As an ABAP developer you will want to use views and database procedures you created in SAP HANA Studio in ABAP. In this sample chapter, you will learn how to consistently transport native SAP HANA development objects, via the Change and Transport System.
ABAP developers want to use views and database procedures that they've created in SAP HANA Studio in ABAP. Developers are also used to a high-performance transport system and expect consistent transport of native SAP HANA development objects via the Change and Transport System.

5 Integrating Native SAP HANA Development Objects with ABAP

Chapter 4 illustrated how you can create analytical models (views) and database procedures using SAP HANA Studio. Now we’ll explain how you can call these native SAP HANA objects from ABAP.

We’ll also discuss how you can transport ABAP programs that use native SAP HANA objects consistently in your system landscape.

5.1 Integrating Analytic Views

In the previous chapter, you learned how to model the different view types in SAP HANA Studio and how to access the results of a view using the Data Preview or Microsoft Excel. In Chapter 4, Section 4.4.4, we also explained how to address the generated column views via SQL.

This section describes how to access the views from ABAP. In this context, we have to differentiate between ABAP Release 7.4 and earlier versions. When working with earlier releases, only Native SQL can be used for access; this will be described briefly in Section 5.1.1. As of ABAP 7.4, you can import the views from the SAP HANA repository into the ABAP Data Dictionary (DDIC) and then access them using Open SQL (explained in detail in Section 5.1.2 and in Section 5.1.3). In the last section, you’ll find some recommendations, tips, and tricks for SAP HANA view modeling.
5.1.1 Access via Native SQL

When activating any of the presented view types in SAP HANA, a column view is created in the database catalog in the _SYS_BIC schema with a public synonym, for example, 'test.a4h.book.chapter04::AT_FLIGHT'.

Using these names, you can access this view from ABAP. Listing 5.1 shows how the AT_FLIGHT attribute view created in Chapter 4, Section 4.4.1, is accessed via ABAP Database Connectivity (ADBC).

```
* Definition of the result structure
TYPES: BEGIN OF ty_data,
  carrid TYPE s_carr_id,
  connid TYPE s_conn_id,
  fdate 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.

* Access to the attribute view
DATA(lv_statement) =
| SELECT carrid, connid, fdate, route |
| FROM "{ gc_view }" |
| WHERE mandt = '{ sy-mandt }' ORDER BY fdate |.
TRY.
  * Prepare SQL connection and statement
  DATA(lo_result_set) =
  cl_sql_connection=>get_connection( )
  ->create_statement( 
    tab_name_for_trace = conv #( gc_view )
  )->execute_query( lv_statement ).
  * Get result
  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).
  * Error handling
  | WHERE mandt = '{ sy-mandt }' ORDER BY fdate |.
  WRITE: | { lo_ex->get_text( ) } |.
ENDTRY.

LOOP AT lt_data ASSIGNING FIELD-SYMBOL(<l>).
```

As you can see, this is a regular access using Native SQL. If an error occurs during execution, the text of the SQL exception points to the cause. In addition to SQL coding errors, which are also visible when accessing views via the SQL console, there may also be errors related to mapping the result to the ABAP data type. Recommendations regarding this topic are given in Section 5.1.4.

5.1.2 External Views in the ABAP Data Dictionary

In ABAP 7.4, external views are a new view type in the DDIC. Using such views, you can import column views defined in the SAP HANA repository into the DDIC. These views are called external views because they aren’t fully defined in the DDIC but are used as a kind of proxy allowing the corresponding column view in the _SYS_BIC schema to be accessed from ABAP.

External views can only be defined using the ABAP Development Tools in Eclipse. To do so, you create a new development object of the Dictionary View type. Figure 5.1 shows the New Dictionary View dialog for the AT_FLIGHT attribute view.

```
WRITE: / <l>-carrid, <l>-connid, <l>-fdate,
  <l>-route .
ENDLOOP.
```

Listing 5.1 Accessing an Attribute View via ADBC

![Figure 5.1 Creating an External View in the ABAP Data Dictionary](create-external-views-in-eclipse.png)
When the view is created, the system checks whether it can be imported into the DDIC. Note that not all SAP HANA data types are supported in ABAP. When defining calculated attributes or accessing tables from views that weren’t created using the DDIC, such potentially unsupported data types may appear. In this case, an error occurs when creating the external view, and the view can’t be imported. The supported data types are listed in Table 5.1 later in this section and are described in Chapter 3, Section 3.1.3.

### View structure and synchronization

After successfully importing the SAP HANA view into the DDIC, the editor displays the structure of the view together with the data type mapping (Figure 5.2). In addition, you can use the Synchronize button to synchronize the view after changing the structure of the corresponding view in SAP HANA Studio. Therefore, if you add attributes to the output structure, delete attributes, or change data types, you need to synchronize the external view to avoid runtime errors. Recommendations on synchronizing developments within a development team are provided in Chapter 14.

As you learned in Chapter 3, Section 3.1.3, SQL data types and DDIC types can’t always be mapped uniquely. However, the data type is decisive for the correct handling of operations (e.g., the calculation of differences for a date). For this reason, the developer must manually map the correct ABAP data type.

Table 5.1 shows the possible data type mappings for some columns of the AT_FLIGHT sample view.

<table>
<thead>
<tr>
<th>Column</th>
<th>SQL Data Type</th>
<th>Possible Dictionary Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARRID</td>
<td>NVARCHAR(3)</td>
<td>CHAR(3), NUMC(3), SSTR, CLNT, UNIT, CUKY</td>
</tr>
<tr>
<td>FLDATE</td>
<td>NVARCHAR(8)</td>
<td>CHAR(8), NUMC(8), SSTR, DATS</td>
</tr>
<tr>
<td>CARRNAME</td>
<td>NVARCHAR(20)</td>
<td>CHAR(20), NUMC(20), SSTR</td>
</tr>
</tbody>
</table>

Table 5.1 Possible Type Mappings

For the external view shown in Figure 5.2, we manually mapped the FLDATE column to the ABAP DATS data type. This may appear strange at first glance because this information is already present in the underlying DDIC table; however, the attributes of column views in SAP HANA don’t have a reference to columns of existing tables that is recognizable by the DDIC. For instance, the FLDATE column could also be a calculated attribute.

The procedure for defining external views based on an analytic or a calculation view is identical to the procedure used for an attribute view. Note that external views in the DDIC currently don’t have a reference to the particular view type; that is, they are just pointing to an arbitrary column view in SAP HANA. The only prerequisite is that the view is defined via the SAP HANA repository. Column views, which solely exist in the database catalog (e.g., generated programmatically), can’t be imported into the DDIC.

The transport of external views (and other SAP HANA-specific developments) is described in Section 5.3.
5.1.3 Options for Accessing External Views

Advantages

The main advantage of external views is that you can use Open SQL to access SAP HANA views. This allows you to benefit from the following advantages:

- Syntax checking by the ABAP Compiler and content assist during development (code completion)
- Automatic client handling
- Iterating through a result set within a SELECT loop
- Use of the INTO CORRESPONDING FIELDS expression for a matching selection in a target structure independent of the sequence in the projection list
- Use of IN for the WHERE condition to transfer selection options

