. Verwenden Sie dazu den Kontextmenüeintrag Edit des Pakets. Alle im System vorhandenen Delivery Units sehen Sie in der Ansicht Quick View über den Menüeintrag Delivery Units. Dort können Sie auch neue Delivery Units anlegen. Abbildung 5.5 veranschaulicht den Zusammenhang zwischen Entwicklungsobjekt, Paket und Delivery Unit am Beispiel des Attribute Views `AT_CUSTOMER` (die Delivery Unit `ZA4H_BOOK_CHAPTER05` ist *nicht* Teil der mit diesem Buch ausgelieferten Beispiele).

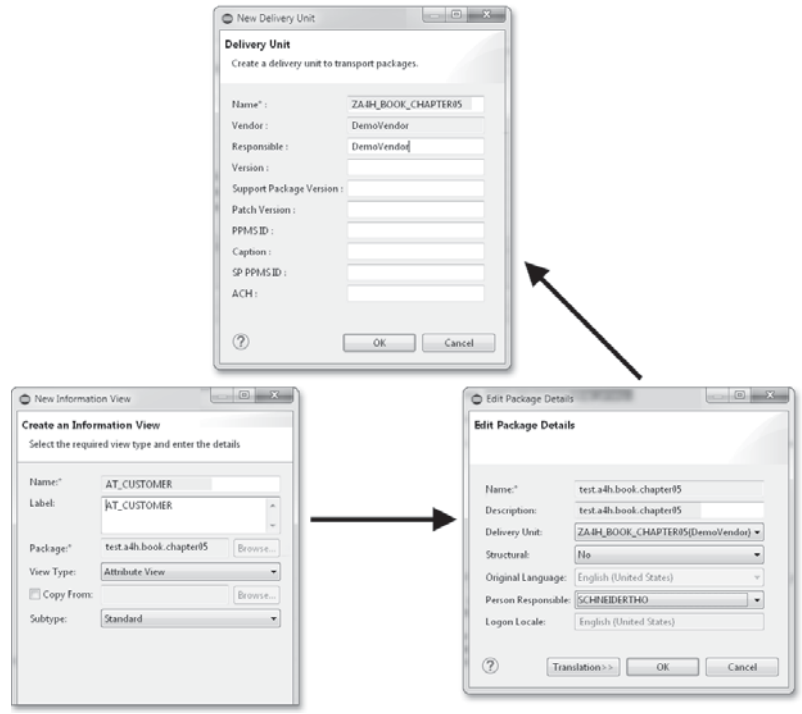

Abbildung 5.5 Entwicklungsobjekt, Paket und Delivery Unit

Import und Export

Im SAP HANA Studio können Sie Entwicklungsobjekte auf zwei Arten transportieren, d. h. exportieren und im Zielsystem importieren:

- Export/Import einer Delivery Unit (optional gekoppelt mit CTS+)
- Export/Import einzelner Objekte (der sogenannte *Developer Mode*)

Für einen konsistenten Transport von HANA-Content (der nicht eng mit einer ABAP-Entwicklung gekoppelt ist) in einer produktiven Systemlandschaft empfehlen wir grundsätzlich den Export/Import auf Basis von Delivery Units und des CTS+.

Schema-Mapping

Eine Besonderheit beim Transport von HANA-Content ist das sogenannte *Schema-Mapping*. Ein Schema-Mapping ist notwendig, wenn die Datenbankschemata im Quellsystem und Zielsystem eines Transports voneinander abweichen. Es handelt sich um eine Abbildung eines Entwicklungsschemas (*Authoring Schema*) auf ein physikalisches Schema (*Physical Schema*).

Sie pflegen ein Schema-Mapping in der Ansicht Quick View über den Menüeintrag Schema Mapping. Bevor wir genauer darauf eingehen, wann und wie das System es auswertet, möchten wir die Notwendigkeit für das Schema-Mapping zunächst anhand des Attribute Views AT_CUSTOMER erläutern. Betrachten Sie hierzu Abbildung 5.6.

