

Reading Sample

This excerpt from Chapter 4 provides a quick glimpse of database views and the view modeler in SAP HANA Studio. This chapter kicks off your introduction to ABAP programming in SAP HANA. You'll also walk along with a helpful example from the SFLIGHT data model that will help you apply theory to reality.



"Introduction"
"View Modeling in SAP HANA Studio"



Contents



Index



The Authors

Thorsten Schneider, Eric Westenberger, Hermann Gahm

ABAP Development for SAP HANA

609 Pages, 2014, \$69.95

ISBN 978-1-59229-859-4



www.sap-press.com/H3320

Introduction

Today's business world is extremely dynamic and subject to constant change, with companies continuously under great pressure to innovate. SAP HANA's vision is to provide a platform that can be used to influence all business processes within a company's value chain in real time. However, what does this key term *real time* mean for business applications?

In technological terms, it describes, in particular, the availability of essential functions without *unwanted* delays. The environment in which a technology is used and the time when this occurs strongly influences the functions needed and what is deemed to be an *acceptable* delay. Before we discuss the software currently used for enterprise management, we wish to illustrate this using an example from daily life, namely telecommunications.

Early forms of communication (for example, telegraphs) were very limited in terms of their usage (range, availability, and manual effort). At that time, however, it was an immense improvement in terms of the speed at which messages were previously exchanged. Then, with advent of the telephone, it became possible to establish flexible connections over long distances. Once again, however, users of this technology had to allow for various delays. Initially, it was necessary to establish a manual connection via a switchboard. Later, and for a very long time after, there were considerable *latencies* with overseas connections, which affected and complicated long-distance telephone conversations. Today, however, telephone connections can be established almost anywhere in the world and done so without any notable delay. Essentially, every leap in evolution has been associated with considerable improvement in terms of *real-time quality*.

In addition to a (synchronous) conversation between two people, asynchronous forms of communication have always played a role historically (for example, postal communication). In this context, the term *real time* has a different meaning because neither the sender nor the receiver needs to actively wait. Asynchronous communication has also undergone

Example: real time in telecommunications

immense changes in recent years (thanks to many new variants such as email, SMS, and so on), which, unlike postal mail, facilitates a new dimension of real-time communication between several people. Furthermore, there is an increasing number of non-human communication users such as devices with an Internet connection, which are known as *smart devices* (for example, intelligent electricity meters).

Most people will testify to the fact that, nowadays, electronic communication is available in real time. Nevertheless, in our daily lives, some things still cannot occur in real time despite the many advances in technology (for example, booking a connecting flight during a trip). It is safe to say that in the future, many as yet inconceivable scenarios will be so widespread that currently accepted limitations will become completely unacceptable.

Real time in business

The above example of telecommunications technology contains some basic principles that can also be applied to business software. On the one hand, there are corporate and economic developments such as globalization and the increasing mobility of customers, and employees who are the driving forces for new types of technology. Companies operate globally and interact in complex networks. Furthermore, customers and employees expect to be able to access products and services at all times, from anywhere in the world.

On the other hand, there are technological innovations that pioneer new paths. The Internet is currently a catalyst for most developments. Enormous volumes of data are simultaneously accessible to a large part of the world's population (that is, *in real time*). The Internet also provides a platform for selling all types of products and services, which has led to a phenomenal increase in the number of business transactions conducted each day. Companies can gain a massive competitive advantage each time a business process (for example, procurement, production, billing, and so on) is optimized. In most industries, there is great potential here, which can be realized by establishing a closer link between operational planning and control in real time.

Today's customers also expect greater customization of products and services to their individual wishes (for example, to their personal circumstances). In particular, companies that are active in industries subject to major changes (for example, the energy industry, financial providers or specific forms of retail) are under a great deal of pressure to act.

The term *real time* shapes the evolution of 40 years of SAP software. Even the letter "R" in SAP's classic product line, R/3, stood for *real time*. SAP's initial concepts in the 1970s, which paved the way for the development of R/1, facilitated the on-screen entry of business data, which, compared to older punch card systems, provided a new quality of real time. Consequently, processes such as payroll accounting and financial accounting were the first to be mapped electronically and automated. With SAP R/2, which was based on a *mainframe* architecture, SAP added further ERP modules (*Enterprise Resource Planning*), for example, Materials Management, to these applications areas. As part of this release, SAP introduced the reporting language ABAP. (Originally, ABAP stood for *Allgemeiner Berichtsaufbereitungsprozessor*, which means "General Report Creation Processor," but this was later changed to *Advanced Business Application Programming*). ABAP *reports* were used to create, for example, a list of purchase orders, which was filtered according to customer and had *drilldown options* for line items. Initially, this was available in the background only (*batch mode*). However, it later became available in *dialog mode*.

Thanks to the *client/server architecture*, in particular, and the related scaling options in SAP R/3, it was possible for a large number of users within a company to access SAP applications. Consequently, SAP software, in combination with consistent use of a database system and an ever-growing number of standard implementations for business processes, penetrated the IT infrastructure of many large companies, thus making it possible to use an integrated system to support transactional processes in real time (for example, a *just-in-time production process*).

Parallel to these developments is the fact that, over the past 20 years, it has become increasingly more important to analyze current business processes, the purpose of which is to continuously obtain information in order to make better operational and strategic decisions. Within this *business intelligence trend*, however, it soon became clear that in many situations, it is technically impractical to perform and integrate the required analyses into a system that already supports business processes. Parallel processing of analyses and transactions involving extremely large amounts of data overloaded most systems, with the database, in particular, emerging as a limiting factor. This was one of the reasons why SAP created a specialized system for analytical scenarios, which you currently know as

Real time at SAP

Importance of business intelligence

SAP NetWeaver Business Warehouse (BW). In addition to new options for consolidating data from multiple systems and integrating external data sources, the use of the data warehouse system for operational scenarios is, unfortunately, fraught with losses when data is processed in real time. First of all, data needs to be extracted and replicated, which, in practice, can cause a time delay, ranging from several hours up to one week, until the current data is available at the correct location. This was SAP's starting point for SAP HANA; in other words, no more delays in receiving key information for a business decision.

SAP HANA as
a database

SAP likes to describe SAP HANA as a platform for real-time data management. To begin with, SAP HANA is a high-end database for business transactions (*Online Transaction Processing*, OLTP) and reporting (*Online Analytical Processing*, OLAP), which can use a combination of in-memory technology and column-oriented storage to optimize both scenarios. In the first step, SAP HANA was used as a *side-by-side scenario* (that is, in addition to an existing traditional database) to accelerate selective processes and analyses. Soon after, it was supported as a new database for SAP NetWeaver BW 7.3. In this way, SAP demonstrated that SAP HANA not only accelerates analytical scenarios, but that it can also be used as a primary database for an SAP NetWeaver system. With the announcement of *SAP Business Suite powered by SAP HANA*, it is now also possible for customers to fully benefit from SAP HANA technology within standard SAP applications. The new SAP NetWeaver release 7.4 underlying this constellation (in particular, SAP NetWeaver Application Server (AS) ABAP 7.4) will therefore play a key role in this book. Furthermore, the sample programs in this book require ABAP 7.4. However, we will always indicate which functions you can also use with earlier releases of SAP NetWeaver. A cloud-based trial version of ABAP 7.4 on SAP HANA is available. For more information, see Appendix E.

SAP HANA as
a platform

Furthermore, SAP HANA provides many more functions that go beyond the usual range of functions associated with a database. In particular, these include extensive data management functions (replication, extraction – transformation – load (ETL), and so on) and data analysis functions (for example, *data mining* by means of a *text search* and *predictive analysis*). Many of these technologies and functions are not exclusively available to SAP HANA. In fact, many software systems now manage data in the

main memory or use column-oriented displays. SAP itself developed and used in-memory technology long before SAP HANA came into being (for example, in *SAP NetWeaver BW Accelerator*). Similarly, a number of software manufacturers (including SAP itself) are involved in data analysis, especially in the context of business intelligence and information management solutions. One key benefit of SAP HANA is the fact that it offers this function in the same system in which business transactions are running. If, for example, you want to run SAP Business Suite on SAP HANA, these enhanced functions are available to you immediately, without the need to extract data. Furthermore, since SAP HANA incorporates the key data structures of the SAP Business Suite, installed functions already exist for some standard operations (for example, currency conversion).

Therefore, what does SAP HANA mean for standard SAP applications that run on the ABAP application server? What changes are occurring in ABAP programming? What new options does SAP HANA open up in terms of ABAP-based solutions? These three questions will be at the heart of this book. Furthermore, we will always use examples to explain the relevant technical backgrounds and concepts, rather than simply introducing you to the technology behind the new tools and frameworks. In particular, we will focus on the basic functions of ABAP development and database access via ABAP. We will introduce existing or planned supports for SAP HANA in ABAP-based *frameworks* as an overview or outlook because a detailed description would generally require an introduction to how these components work. (Examples here include *Embedded Search* and *BRFplus*.) In the examples contained in this book, we will use simple ABAP reports as the user interfaces, for the most part. In two detailed examples, however, we will also create web-based interfaces with Web Dynpro ABAP and HTML5.

We made the decision to divide this book into three parts. In Part I, "Basic Principles," we will introduce you to the basic principles of in-memory technology. Here, you will get to know the development tools as well as refresh your knowledge of ABAP database programming. In **Chapter 1**, "Overview of SAP HANA," we will start with an overview of the components of SAP HANA and potential usage scenarios in conjunction with ABAP. In **Chapter 2**, "Introducing the Development Environment," we will introduce you to the development environment, which comprises

ABAP development
on SAP HANA

Structure of the
book: Part I

SAP HANA Studio and the ABAP development tools for SAP NetWeaver (also known as ABAP in Eclipse). **Chapter 3**, “Database Programming using SAP NetWeaver AS ABAP,” will discuss the use of Open SQL and Native SQL to access the HANA database from ABAP programs.

Part II In the second part of the book, “Introduction to ABAP Programming with SAP HANA,” you’ll learn how to store data from an ABAP application (for example, certain calculations) in SAP HANA, thus achieving considerably better performance. Here, the focus will be on programming and modeling SAP HANA, as well as accessing SAP HANA from ABAP programs. In **Chapter 4**, “View Modeling in SAP HANA Studio,” we will discuss the various ways in which you can create data views, which can then be used to conduct calculations and analyses in relation to ABAP table content. Then, in **Chapter 5**, “Programming Options in SAP HANA,” you will learn about SQLScript, which is the programming language for database procedures in SAP HANA. You’ll also learn how to use ABAP to access these procedures. In **Chapter 6**, “Application Transport,” we will explain how you can transport ABAP development objects alongside the objects contained in the SAP HANA Repository. Together with the tools in **Chapter 7**, “Runtime and Error Analysis on SAP HANA,” you now have the basic tools that we, as ABAP developers, believe you need to know within the context of SAP HANA. Part II of this book will conclude with **Chapter 8**, “Sample Scenario: Optimizing an Existing Application,” where we will use the technologies and tools introduced earlier in this book to optimize an existing ABAP implementation for SAP HANA, step by step.

Part III In Part III of this book, “Advanced Techniques for ABAP Programming for SAP HANA,” we will introduce you to some advanced SAP HANA functions, which are not available in classic ABAP development. Even though the chapters contained in Part III of this book are based on the content of the preceding part, Part III can be read in isolation. In **Chapter 9**, “Text Search and Analysis of Unstructured Data,” we will start by describing the *fuzzy search* in SAP HANA and we will show you how you can use it to improve, for example, input helps within an ABAP application. Then in **Chapter 10**, “Integrating Analytical Functionality,” we will introduce you to the capabilities of the embedded SAP NetWeaver BW technology in conjunction with ABAP developments on SAP HANA and existing SAP Business Intelligence products. You can then use decision tables, whose

usage we will discuss in **Chapter 11**, “Decision Tables in SAP HANA,” to use rules that enable you to design parts of an application in a very flexible manner. As a final element, we will show you in **Chapter 12**, “Function Libraries in SAP HANA,” how you can, for example, incorporate statistical functions for *predictive analysis* into an ABAP application. In **Chapter 13**, “Sample Scenario: Development of a New Application,” we will create a small sample application that connects innovations achieved with SAP HANA to ABAP transactions. The book concludes with **Chapter 14**, “Practical Tips,” which contains our recommendations for optimizing ABAP applications on SAP HANA as well as some new developments in relation to ABAP applications on SAP HANA.

As you will see while reading this book, the HANA platform provides a whole host of options. You do not necessarily have to use all of the elements introduced here in ABAP custom developments on SAP HANA. For some new types of functions, the use of *low-level technologies*, which you may only have used occasionally in the past, is currently necessary in the ABAP application server (for example, Native SQL). However, we are convinced that the use of new options holds great innovation potential in terms of new developments. For this reason, we strive to adopt a certain pioneering approach, which is evident in some of the examples provided in this book.

As an example, we will use the *flight data model* in SAP NetWeaver (also known as the `SFLIGHT` model), which was and remains the basis for many training courses, documentation, and specialist books relating to SAP ERP. Thanks to its popularity, the new features and paradigm shifts involved with SAP HANA can be explained very well using this example. The underlying business scenario (airlines and travel agencies) is also very well suited to explaining aspects of real time because, in recent years, the travel industry has been subject to great changes as a result of globalization and the Internet. Furthermore, the volume of data in the context of flight schedules, postings, and passengers has continued to grow.

Throughout this book, you will find several elements that will make it easier for you to work with this book.

Highlighted information boxes contain helpful content that is worth knowing, but lies somewhat outside the actual explanation. In order to

Deploying new technologies

Sample data model

How to use this book

help you immediately identify the type of information contained in the boxes, we have assigned symbols to each box:

- [+]** Tips marked with this symbol will give you special recommendations that may make your work easier.
- [»]** Boxes marked with this symbol contain information about additional topics or important content that you should note.
- [!]** This symbol refers to specifics that you should consider. It also warns about frequent errors or problems that can occur.
- [Ex]** Examples marked with this symbol make reference to practical scenarios and illustrate the functions shown.