Access via Open SQL

Listing 5.2 shows how the access to the external view from Figure 5.2 is implemented. From a functional perspective, this corresponds to the ADBC access variant from Listing 5.1. As you can see, the ABAP code required for access is significantly shorter and corresponds to the access for a standard DDIC view.

```
REPORT ZR_A4H_CHAPTER4_VIEW_OPEN.
DATA: wa TYPE zev_a4h_flights.
  " Read data from external view
  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 Accessing an External View via Open SQL

### Possible Runtime Errors When Accessing External Views

When using Open SQL to access an external view, an SQL query is executed for the corresponding column view in SAP HANA. The same rules apply as when accessing the view using Native SQL.

As explained in Chapter 4, Section 4.4.4, you must consider certain limitations when accessing analytic views via SQL. An unsupported query via Open SQL leads to a runtime error. Because these errors rarely occur when accessing ABAP tables using Open SQL, ABAP developers should use caution when following this approach. The troubleshooting tools and possible runtime errors during SQL access are explained in more detail in Chapter 7, Section 7.2.

In addition to Open SQL, you can also address external views using Native SQL. This variant, which seems somewhat awkward at first glance, is useful if you want to use an SQL query to access an SAP HANA view in a way that isn’t supported using Open SQL, such as a fuzzy search in an attribute view (see Chapter 10, Section 10.4). Compared to accessing the generated column view in the _SYS_BIC schema via Native SQL, the external view has an advantage in that a suitable target structure for a selection via ADBC already exists in the DDIC.

5.1.4 Recommendations

This section concludes with some recommendations for using SAP HANA views. These are limited to functional recommendations. Tools and recommendations for performance analysis are discussed in Chapter 7 and Chapter 14, where we’ll also deal with design aspects such as naming conventions.

If the scope of functions provided by standard DDIC views is sufficient for your purposes and you’ve used these views in the past, there’s no need to change your application using native SAP HANA views. The next chapter presents Core Data Services (CDS) views, which enable you to define complex views with calculated fields directly in ABAP.

The modeled SAP HANA views provide simple access for special analytical scenarios. The following questions can help to determine the best view type in SAP HANA for your scenario:

- Are you dealing with master data views that might be extended by calculated attributes? In this case, you should start with an attribute view.
- Are you performing an analysis of transaction data based on a star schema? In this case, you should choose an analytic view and implement the dimensions as attribute views.
Do you have to combine or adapt the results from different tables and SAP HANA views? In this case, you should use the modeled calculation view. If the modeled variant isn’t sufficient for some part of your scenario, you can use a SQLScript-based implementation for that part.

Client handling
When modeling views, you should make sure that the client field is handled correctly. In particular, it’s advisable to add the client field as the first field of the view and to make sure that the client is included in the join definition. In most cases, the SESSION CLIENT configuration value is the correct setting for views based on ABAP tables from the same system. If tables are replicated from a different system, it may be useful to use a fixed value for the client. Cross-client access is useful only in rare cases.

Schema mapping
You should always choose the correct default schema for analytic views and calculation views. This schema is taken into account in particular for the relevant Customizing for conversions, that is, if no special setting was configured for the attribute. Specifying the correct default schema is even more important when dealing with the implemented variant of calculation views.

Define external views
External views should only be defined for SAP HANA views that will be used for access via ABAP because these views have to be synchronized manually after changing the corresponding structures. Moreover, you should define a maximum of one external view for each SAP HANA view.

Troubleshooting
If error messages are displayed when activating an SAP HANA view, the error text usually includes information on the root cause. In some cases, however, you may need some experience to interpret the error message correctly. For this reason, we recommend following a heuristic approach to error analysis. As a first step, you should make sure that you mark at least one field of an attribute view as a key field and that you define at least one key figure for an analytic view. If your view contains calculated attributes, you should check if you correctly defined the corresponding expression.

If you come to a dead end during error analysis, you can try to remove the corresponding attribute (e.g., in a copy of the view). If an error message or unexpected data appears when calling the DATA PREVIEW, this is often an indication of a problem in the join modeling. For currency conversions, a missing client context may result in an error.

When accessing an SAP HANA view from ABAP using Native SQL, you should pass the name of the view (via the tab_name_for_trace parameter as shown in Listing 5.1 or via the SET_TABLE_NAME_FOR_TRACE method). This facilitates the error analysis in a support scenario.

5.2 Integrating Native Procedures with ABAP

In Chapter 4, you learned what SQLScript is and how you can use it for implementing database procedures. Now we want to explain how to call database procedures from ABAP. You have two options:

- Native SQL and ADBC (see also Chapter 3)
- Database procedure proxies

Requirements
As of ABAP Release 7.0 and SAP Kernel 7.20, it’s possible to use ADBC to call database procedures in SAP HANA. Database procedure proxies are available as of Release 7.4 and require that SAP HANA be used as the primary database. Moreover, database procedure proxies only support the XML file format (.procedure), which is actually outdated.

5.2.1 Access via Native SQL

As already described in Chapter 4, Section 4.3, the system generates different runtime objects in the _SYS_BIC schema when activating a database procedure. It also generates a public synonym. Here, you can use Native SQL to access the database procedure from ABAP.

However, the use of Native SQL to call a database procedure is relatively time-consuming and prone to errors. Later in this section, you’ll see how you can only use temporary tables to exchange tabular input and output parameters with the database procedure. Furthermore, SAP NetWeaver AS ABAP doesn’t detect syntax errors in Native SQL statements until runtime. For more information, refer to the explanations provided in Chapter 3.
We’ll now use several examples to provide a detailed description of how to use Native SQL to access database procedures. First, we’ll consider a database procedure that determines the name of an airline on the basis of the ID. For the remaining examples, we’ll revert to the database procedures from Chapter 4.

Example 1: Calling a Database Procedure

If you use ADBC to call a database procedure, the CL_SQL_STATEMENT class makes the EXECUTE_PROCEDURE method available. You can use this as long as a database procedure doesn’t have a tabular input/output parameter.

Sample call

Program ZR_A4H_CHAPTER5_CARRNAME_ADBC shows an example of the EXECUTE_PROCEDURE method (see Listing 5.3). It calls the DETERMINE_CARRNAME database procedure, which has the following input and output parameters:

- /L50776 IV_MANDT: Client.
- /L50776 IV_CARRID: ID of an airline.
- /L50776 EV_CARRNAME: Name of an airline.

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.abi-book.chapter04::DETERMINE_CARRNAME" ).
  " error handling
  CATCH cx_sql_exception INTO DATA(lo_ex).
  WRITE: | { lo_ex->get_text( ) } |.
  ENDTRY.
  WRITE: / lv_carrname.

Listing 5.3 Using Native SQL to Call a Database Procedure

First, the program generates an instance of the CL_SQL_STATEMENT class. Then, it calls the SET_PARAM method to bind the input and output parameters of the database procedures to the actual parameters. It then calls the EXECUTE_PROCEDURE method.

Example 2: Tabular Output Parameters

Alternatively, you can use the EXECUTE_QUERY method (together with the WITH OVERVIEW addition) to execute a database procedure. This also works for database procedures that have tabular input and output parameters.

Sample call

Program ZR_A4H_CHAPTER5_TOP_ADBC in Listing 5.4 shows an example of the EXECUTE_QUERY method, in which the DETERMINE_TOP_CONNECTIONS database procedure is called. This database procedure determines an airline’s top connections and has the following input and output parameters:

- /L50776 IV_MANDT: Client.
- /L50776 IV_CARRID: ID of an airline.
- /L50776 IV_ALGORITHM: Controls how the top connections are determined.
- /L50776 ET_CONNECTIONS: A table parameter that contains the airline’s ID CARRID and connection code CONNID.

PARAMETERS: p_carrid TYPE s_carr_id.

" Definition of the result structure
TYPES: BEGIN OF ty_connections,
```abap
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.
  * Delete local temporary table
  lv_statement = | DROP TABLE #ET_CONNECTIONS |.
  cl_sql_connection=>get_connection( )->create_statement( )->execute_ddl( lv_statement ).
  CATCH cx_sql_exception.
    * The local temporary table may not exist,
    * we ignore this error
  ENDTRY.