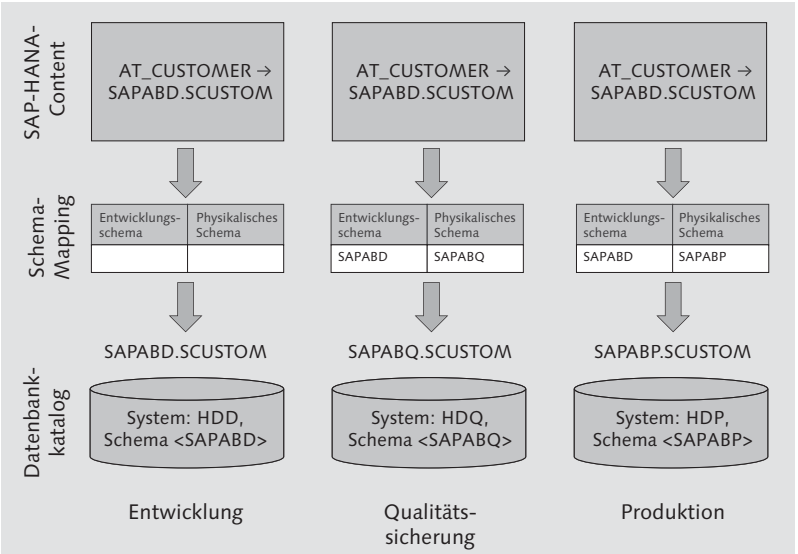

Abbildung 5.6 Prinzip des Schema-Mappings

Beispiel zum Schema-Mapping

Der Attribute View AT_CUSTOMER liest Kundendaten aus der Datenbanktabelle SCUSTOM. Diese Tabelle gehört zum Flugdatenmodell des AS ABAP und liegt im Entwicklungssystem im Datenbankschema SAPABD (weil die Systemkennung des ABAP-Systems ABD ist). Folglich verweist der Attribute View auf SAPABD.SCUSTOM.

Im Qualitätssicherungs- und Produktivsystem gibt es die Tabelle SAPABD.SCUSTOM nicht. Aufgrund der abweichenden Systemkennungen liegt die Datenbanktabelle im Qualitätssicherungssystem im Schema SAPABQ und im Produktivsystem im Schema SAPABP.

Das Schema-Mapping ermöglicht Ihnen, das Schema SAPABD im Qualitätssicherungssystem auf SAPABQ und im Produktivsystem auf SAPABP abzubilden.

Schema-Mapping-Pflege

Bei der Pflege eines Schema-Mappings müssen Sie einige Aspekte beachten:

- Das Schema-Mapping steuert letztlich, in welchem Datenbankschema ein Entwicklungsobjekt des SAP HANA Repositorys ein Objekt des Datenbankkatalogs sucht.
- Wenn kein Schema-Mapping gepflegt ist, sind Entwicklungsschema und physikalisches Schema identisch.
- Sie können mehrere Entwicklungsschemata auf das gleiche physikalische Schema abbilden.
- Sie können einem Entwicklungsschema *nicht* mehrere physikalische Schemata zuordnen.
- Der HANA-Content speichert Verweise zu Datenbankobjekten mit dem Entwicklungsschema. Wenn dieses (aufgrund einer Mehrfachzuordnung) nicht eindeutig ermittelt werden kann, speichert das System den Verweis mit dem physikalischen Schema.

Schema-Mapping bei Installation von SAP NetWeaver AS ABAP 7.4 [«]

Wenn Sie SAP NetWeaver AS ABAP 7.4 auf einer HANA-Datenbank installieren, erzeugt das Installationsprogramm das ABAP-Schema SAP<SID>. Außerdem legt das Installationsprogramm auch (mindestens) ein Schema-Mapping an, nämlich vom Entwicklungsschema ABAP auf das physikalische Schema SAP<SID>.