In addition, you will find the code samples used throughout as a download on this book's web page at *www.sap-press.com*.

We hope that, with this book, we can give you a comprehensive tool that will support you in using the HANA technology in ABAP programs. Finally, we hope that you enjoy reading this book.

Acknowledgments

We wish to thank the following people who supported us by partaking in discussions and providing advice and feedback during the writing of this book:

Arne Arnold, Dr. Alexander Böhm, Ingo Bräuninger, Ralf-Dietmar Dittmann, Franz Färber, Markus Fath, Dr. Hans-Dieter Frey, Boris Gebhardt, Dr. Heiko Gerwens, Dr. Jasmin Gruschke, Martin Hartig, Vishnu Prasad Hegde, Rich Heilman, Thea Hillenbrand, Dr. Harshavardhan Jegadeesan, Thomas Jung, Bernd Krannich, Dr. Willi Petri, Eric Schemer, Joachim Schmid, Sascha Schwedes, Christiaan Edward Swanepoel, Welf Walter, Jens Weiler, Stefan Weitland, Tobias Wenner, and Andreas Wesselmann.

Thank you so much—this book would not have been possible without your help.

Thorsten Schneider, Eric Westenberger, and Hermann Gahm

Using SAP HANA, you can perform business calculations directly on the original data in the main memory without the need to transform data. Many of these calculations can be modeled graphically as special data views in the SAP HANA Studio without having to write program code. When using ABAP 7.4, these views can then be imported to the ABAP Data Dictionary.

4 View Modeling in SAP HANA Studio

In this chapter, we'll kick off Part II of this book by looking in detail into *database views*. You may be asking yourself why exactly this topic plays such a big role in the context of SAP HANA. To answer this question, we would like to go back a little and briefly explain the underlying reasoning.

The business data of a domain are stored (usually in a normalized form) in a set of database tables that are connected via foreign key relationships (a so-called *entity-relationship model*). Using this data model, single records can be efficiently created, selected, and modified. However, if data access becomes more dynamic and complex, or if certain analyses or checks are necessary, the data must be transformed.

So far, the pattern that was most commonly used for these transformations is that the data is read from the database and used by a program for calculations before storing the result back in the database. This is referred to as *materialization* of the transformed data.

A simple example is the materialization of a totals calculation in a special column or *totals table*. In principle, the same pattern is used for data structures of a *business intelligence system*, where the original data is transformed into a form that can be used more efficiently for analyses (*star schema*). This materialization was primarily done for performance reasons in the past, since it was not possible to perform the transformations *on the fly* at runtime when users submitted a query. However, since the different data structures had to be synchronized (which is usually done with

some time offset), this performance gain also led to higher complexity and prevented a real-time experience for users. Using SAP HANA, this redundancy can now be eliminated in many scenarios. From a technical perspective, this means that the transformations are performed in real time, using the original data. As a consequence, database views are an important element, used in this context to express transformations for read accesses.

SQL views Every relational database system provides an option for defining views. These *standard views* (also referred to as *SQL views*) are defined in the database catalog using the `CREATE VIEW` statement essentially as an alias for a SQL query:

```
CREATE VIEW <name> AS SELECT <SQL query>
```

Being a relation database, SAP HANA also supports SQL views; these views differ from the views of other databases only in their SAP HANA-specific SQL dialect.

Column views In addition to these views, SAP HANA also supports so-called *column views*, which usually provide a better performance and a significantly wider scope of functions. Moreover, these views use the *engines* described in Section 1.3 when queries are executed. The currency conversion of monetary amounts is a good example for functionality that is not available directly via SQL. A prerequisite for using column views is that all involved tables are stored in the column store in SAP HANA, which should be the standard for pretty much all business data (see also Chapter 14). In SAP HANA Studio, both the existing SQL views and the column views are visible in the database catalog (Figure 4.1).

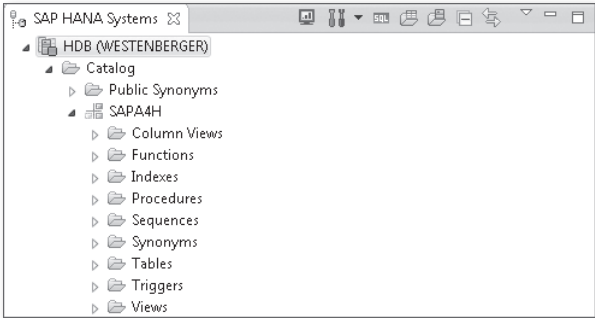


Figure 4.1 SQL Views and Column Views in the Database Catalog

In Section 3.2.2, we explained how simple operations (e.g., for summation or existence checks) can be expressed using Open SQL. However, the key figures in real business applications are usually much more complex. Units of measure and currencies, for example, play an important role and may have to be considered in mathematical operations by using conversions. Time stamps (day, time) for business processes are also very important—including the fiscally correct handling of (business) year, month, or quarter.

When dealing with these operations, standard SQL-based table access reaches its limits. And this is where one of the greatest advantages of SAP HANA comes into play: The integrated engines (see Section 1.3) provide reusable functions tailored for business processes which can be integrated in column views and then accessed using standard SQL. Column views thus enhance the scope of functions for defining database views.

In the scope of this chapter, we will create relatively simple analyses of flight bookings and the seat utilization of flights based on the `SFLIGHT` data model. In addition to some master data of a flight connection (airline, departure, and destination location), statistical information on seat utilization, revenues, and baggage should also be displayed per quarter. To create these analyses, we will use the different modeling options provided by SAP HANA and explain their properties and areas of use.

Reference example
for this chapter

The following types of views will be discussed:

View types

- *Attribute views* to define master data views (see Section 4.1). We will introduce the different options available to create table joins and explain how calculated attributes can be added to a view.
- *Analytic views* can be used for calculations and analyses based on transaction data using a star schema (see Section 4.2). We will explain how you can define simple and calculated key figures and add dimensions. As a special case of calculated key figures, we will describe currency conversion and unit conversions.
- Using *calculation views*, you can flexibly combine views and basic data operations (see Section 1.3). We will describe both the modeling and the implementation of calculation views using SQLScript. Since SQLScript will be described in detail in Chapter 5, the sample implementation used in this chapter will be kept rather simple.

After demonstrating how to define and test these views in SAP HANA Studio, we will describe external access. You will first learn how to access the views from Microsoft Excel, which provides a simple option for first tests and analyses.

In the remaining sections of this chapter, we will describe how these views are accessed from ABAP. We will explain both native access via ABAP Database Connectivity (ADBC)—the only option for ABAP releases before ABAP 7.4—and the new options available as of ABAP 7.4.

4.1 Attribute Views

Overview and usage scenarios

Attribute views comprise a number of fields (columns) from database tables, which are linked through foreign key relationships. Moreover, attribute views provide a way to define calculated columns and hierarchical relationships between individual fields (e.g., parent-child relationships). They are especially relevant in the following scenarios:

- ▶ As components of other view types, especially as *dimensions* of analytic views (see Section 4.2) or for a more general purpose as *nodes* in calculation views (see Section 4.3).
- ▶ As a data provider for text searches across several tables (see Chapter 9).

Reference examples for this section

In this section, we will create a number of such views to demonstrate different functional aspects. The reason for creating several views is that it is not possible or not useful to use all functions for all tables. To give you an overview of the views used in the examples of this section, they are listed in Table 4.1 together with a description and the corresponding functionality.

Column	Description	Functionality
AT_FLIGHT_BASIC	Simple view for table SFLIGHT	First basic example
AT_FLIGHT	Flight data plus information from the flight plan and information on the airlines	Different join types and calculated fields

Table 4.1 Sample Attribute Views Used in this Section

Column	Description	Functionality
AT_MEAL	List of meals served on flights with (language-dependent) description	Text joins and filter values
AT_PASSENGER	View of passenger data and address information	Hierarchy
AT_FLIGHT_FISCAL	Flight data with assignment to accounting periods	Fiscal calendar
AT_FLIGHT_GREG	Flight data with assignment to year, quarter, calendar week	Gregorian calendar
AT_TIME_GREG	Pure time hierarchy (year, quarter, calendar week)	Attribute view of type TIME

Table 4.1 Sample Attribute Views Used in this Section (Cont.)

The views AT_FLIGHT, AT_PASSENGER, and AT_TIME_GREG will also be used in Section 4.2.

4.1.1 Basic Principles

Before describing how attribute views are modeled, let's take a quick look at the most important concepts. Since attribute views can be used to create data views based on several tables that are linked via different types of joins, they can also be referred to as *join views*. The different join types will be introduced in this section. Because joins play a major role when dealing with attribute views, accesses to attribute views are handled by the *join engine* in SAP HANA.

Join views

When modeling attribute views, we differentiate between the following concepts:

Modeling concepts

- ▶ *Attributes* refer to the columns of the attribute view. You can add columns from one or several physical tables or define additional calculated columns.
- ▶ *Key attributes* are those attributes of the view that uniquely specify an entry. These play an important role when the view is used as a *dimensions* of an analytic view (see Section 4.2.2).

- *Filters* define restrictions applied to the values of a column (similar to a WHERE condition in a SELECT statement).
- *Hierarchies* are relations defined for the attributes such as a parent-child relationship (see Section 4.1.4).

The main advantage of attribute views is the possibility to define a view based on fields from several tables. In contrast to the ABAP Data Dictionary views presented in Section 3.2.3, which can comprise only inner joins, attribute views in SAP HANA allow you to use a greater variety of join types.

Sample data

Before describing the details of join modeling, we would like to first introduce the different join types of the SQL standard. To do so, we will use the known tables SFLIGHT (flights) and SCARR (airlines) with a foreign key relationship via the CARRID field (for the sake of simplicity, the client is disregarded in the excerpt in Table 4.2). The tables have an n:1 relationship and the SCARR table may contain airlines for which no flight is entered in the SFLIGHT table (e.g., the airline "UA" in Table 4.2).

Table SFLIGHT			Table SCARR	
CARRID	CONNID	FLDATE	CARRID	CARRNAME
AA	0017	20130101	AA	American Airlines
...
LH	400	20130101	LH	Lufthansa
LH	400	20130102
...	UA	United Airways

Table 4.2 Sample Data from the Tables SFLIGHT and SCARR to Explain Join Types

Inner/outer joins

When defining joins, we differentiate between inner and outer joins. In case of an *inner join*, all combinations are included in the result if there is a matching entry in both tables. With an *outer join*, results that are present only in the left table (*left outer join*), only in the right table (*right outer join*), or in any of the tables (*full outer join*) are also included. To differentiate between left and right, the join order is used. Full outer joins are not supported for attribute views.

The differences between the join types will be explained based on the following SQL examples for selecting flights and the corresponding airline names. The first example comprises an inner join. Since the airline "UA" is not present in the sample data for the sample data for the SFLIGHT table, there is no matching entry in the result set:

```
select s.carrid, s.connid, c.carrname from sflight as s inner
join scarr as c on s.carrid = c.carrid
```

In case of a right outer join, where SCARR is the right-hand table, an entry for the airline "UA" is displayed in the result set, even though there is no corresponding entry in the SFLIGHT table. The columns carrid and connid thus display the value NULL:

```
select s.carrid, s.connid, c.carrname from sflight as s right
outer join scarr as c on s.carrid = c.carrid
```

Similarly, "UA" is also included in the result set in case of a left outer join with SCARR as the left-hand table. If the data model assumes that a corresponding airline exists for every entry of a flight (but not necessarily the other way around), the two outer join variants are functionally equivalent.

```
select s.carrid, s.connid, c.carrname from scarr as c left
outer join sflight as s on s.carrid = c.carrid
```

In addition to the presented standard joins, two other special join types are used when modeling attribute views in SAP HANA:

- *Text joins* can be used to read language-dependent texts from a different table. For this purpose, the column with the language key must be included in the text table; at runtime, a filter for the correct language is then applied based on the context. The next section shows an example for using text joins.
- *Referential joins* provide a special way of defining an inner join; with this join type, *referential integrity* is assumed implicitly (which has advantages with regard to performance). So, when using a referential join and no field from the right-hand table is queried, it is not checked if there is a matching entry. It is assumed that the data is consistent. Referential joins are often a useful standard when defining joins in attribute views.

SQL examples

Text joins and referential joins

[>>]

Attribute Views Only Support Equi-Joins

When formulating join conditions, you can use further expressions (e. g., <, >) in SQL that go beyond checking the equality of columns (*equi-join*), as shown in the following example:

```
SELECT ... FROM ... [INNER|OUTER] JOIN ... ON col1 < col2 ...
```

However, attribute views support only equi-joins.

4.1.2 Creating Attribute Views

Attribute views can be defined via the **MODELER** perspective in SAP HANA Studio, which was introduced in Section 2.4.3. To create a view, select **NEW • ATTRIBUTE VIEW** from the context menu of a package in the **CONTENT** node. You first have to specify a name and a description in the dialog shown in Figure 4.2.

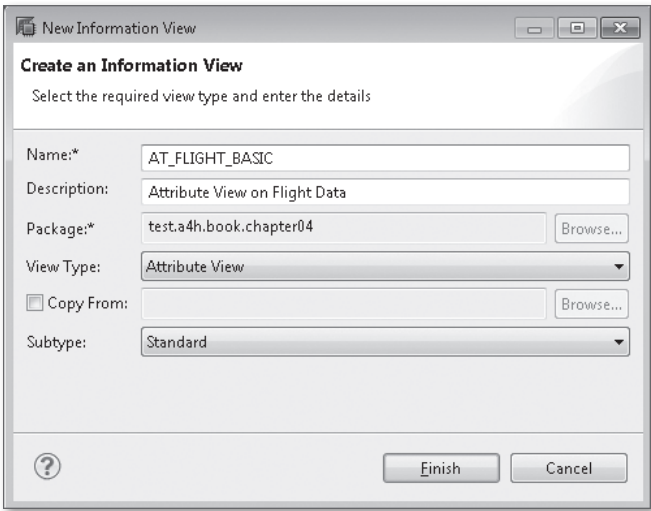


Figure 4.2 Creating an Attribute View

In this dialog, you can also copy an existing view as basis for a new attribute view. When selecting **SUBTYPE**, you can create special types of attribute views (e.g., for time hierarchies, which will be explained in more detail

in Section 4.1.5). When clicking the **FINISH** button, the attribute view is created and the corresponding modeling editor opens.