TRY.
  * Create local temporary table
  lv_statement = | CREATE LOCAL TEMPORARY ROW|
  && | TABLE #ET_CONNECTIONS LIKE "_SYS_BIC".|
  && | "test.a4h.book.chapter04::GlobalTypes.t|
  && | lt_connections" |.
  cl_sql_connection=>get_connection( )->create_statement( )->execute_ddl( lv_statement ).
  * Call database procedure
  lv_statement = | CALL "test.a4h.book.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( ).
  * Read local temporary table
  lv_statement = | SELECT * FROM #ET_CONNECTIONS |
  lo_result_set = cl_sql_connection=>get_connection( )->create_statement( )->execute_query( lv_statement ).
  * Read result
  GET REFERENCE OF lt_connections INTO lo_connections.
  lo_result_set->set_param_table( lo_connections ).
  lo_result_set->next_package( )
```

Temporary Tables

Many databases, including the SAP HANA database, enable you to save temporarily the interim and final results of calculations in temporary tables. For this use case, temporary tables have many different advantages over conventional tables:

- The table definition and table contents are deleted automatically from the database if they are no longer required.
- The database automatically isolates data in parallel sessions from one another. It's neither necessary nor possible to place locks on temporary tables.
- The database doesn't write a transaction log for temporary tables.
- Generally, it's more efficient to use temporary tables than conventional tables.

SAP HANA supports global and local temporary tables:

- **Global temporary tables**
  The table definition can be used in different sessions. The table contents can only be displayed for the current session. At the end of the session, the table contents are deleted from the database automatically.

- **Local temporary tables**
  Both the table definition and the table contents are only valid for the current session. In other words, both are deleted from the database automatically at the end of the session.
Usage in AS ABAP

When using temporary tables to transfer data between AS ABAP and a database procedure, note the following:

- If you work with global temporary tables, you can create these once (because they can be used in different sessions). Organizationally, however, you must ensure that the table name isn’t used for different use cases (that require a different table structure).
- You can create global temporary tables at design time. Then you must ensure that the tables are also available in the test and production systems after a transport.
- If you decide to create global temporary tables at runtime, you must ensure that—before you call a database procedure—the table structure is suitable for the interface of the database procedure called (because this may have changed in the meantime).
- You must create local temporary tables at least once for each session (also note the following explanations in relation to the ABAP work process and database connection). Consequently, you can only create local temporary tables when an ABAP program is running.
- Because each ABAP work process has only one connection with the database, multiple ABAP programs processed by the same work process subsequently, are one session for the database. Therefore, after an ABAP program ends, neither the definition nor the contents of local (and global) temporary tables are deleted automatically.
- For global and local temporary tables, you should delete the contents (of the current session) before you call the database procedure.

Explanation of the program

Program ZR_A4H_CHAPTER5_TOP_ADBC in Listing 5.4 works with a local temporary table. First, it uses DROP TABLE #ET_CONNECTIONS to delete local temporary table #ET_CONNECTIONS if it exists. Then it uses the CREATE LOCAL TEMPORARY ROW TABLE statement to create a (new) local temporary table with the name #ET_CONNECTIONS. Here, the program refers to the table type that the system automatically created for the ET_CONNECTIONS output parameter when the database procedure was activated. This approach enables the program to ensure that, before the database procedure is called, the temporary table is empty and suitable for the current structure of the ET_CONNECTIONS output parameter.

Example 3: Tabular Input Parameters

If a database procedure has tabular input parameters, you can proceed in the same way as for tabular output parameters. Program ZR_A4H_CHAPTER5_KPIS_ADBC in Listing 5.5 shows how to call the GET_KPIS_FOR_CONNECTIONS database procedure for a set of flight connections. The database procedure determines some key performance indicators (KPIs) for each connection transferred.

The program now uses the EXECUTE_QUERY method to call the database procedure. It transfers SY-MANDT, P_CARRID, and ‘P’ to the input parameters, and it transfers temporary table #ET_CONNECTIONS to the output parameter for the database procedure.

After the database procedure has been called, the program reads the contents of temporary table #ET_CONNECTIONS, which correspond to the transferred airline’s top connections.

Example of an input parameter

Program ZR_A4H_CHAPTER5_KPIS_ADBC in Listing 5.5 works with a local temporary table. First, it uses DROP TABLE #ET_CONNECTIONS to delete local temporary table #ET_CONNECTIONS if it exists. Then it uses the CREATE LOCAL TEMPORARY ROW TABLE statement to create a (new) local temporary table with the name #ET_CONNECTIONS. Here, the program refers to the table type that the system automatically created for the ET_CONNECTIONS output parameter when the database procedure was activated. This approach enables the program to ensure that, before the database procedure is called, the temporary table is empty and suitable for the current structure of the ET_CONNECTIONS output parameter.
**Explanation of the program**

Before the database procedure is called, the program fills local temporary table `#IT_CONNECTIONS` with the relevant flight connections. `EXECUTE_QUERY` is used to call the database procedure.

### 5.2.2 Defining Database Procedure Proxies

As of ABAP Release 7.4, you can define a database procedure proxy to access database procedures from ABAP. Note that only the XML file format (`.procedure`) is supported (see Chapter 4, Section 4.3).

A database procedure proxy is a `proxy object` that represents a database procedure in the DDIC.

#### Multiple Proxy Objects for one Database Procedure

Technically, it’s possible to create multiple database procedure proxies for one database procedure. However, we don’t recommend this. In the DDIC, you should never create more than one proxy object for a database procedure.

#### Interface of the proxy object

The system also automatically creates an interface for each database procedure proxy. You can use this interface to influence the parameter names and data types used when calling the database procedure with ABAP:

- You can change the names of the input and output parameters as soon as they exceed 30 characters. In this case, the system initially abbreviates the parameter names. You can then overwrite these abbreviated names, if necessary.
- You can always overwrite the component names of table parameters.
- You can assign the relevant data type to each parameter. This is important because SQL data types aren’t uniquely mapped to ABAP data types and DDIC types. Consequently, when creating a proxy object, the system can’t (always) derive the correct ABAP data type and/or DDIC type.

#### Create a database procedure proxy

We’ll now explain how to create a proxy object for the `DETERMINE_TOP_CONNECTIONS_XML` database procedure. To do this, open the ABAP Development Tools in Eclipse, and choose the menu option, File • New • Other. Then, choose DATABASE PROCEDURE PROXY, and click Next. Figure 5.3 shows the window that opens.

In this window, enter the following data for the database procedure proxy:

- **NAME**
  Used to call the database procedure later in ABAP.

- **DESCRIPTION**
  Piece of explanatory text.

- **SAP HANA PROCEDURE**
  Name of the existing database procedure in the SAP HANA repository.

- **PARAMETER TYPES INTERFACE**
  Name of the interface that is automatically created when you create the proxy object (see Listing 5.6).

After you choose Next and Finish, the system creates the database procedure proxy and the corresponding interface.

The Project Explorer contains the database procedure proxy in the corresponding package below the Dictionary • DB Procedure Proxies node. Just like the other interfaces, the parameter type interface is located in the corresponding package below the Source Library node.
Adjust the interface

Figure 5.4 shows the database procedure proxy for the DETERMINE_TOP_CONNECTIONS_XML database procedure. If you want to adjust parameter names or data types, you can do this in the ABAP NAME, ABAP TYPE, and DDIC TYPE OVERRIDE columns. For example, you can map the CARRID column in the table-based ET_CONNECTIONS output parameter to the S_CONN_ID data element (and therefore to the ABAP data type N LENGTH 4).

Listing 5.6 shows the interface that the system automatically creates after the data types have been adjusted.