Sollten Sie sich für weiterführende Informationen zur Entwicklungsorganisation und zum Transport in SAP HANA interessieren, konsultieren Sie bitte die Dokumentation der HANA-Datenbank.

5.3.2 Nutzung des SAP-HANA-Transportcontainers

Nun betrachten wir den Transport von ABAP-Programmen, die native HANA-Objekte nutzen, über den HANA-Transportcontainer. Hierzu verwenden wir den Report ZR_A4H_CHAPTER5_LIST_CUSTOMER. Dieser greift über den externen View ZEV_A4H_CUSTOMER des ABAP Dictionarys auf den Attribute View AT_CUSTOMER des SAP HANA Repositorys zu. Den Quelltext des Reports finden Sie in Listing 5.8.

```
REPORT zr_a4h_chapter5_list_customer.

DATA: lt_customer TYPE STANDARD TABLE OF
      zpv_a4h_customer,
      ls_customer TYPE zpv_a4h_customer.

IF cl_db_sys=>dbsys_type = 'HDB'.
```

```
  SELECT * FROM zev_a4h_customer
    INTO TABLE lt_customer.
ELSE.
  SELECT * FROM zpv_a4h_customer
    INTO TABLE lt_customer.
ENDIF.
LOOP AT lt_customer INTO ls_customer.
  WRITE: / ls_customer-id, ls_customer-name.
ENDLOOP.
```

Listing 5.8 Zu transportierender Beispielreport

Probleme beim Transport

Sowohl der Report ZR_A4H_CHAPTER5_LIST_CUSTOMER als auch der externe View ZEV_A4H_CUSTOMER können durch die Änderungsaufzeichnung und das Transportwesen des SAP NetWeaver AS ABAP problemlos transportiert werden (das geschieht im Prinzip »automatisch«). Der dem externen View zugrunde liegende Attribute View AT_CUSTOMER unterliegt hingegen nicht der Änderungsaufzeichnung und dem Transportwesen des Applikationsservers. Damit fehlt er (ohne dass wir entsprechende Maßnahmen ergreifen) nach einem Transport im Zielsystem. Daher kommt es im Zielsystem beim Aufruf des Reports zu einem Laufzeitfehler. Abhilfe kann der HANA-Transportcontainer schaffen.

Grundlegende Funktionsweise

Der HANA-Transportcontainer steht im Release SAP NetWeaver 7.31 ab dem Support Package 5 sowie ab dem Release 7.4 zur Verfügung. Er kann genutzt werden, wenn SAP HANA die Primärdatenbank ist.

Der HANA-Transportcontainer erlaubt Ihnen, über das SAP HANA Studio angelegte Entwicklungsobjekte des SAP HANA Repositorys mit den Mechanismen des Change and Transport Systems des ABAP-Applikationsservers (und insbesondere ohne die Notwendigkeit für einen Java-Stack, wie er für CTS+ benötigt würde) zu transportieren.

Technisch betrachtet handelt es sich beim HANA-Transportcontainer um ein logisches Transportobjekt, das als Proxy-Objekt für genau eine Delivery Unit agiert. Die Funktionsweise des HANA-Transportcontainers veranschaulicht Abbildung 5.7.