The editor used to define an attribute view comprises two sections: **DATA FOUNDATION** and **SEMANTICS**. These are displayed as boxes in the **SCENARIO** pane on the left-hand side (see Figure 4.3). By selecting each node, you can switch between defining the data basis (**DATA FOUNDATION**) and the semantic configuration (**SEMANTICS**).

The **DATA FOUNDATION** is used to add tables, define joins, and add attributes. Figure 4.3 shows a simple example based on the **SFLIGHT** table.

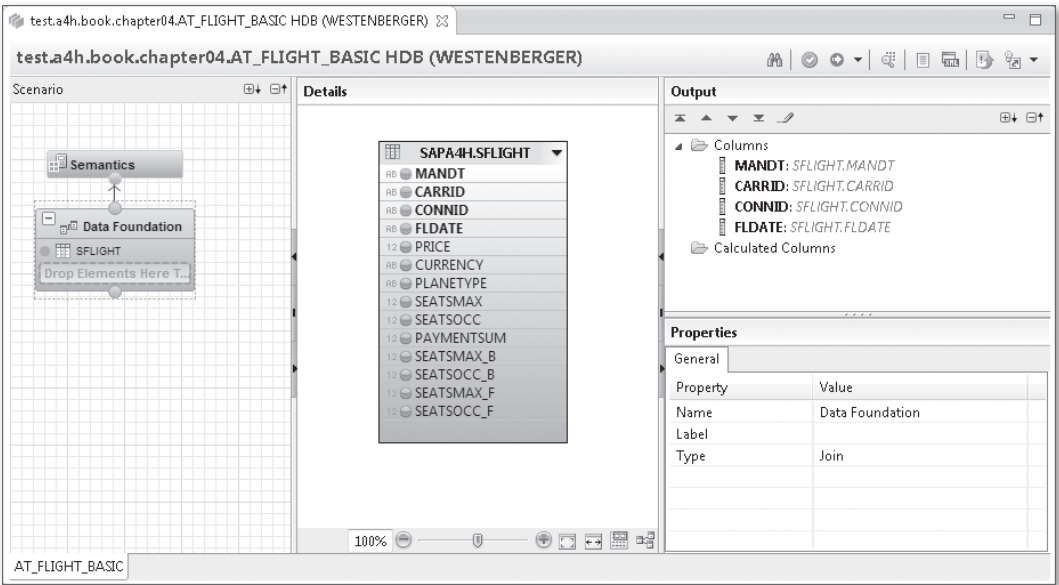


Figure 4.3 Definition of the Data Foundation

By selecting the node **SEMANTICS**, you can maintain further metadata for the attribute view. You can, for example, specify the following:

- You can specify if an attribute is a key field of the view. Note that every attribute view must contain at least one key field. In addition, you can define texts (*labels*) for attributes or hide attributes, which can be useful in the context of calculated fields (see Section 4.1.3).

- ▶ You can specify how the client field is handled (static value or dynamically). Client handling will be discussed in detail at the end of this section.
- ▶ You can define hierarchies (see Section 4.1.4).

The layout of the SEMANTICS section is shown in Figure 4.4.

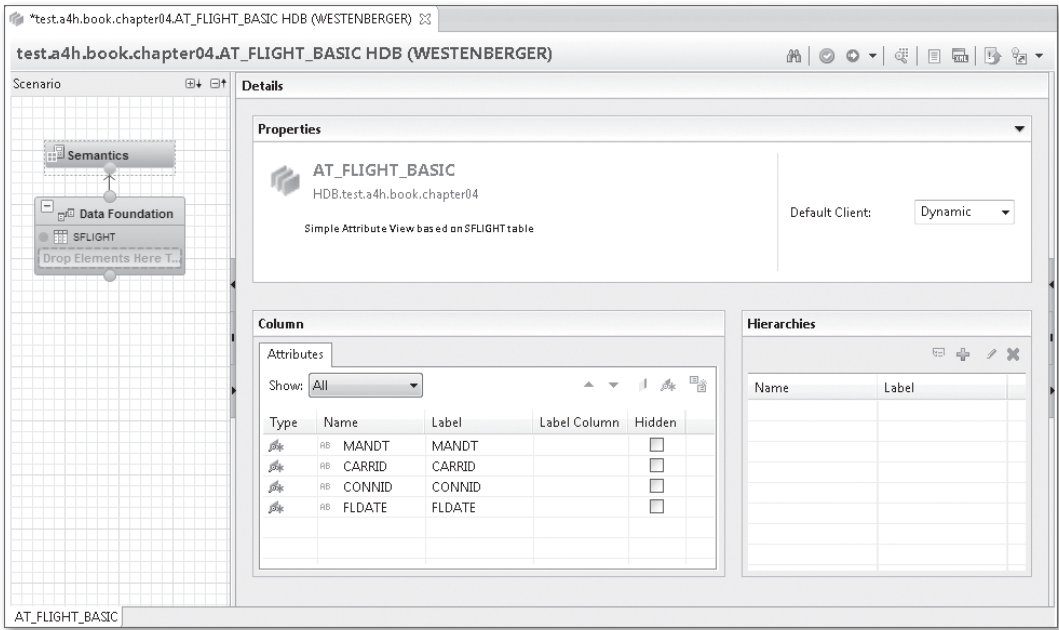


Figure 4.4 Further Semantic Configuration of the Attribute View

The selected columns from the SFLIGHT table are marked as key fields. As described in Section 2.4.3, you now have to save and activate the ATTRIBUTE view to be able to use it.

Activation errors

If the view was not modeled properly, an error will be displayed during activation. Typical errors are caused by missing key fields, invalid joins, or calculated fields that were not defined correctly. Figure 4.5 shows an example of an activation error. The cause of an error may not always be as obvious. Section 4.5.4 provides some troubleshooting tips.

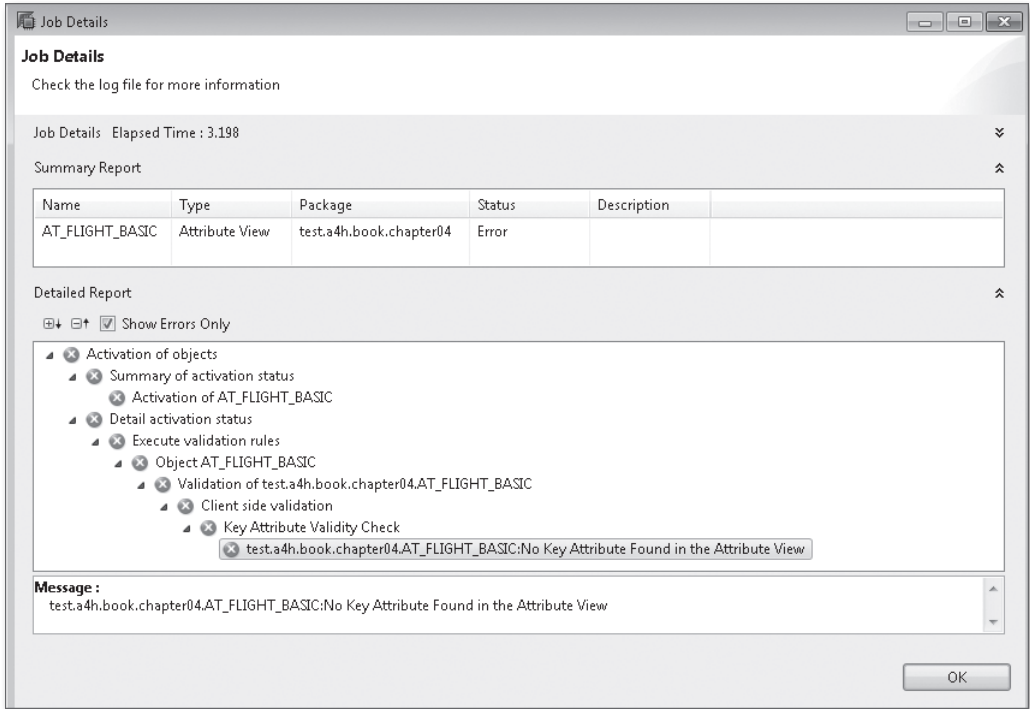


Figure 4.5 Example of an Activation Error

If the tables used are client-dependent, you can specify if the client should be automatically included in the filter condition based on the current context (DYNAMIC DEFAULT CLIENT). Alternatively, it can be defined as CROSS-CLIENT to access the data for all clients. It is also possible to specify a static value for the client. Usage tips can be found in Section 4.5.4.

Client handling

Background Information: Determining the Client

There is a so-called *session context* for every database connection, which stores certain properties of the current connection. In particular, this information comprises the current client, which is set by the DBSL in case of a connection via the SAP NetWeaver AS ABAP. When using the Data Preview or a connection via the SQL console in SAP HANA Studio, the client is determined from the user settings. When configuring these settings, you can specify a default client for a user. If no client is specified, there is no client context; this means that all data is displayed (cross-client) when using the Data Preview. The session context is explained in more detail in Chapter 5.

[«]

View SFLIGHTS as attribute view

Following this brief summary of the available join types, we will now define attribute views. As our first example, we want to define the SFLIGHTS view from the ABAP Data Dictionary, which you have already seen in Section 3.2.3 as an attribute view. Based on our example from Figure 4.3, we can add further tables to the DATA FOUNDATION. You can either manually select those tables or have the system propose tables based on the metadata maintained in the ABAP Data Dictionary. For the latter option, select the table and then choose PROPOSE TABLES from the context menu. The selection dialog opens the screen shown in Figure 4.6.

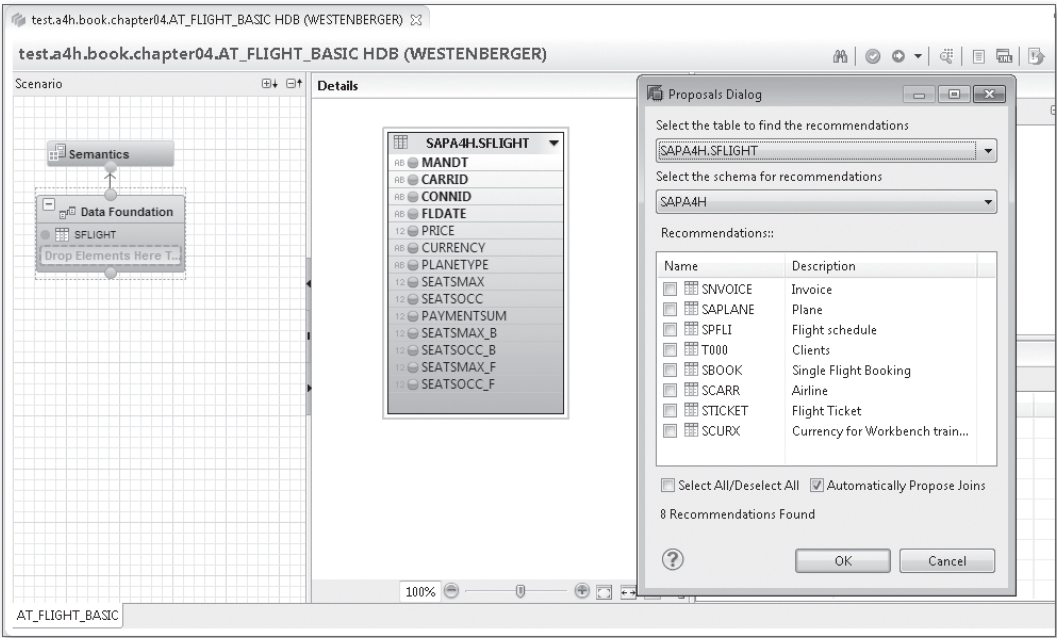


Figure 4.6 Proposed Values for Defining Joins

Selecting tables and defining joins

To reproduce the SFLIGHTS view, we will add the tables SCARR and SPFLI and define the joins as shown in Figure 4.7. If you want to define a new join, simply drag a connecting line between the corresponding attributes of two tables while holding the mouse button down. To define the properties of a join, you first have to select the join and then configure it in

the PROPERTIES section (JOIN TYPE, CARDINALITY). For our example, a referential join and a cardinality of n:1 is used.

In the next step, you add the desired attributes from the tables via the context menu of the output structure of the view. The selected attributes will then be highlighted and displayed in the OUTPUT section in the right-hand pane of the editor.

Adding attributes

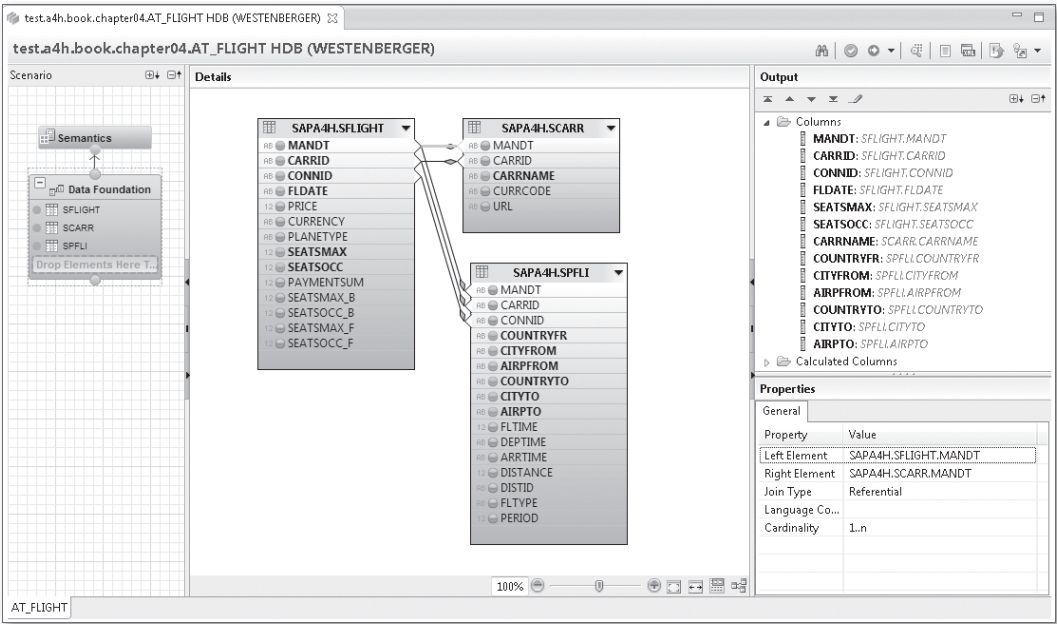


Figure 4.7 Attribute View Analogous to the DDIC View SFLIGHTS

Since we already defined the key fields, and they were not changed by adding tables, we can now activate and test the view. The result shows the name of the airline and information on the departure and destination location for every flight (see Figure 4.8).