```
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 of the Proxy Object

5.2.3 Calling Database Procedure Proxies

Now that you’ve activated the database procedure proxy, you can use the proxy object to call the database procedure. Program ZR_A4H CHAPTER5_TOP_PROXY in Listing 5.7 shows an example of this usage.

```
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).
    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 Calling a Database Procedure Proxy

The program uses the CALL DATABASE PROCEDURE statement to call the DETERMINE_TOP_CONNECTIONS XML database procedure via the ZDP DETERMINE_TOP_CONNECTIONS proxy. When defining internal table LT_CONNECTIONS, the program refers to the ZIF徐MINE_TOP_CONNECTIONS interface. The program catches any problems that may occur when calling the database procedure (exceptions of the type CX_SY_DB_PROCEDURE_SQL_ERROR and CX_SY_DB_PROCEDURE_CALL).

5.2.4 Adjusting Database Procedure Proxies

If you change a database procedure (or more accurately, the interface of a database procedure) in SAP HANA Studio, you must synchronize the proxy object with the SAP HANA repository via the SYNCHRONIZE button (refer to Figure 5.4).
During the synchronization process, you can decide whether you want to retain or overwrite the adjustments made to the proxy object (component names or data types).

Chapter 6 introduces you to ABAP Managed Database Procedures (AMDP). When used within the scope of ABAP, they have several advantages compared with procedures that you’ve created via SAP HANA Studio. For this reason, we generally recommend the usage of ABAP database procedures if you want to use SQLScript within ABAP.

5.3 Transport of Native Development Objects

In this section, we discuss how you can transport ABAP programs that use native SAP HANA objects consistently in your system landscape. For this purpose, we’ll discuss SAP HANA transport containers. We won’t outline the advanced Change and Transport System (CTS+), which offers options too.

For our descriptions, we assume that you’re already familiar with the development organization and transport in AS ABAP.

5.3.1 Digression: Development Organization and Transport in SAP HANA

To understand the functioning of the SAP HANA transport container better, this section provides some background information on the development organization and transport in SAP HANA.

Development Organization

The development organization in SAP HANA is similar in many ways to AS ABAP. However, it also differs in some essential aspects. As described in Chapter 2, the SAP HANA repository is the central storage of the SAP HANA database development objects.

Within the repository, SAP delivers content below the sap root package. Thus, no customer developments can be created under this package because they might be accidentally overwritten. You can build a parallel package hierarchy for customer developments instead. As a root package, for example, use your domain name.

The system-local package represents a special case. It’s similar to the concept of local packages of AS ABAP. You can use it for development objects that won’t be transported.

Transport

A transport usually takes place in SAP HANA on the basis of a delivery unit. A delivery unit combines packages that are to be transported or delivered together. Conceptually, it broadly corresponds to a software component in the sense of AS ABAP. While you usually work in AS ABAP with the HOME software component, you must always create your own delivery units for customer developments in SAP HANA. To do so, you or an administrator are required to have maintained, in advance, the content_vendor system parameter in the indexserver.ini file using the Administration Console of SAP HANA Studio.

Let’s consider the assignment of a delivery unit and the subsequent transport using an AT_CUSTOMER attribute view. When you create the AT_CUSTOMER attribute view, you assign a package to it. You can maintain a delivery unit in the package properties. To do so, use the context menu entry Edit of the package. You see all existing delivery units in the system in the Quick View using the menu entry DELIVERY UNITS. You can also create new delivery units there. Figure 5.5 shows the relationships among the development object, package, and delivery unit using the example of the AT_CUSTOMER attribute view (the ZA4H_BOOK CHAPTER05 delivery unit isn’t part of the examples provided with this book).
In SAP HANA Studio, you have two options to transport development objects, that is, you can export and import them in the target system:

- Exporting/importing a delivery unit (optionally coupled with CTS+)
- Exporting/importing individual objects (the developer mode)

For a consistent transport of SAP HANA content (which isn’t closely coupled with an ABAP development) in a production system landscape, we always recommend exporting/importing based on delivery units and CTS+.

Figure 5.6 Principle of Schema Mapping

Remember that the AT_CUSTOMER attribute view reads customer data from database table SCUSTOM. This table is part of the flight data model of AS ABAP and is located in the development system in the SAPABD database schema (because the system ID of the ABAP system is ABD). As a result, the attribute view refers to SAPABD.SCUSTOM.
Table SAPABD.SCUSTOM doesn’t exist in the quality assurance system or production system. Due to the different system IDs, the database table resides in the SAPABQ schema in the quality assurance system and in the SAPABP schema in the production system.

Schema mapping enables you to map the SAPABD schema to the SAPABQ schema in the quality assurance system and to the SAPABP schema in the production system.

When maintaining schema mapping, you must consider the following:
- Schema mapping ultimately controls where—that is, in which database schema—an SAP HANA repository development object searches for a database catalog object.
- If no schema mapping is maintained, the authoring schema and physical schema are identical.
- You can map multiple authoring schemas to the same physical schema.
- You cannot assign multiple physical schemas to an authoring schema.
- The SAP HANA content stores references to database objects with the authoring schema. If this can’t be clearly determined (due to a multiple assignment), the system stores the reference with the physical schema.

If you install AS ABAP 7.4 on a SAP HANA database, the installation program creates the SAP<SID> ABAP schema. Furthermore, the installation program also creates at least one schema mapping—that is, from the ABAP authoring schema to the SAP<SID> physical schema.

If you’re interested in further information on the development organization and transport in SAP HANA, please refer to the documentation of the SAP HANA database.

5.3.2 Using the SAP HANA Transport Container
Let’s now discuss the transport of ABAP programs that use native SAP HANA objects via the SAP HANA transport container. For this purpose, we use Program ZR_A4H_CHAPTER5_LIST_CUSTOMER, which accesses the AT_CUSTOMER attribute view of the SAP HANA repository via the ZEV_A4H_CUSTOMER external view of DDIC. The source text of the program is available in Listing 5.8.

Listing 5.8 Sample Report to Be Transported

Both Program ZR_A4H_CHAPTER5_LIST_CUSTOMER and the ZEV_A4H_CUSTOMER external view can be transported readily using the change recording and the transport system of AS ABAP (in principle, this occurs automatically). The AT_CUSTOMER attribute view that forms the basis of the external view, however, isn’t subject to the change recording and transport system of the application server. For this reason, it isn’t available in the target system after a transport (unless you take appropriate measures). Thus, a runtime error occurs in the target system when calling the report. The SAP HANA transport container provides relief here.