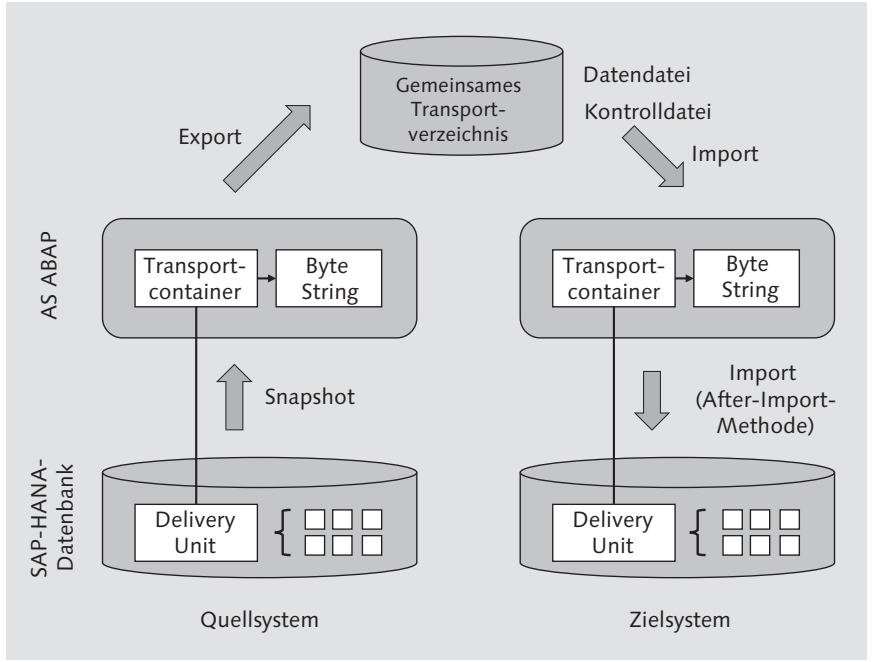

Abbildung 5.7 Funktionsweise des HANA-Transportcontainers

Anlage des Transportcontainers

Sie können einen HANA-Transportcontainer über die ABAP Development Tools (und nur dort) anlegen. Dazu folgen Sie in der ABAP-Perspektive z. B. dem Menüpfad File • New • Other... • ABAP • SAP HANA Transport Container. Anschließend geben Sie den Namen der Delivery Unit ein, für die Sie den Transportcontainer anlegen möchten. Das System leitet daraus automatisch den Namen des Transportcontainers ab (siehe Abbildung 5.8; der HANA-Transportcontainer ZA4H_BOOK_CHAPTER05 ist *nicht* Teil der mit diesem Buch ausgelieferten Beispiele).

Nutzung eines Präfixnamensraums

Falls Sie in ABAP einen Präfixnamensraum verwenden möchten, müssen Sie vor der Anlage des Transportcontainers dem Namen des content_vendor (siehe Abschnitt 5.3.1, »Exkurs: Entwicklungsorganisation und Transport in SAP HANA«) den gewünschten Präfixnamen zuordnen. Dazu können Sie die Datenbanktabelle SNHI_VENDOR_MAPP über die Tabellensicht-Pflege füllen.

Änderungsaufzeichnung

Wenn die Transporteigenschaften des verwendeten Pakets – im Beispiel TEST_A4H_BOOK_CHAPTER05 – entsprechend gepflegt sind, zeichnet das System die Anlage des Transportcontainers in einem transportierbaren Änderungsauftrag auf.

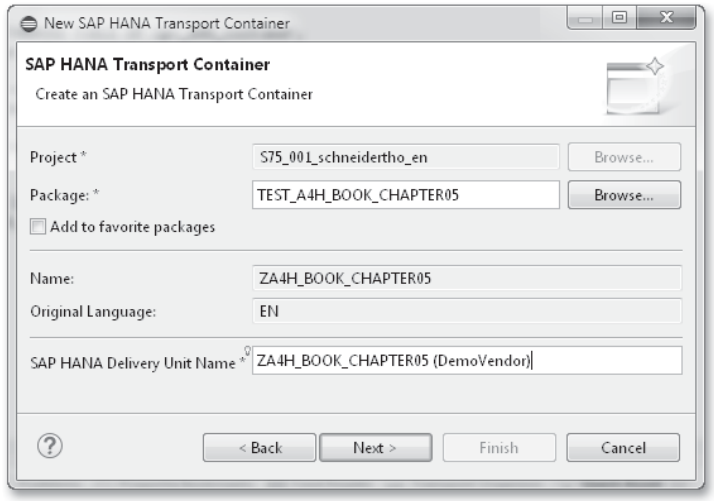

Abbildung 5.8 Anlage eines Transportcontainers

Synchronisation

Beim Anlegen eines Transportcontainers synchronisiert das System den Inhalt dieses Containers einmalig automatisch mit dem Inhalt der Delivery Unit. Das bedeutet, dass alle Objekte der Delivery Unit als gepackte Datei auf den ABAP-Applikationsserver geladen und dort als *Byte String* in einer Datenbanktabelle (nämlich der Tabelle `SNHI_DU_PROXY`) abgelegt werden. Genau genommen liegt der Inhalt der Delivery Unit anschließend zweimal in der HANA-Datenbank:

- im SAP HANA Repository
- über die Datenbanktabelle `SNHI_DU_PROXY`

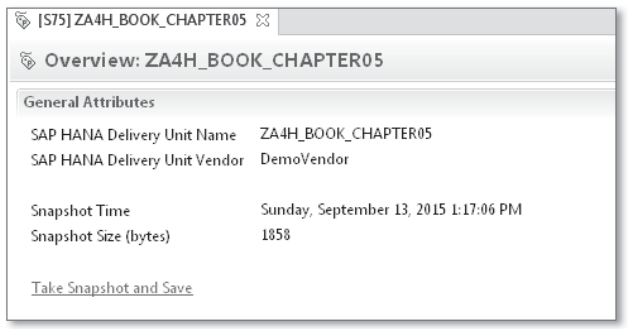

Abbildung 5.9 Synchronisation und Inhalt eines Transportcontainers

Wenn Sie den Transportcontainer nach der Anlage – z. B. weil Sie Änderungen am Attribute View `AT_CUSTOMER` vorgenommen haben –

mit der Delivery Unit synchronisieren möchten, müssen Sie dies manuell tun. Verwenden Sie dazu den Link Take Snapshot and Save. Den aktuellen Inhalt des Transportcontainers können Sie sich über die Registerkarte Contents anschauen (beides ist in Abbildung 5.9 dargestellt).

Export und Import

Der Transport vom Entwicklungs- ins Qualitätssicherungs- und Produktivsystem erfolgt mit den Mechanismen des CTS:

- Beim Export (genauer beim *Export Release Preprocessing*) schreibt das System den Inhalt des Transportcontainers in die Datendatei im gemeinsamen Transportverzeichnis der am Transport beteiligten Systeme.
- Beim Import (genauer gesagt in einer *After-Import-Methode*) liest das System den Inhalt des Transportcontainers aus der Datendatei und importiert die Delivery Unit in die HANA-Datenbank des Zielsystems. Eine Aktivierung des Contents findet dabei nur statt, wenn Sie dies für die Softwarekomponente des Transportcontainers in der Tabelle `SNHI_DUP_PREWORK` aktiviert haben (und zwar im Zielsystem).

Sie können die beiden Schritte anhand des Transportprotokolls jederzeit nachvollziehen.

Gemischte Systemlandschaften

Einen Sonderfall bei der ABAP-Entwicklung auf SAP HANA stellen gemischte Systemlandschaften dar. Stellen Sie sich dazu vor, dass Sie als ABAP-Entwickler ein Programm für SAP HANA optimieren und dabei von spezifischen Möglichkeiten der HANA-Datenbank Gebrauch machen möchten. Gleichzeitig soll dieses Programm aber auch auf traditionellen Datenbanken lauffähig sein, z. B. weil Ihr Arbeitgeber nur in Teilbereichen des Unternehmens SAP HANA als Datenbank nutzt. Eine Systemlandschaft könnte in diesem Fall (vereinfacht) wie in Abbildung 5.10 aussehen.

Durch eine Fallunterscheidung können Sie – um beim Beispiel des Programms `ZR_A4H_CHAPTER5_LIST_CUSTOMER` zu bleiben – einmal den Projektions-View `ZPV_A4H_CUSTOMER` und einmal den externen View `ZEV_A4H_CUSTOMER` aufrufen (siehe Listing 5.8). Dadurch stellen Sie sicher, dass *zur Laufzeit* keine Fehler auftreten.