Activate/test

MANDT	CARRID	CONNID	FLDATE	SEATSM...	SEATSOCC	CARRNAME	COUNTRYFR	CITYFROM	AIRPFROM
001	LH	0400	20040101	280	257	Lufthansa	DE	FRANKFURT	FRA
001	AA	0017	20040101	220	174	American Airlines	US	NEW YORK	JFK
001	AZ	0555	20040101	189	150	Alitalia	IT	ROME	FCO
001	LH	2402	20040101	130	109	Lufthansa	DE	FRANKFURT	FRA
001	UA	0941	20040101	380	380	United Airlines	DE	FRANKFURT	FRA
001	AZ	0789	20040101	385	298	Alitalia	JP	TOKYO	TYO
001	LH	0402	20040101	380	321	Lufthansa	DE	FRANKFURT	FRA
001	QF	0005	20040101	385	361	Qantas Airways	SG	SINGAPORE	SIN
001	SQ	0015	20040101	220	187	Singapore Airlines	US	SAN FRANCISCO	SFO
001	SQ	0002	20040101	380	377	Singapore Airlines	SG	SINGAPORE	SIN
001	LH	0401	20040101	280	266	Lufthansa	US	NEW YORK	JFK
001	DL	0106	20040101	380	339	Delta Airlines	US	NEW YORK	JFK
001	JL	0407	20040101	380	315	Japan Airlines	JP	TOKYO	NRT
001	JL	0408	20040101	380	380	Japan Airlines	DE	FRANKFURT	FRA
001	AA	0064	20040101	380	331	American Airlines	US	SAN FRANCISCO	SFO
001	DL	1699	20040101	385	363	Delta Airlines	US	NEW YORK	JFK
001	DL	1984	20040101	280	266	Delta Airlines	US	SAN FRANCISCO	SFO
001	LH	2407	20040101	130	122	Lufthansa	DE	BERLIN	TXL
001	UA	3504	20040101	385	385	United Airlines	US	SAN FRANCISCO	SFO
001	AZ	0788	20040101	385	300	Alitalia	IT	ROME	FCO
001	AZ	0790	20040101	280	251	Alitalia	IT	ROME	FCO

Figure 4.8 Result of the Attribute View

Using text joins

To illustrate the usage of the aforementioned text join, we will create another attribute view and read the corresponding texts (table SMEALT) for the in-flight meals (table SMEAL). The required modeling is shown in Figure 4.9. Since filtering is done based on the language, the cardinality for this join is always 1:1.

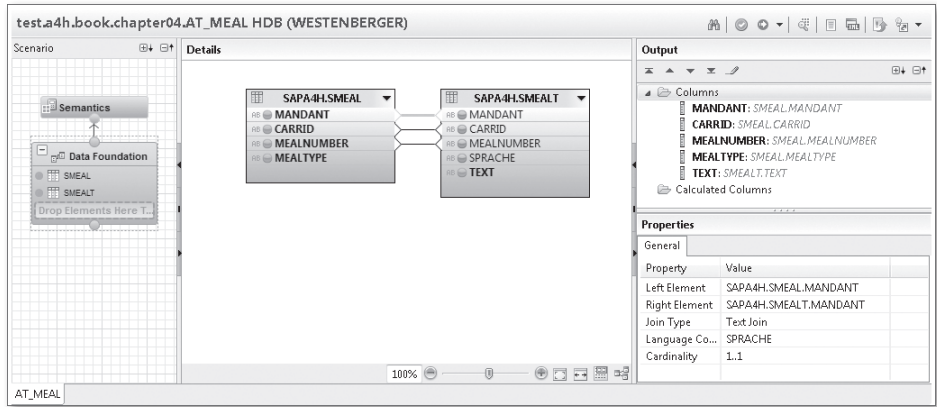


Figure 4.9 Using a Text Join

Defining filter values

As in case of normal SQL views, you can also specify filter values for columns when working with attribute views. To define the filter, you open the filter dialog for an attribute via the context menu item APPLY

FILTER. Attributes with an existing filter are marked with a filter symbol (as shown in Figure 4.10).

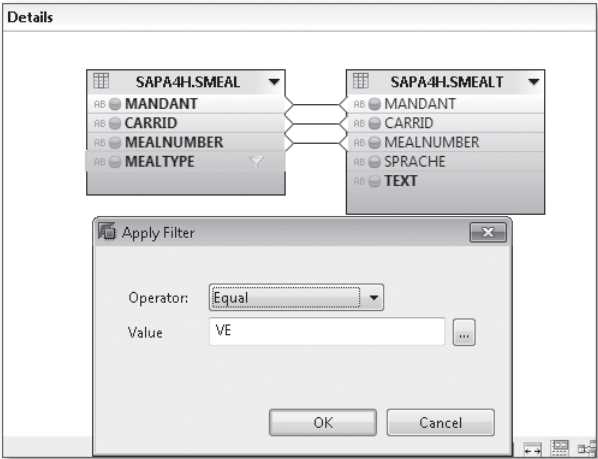


Figure 4.10 Filter for an Attribute

For the example using the meals served on the flight, we define a filter for the attribute MEAL_TYPE with an equals operator and the value “VE” (vegetarian), as shown in Figure 4.11. Alternatively, you can also try other comparison operators. The Data Preview displays all vegetarian meals with the corresponding texts in the correct language.

MANDANT	CARRID	MEALNUMBER	MEALTYPE	TEXT
001	AA	00000001	VE	SEASONAL SALAD
001	AA	00000002	VE	NICE SALAD
001	AA	00000003	VE	PARIS SALAD
001	AA	00000004	VE	HAMBURG SALAD WITH FRESH SHRIMPS
001	AA	00000023	VE	CHOCOLATE ICE CREAM
001	AA	00000024	VE	VANILLA ICE CREAM
001	AA	00000025	VE	VANILLA ICE CREAM WITH HOT CHERRIES
001	AA	00000026	VE	VANILLA ICE CREAM WITH HOT RASPBERRIES
001	AA	00000027	VE	APPLE STRUDEL
001	AA	00000028	VE	RASPBERRY SORBET
001	AA	00000029	VE	STRAWBERRY SORBET
001	AC	00000001	VE	SEASONAL SALAD
001	AC	00000002	VE	NICE SALAD
001	AC	00000003	VE	PARIS SALAD
001	AC	00000004	VE	HAMBURG SALAD WITH FRESH SHRIMPS
001	AC	00000023	VE	CHOCOLATE ICE CREAM

Figure 4.11 Example of a Text Join with an Additional Filter

4.1.3 Calculated Fields

Virtual attributes

Having explained how an attribute view can be used to read data from different tables using different join types, we will now go one step further and dynamically calculate some of the view columns. Compared to classic ABAP Data Dictionary views, these *virtual attributes* (i.e., attributes that do not belong directly to a column of one of the physical tables) are a powerful new opportunity for expressing data processing logic.

As a first example, we will now add a calculated attribute to the attribute view AT_FLIGHTS from Figure 4.7, which will contain the full flight connection (departure location and airport plus destination location and airport) as its value, e.g. NEW YORK (JFK)—SAN FRANCISCO (SFO).

Defining calculated attributes

To do so, we define a calculated attribute in the DATA FOUNDATION via the node CALCULATED COLUMNS of the OUTPUT section and specify a name, a description, and a data type (see Figure 4.12).

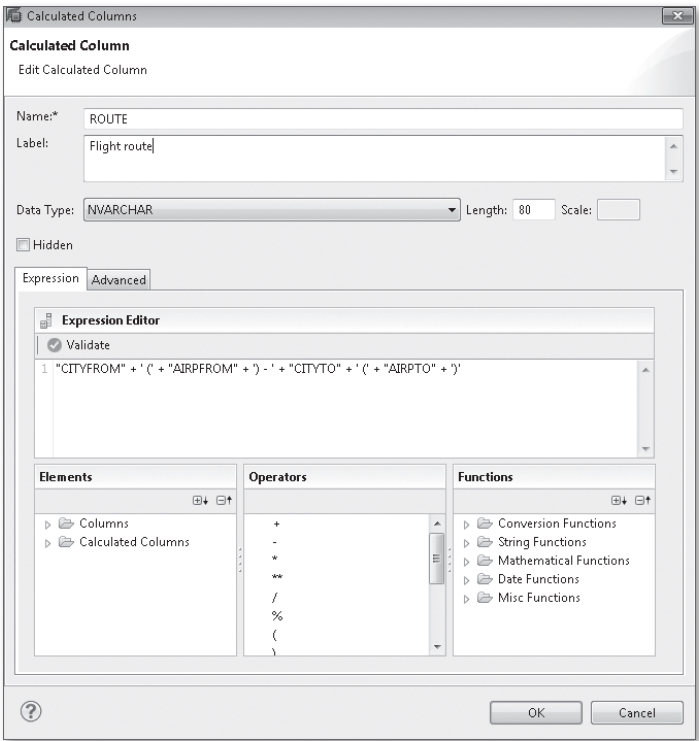


Figure 4.12 Definition of a Calculated Field

Using the EXPRESSION EDITOR, you can specify an expression that will be used to determine the value. This provides a variety of functions (conversions, mathematical operations, string operations, date calculations, and even simple case distinctions). In our example, we will only use a simple concatenation of strings for now (see Listing 4.1):

```
"CITYFROM" + ' ( ' + "AIRPFROM" + ') - ' + "CITYTO" + ' ( ' + "AIRPTO" + ')'
```

Listing 4.1 Example of an Expression for a Calculated Field

Defining expressions for calculations

Attribute References and Constants in Expressions

When defining *expressions* for calculated attributes, you must make sure to use the correct type of quotation marks. For references to attributes of the view (e.g., "CITYFROM" in Listing 4.1), double quotes must be used. It is recommended to use the drag-and-drop function via the formula editor. For text constants, by contrast, simple quotes must be used (as shown in the parentheses in Listing 4.1).

Using the wrong quotation marks usually leads to an activation error.

[!]

After activating the attribute view, the calculated column is displayed in the output (see Figure 4.13). Calculated columns can be queried via SQL just like normal columns, which will be demonstrated in Section 4.1.6.

Output of the calculated field

Analysis Distinct values Raw Data					
Filter pattern 200 rows retrieved - 618 ms					
RB	CARRID	RB	CONNID	FLDATE	RB CARRNAME
LH	0400		20040101	Lufthansa	FRANKFURT (FRA) - NEW YORK (JFK)
AA	0017		20040101	American Airlines	NEW YORK (JFK) - SAN FRANCISCO (SFO)
AZ	0555		20040101	Alitalia	ROME (FCO) - FRANKFURT (FRA)
LH	2402		20040101	Lufthansa	FRANKFURT (FRA) - BERLIN (SXF)
UA	0941		20040101	United Airlines	FRANKFURT (FRA) - SAN FRANCISCO (SFO)
AZ	0789		20040101	Alitalia	TOKYO (TYO) - ROME (FCO)
LH	0402		20040101	Lufthansa	FRANKFURT (FRA) - NEW YORK (JFK)
QF	0005		20040101	Qantas Airways	SINGAPORE (SIN) - FRANKFURT (FRA)
SQ	0015		20040101	Singapore Airlines	SAN FRANCISCO (SFO) - SINGAPORE (SIN)
SQ	0002		20040101	Singapore Airlines	SINGAPORE (SIN) - SAN FRANCISCO (SFO)
LH	0401		20040101	Lufthansa	NEW YORK (JFK) - FRANKFURT (FRA)
DL	0106		20040101	Delta Airlines	NEW YORK (JFK) - FRANKFURT (FRA)
JL	0407		20040101	Japan Airlines	TOKYO (NRT) - FRANKFURT (FRA)

Figure 4.13 Output of the Calculated Field

Calculated fields are also supported for the other view types (see Section 4.2), where these fields are used especially for the calculations and conversions of currencies and units that we already mentioned.

4.1.4 Hierarchies

A lot of data has hierarchical relationships. The place of residence or principal office of customers is structured geographically by country, region, and city; the hierarchical structure of a creation date comprises the year, quarter, and month; a product catalog can consist of several categories, etc.

Data analysis

Hierarchies play an important role in data analyses. You can start with an aggregated view of the data and then navigate within the hierarchical structures. This is referred to as a *drilldown* (or *drillup* when data is aggregated). Every OLAP infrastructure (like SAP NetWeaver BW) provides built-in support for hierarchies.

Hierarchies in SAP HANA

For attribute views, hierarchies are defined in the SEMANTICS section. SAP HANA currently supports two types of hierarchies:

► Parent-child relationships

For this type, two attributes with a parent-child relationship must be defined. An example would be storing a directory structure in a table. In this context, it must be noted that this is a *full* and *consistent* self-referential relation. Each parent node must exist and (except for a special root node) must be the child node of another node. This rather limits the use of this hierarchy type, especially for ABAP tables. An example would be the ABAP hierarchy of packages, where the corresponding database table (TDEVIC) comprises columns for the package name and the name of the superpackage. These columns form a parent-child relationship.