Basic Functions
The SAP HANA transport container is available in SAP NetWeaver 7.31 as of SP 5 and as of Release 7.4. It can be used if SAP HANA is the primary database.

The SAP HANA transport container allows you to transport development objects created via SAP HANA Studio using the mechanisms of the
CTS of the ABAP AS (and without the need for a Java stack, which is required for CTS+).

From a technical perspective, the SAP HANA transport container is a logical transport object that acts as a proxy object for exactly one delivery unit. Figure 5.7 illustrates how the SAP HANA transport container works.

Create the transport container

You can only create a SAP HANA transport container using the ABAP Development Tools. In the ABAP perspective, for example, choose the menu path, File • New • Other • ABAP • SAP HANA TRANSPORT CONTAINER. Then enter the name of the delivery unit for which you want to create the transport container. The system automatically derives the name of the transport container (see Figure 5.8; the SAP HANA transport container ZA4H_BOOKCHAPTER05 isn’t part of the examples provided with this book).

Use a prefix namespace

If you want to use a prefix namespace in ABAP, you must assign the desired prefix name to the name of the content_vendor (refer to Section 5.3.1) before creating the transport container. To do so, you can fill database table SNH1_VENDOR_MAPP using the TABLE VIEW MAINTENANCE drop-down.

If the transport properties of the package that is used—in the example, TEST_A4H_BOOKCHAPTER05—are maintained accordingly, the system records the creation of the transport container in a transportable change request.

Synchronization

When you create a transport container, the system automatically synchronizes the contents of this container (once) with the contents of the delivery unit. This means that all objects of the delivery unit are loaded as a packed file on the ABAP AS and are stored there as a byte string in a database table (i.e., table SNH1_DU_PROXY). Strictly speaking, the content of the delivery unit then appears twice in the SAP HANA database:

- In the SAP HANA repository
- Via database table SNH1_DU_PROXY

If, after creating the transport container, you want to synchronize it with the delivery unit—because you’ve made changes to the AT_CUSTOMER...
attribute view, for example—you must do so manually. Use the **Take Snapshot** and **Save** link in this case. You can view the current content of the transport container using the **Contents** tab (Figure 5.9).

**Figure 5.9** Synchronization and Content of a Transport Container

The transport from the development system to the quality assurance and production systems takes place via the CTS mechanisms:

- When exporting (more precisely, during **export release preprocessing**), the system writes the content of the transport container in the data file to the common transport directory of the systems involved in the transport.
- When importing (more precisely, in an **after-import method**), the system reads the transport container’s content from the data file and imports the delivery unit in the SAP HANA database of the target system. Activation of content occurs only if you’ve activated this for the software component of the transport container in table **SNH1_DUP_PREWORK** (in the target system).

You can reproduce the two steps at any time using the transport log.

**Mixed System Landscapes**

Mixed system landscapes represent a special case of the ABAP development on SAP HANA. Imagine that as an ABAP developer, you want to optimize a program for SAP HANA and make use of specific SAP HANA database options. At the same time, however, this program should also be able to run on traditional databases, for example, because your employer uses SAP HANA as a database only in certain areas of the company. In this case, a simplified system landscape might look like that shown in Figure 5.10.

Using a case distinction, you can—to stick with the example of Program **ZR_A4HCHAPTER5_LIST_CUSTOMER**—call the **ZPV_A4H_CUSTOMER** projection view once and the **ZEV_A4H_CUSTOMER** external view once (see Listing 5.8). As a result, you ensure that no errors occur at runtime.

**Figure 5.10** Mixed System Landscape

The implementation of the transport container ensures that no errors occur during the transport, and the SAP HANA content is only imported if the target system of the import is a SAP HANA-based system.

**Recommendations for Using the Transport Container**

When using the transport container, you should note some restrictions:

- When using the transport container, you always transport the complete delivery unit. You can’t transport only the contents of a delivery unit that were changed in a specific time interval.
Unlike development objects that are managed in AS ABAP, the system doesn’t automatically record changes to the content of a delivery unit, and the objects of a delivery unit aren’t locked exclusively for a transport request. It’s thus your responsibility to synchronize the transport container with the delivery unit manually.

When exporting the development objects from the source system, the transport considers only the active objects.

The transport system doesn’t recognize any dependencies between multiple transport containers that are transported simultaneously.

Within the restrictions, the transport container allows you to transport applications consistently that consist partly of ABAP objects and partly of SAP HANA content. We recommend its use if the prerequisites that are described at the start of Section 5.3.2 are fulfilled.

You don’t require the SAP HANA transport container if you use the options described in Chapter 6.
Contents at a Glance

PART I  Basic Principles
  1 Overview of SAP HANA .................................................. 29
  2 Introducing the Development Environment .................... 71
  3 Database Programming Using SAP NetWeaver Application Server ABAP ................................................ 109

PART II  Introduction to ABAP Programming with SAP HANA
  4 Native Database Development Using SAP HANA ............ 161
  5 Integrating Native SAP HANA Development Objects with ABAP ................................................. 231
  6 Advanced Database Programming with ABAP 7.4 ......... 261
  7 Runtime and Error Analysis with SAP HANA ............... 319
  8 Sample Scenario: Optimizing an Existing Application ...... 385

PART III  Advanced Techniques for ABAP Programming for SAP HANA
  9 Integrating Analytical Functionality ............................. 417
  10 Text Search and Analysis of Unstructured Data .......... 441
  11 Decision Tables in SAP HANA .................................. 483
  12 Function Libraries in SAP HANA ............................... 501
  13 Processing Geoinformation ...................................... 523
  14 Practical Tips ............................................................ 539
Dear Reader,

What does SAP HANA mean for you? From high-speed in-memory processing to real-time calculation capabilities, SAP HANA is changing the world of SAP. For many developers it means new concepts to learn, and old concepts to apply to new scenarios. For me, it means a wonderful book to work on and an inspiring author team to collaborate with.

Thorsten Schneider, Eric Westenberger, Hermann Gahm, and Christiaan Swanepoel have made it their mission to make sure that, whatever SAP HANA means for you as a developer, you are prepared to meet its challenges head-on. With thorough examples, they introduce you to the intricacies of ABAP development for SAP HANA, and ensure that as you make your way through this book you will become a more confident user of the SAP HANA development environment.