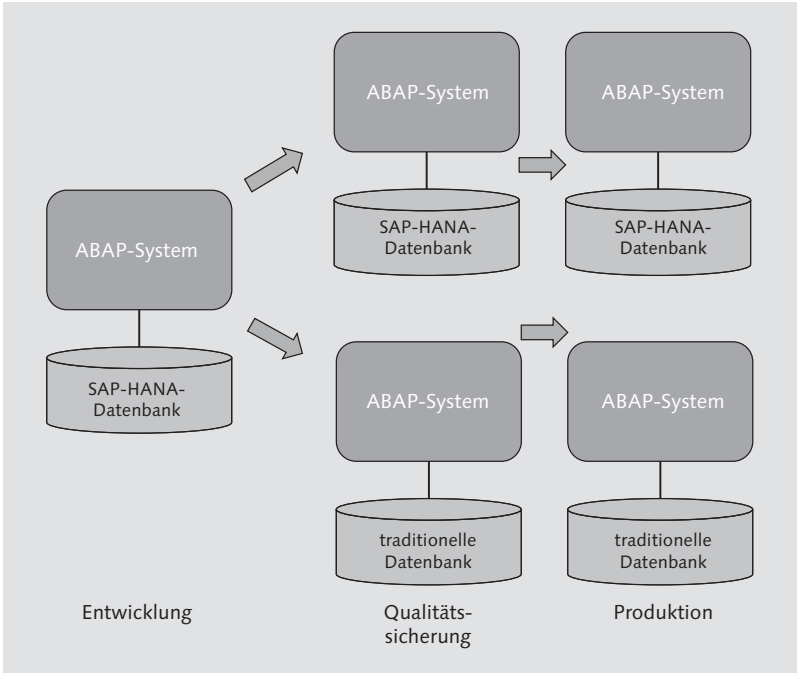

Abbildung 5.10 Gemischte Systemlandschaft

Systeme ohne HANA-Datenbank

Die Implementierung des Transportcontainers sorgt dafür, dass *beim Transport* keine Fehler auftreten und der HANA-Content nur dann importiert wird, wenn es sich beim Zielsystem des Imports um ein HANA-basiertes System handelt.

Empfehlungen zur Verwendung des Transportcontainers

Einschränkungen

Bei der Verwendung des Transportcontainers sollten Sie einige Einschränkungen beachten:

- Bei der Verwendung des Transportcontainers transportieren Sie immer die komplette Delivery Unit. Es besteht keine Möglichkeit, nur den Inhalt einer Delivery Unit zu transportieren, der in einem bestimmten Zeitintervall geändert wurde.
- Anders als bei Entwicklungsobjekten, die im SAP NetWeaver AS ABAP verwaltet werden, zeichnet das System Änderungen am Inhalt einer Delivery Unit nicht automatisch auf, und die Objekte einer Delivery Unit werden nicht exklusiv für einen Transportauftrag gesperrt. Es liegt folglich in Ihrer Verantwortung, den Transportcontainer manuell mit der Delivery Unit zu synchronisieren.

- Beim Export der Entwicklungsobjekte aus dem Quellsystem berücksichtigt der Transport nur die aktiven Objekte.
- Das Transportsystem erkennt keine Abhängigkeiten zwischen mehreren gleichzeitig transportieren Transportcontainern.

Im Rahmen der Einschränkungen erlaubt Ihnen der Transportcontainer, Anwendungen, die zum Teil aus ABAP-Objekten und zum Teil aus HANA-Content bestehen, konsistent zu transportieren. Wir empfehlen Ihnen seine Verwendung, wenn die zu Beginn des Abschnitt 5.3.2, »Nutzung des SAP-HANA-Transportcontainers«, beschriebenen Voraussetzungen erfüllt sind.

Wenn Sie von den in Kapitel 6, »Erweiterte Datenbankprogrammierung mit ABAP 7.4«, beschriebenen Möglichkeiten Gebrauch machen, brauchen Sie den HANA-Transportcontainer nicht.