► Level hierarchy

With this hierarchy type, you define hierarchy levels based on normal or calculated attributes. If a table for example comprises columns for the country and the city, these attributes define a hierarchy of several levels (the countries at the upper level and the corresponding cities at the lower levels). However, these attributes do not have a parent-child

relationship, since this would require the city values to also appear as countries (this is not a self-referential relation as described previously).

Existing hierarchies are displayed in the SEMANTICS section, where you can also create new hierarchies. Figure 4.14 shows a level hierarchy based on the attributes of the departure location (country, city, airport) from table SPFLI. Hierarchies can also be defined for *calculation views* (see Section 4.3).

Creating hierarchies

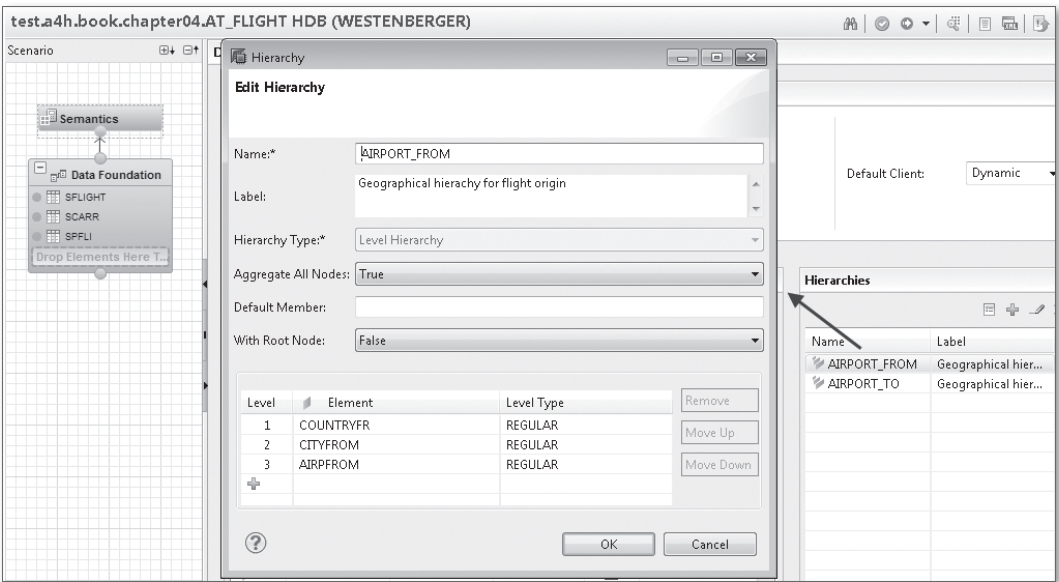


Figure 4.14 Hierarchy of an Attribute View

There are various options for using the modeled hierarchies. This information is evaluated in particular by the supported *business intelligence* clients. One particular variant (access via Microsoft Excel) will be shown in Section 4.4.

SAP HANA thus provides basic support for simple hierarchies, but compared to the comprehensive hierarchy modeling that's available in SAP NetWeaver BW (as an example), the options are rather limited. In many real-life scenarios, hierarchies are much more complex, and there are special cases like external or incomplete hierarchies. This topic is described in detail in the book *Data Modeling in SAP NetWeaver BW* by Frank K. Wolf and Stefan Yamada (SAP PRESS 2011).

Limitations of hierarchy support

4.1.5 Attribute Views for Time Values

Most business data have a time reference (e.g., a creation date or a validity period). These references are usually implemented as date fields or time stamps in the data model. The flight data model, for example, comprises the flight date in the `SFLIGHT` table and the booking time in the `SBOOK` table. For many analyses, this point in time must be mapped to a certain time interval. In the simplest case, this can be the corresponding year, month, quarter, or calendar week. However, there are also more complicated or configurable time intervals like the *fiscal year*, which is the calendar to be used for certain scenarios.



Customizing of the Fiscal Year

Fiscal years and periods are configured via the ABAP Customizing. Using the ABAP Customizing, you can configure comprehensive settings or variants and also define special cases (e.g., a short fiscal year when a company is founded). These settings are configured via the entry `MAINTAIN FISCAL YEAR VARIANT` of Transaction `SPRO`.

The SAP standard provides several function modules to convert a normal date (e.g., of type `DATS`) into the corresponding fiscal year or period.

From a technical perspective, the corresponding Customizing is stored particularly in the tables `T009` and `T009B`. These tables were previously pool/cluster tables and therefore not available directly in the database. Such tables are converted into normal database tables when performing a migration to SAP HANA (see Section 3.2.1) so that such data can also be accessed natively in the database.

Mapping of the fiscal year

In the past, when determining the corresponding fiscal year for a date in ABAP, the data first needed to be transferred to the application server in order to perform the conversion. There was therefore no way to simply create an aggregated set of records by fiscal year via Open SQL. The determination of the fiscal year had always to be done in ABAP. Using attribute views in SAP HANA, you can define these mappings to intervals of both the normal calendar (Gregorian calendar) and the fiscal calendar.

Generating calendar data

To do so, we first generate time data in special technical tables in SAP HANA. You can select the entry `GENERATE TIME DATA` on the initial screen of the `MODELER` perspective for this purpose. Subsequently, you specify the details for calendar type and time period. In our example, we specify the configuration shown in Figure 4.15 to create the fiscal calendar from 2000 to 2020.

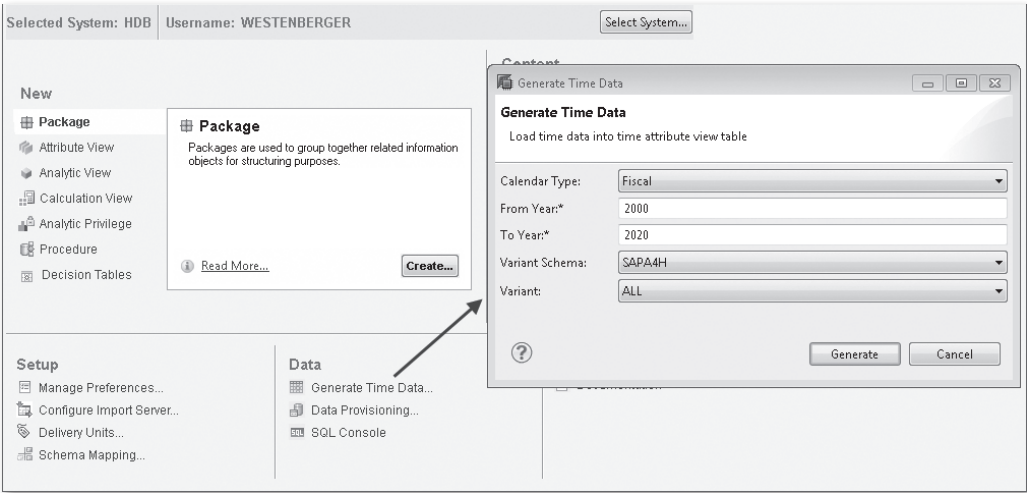


Figure 4.15 Generating the Data for the Fiscal Calendar

You can now use the underlying table `M_FISCAL_CALENDAR` (schema `_SYS_BI`) in attribute views. In the example shown in Figure 4.16, we use the attribute view to determine the fiscal year and period for every flight in the `SFLIGHT` table. Since we want to use only a fixed variant from the ABAP Customizing, we define a static filter for the field `CALENDAR_VARIANT`.

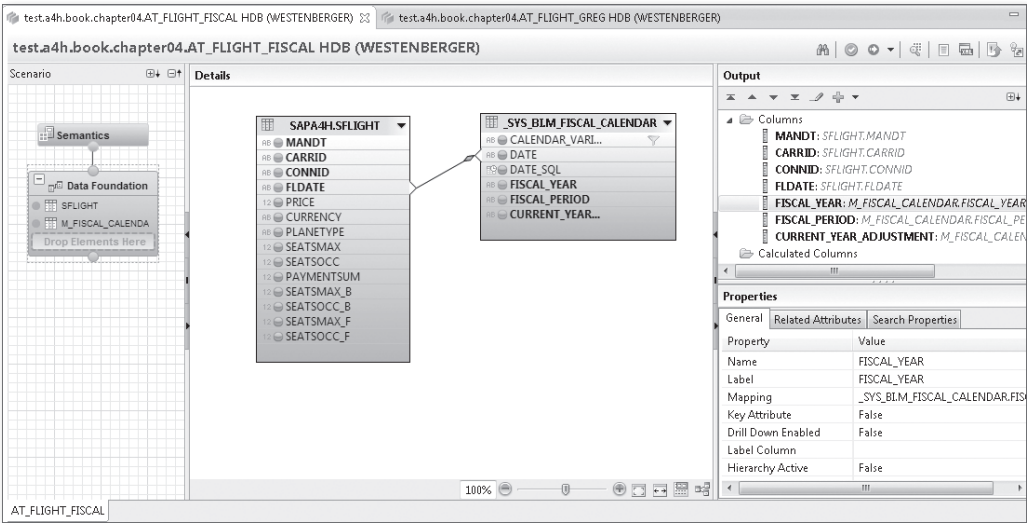


Figure 4.16 Determining the Fiscal Periods for Flight Data

Determining the quarter or calendar week

Another sample scenario would be to determine the quarter or the calendar week for a given date using an attribute view. For this scenario, the data from the Gregorian calendar is needed; this is stored in SAP HANA in the technical table `M_TIME_DIMENSION`, which is part of the `_SYS_BI` schema as well. This means that you will have to generate data first—as in case of the fiscal calendar. The use of table `M_TIME_DIMENSION` can be seen in Figure 4.17.

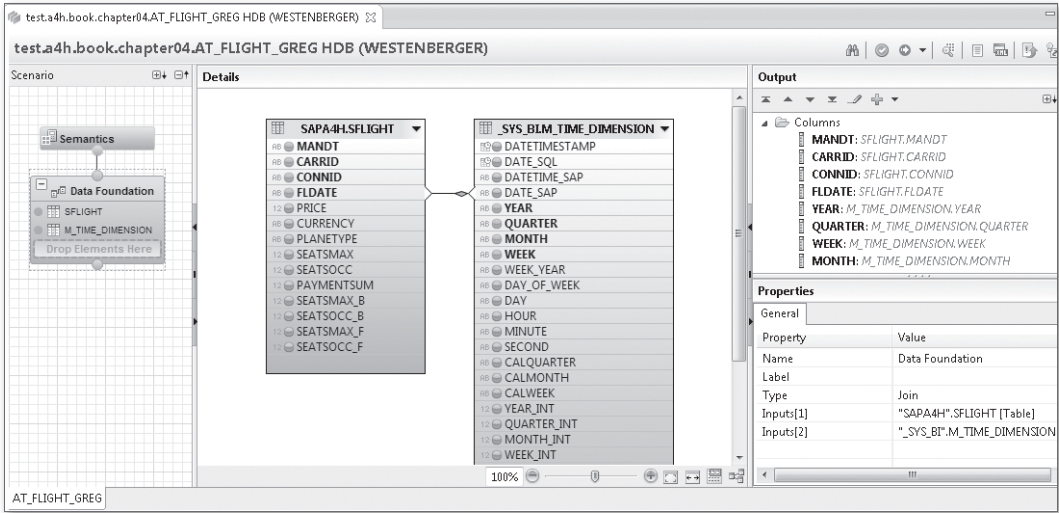


Figure 4.17 Determining the Quarter and Calendar Week

Attribute view of type “Time”

You can also define an attribute view containing only time data. To do so, you select the type `TIME` and specify the desired details for the calendar when creating an attribute view. Figure 4.18 shows how the attribute view `AT_TIME_GREG` is created for a day-based Gregorian calendar.

Since the view contains the date as a key field, joins can be created for a date column in the business data. This means that you can use these views as time dimensions in an *analytic view* if the date is part of the fact table. This will be described in detail in Section 4.2.

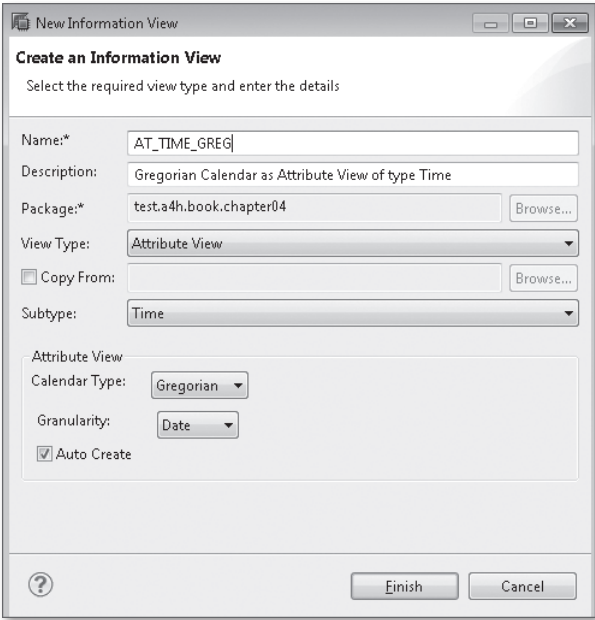


Figure 4.18 Attribute View for a Gregorian Calendar

4.1.6 Runtime Artifacts and SQL Access for Attribute Views

As described in Section 2.4.3, column views are created in the schema `_SYS_BIC` when activating views from the SAP HANA Repository that can be accessed via normal SQL. These column views also form the basis for ABAP access, as shown in Section 4.5. The exact *runtime artifacts* depend on the view type and the concrete modeling. Usually, there is a leading object that serves as the primary interface for data access, and further additional technical artifacts for specific aspects.

This section describes the specifics of attribute views. Every attribute view has a corresponding column view. In addition to this view, another column view is created for every hierarchy. For our attribute view `AT_FLIGHT`, the column views listed in Figure 4.19 exist in the database catalog in the `_SYS_BIC` schema.