What did you think about ABAP Development for SAP HANA? Your comments and suggestions are the most useful tools to help us make our books the best they can be. Please feel free to contact me and share any praise or criticism you may have.

Thank you for purchasing a book from SAP PRESS!

Hareem Shafi
Editor, SAP PRESS
Rheinwerk Publishing
Boston, MA
hareems@rheinwerk-publishing.com
www.sap-press.com

Contents

Foreword ......................................................................................... 15
Preface ............................................................................................. 17
Introduction ..................................................................................... 19

PART I Basic Principles

1 Overview of SAP HANA .............................................................. 29
  1.1 Software Components of SAP HANA .................................. 30
    1.1.1 SAP HANA Database ........................................ 30
    1.1.2 SAP HANA Studio ......................................... 32
    1.1.3 SAP HANA Client ............................................. 33
    1.1.4 SAP HANA Extended Application Services .......... 35
    1.1.5 Additional Options ........................................... 36
  1.2 Basic Principles of In-Memory Technology ....................... 39
    1.2.1 Hardware Innovations ...................................... 40
    1.2.2 Software Innovations ........................................ 44
  1.3 Architecture of the In-Memory Database ....................... 55
  1.4 Application Cases and Deployment Options for SAP HANA ......................................................... 58
    1.4.1 Application Cases ............................................. 58
    1.4.2 Deployment Options ........................................ 61
  1.5 How SAP HANA Affects Application Development ........ 63
    1.5.1 New Technical Options .................................... 63
    1.5.2 Code Pushdown ............................................... 64
    1.5.3 Database as White Box ..................................... 66
    1.5.4 Required Qualifications for Developers ............. 68

2 Introducing the Development Environment ....................... 71
  2.1 Overview of Eclipse ....................................................... 71
  2.2 SAP's Eclipse Strategy .................................................... 74
    2.2.1 Unbundling of Eclipse and SAP Software .......... 75
    2.2.2 Central Update Site .......................................... 76
  2.3 Installing the Development Environment ....................... 77
    2.3.1 Installing the Eclipse IDE for Java Developers ... 77
    2.3.2 Installing SAP HANA Studio ................................ 78
Index

@ (for annotations), 294
$parameters, 283

A

ABAP
ABAP code analysis, 386
ABAP project, 84
ABAP runtime environment, 112
ABAP trace, 342, 406
ABAP Unit, 323
code analysis, 321
schema, 115
source code search, 474
type system, 119
ABAP 7.4, 603
ABAP application transport, 549
ABAP buffer cross-user, 577
ABAP CDS, 263
modeling, 300
ABAP Core Development Tools, 78, 79
ABAP Data Dictionary (DDIC), 85, 110, 124, 450
type system, 120
ABAP Database Connectivity — ADBC
ABAP Development Tools for SAP NetWeaver, 32, 233
ABAP resource URL, 86
authorizations, 84
code analysis, 334
components, 79
create program, 88
debugger, 91
eexecute program, 91
favorite packages, 85
perspective, 83
project, 84
Project Explorer, 85
SAP GUI integration, 85
system library, 85
ABAP Development Tools for SAP NetWeaver (Cont.)
template, 89
user settings, 86
ABAP Managed Database Procedure (AMDP), 140, 164, 262, 307, 535
framework, 307
ABAP memory, 580
ABAP profiler, 321, 344
ABAP program analysis, 399
runtime, 399
ABAP table buffer — Table buffer
ABAP Test Cockpit, 89, 321, 332, 335
Accelerator, 59
Access operator, 187
Access time, 41
CPU cache, 42
flash memory, 42
hard disk, 42
main memory, 42
ACID principle, 30
Activation log, 300
ADBC, 139, 239, 452, 466, 546, 584
Administration Console, 93
AFL, 501, 504
Software Development Kit, 503
After-import method, 258
Aggregate function, 132, 289
Aggregation, 286, 299
Alias, 286
Alternative implementation, 542
ALV — SAP List Viewer
AMDP, 466, 508
BADI, 314
call database procedure, 311
class, 308
database connection, 311
enhance, 314
method, 307, 309
tips, 318
troubleshooting, 311

629
AMDP framework → ABAP-Managed Database Procedure framework
Analytic Engine, 428
formula, 439
hierarchy processing, 439
report-report interface, 439
Analytic functionality, 418
Analytic privilege, 102
Analytic view, 102, 195, 210, 228, 585, 600
create, 212
Analytical index, 429
Analytical query → SAP BW Query
Analytical search, 412
Annotation, 263, 294
post-annotation, 295
scope, 294
Append view, 292
Appliance, 29
Application
layer, 64
logic, 64
optimization, 385, 389
Application Function Modeler, 506, 504
Application Function Modeler, 506, 510, 504
Application Function Modeler, 506, 512, 515
Array interface, 584
Association, 274
benefits, 279
filter, 282
local, 276
name, 280
prefix, 280
usage, 278
visible, 276
Attribute, 196
attribute view, 196, 588
Authoring schema, 253
Authorization
analytical, 94, 557
check, 556
package authorization, 94
Authorization (Cont.)
SAP HANA Studio, 94
system authorization, 94
B
B tree index, 613
BADI, 543
BFL, 502, 506
Blocking factor, 154
Breakpoint, 312
dynamic, 91
external, 92
static, 91
BRPlus, 484
Buffer access, 404
Buffer trace, 157
Business Function Library, 502, 506
Business logic, 176
Business process, 483
Business rule management system, 483
Business rule mining, 502
C
Calculation engine, 177, 187
Calculation logic, 65
Calculation view, 102, 171, 195, 219, 429, 497, 585
modeled, 219
SQLScript, 219
Call hierarchy, 349, 402
Call stack, 349, 402
Cardinality, 276, 281
CASE statement, 285
Cash flow, 506
Catalog object, 302
CDS entity, 265, 298
CDS object, 265, 302
CDS → Core Data Service
CDS view, 264, 456
activation log, 300
create, 266
extend, 292
name, 298
outer join, 469
CDS view (Cont.)
usage with parameters, 284
with parameters, 282
CE function, 408
CE plan operator
access operator, 187
CE.VERTICAL_UNION, 188, 508
Change request, 257
Class
CL_ABAP_DBFEATURES, 284
CL_SALV_GUI_TABLE, 299
Client dependency, 272
Client handling, 114, 238, 272, 548
attribute view, 200
Client/server, 21
Cloud deployment, 62
Cluster encoding, 50
Cluster table, 128, 547
COALESCE function, 271
Code
completion, 88, 100, 236
Inspector, 321, 328, 329, 387, 401
pattern, 69
pushdown, 64, 580
template, 267
Code-to-data paradigm, 64, 66, 397
Collective search help, 467
column store, 46, 98, 127, 165, 609
composite index, 617
data type, 448
INSERT ONLY, 613
inverted index, 616
merge, 614
read access, 610
recommendation, 540
write access, 612
column view, 97, 163, 227, 232
column-based data storage, 98
Commit
implicit, 137
Compression
delta compression, 49
technique, 47, 49
Constructor expression, 605
CONTAINS, 454
Content, 33, 101
Control structure, 184
Core Data Service, 35, 138, 262
ABAP, 263
element, 270
SAP HANA, 263, 301
CPU
cache, 41, 42
core, 39
time, 390, 412
Currency conversion
Customizing, 168
parameterize, 218
Cursors, 135, 185, 588
D
Data aging, 38
Data analysis, 418, 421
data class, 127
data control language (DCL), 123
data declaration, 603
data definition language (DDL), 123,
265, 528
Data Dictionary → ABAP Dictionary
Data exploration, 421
Data inconsistency, 551, 554
Data layout, 44
Data manipulation language (DML), 123
Data model
enhancement, 264
syntax, 263
target data source, 276
target data source, 276
type, 117, 125, 166, 246, 299
conversion, 493
geo-spatial, 526
integer, 47
**Data type (Cont.)**

- SHORTTEXT, 166, 448
- ST_GEOMETRY, 166, 527
- TEXT, 166, 448
- user-defined, 263