Addressing via SQL

Column views

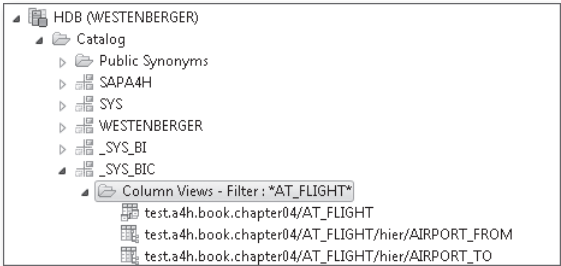


Figure 4.19 Column Views Generated for the Attribute View AT_FLIGHT

Please note that the names of the runtime artifacts always contain the package names. This is necessary because you can create objects with the same name in different packages.

Public synonym In addition, there is a *public synonym* that can also be used to access the views:

```
"test.a4h.book.chapter04::AT_FLIGHT"
```

Attribute views can be accessed using regular SQL. However, please note that attribute views are not optimized for calculations like column aggregations, but rather for efficient join calculations. In other words, not every SQL statement should be used for every view type in SAP HANA. Recommendations can be found in Section 4.5.4.

Contents

Foreword	15
Preface	17
Introduction	19

PART I Basic Principles

1 Overview of SAP HANA	29
1.1 Software Components of SAP HANA	29
1.1.1 SAP HANA Database	31
1.1.2 SAP HANA Studio	31
1.1.3 SAP HANA Client	33
1.1.4 SAP HANA Function Libraries	34
1.1.5 Software for Data Replication	34
1.1.6 Software for Direct Data Access	35
1.1.7 Lifecycle Management Components	36
1.2 Basic Principles of In-Memory Technology	37
1.2.1 Hardware Innovations	37
1.2.2 Software Innovations	41
1.3 Architecture of the In-Memory Database	51
1.4 Application Cases for SAP HANA	53
1.5 How SAP HANA Affects Application Development	56
1.5.1 New Technical Options	56
1.5.2 Code Pushdown	57
1.5.3 Database as Whitebox	59
1.5.4 Required Qualifications for Developers	61

2 Introducing the Development Environment	63
2.1 Overview of Eclipse	63
2.2 SAP's Eclipse Strategy	66
2.2.1 Unbundling of Eclipse and SAP Software	67
2.2.2 Central Update Site	67
2.3 Installing the Development Environment	69
2.3.1 Installing SAP HANA Studio	69

2.3.2	Installing the ABAP Development Tools for SAP NetWeaver	70
2.4	Getting Started in the Development System	72
2.4.1	Basic Principles of Eclipse	72
2.4.2	ABAP Development Tools for SAP NetWeaver ...	75
2.4.3	SAP HANA Studio	85

**3 Database Programming Using SAP NetWeaver
AS ABAP 103**

3.1	SAP NetWeaver AS ABAP Architecture	105
3.1.1	Database Interface	107
3.1.2	Role of the Database for the ABAP Application Server	109
3.1.3	Data Types	110
3.2	ABAP Database Access	116
3.2.1	ABAP Data Dictionary	117
3.2.2	Open SQL	122
3.2.3	Database Views in the ABAP Data Dictionary ...	132
3.2.4	Database Access via Native SQL	133
3.2.5	Secondary Database Connections	139
3.3	Analyzing Database Accesses Using the SQL Trace	143
3.3.1	Statement Transformations	143
3.3.2	Secondary Connections	150
3.3.3	Native SQL	151
3.3.4	Buffer	152

**PART II Introduction to ABAP Programming with
SAP HANA**

4 View Modeling in SAP HANA Studio 157

4.1	Attribute Views	160
4.1.1	Basic Principles	161
4.1.2	Creating Attribute Views	164
4.1.3	Calculated Fields	172
4.1.4	Hierarchies	174
4.1.5	Attribute Views for Time Values	176

4.1.6	Runtime Artifacts and SQL Access for Attribute Views	179
4.2	Analytic Views	180
4.2.1	Basic Principles	181
4.2.2	Creating Analytic Views	183
4.2.3	Calculated Key Figures	186
4.2.4	Currency Conversion and Unit Conversion	187
4.2.5	Runtime Artifacts and SQL Access for Analytic Views	191
4.3	Calculation Views	192
4.3.1	Basic Principles	193
4.3.2	Graphical Modeling of Calculation Views	195
4.3.3	Implementing Calculation Views via SQLScript	197
4.3.4	Runtime Artifacts and SQL Access for Calculation Views	202
4.4	Accessing Column Views via Microsoft Excel	203
4.5	Using SAP HANA Views in ABAP	205
4.5.1	Access via Native SQL	205
4.5.2	External Views in the ABAP Data Dictionary	207
4.5.3	Options for Accessing External Views	210
4.5.4	Recommendations	211

5 Programming Options in SAP HANA 215

5.1	Overview of SQLScript	215
5.1.1	Qualities of SQLScript	216
5.1.2	Processing SQLScript	222
5.2	Implementing Database Procedures	223
5.2.1	Basic Principles of Database Procedures	223
5.2.2	Creating Database Procedures	225
5.2.3	Using Variables	237
5.2.4	Calculation Engine Plan Operator	239
5.2.5	Imperative Enhancements	250
5.2.6	Accessing System Fields	252
5.2.7	Error Handling	254
5.3	Using Procedures in ABAP	255
5.3.1	Access Using Native SQL	256

- 5.3.2 Defining Database Procedure Proxies 263
- 5.3.3 Calling Database Procedure Proxies 265
- 5.3.4 Adjusting Database Procedure Proxies 267

6 Application Transport 269

- 6.1 Basic Principles of the Transport System 271
 - 6.1.1 Transport in SAP NetWeaver AS ABAP 271
 - 6.1.2 Transport in SAP HANA 276
- 6.2 Combined ABAP/SAP HANA Transport 285
 - 6.2.1 HANA Transport Container 286
 - 6.2.2 Enhanced Transport System 292

7 Runtime and Error Analysis with SAP HANA 293

- 7.1 Overview of the Tools Available 294
- 7.2 Error Analysis 296
 - 7.2.1 Unit Tests 296
 - 7.2.2 Dump Analysis 299
 - 7.2.3 Tracing in SQLScript 301
 - 7.2.4 Debugging SQLScript 302
- 7.3 ABAP Code Analysis 305
 - 7.3.1 Checks and Check Variants 305
 - 7.3.2 Checks in the Development Infrastructure 309
 - 7.3.3 Global Check Runs in the System 311
- 7.4 Runtime Statistics and Traces 313
 - 7.4.1 Runtime Statistics 314
 - 7.4.2 ABAP Trace and ABAP Profiler 318
 - 7.4.3 SQL Trace 326
 - 7.4.4 Single Transaction Analysis 330
 - 7.4.5 Explain Plan 331
 - 7.4.6 SAP HANA Plan Visualizer 333
- 7.5 System-Wide SQL Analyses 337
 - 7.5.1 DBA Cockpit 338
 - 7.5.2 SQL Monitor 342
- 7.6 SQL Performance Optimization 346

8 Sample Scenario: Optimizing an Existing Application 351

- 8.1 Optimization Procedure 351
 - 8.1.1 Migrating to SAP HANA 352
 - 8.1.2 System Optimization 353
 - 8.1.3 Application Optimization 355
- 8.2 Scenario and Requirements 357
 - 8.2.1 Initial Situation 358
 - 8.2.2 Technical Implementation 359
 - 8.2.3 Current Problems 362
- 8.3 Meeting the Requirements 362
 - 8.3.1 Narrowing Down the Problem Using the Runtime Statistics 363
 - 8.3.2 Detailed Analysis of the ABAP Program Using Transaction SAT 364
 - 8.3.3 Detailed Analysis of Database Accesses 366
 - 8.3.4 Analysis Result 368
 - 8.3.5 Optimization Using Open SQL 369
 - 8.3.6 Analysis of the First Optimization 371
 - 8.3.7 Analysis Result 372
 - 8.3.8 Optimization Using an Analytic View 373
 - 8.3.9 Analysis of the Second Optimization 374
 - 8.3.10 Analysis Result 378

PART III Advanced Techniques for ABAP Programming for SAP HANA

9 Text Search and Analysis of Unstructured Data 383

- 9.1 Basic Principles of the Text Search in SAP HANA 385
 - 9.1.1 Technical Architecture 386
 - 9.1.2 Error-Tolerant Search 387
 - 9.1.3 SAP Components and Products for Search 389
- 9.2 Types of Text Data and Full Text Indexes in SAP HANA 390
- 9.3 Using the Text Search via SQL 395
 - 9.3.1 Fuzzy Search 397

9.3.2	Synonyms and Noise Words	401
9.3.3	Searching Across Date Fields and Address Data	404
9.4	Using the Text Search in ABAP	407
9.4.1	Calling the Text Search from ABAP via SQL	408
9.4.2	Freely Defined Input Helps	409
9.5	Text Analysis	416
9.6	Resource Consumption and Runtime Aspects of the Text Search	418
10 Integrating Analytical Functionality		423
10.1	Introduction	423
10.1.1	What is Analytical Functionality?	424
10.1.2	Digression: SAP NetWeaver Business Warehouse	427
10.2	Overview of Possible Architectures	429
10.2.1	Direct Access to Analytical Functionality in SAP HANA	430
10.2.2	Access via the SAP NetWeaver AS ABAP	434
10.3	Selected Technologies and Tools	439
10.3.1	InfoProviders when Using SAP HANA	440
10.3.2	SAP BusinessObjects Portfolio	447
10.3.3	Easy Query Interface	451
10.4	User Interface Building Blocks	453
11 Decision Tables in SAP HANA		455
11.1	Basic Principles of Decision Tables	456
11.2	Creating Decision Tables in SAP HANA Studio	459
11.3	Decision Tables Based on SAP HANA Views	465
11.4	Runtime Artifacts and SQL Access for Decision Tables ...	468
11.5	Access to Decision Tables from ABAP	468
12 Function Libraries in SAP HANA		473
12.1	Basics of the Application Function Library	476
12.1.1	Technical Basics	476
12.1.2	Business Function Library	477

12.1.3	Predictive Analysis Library	478
12.2	Use of Application Function Library Functions in SQLScript	483
12.3	Integration of Function Libraries in ABAP	487
13 Sample Scenario: Development of a New Application		491
13.1	Scenario and Requirements	491
13.2	Application Design	492
13.2.1	Management of Discounts by the Travel Company Owner	493
13.2.2	Additional Evaluations via a Side Panel Application	494
13.2.3	Mobile Application for the Air Passenger	496
13.3	Implementation of the Application	497
13.3.1	SAP HANA Views and Procedures	498
13.3.2	Core of the ABAP Application	499
13.3.3	User Interfaces	501
13.4	Using the Applications	506
14 Practical Tips		509
14.1	General Recommendations	510
14.1.1	Recommendations for Column and Row Store	510
14.1.2	SAP HANA-Specific Implementations	511
14.1.3	Checklist for Database-Specific Implementations	513
14.1.4	Recommendations for Migration	515
14.1.5	Development in Landscapes	517
14.1.6	Modifying Data in SQLScript or Native SQL	518
14.2	Conventions	520
14.2.1	Naming Conventions	521
14.2.2	Encapsulating Packages	522
14.3	Quality Aspects	523
14.3.1	Testing Views and Procedures	523
14.3.2	Robust Programming	524
14.3.3	Security Aspects	525

14.4	Performance Recommendations for Open SQL	526
14.4.1	Rule 1: Keeping Result Sets Small	527
14.4.2	Rule 2: Keeping Transferred Datasets Small	530
14.4.3	Rule 3: Reducing Number of Queries	537
14.4.4	Rule 4: Minimizing Search Effort	543
14.4.5	Rule 5: Reducing Load on Database	546
14.4.6	Summary of Rules	551
14.5	Performance Recommendations for Native Implementations in SAP HANA	551
14.5.1	Recommendations for Native SQL	552
14.5.2	Recommendations for SAP HANA Views	553
14.5.3	Recommendations for SQLScript	556
14.6	Summary of Recommendations	558
Appendices		561
A	Flight Data Model	563
A.1	Basic Principles of the Flight Data Model	563
A.2	Database Tables for the Flight Data Model	564
A.2.1	Customizing	564
A.2.2	Master Data	565
A.2.3	Transaction Data	566
A.2.4	Designing the SFLIGHT Data Model	568
A.3	Data Generation	569
B	What's New in ABAP in SAP NetWeaver 7.4	573
B.1	Inline Declarations	573
B.2	Constructor Expressions	575
B.3	Internal Tables	577
C	Read and Write Access in the Column Store	579
C.1	Basic Principles	579
C.2	Read Access without an Index	580
C.3	Write Access without an Index	582
C.4	Read Accesses with an Index	585
D	SAP Business Application Accelerator Powered by SAP HANA	589
E	Installing the Sample Programs	593
F	The Authors	595
	Index	597

Index

A

- ABAP, 21
 - code analysis*, 295, 352
 - profiler*, 295, 320
 - project*, 76
 - proxy*, 522
 - reports*, 21
 - resource URL*, 83
 - runtime environment*, 105
 - schema*, 109
 - table buffer*, 307
 - trace*, 372
 - type system*, 112
- ABAP 7.4, 573
- ABAP application
 - transport*, 518
- ABAP-based frameworks, 23
- ABAP Connectivity and Integration
 - Development Tools, 71
- ABAP Core Development Tools, 71
- ABAP Database Connectivity, 255
- ABAP Data Dictionary, 77, 104, 111, 117, 206, 392
 - input help*, 409
 - type system*, 113
- ABAP Development Tools for SAP
 - NetWeaver, 32, 207
 - ABAP resource URL*, 78, 79
 - authorizations*, 76
 - code analysis*, 310
 - components*, 71
 - create program*, 80
 - debugger*, 83
 - downward compatibility*, 72
 - execute program*, 83
 - favorite packages*, 77
 - perspectives*, 75
 - project*, 76
 - Project Explorer*, 77
 - SAP GUI integration*, 77
 - system library*, 77
 - templates*, 81
 - user settings*, 78
- ABAP List Viewer, 453
- ABAP Memory, 549
- ABAP program
 - analysis*, 364
 - runtime*, 364
- ABAP Test Cockpit, 82, 275, 295, 308, 311
 - trace*, 318
- ABAP Unit, 296
- Access function, 241
- Access time, 38
 - CPU cache*, 39
 - Flash memory*, 38
 - hard disks*, 38
 - main memory*, 39
- ACID principle, 31
- ADBC, 133, 255, 408, 516, 553
 - prepared statement*, 136, 552
- Add-on Assembly Kit, 273
- Administration Console, 85
- After-import method, 276, 290
- Aggregate function, 125
- Alternative implementation, 511
- ALV, 469, 471
 - integrated data access*, 494
- Analytical functionality, 424
- Analytical index, 440
- Analytical search, 377
- Analytic Engine, 428, 436
 - formulas*, 439
 - hierarchy processing*, 438
 - report-report interface*, 439
- Analytic privilege, 94
- Analytics list component, 453
- Analytic view, 94, 180, 554, 439
 - call*, 373
 - create*, 183
- Appliance, 29
- Application
 - optimization*, 351, 355
- Application Function Library, 34, 473
 - installation*, 476
- Application layer, 58

Application logic, 58
Application scenario
 accelerator scenarios, 54
 integrated, 53, 55
 side-by-side scenarios, 53
Array interface, 553
Attribute, 161
 calculated, 172
 vector, 45, 579
 virtual, 172
Attribute view, 94, 161, 499, 554
 fuzzy search, 398
Authorization
 analytical, 86, 526
 check, 525
 package authorization, 86
 SAP HANA Studio, 86
 SQL authorization, 86
 system authorization, 86

B

BAdI, 513
BEx Query Designer, 446
BFL, 34, 474
Blocking factor, 148
Breakpoint
 dynamic, 83
 external, 83
 static, 83
BRFplus, 456
BSP Framework, 495
B* tree index, 583
Buffer
 access, 369
 cross-user, 546
 trace, 152
Business Function Library, 474
Business functions, 31
Business intelligence, 21
Business logic, 221
Business process, 455
Business rule management system, 455
Business rule mining, 474

Business Server Pages (BSP), 495
BW query, 428, 446, 451

C

Calculation engine, 223, 241
Calculation logic, 58
Calculation view, 94, 215, 247, 440,
 468, 499, 554
 implemented, 197
 SQLScript, 192
Calendar
 fiscal, 176
 Gregorian, 178
Call hierarchy, 325, 367
Cash flow, 477
CE Plan Operator, 239, 373, 555
 CE_AGGREGATION, 240, 244
 CE_CALC, 244
 CE_CALC_VIEW, 242
 CE_COLUMN_TABLE, 240, 241
 CE_CONVERSION, 246
 CE_JOIN, 242
 CE_JOIN_VIEW, 241
 CE_OLAP_VIEW, 242
 CE_PROJECTION, 240, 243
 CE_UNION_ALL, 244
 CE_VERTICAL_UNION, 245, 483
 data source access operator, 241
 inner join, 242
 other, 245
 relational, 241, 242
 special operator, 241
 TRACE, 301
Change recording, 269, 273, 278
Change request, 273, 288, 289
Change transport, 269
Chart component, 454
CHIP, 494
Class
 CL_PREPARED_STATEMENT, 553
 CL_SQL_STATEMENT, 256, 553
Client handling, 107, 212, 252, 518
 attribute view, 167
 automatic, 232

Client/server, 21
Cluster encoding, 47
Cluster table, 121, 516
Code completion, 80, 93, 210
Code Inspector, 295, 305, 352, 366
 check variant, 305
Code pattern, 62
Code pushdown, 57, 549
Code-to-data paradigm, 57, 59, 362
Collective search help, 410
Column-based data storage, 42
Column-oriented data storage, 91
Column store, 42, 91, 120, 215, 579
 composite index, 587
 data types, 390
 inverted index, 586
 merge, 584
 read access, 580
 recommendation, 510
 write access, 582
Column view, 90, 179, 183, 205, 210,
 269
Commit
 implicit, 130
Composite index, 587
Compression
 techniques, 43, 46
Constructor expression, 206, 575
CONTAINS key word, 395
Content, 32
Control file, 276
Control structure, 250
Conversion exit, 514
Counter, 193
CPU cache, 38
CPU core, 37
CPU time, 377
CTS, 275
 Deploy Web Service, 281
 plug-in, 280
CTS+, 270, 292,
Currency conversion
 Customizing, 188
 parameterization, 189
Cursor, 128, 251, 557

D

DATA(), 574
Data analysis, 424
Data analyst, 473
Database
 catalog, 89, 158, 205
 index, 543
 interface, 107, 516, 519
 layer, 58
 object, 90
 optimizer, 50, 219, 331
 relational, 31, 41, 158
 type system, 113
Database connection
 secondary, 139, 150
 standard, 151
Database procedure, 52, 90, 95, 215,
 216, 499, 522
 compilation, 222
 control structure, 250
 create, 226
 execution, 222
 input parameter, 229
 output parameter, 223, 229
 processing logic, 230
 proxy, 500
 test, 523
 types, 224
Database procedure proxy, 255, 263,
 487, 518, 522
 adjusting, 267
 calling, 265
 creating, 263
 synchronization, 267
Database programming
 tools, 143
Database schema, 89, 96
 table, 90
 technical, 89
 trigger, 90
 view, 118, 132
Data class, 120
Data Control Language (DCL), 116
Data declaration, 573
Data Definition Language (DDL), 116

Data file, 275
Data inconsistency, 520, 523
Data layout, 41
Data Manipulation Language (DML), 116
Data mart, 427
Data model
 virtual, 431
Data Modeler, 564
Data preview, 92, 100
Data reference, 576
Data replication, 34
 Direct Extractor Connection (DXC), 35
 SAP Landscape Transformation Replication Server, 35
DataSource, 428, 445
Data-to-code paradigm, 57
Data type, 110, 119, 263
 conversion, 464
 SHORTTEXT, 390
 TEXT, 390
Data warehouse, 426
DBA Cockpit, 140, 338
DBI, 107
DBSL, 33, 108
DDL, 224
 statement, 50
Debug trace, 302
Decision rule, 455
Decision table, 95, 455, 457, 459, 494
 actions, 457
 conditions, 457
 create, 459
 transport, 471
Declarative language element, 557
Declarative programming, 220
Decoupling, 512, 523
Default schema, 228, 231
Delivery unit, 94, 278, 522
Delta compression, 46
Delta load, 34
Delta merge, 421
Delta store, 48, 421, 582
Design time, 260
 object, 98

Development environment
 ABAP Development Tools, 70, 75
 installation, 69
 SAP HANA Studio, 69
Development landscape
 mixed, 518
Development object, 80, 89, 94
 ABAP, 521
 activate, 98
 naming convention, 521
 SAP HANA, 521
 store, 96
 test, 100
 validate, 96
Development organization, 270, 271, 276
 delivery unit, 278
 package, 271
 package hierarchy, 271, 277
 package interface, 273
 software component, 272
 use access, 273
Diagnostics Agent, 36
Dictionary encoding, 43
Dictionary vector, 44, 579
Dimension, 181
Direct Extractor Connection (DXC), 35
Discount scheme, 492
Document Analysis Toolkit, 386
Domain, 119
DRAM, 39
Drilldown, 174
Dump, 299, 525

E

Easy Query, 437, 451
Eclipse, 31
 ABAP development environment, 31
 composite release, 65
 editor, 74
 extension point, 64
 Foundation, 32, 63, 65
 framework, 63
 menu bar, 75

Eclipse (Cont.)
 perspective, 73
 platform, 63, 65
 plug-in, 64
 project, 65
 Release Train, 65
 repository, 67
 SAP, 66
 SAP Release Train for Eclipse, 67
 SDK, 64
 toolbar, 75
 update site, 67
 view, 74
 window, 73
 Workbench, 72
 workspace, 75
Elementary search help, 410
Embedded reporting, 437
Embedded Search, 389
Encapsulation, 522
Encoding
 dictionary encoding, 43
 indirect, 47
 prefix encoding, 46
 run-length encoding, 47
 sparse encoding, 47
Engine, 52, 555
Enqueue server, 105
Enqueue Service, 519
Enqueue work process, 110
Enterprise Data Warehousing, 427
Enterprise Search, 389
Entity-relationship model, 157
Equi-join, 164
Error analysis, 293, 294, 296
Error handling, 524
ETL, 22
Exact search, 396
Exception handler, 254
Existence check, 307
Expensive SQL statement trace, 295
Explain plan, 295, 331
 call, 332
 output, 332
Exporting, 275

Export release preprocessing, 290
Extension index, 392

F

Factory class, 501
Factory pattern, 512
Fact table, 178, 181
Feeder class, 502
Field
 calculated, 186
 list, 533
 symbol, 574
Filter, 162
 value, 170
Fiscal year, 176
Flight data model, 105, 25
 Customizing, 564
Floorplan Manager, 453, 502
 feeder class, 502
FOR ALL ENTRIES, 124, 148, 540
 driver table, 149, 307
Foreign key relationship, 118, 160
Forward navigation, 82
Full outer join, 243
Full text index, 391, 417, 499
 displaying, 420,
Function
 user-defined, 90
 library, 89, 222
Function module
 call, 368, 369
Fuzzy score, 401
Fuzzy search, 384, 385, 494, 499
 index, 419, 420, 422
 parameters, 401

G

GET_AGENCIES_FOR_CONNECTIONS, 217
Golden rules for database programming, 527
GUID, 569

H

HANA transport container, 270, 286, 518
Hardware
 certified, 37
 innovations, 37
 trends, 37
Hash, 51
 partitioning, 51
HAVING Clause, 528, 543
Hierarchy, 162, 174, 182
 level, 174
 parent-child, 174
Hint, 131, 516
Hit list, 323
HTML5, 53, 433
Hybrid application, 57
Hypernym, 404
Hyponym, 404

I

Identical select, 328, 329
Imperative language element, 557
Imperative programming, 220, 250
Importing, 275
Index, 90
 composite, 545
 exclusion list, 121
 inclusion list, 121
 inverted, 585
 primary index, 543
 server, 52
Indirect encoding, 47
InfoObject, 428, 441
 virtual, 443
InfoProvider, 427, 440
 transient, 440
 virtual, 441
InfoSet
 classic, 445
Initial load, 34
Inline declaration, 206, 573

IN list, 149
In-memory database, 51
In-memory technology, 37
Input help, 383, 409
Input parameter, 189, 522
Insight to action, 424
Integer, 43
Integrated scenario, 55
Internal table, 549
Internet Communication Framework, 505
Inverted index, 586

J

Java Runtime Environment (JRE), 69
JDBC, 33
Job log, 97
Join, 92, 245, 372
 complex, 556
 full outer join, 162
 inner, 122
 inner join, 162, 242
 join types, 162
 left outer, 122, 243
 left outer join, 162
 outer join, 162
 referential join, 163, 169
 right outer, 243
 right outer join, 162
 self join, 200
 text join, 163
Join Engine, 161, 557

K

Kernel, 105
Key field, 161
Key figure, 181
 calculated, 186
K-means, 479, 484

L

Large object, 421,
Left outer join, 243
Linguistic search, 396
List of suggestions, 385
List of synonyms, 388, 403
L node, 222
Load distribution, 49
Lock, 519
 indicator, 274
 object, 110, 119
Logical Unit of Work, 109
Loop, 557
LOOP loop, 542
Low-level technologies, 25
L (programming language), 222, 225
LUW concept, 109

M

Mainframe architecture, 21
Main memory, 37
Main store, 48, 582
Manifest, 64
Mass operation, 553
Master data, 181
Materialization, 157
MDX, 33, 52, 205
Measure, 181
 restricted, 183
Merge, 584
Message server, 105
Mobile application, 496
Modeler, 85
Modification Assistant, 276
MODIFY, 148
Modularization, 523
 unit, 307
Monitoring view, 420

N

Name server, 53
Namespace, 521

Native SQL, 373, 408, 552
 ABAP tables, 518
 ADBC, 133
Near real time, 589
Negative test, 297
NEW operator, 575
Noise words, 401
NUMA architecture, 39

O

Object instance
 create, 575
OData, 53, 494
 service, 433
ODBC, 33
ODBO, 33
OLAP, 57, 426
OLAP Engine, 557
OLTP, 57, 426
Online Analytical Processing (OLAP), 22
Online Transaction Processing (OLTP), 22
On the fly, 426
Open SQL, 59, 116, 219, 369, 103
 array operations, 130
 dynamic, 128
 existence check, 127
 hints, 131
 package size, 129
 transaction control, 130
Operational Data Provisioning, 445
Orchestration logic, 58
OR combination, 148
Original system, 276, 282
Outer join, 115
Output parameter, 522

P

Package, 94, 522
 development package, 272, 277
 encapsulated, 273
 interface, 273
 main package, 272

Package (Cont.)
 SAP HANA, 521
 structure package, 272, 277
 system-local, 278
 test package, 273
PAL, 34, 475
Parallelization, 50
Parameter
 marker, 553
 stopwordListId, 403
 stopwordTable, 403
 textsearch, 403
Parameter types interface, 265
Partitioning, 48
 explicit partition handling, 50
 hash partitioning, 51
 horizontal, 48
 partition pruning, 50
 range partitioning, 51
 round-robin partitioning, 49, 51
 types, 51
 vertical, 48
Partition pruning, 51
PBO module, 469
Performance, 526
Phrase index, 419
Phrase-index ratio, 419
Planning engine, 52
PlanViz, 295, 333, 376,
 analysis, 333
 recording, 333
PMML, 479
Pool table, 516
Predictive analysis, 31, 473
Predictive Analysis Library, 475
Predictive Model Markup Language, 479
Prefix encoding, 46
Prefix namespace, 287
Prepared statement, 526
Prepare phase, 552
Preprocessor server, 53
Presentation layer, 58
Pretty Printer, 81
Primary database, 53, 256, 442
Program
 create, 80
Program (Cont.)
 execute, 83
 hdbstudio, 72
 regi, 236
Projection, 240
 view, 118
Proxy object, 263
Public synonym, 99, 232, 233