**Data warehouse**, 60, 420

**DATAO**, 604

**Database relational**, 30, 44, 166

**type system**, 120

**Database catalog**, 96, 162, 232

**Database connection secondary**, 145, 156

**Database independence**, 264

**Database index**, 574

**Database interface** → DBI

**Database layer**, 64

**Database object**, 97

**Database optimizer**, 53, 173, 355

**Database procedure**, 56, 97, 102, 171, 552

**ABAP-managed**, 262

**compilation**, 176

**control structure**, 184

**execution**, 176

**test**, 554

**type**, 178

**Database procedure proxy**, 239, 261, 284, 549, 552

**synchronization**, 249

**Database programming tool**, 149

**Database schema**, 96, 104, 162

**Database Shared Library** → DBSL

**Database table**, 98

**Database trigger**, 98

**Database view**, 125, 138

**DataSource**, 428, 433

**Data-to-code paradigm**, 64

**DBA Cockpit**, 146, 362

**DBI**, 112, 546, 549

**DBSL**, 34, 114

**DCL → data control language (DCL)**

**DDIC**, input help, 467

**DDIC Editor**, 267

**DDL → data definition language (DDL)**

**DDL source**, 265

**create**, 266

**DDL statement**, 54

**Debugging**, 327

**Decision rule**, 483, 493

**Decision table**, 102, 483, 485, 487

**actions**, 485

**conditions**, 485

**create**, 487

**transport**, 499

**Declarative programming**, 175, 588

**Decoupling**, 543, 554

**Delivery unit**, 101, 553

**Delta compression**, 49

**Delta load**, 37

**Delta merge**, 482

**Delta store**, 51, 481, 612

**Design pattern**, 324

**Design time**, 244

**Design time object**, 106

**Development environment**

**ABAP Development Tools**, 79, 83

**installation**, 77

**SAP HANA Studio**, 78

**Development landscape**

**mixed**, 549

**Development object**, 88, 101, 102

**ABAP**, 552

**activate**, 106

**naming convention**, 551

**SAP HANA**, 551

**storage**, 103

**test**, 107

**validate**, 104

**Diagnostics Agent**, 31

**Dictionary encoding**, 47

**Dictionary vector**, 47, 609

**Dimension**, 211

**Direct Extractor Connection**, 38

**DISTINCT**, 286

**Document Analysis Toolkit**, 444

**Domain**, 126

**DRAM**, 42

**Drilldown**, 206

**Dump**, 325, 556

**Dynamic database view**, 484

**Dynamic Random Access Memory**, 42

**E**

**Easy Query**, 439

**Eclipse**, 32

**ABAP development environment**, 32

**composite release**, 73

**editor**, 82

**enhancement point**, 72

**framework**, 71

**menu bar**, 82

**perspective**, 81

**platform**, 32, 71, 73

**plug-in**, 72

**project**, 73

**repository**, 76

**SAP Release Train for Eclipse**, 75

**update site**, 76

**view**, 82

**window**, 81

**Workbench**, 80

**workspace**, 83

**Eclipse Foundation**, 32, 71, 73

**Eclipse Release Train**, 73

**Eclipse SDK**, 72

**Element info**, 293

**Elementary search help**, 467

**Embedded Reporting**, 426

**Embedded Search**, 447

**Encapsulation**, 553

**Encoding**

**cluster encoding**, 50

**dictionary encoding**, 47

**indirect**, 50

**parse encoding**, 50

**prefix encoding**, 50

**run-length encoding**, 50

**regexp**, 556

**Enqueue server**, 111

**Enqueue service**, 550

**Enqueue work process**, 117

**Enterprise Information Management**, 518

**Enterprise search**, 448

**Entity name**, 268

**Entity-relationship model**, 194

**Enumeration value**, 294

**Equi-join**, 202

**Error handling**, 555

**ESRI Shapefile**, 529

**ETL processes**, 37

**Exception**, CX_SY_SQL_UNSUPPORTED_FEATURE, 284

**Execution plan** → Explain plan

**Existence check**, 331

**Expensive SQL statement trace**, 322

**Explain plan**, 321, 355

**call**, 355

**output**, 356

**Export release preprocessing**, 258

**Extended storage**, 38

**Extension index**, 450

**F**

**Fact table**, 210, 211

**Factory pattern**, 542

**Field calculated**, 215

**reference**, 292

**Field list**, 564

**Field symbol**, 604

**Filter**, 197, 286, 299

**Filter condition**, 282

**Filter value**, 203

**Fiscal year**, 208

**Flash memory**, 42

**Flight data model**, 111, 595, 596

**Flowgraph**, 515

**FOR ALL ENTRIES**, 171, 552

**FOR ALL ENTRIES**, 171, 552

**Fuzzy search**, 442, 443, 445, 459

**stopwordListId**, 460
<table>
<thead>
<tr>
<th>Page</th>
<th>Index</th>
</tr>
</thead>
</table>
| 634  | Fuzzy search (Cont.)  
|      | stopwordTable, 460  
|      | textsearch, 461  
|      | threshold value, 445  
|      | Fuzzy search index, 447, 480, 481  |
| 635  | Geo-coding, 534  
|      | Geodata type, 526  
|      | convert, 530  
|      | Geoinformation, 523, 524  
|      | GeoJSON, 529  
|      | Geo-Spatial Engine, 523, 526  
|      | constructor, 528  
|      | create table, 528  
|      | presentation format, 529  
|      | ST_Contains, 532  
|      | Geospatial index, 534  
|      | GIS → Geoinformation system  
|      | Golden rules for database programming, 558  
| GROUP BY, 286  
| Grouping, 299  
| Grouping set, 169  
| GUID, 600  |
| 636  | Hadoop, 38  
|      | Hard disk, 42  
|      | Hardware innovation, 40  
|      | Hardware trend, 39  
|      | Hash partitioning, 54  
|      | Hash value, 54  
|      | HAVING clause, 286, 289  
|      | Help view, 469  
|      | Hierarchy, 197, 206, 212  
|      | level, 206  
|      | parent-child, 206  
|      | High Performance Analytical Appliance  
|      | → SAP HANA  
|      | Hint, 137, 546  
|      | Hit list, 347  |
| 637  | Host variable, 284  
|      | Hybrid application, 64  
|      | Hypernym, 462  
|      | Hyponym, 462  
|      | Join, 100, 188, 271, 305, 407  
|      | complex, 587  
|      | equi-join, 202  
|      | from association, 276  
|      | full outer join, 169  
|      | inner join, 129, 169, 271  
|      | join type, 168  
|      | left outer join, 129, 169  
|      | outer join, 169  
|      | referential join, 197, 201  
|      | right outer join, 169, 282  
|      | self-join, 225  
|      | text join, 197  
|      | type, 281  
|      | Join view → Attribute view  |
| 638  | Kerberos, 57  
|      | Kernel, 112  
|      | Key, 289  
|      | Key attribute, 196, 289  
|      | Key figure, 211  
|      | calculated, 215  
|      | restricted, 212  
|      | Keyword  
|      | Sprojection, 277  
|      | K-means, 511, 515  |
| 639  | L (programming language), 176, 179  
|      | L node, 176  
|      | Large object, 448, 481  
|      | Latency, 19  
|      | Lifecycle management, 31  
|      | List, 299  
|      | List of synonyms, 446, 461  
|      | Literal, 282, 285, 294  
|      | Load distribution, 53  
|      | Local temporary table, 328  
|      | Lock, 550  
|      | Lock object, 117, 126  
|      | Logical unit of work (LUW), 116  
|      | Loop, 588  
|      | LOOP loop, 572  
|      | LUW concept, 116  |
| 640  | Main memory, 39, 42  
|      | Main store, 51, 612  
|      | Mainframe, 21  
|      | MANDT, 273  
|      | Manifest, 72  
|      | Map  
|      | external, 532  
|      | Mass operation, 584  
|      | Master data, 211  
|      | Materialization, 194  
|      | MDX, 33, 56, 170  
|      | Measure → s. key figure, 211  
|      | Merge, 481  
|      | Message server, 111  
|      | Metadata, 263  
|      | Method  
|      | DB_CAPABILITIES, 299  
|      | Modeler perspective, 93  
|      | MODIFY statement, 153  
|      | Modularization, 554  
|      | Modularization unit, 332  
|      | Monitoring view, 480  
|      | Multidimensional expressions → MDX  
|      | Multitenant database container, 56  |
| 641  | Name server, 57  
|      | Namespace, 552  
|      | Native SQL, 466, 582  
|      | ABAP table, 549  
|      | NEW operator, 605  
|      | node.js, 34  
|      | NUMA architecture, 42  |
| 642  | Object instance  
|      | create, 605  
|      | ODBC, 33  
|      | ODBO, 33  |

Index
SAP HANA Repository, 93, 101, 551
SAP HANA repository, 250, 553
SAP HANA repository view, 233, 235
SAP HANA software component, 30
SAP HANA Studio, 30, 32, 93
database catalog, 96
perspective, 93
Program HDBINST, 78
Program HDBSETUP, 78
SQL console, 100, 178
SQL statement, 412
system connection, 95
template, 100
user settings, 96
view modeling, 195
workplace, 95
SAP HANA transport container, 255, 549
SAP HANA UI for Information
Access, 448
SAP HANA view, 67, 232, 548
performance, 584
selection, 237
test, 554
type, 584
SAP HANA Web Workbench, 35
SAP HANA XS Engine, 93, 557
SAP HANA XS → SAP HANA Extended
Application Services
SAP Host Agent, 31
SAP InfiniInsight, 503
SAP IQ (database), 38
SAP Landscape Transformation
Replication Server, 37
SAP List Viewer, 299, 498, 499
with Integrated Data Access, 299
SAP List Viewer with Integrated Data
Access, 289
SAP Lumira, 425
SAP Management Console, 31
SAP memory, 580
SAP NetWeaver AS ABAP, 34
74, 284
7.5, 277, 284
SAP Operational Process
Intelligence, 487
SAP Predictive Analysis, 503
SAP Replication Server, 37
SAP S/4HANA, 61
SAP Solution Manager, 31
Scalar parameter, 180
Scale-out, 41, 117
Schema mapping, 238, 253, 548
Schema → Database schema
Scope
element, 294
extend view, 294
view, 294
Scope operator, 283
Scorecard, 502
Script server, 505
Scrolling, 299
SDK → Software Development Kit
Search
exact, 454
freestyle, 443
fuzzy, 442, 443
linguistic, 443, 446, 454
synonym search, 443
Search facets, 444
Search help, 125, 442, 467
elementary, 467
search help exit, 469
Search case, 285
Secondary connection, 156
Secondary database, 145, 437
Secondary index, 574
Segmentation, 511
SELECT statement, 558
SELECT* statement, 404
Selectivity, 447
Sentiment analysis, 442, 444, 477
Sequence, 98, 163
Server component, 56
Session context, 200
Session variable, 285
Set operation, 568
SQL, 123
SQL statement
for schema
1999 (standard), 164
ADBC, 139
dynamic, 185
dynamic
enforcement, 263
injection, 185
native, 67, 139, 239
syntax, 466
SQL cache, 322, 366, 583
SQL console, 100, 178
SQL injection, 557
SQL performance optimization tool, 387
SQL processor, 56
SQL profile, 401
SQL analysis
system-wide, 362
SQL dialect, 161
SQL expression, in Open SQL, 304
SQL Injection, 135
SQL Monitor, 322, 367, 386, 401
activate, 367
analysis, 369
table, 367
entry point, 370
SQL Performance Tuning Worklist, 379
SQL statement
analysis, 389, 402
CREATE FULLTEXT INDEX, 450
EXEC, 556
EXEC SQL, 546
FOR ALL ENTRIES, 571
SQL statement (Cont.)
ORDER BY, 547
SELECT* , 404
SELECT COUNT(*), 565
UP TO n ROWS, 562
UPDATE, 566
UPDATE ... SET, 566
SQL trace, 151, 321, 350, 401, 407
analyzer, 351
record, 151, 350
SQL view, 166, 265, 280
definition, 273
SQL-92 (standard), 164
SQLDRC library, 33
SQLView, 67, 408
ABAP tables, 549
BREAK statement, 184
case distinction, 175, 184
CREATE PROCEDURE, 180
CREATE TYPE, 181
Debugger, 312
dynamic, 556
EXEC statement, 185
EXECUTE IMMEDIATE statement, 185
explicit typing, 182
implicit typing, 182
input parameter, 245
loop, 184
optimization, 185
output parameter, 241
qualities, 170
reuse, 172
rules, 587
scalar parameter, 180
table parameter, 180
table type, 180
table variable, 172
UDF, 179
user-defined function, 179
WITH RESULT VOW, 181
SRS → Spatial reference system
ST_GEOMETRY, 166
Stack trace, 352
Standard database connection, 116
Standard view, 166
Star schema, 194, 210, 211
SFLIGHT, 595, 596
Shadow column, 449
Shadow column, 580
Shared buffer, 580
Shared memory, 580
Shared object, 580
Side panel, 440
Side-by-side scenario, 22
Simple case, 285
Single transaction analysis, 321, 354
Size category, 127
Slice-and-dice operation, 211
Smart Data Access, 38
Smart Data Access (SDA), 165
Software Development Kit, 503
Software innovation, 44
Sort behavior, 329, 547
Sorting, 299
Sparse encoding, 50
Spatial reference identifier, 524
Spatial reference system, 524
SQL Monitor, 322, 367, 386, 401
SQL Profile, 502
SQL processor, 56
SQL profile, 401
SQL statement
for schema
1999 (standard), 164
ADBC, 139
dynamic, 185
dynamic
enforcement, 263
injection, 185
native, 67, 139, 239
syntax, 466
SQL cache, 322, 366, 583
SQL console, 100, 178
SQL injection, 557
SQL performance optimization tool, 387
SQL processor, 56
SQL profile, 401
SQL analysis
system-wide, 362
SQL dialect, 161
SQL expression, in Open SQL, 304
SQL Injection, 135
SQL Monitor, 322, 367, 386, 401
activate, 367
analysis, 369
table, 367
entry point, 370
SQL Performance Tuning Worklist, 379
SQL statement
analysis, 389, 402
CREATE FULLTEXT INDEX, 450
EXEC, 556
EXEC SQL, 546
FOR ALL ENTRIES, 571
SQL statement (Cont.)
ORDER BY, 547
SELECT * , 404
SELECT COUNT(*), 565
UP TO n ROWS, 562
UPDATE, 566
UPDATE ... SET, 566
SQL trace, 151, 321, 350, 401, 407
analyzer, 351
record, 151, 350
SQL view, 166, 265, 280
definition, 273
SQL-92 (standard), 164
SQLDRC library, 33
SQLScript, 67, 408
ABAP tables, 549
BREAK statement, 184
case distinction, 175, 184
CREATE PROCEDURE, 180
CREATE TYPE, 181
Debugger, 312
dynamic, 556
EXEC statement, 185
EXECUTE IMMEDIATE statement, 185
explicit typing, 182
implicit typing, 182
input parameter, 245
loop, 184
optimization, 185
output parameter, 241
qualities, 170
reuse, 172
rules, 587
scalar parameter, 180
table parameter, 180
table type, 180
table variable, 172
UDF, 179
user-defined function, 179
WITH RESULT VOW, 181
SRS → Spatial reference system
ST_GEOMETRY, 166
Stack trace, 352
Standard database connection, 116
Standard view, 166
Star schema, 194, 210, 211
Statement
FOR ALL ENTRIES, 331
Statement transformation, 149
Statistic record, 321
Statistics server, 37
Stop word, 447, 460
Stored procedure → Database procedure
String, 49
Structure, 298, 606
Structure component, 298
Structured Query Language → SQL
Subquery, 133, 571
Subselect, 286
Synonym, 98, 106, 163
Syntax check, 89
Syntax warning prevent, 284
System landscape, mixed, 258
System optimization, 388
System schema, 96, 115
Table, 162
internal, 606
replicated, 621
SFLIGHT, 111
table buffer, 114, 331, 549, 573, 579
table definition, 98
table parameter, 180
table statistics, 398
table type, 180
table variable, 172
temporary table, 243
Target cardinality, 281
Target data source, 276
cardinality, 276
Temporary table, 165
Term mapping, 461
Text, 555
Text analysis, 442, 444, 476
Text join, 197
Text mining, 36
Text search, 444, 448
Time data
generate, 208
Time zone, 544
Token, 448, 476
Tools for performance analysis
runtime check monitor (Transaction SRTCM), 377
Totals table, 194
Trace, 337
Tracing, 328
Transaction
ATC, 321
DBACOCKPIT, 146, 322, 362
RFAI, 430
RSDD_HM_PUBLISH, 429
SAT, 321, 342, 399, 406
SCI, 321, 329
SE11, 85
SE00, 473
SFW5, 475
SQLM, 322, 367
SRTCM, 377
ST04, 362
ST05, 151, 156, 157, 321, 374, 407
ST12, 321, 354
ST22, 325
STAD, 321, 337, 390, 397, 406
STATS, 342
SWLT, 322, 379, 380, 401
Transaction data, 211, 598
Transaction SRTCM, check, 377
Transactional system, 420
Transport
change request, 257
developer mode, 252
log, 258
logical transport object, 256
mixed system landscape, 258
recommendations, 259
synchronization, 257
transport container, 255
Transport container, 254
TREX, 447
Trigger, 163
Troubleshooting, 319, 321, 322
Truth value, 294
U
UNION, 286
UNION ALL, 286, 290
Unit conversion, 217
Unit test, 322, 324, 391, 555
Update, 111
Update module, 137
V
Validation, 105
Value suggestion, 443
Variable
global, 285
scope, 505
session, 285
Version history, 108
View, 98, 162, 194, 265
analytic view, 195, 210
attribute view, 195
calculation view, 195, 219
database view, 138
dictionary view, 233
e external, 238, 261, 549, 552
V (Cont.)
SQL view, 166
View-on-view schema, 300
W
Web Dynpro ABAP, 472
context, 472
Eclipse, 473
Weighted average, 506, 507
WHERE, 282, 286
clause, 558, 574
Where-used list, 90
White list, 557
Wildcard, 441
WITH PARAMETERS, 283
Word dictionary, 479
Work process, 111, 244
Wrapper function
AFL, 506
X
XS Engine, 35
XS Server, 57
ABAP Development for SAP HANA

Hermann Gahm, Thorsten Schneider, Christiaan Swanepoel, Eric Westenberger

625 Pages, 2016, $79.95
ISBN 978-1-4932-1304-7

www.sap-press.com/3973

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.