R

R, 482
RAM, 37
Range, 124
 partitioning, 51
Read-only procedures, 224
Read/write procedures, 224
Real time, 426, 19
Redirected Database Access (RDA), 140
Refactoring, 296
REF operator, 576
Relational operator, 241, 242
Repair flag, 276
Reporting, 424
 operational, 426
Repository, 32
RESTful Service, 53
Result view, 468
 wrapping, 468
Revision, 70
Right outer join, 243
Robust programming, 524
Role
 management, 494
 SAP HANA Studio, 86
Rollback, 130
Round-robin partitioning, 49, 51
Round trip, 148
Row-based data storage, 41
Row store, 41, 120, 215
R (programming language), 225
Rule, 458, 465
Run-length encoding, 47
Runtime, 260
 analysis, 293, 295, 371
 artifact, 179

Runtime (Cont.)
 error, 299
 object, 98, 232
Runtime statistics, 313, 314, 356, 363
 analysis, 314
 selection, 314

S

SAP Business Application Accelerator,
 589
SAP Business Explorer (BEx), 428
SAP BusinessObjects, 429
SAP BusinessObjects Business
 Intelligence platform, 203, 431
SAP Business Suite, 55
 powered by SAP HANA, 22, 513
SAP Community Network, 495
SAP CO-PA Accelerator, 54
SAP Crystal Reports, 453
SAP Data Services, 35
SAP HANA, 22
 advanced functions, 24
 application cases, 53
 applications, 55
 client software, 70
 development, 86
 Extended Application Services, 86,
 433, 526
 function libraries, 34
 Live, 431
 MDX Provider, 203
 migration, 352, 515
SAP HANA Client, 33
 HTTP, 34
 Package for Microsoft Excel, 35, 203
SAP HANA database, 31
 architecture, 51
SAP HANA Repository, 85, 94, 277,
 521, 522
 Client, 70, 236
 view, 207
SAP HANA software component, 29
 core components, 30
 direct data preparation, 30
 lifecycle management component, 30
SAP HANA software component (Cont.)
 lifecycle management components, 36
 software for data replication, 30
SAP HANA Studio, 31, 32, 85
 authorizations, 86
 database catalog, 89
 hdbinst, 69
 hdbsetup, 69
 perspective, 85
 SQL statement, 376
 system connection, 87
 templates, 93
 user settings, 88
 view modeling, 160, 164
 workspace, 87
SAP HANA UI for Information Access,
 35
SAP HANA view, 60, 205, 373, 518
 performance, 553
 selection, 212
 test, 523
 type, 554
SAP Host Agent, 36
SAP Landscape Transformation
 Replication Server, 35
SAP Lumira, 475
SAP Management Console, 36
SAP Memory, 549
SAP NetWeaver AS ABAP, 33
 architecture, 105
SAP NetWeaver Business Client, 493,
 494, 506
SAP NetWeaver Business Warehouse
 (BW), 22, 426
SAP NetWeaver Gateway, 437, 494
 Service Builder, 496
SAP NetWeaver Operational Process
 Intelligence, 458
SAP Predictive Analysis, 475
SAP software
 real time, 21
SAP Solution Manager, 36
SAPUI5, 433, 494, 495
 application, 504
 Model View Controller, 505
Scalar parameter, 226
Scalar variable, 227

Scale-out, 38, 105
Scale-up, 38
Schema, 89
 handling, 518
 mapping, 232, 283, 518
Scorecard, 474
Script server, 476
Search
 facet, 386
 freestyle, 385
 fuzzy, 384, 385
 linguistic, 385, 388
 sentiment analysis, 386
 synonym search, 385
Search help, 118, 410
 default value, 415
 exit, 410
Secondary connection, 150
Secondary database, 53, 139, 434
Secondary index, 543
Selectivity, 389
SELECT * statement, 369
SELECT statement
 nested, 369, 540
Sentiment analysis, 384, 386, 417
Sequence, 90
Server component, 52
Service Builder, 503
Session context, 167
Set operation, 538
Shadow column, 391
Shared buffer, 549
Shared memory, 549
Shared objects, 549
Side-by-side scenario, , 22
Side panel, 492
 configuration, 505
Single transaction analysis, 295, 330
Size category, 120
Slice and dice, 181
Smart device, 20
Software component, 272
Software innovation, 41
Software Update Manager for SAP
 HANA (SUM), 36
Sort Behavior, 517
Sparse encoding, 47

SQL, 116
 analysis, system-wide, 337
 cache, 295, 340, 553
 console, 92, 224
 data type, 230, 231
 dynamic, 252, 557
 error code, 254
 injection, 252, 526
 Native, 60, 133, 255
 Open, 59
 performance optimization tool, 353
 processor, 52
 profile, 366
 view, 158
SQL92, 215
SQL99, 215
SQLDBC library, 33
SQL Monitor, 296, 342, 352, 366
 activate, 342
 analysis, 342
 entry point, 344
SQL Performance Tuning Worklist, 346, 356
SQLScript, 31, 60, 194, 373, 190
 ABAP tables, 518
 accessing the business logic, 221
 activating, 222
 basic principles, 223
 BREAK, 250
 CALL, 233
 calling, 222
 case distinction, 220, 250
 case sensitivity, 232
 CE Plan Operator, 239
 client handling, 232, 252
 CONTINUE, 250
 control structures, 250
 CREATE PROCEDURE, 226, 236
 CREATE TYPE, 227
 cursor processing, 251
 custom exceptions, 254
 debugger, 235
 default exception, 254
 dynamic, 525
 dynamic SQL, 252
 error handling, 254
 EXEC, 252

SQLScript (Cont.)
 EXECUTE IMMEDIATE, 252
 explicit typing, 237
 implicit typing, 237
 input parameter, 229, 262
 loop, 220, 250
 modularization, 216
 optimizations, 247, 252
 orchestration, 220
 output parameter, 229, 258
 parallelization, 219
 performance, 249
 processing, 222
 processing logic, 230
 qualities, 216
 reuse, 217
 rules, 556
 scalar parameter, 226
 scalar variable, 227, 238
 SESSION_CONTEXT, 252
 splitting up, 217
 SQL versus CE Plan Operators, 246
 system fields, 252
 table parameter, 226
 table type, 226, 227, 235
 table variable, 217, 227, 237
 typing, 237, 238
 UDF, 225
 user-defined functions, 225
 variable, 237
 WITH RESULT VIEW, 227
SQLScript debugger, 302
 standard, 215
SQL statement
 analysis, 354
 BINARY SEARCH, 517
 CREATE FULLTEXT INDEX, 391
 EXEC, 525
 EXEC SQL, 516
 FOR ALL ENTRIES, 540, 541
 INTO CORRESPONDING FIELDS OF, 533
 ORDER BY, 517
 SELECT COUNT()*, 534
 UPDATE, 535
 UPDATE ... SET, 536

SQL trace, 295, 326, 356, 366, 372
 UP TO n ROWS, 531
 analyze, 326
 record, 326
 recording, 145
Stack trace, 328
Standard database connection, 109
Star schema, 157, 182
Statement transformation, 143
Statistic record, 295
Statistics server, 53
Stop word, 389, 401
String, 46
 templates, 573
Structure, 576
Subquery, 127, 541
 scalar, 127
Synonym, 90, 99
 public, 99
Syntax check, 81
System field, 512
System landscape
 mixed, 290
System optimization, 353
System schema, 89, 109

T

Table, 117
 internal, 576, 577
 replicated, 589
 access statistics, 364
 buffer, 107, 120, 519, 542, 548
 contents, 91
 definition, 91
 parameter, 226
Tablet, 496
 type, 226, 227, 235
 variable, 217, 227, 237
Temporary table, 260
 global, 260
 local, 260
Term mapping, 403
Test, 524

Text analysis, 384, 386, 416
Text search, 386, 390
Thread, 340
Time data
 generate, 176
Time zone, 513
Token, 390, 416
Totals table, 157
Trace, 313
Transaction
 ATC, 295
 DBACOCKPIT, 140, 295, 338
 EQPREVIEW, 452
 PFCG, 494, 505
 RSDD_HM_PUBLISH, 440
 SAT, 295, 318, 364, 372
 SCI, 295, 305
 SE11, 77
 SE80, 414
 SEGW, 503
 SQLM, 296, 342
 ST04, 338
 ST05, 145, 151, 295, 372
 ST12, 295, 330
 ST22, 299
 STAD, 295, 314, 356, 363, 371
 SWLT, 346, 293, 366
Transactional system, 426
Transaction data, 181
Transport, 269
 change recording, 269, 273, 278
 change request, 288
 combined, 285
 control file, 276
 CTS+, 292,
 CTS Deploy Web Service, 281
 CTS plug-in, 280
 data file, 275
 developer mode, 280
 exporting, 280
 HANA transport container, 270
 importing, 280
 lock indicator, 274
 log, 276
 logical transport object, 287
 mechanisms, 270
 mixed system landscape, 290

Transport (Cont.)
 object list, 274
 original system, 282
 possible problems, 269
 properties, 273
 recommendations, 291
 relocation, 276
 schema mapping, 232
 synchronization, 289
 transport container, 286
 transport directory, 275
 Transport Domain Controller, 275, 280
 transport layer, 274
 transport log, 290
 transport properties, 273, 278
 transports of copies, 276
 transport system, 274, 278
TREX, 389

U

Unit test, 296, 357, 524
Update, 105
UPSERT, 147
Use access, 273
User interface
 building block, 453

V

Validation, 96
VALUE operator, 576
Variable, 237
 declare, 573
 scalar, 238
 scope, 575
 table variable, 237
Version history, 101
View, 90, 118, 157
 analytic view, 159
 attribute view, 159
 calculation view, 159, 192
 column view, 158, 269
 database view, 132

View (Cont.)
 Dictionary View, 207
 external, 207, 213, 269, 500, 518, 522
 SQL view, 158
Virtual Data Model (VDM), 55

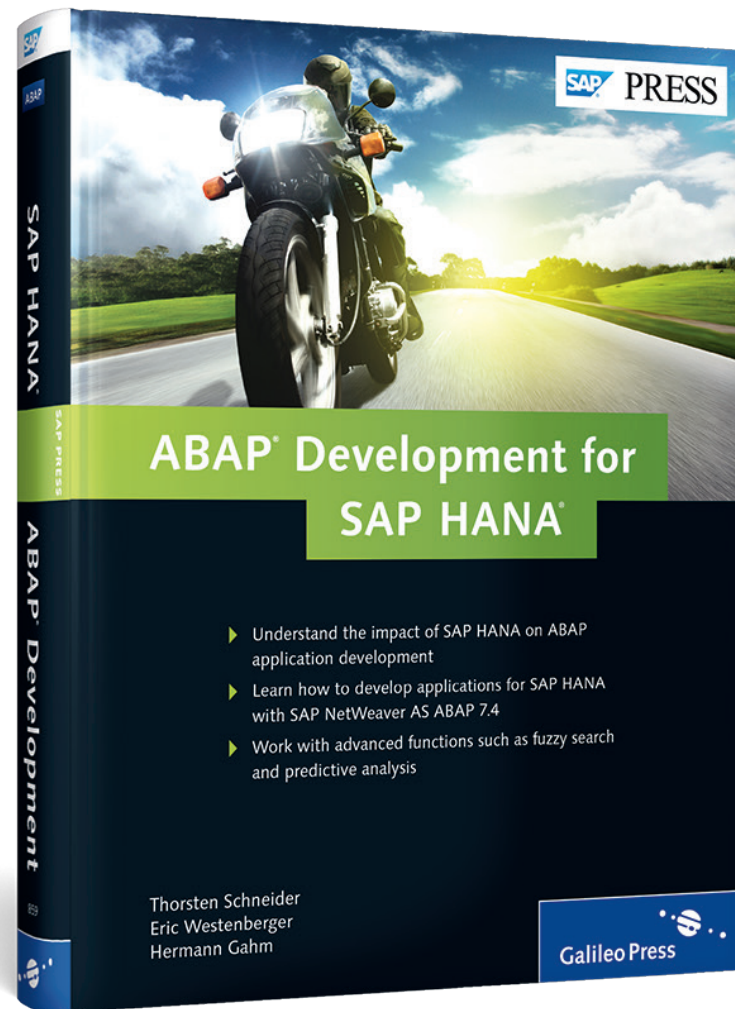
W

Web Dynpro ABAP, 414, 453, 493
 context, 414
 Eclipse, 414
 Floorplan Manager, 498, 502
Web Dynpro ABAP Tools, 71

Weighted average, 483
 BFL, 477
WHERE clause, 527, 543
Where-used list, 82
White list, 526
Widget, 493
Wild card, 383, 396
Word dictionary, 419
Work process, 105, 261
Wrapper function
 AFL, 477

X

XS Engine, 34, 53, 90, 215



Thorsten Schneider, Eric Westenberger, Hermann Gahm

ABAP Development for SAP HANA

609 Pages, 2014, \$69.95

ISBN 978-1-59229-859-4

 www.sap-press.com/H3320



Thorsten Schneider is a product manager in the Product & Innovation HANA Platform department at SAP AG. In this position, he deals with application development using the new in-memory database technology. His main focus is the implementation of business applications based on ABAP and SAP HANA.



Eric Westenberger has worked for SAP AG since 2005, where he is currently a product manager for SAP HANA and SAP NetWeaver. Prior to this, he was involved in the development of several components of the SAP NetWeaver basis technology as a developer and software architect for several years.



Hermann Gahm is a principal consultant in the performance CoE of SAP Global IT Application Services. In this position, he is primarily responsible for performance analysis and optimization of the internal SAP ABAP systems powered by SAP HANA. He's helped SAP customers solve performance problems in the context of ABAP developments and system, database, and ABAP program tuning.

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.