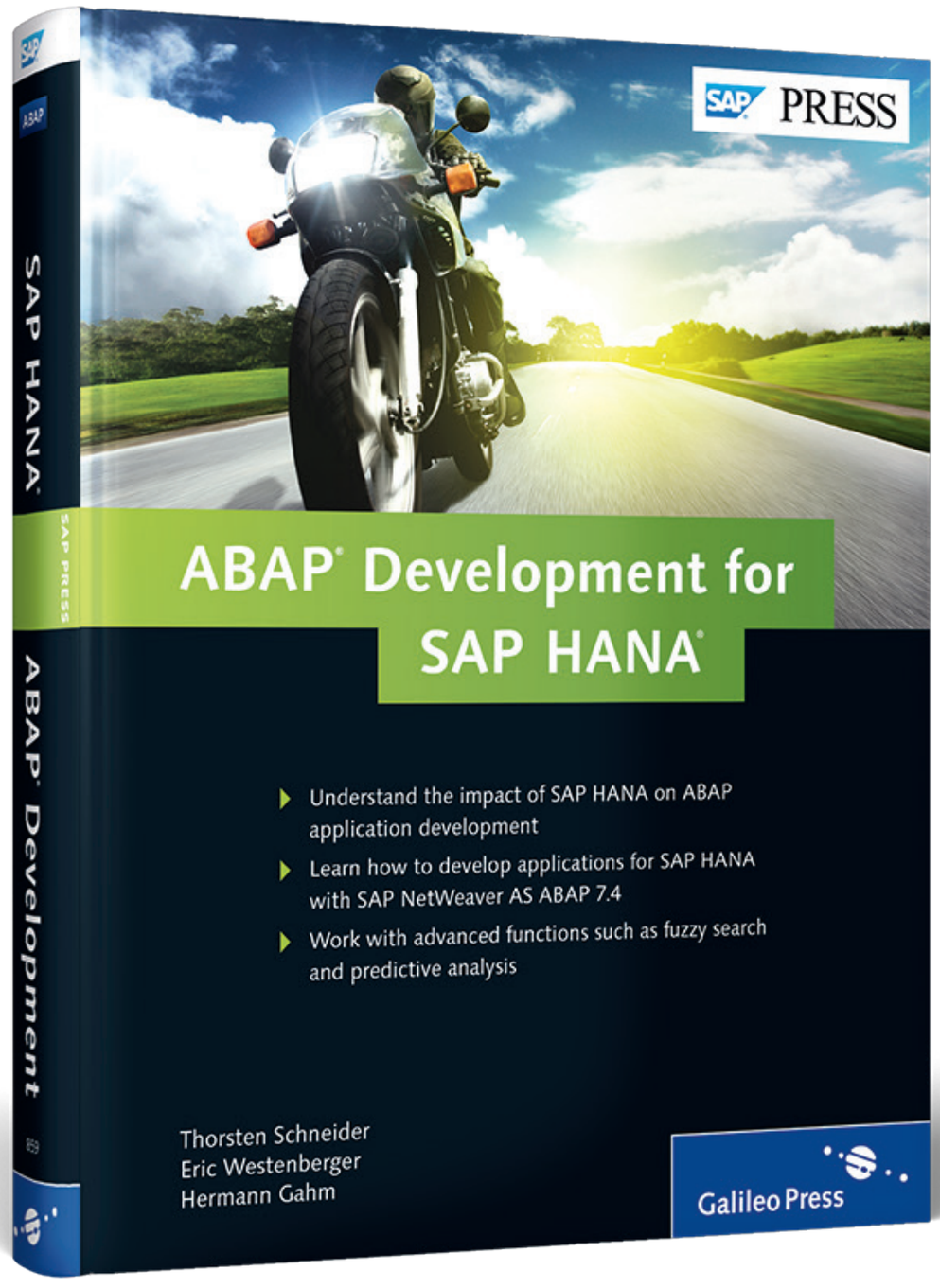


Reading Sample

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.

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.

Example: real time in 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

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.

Real time at SAP

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*).

Importance of business intelligence

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

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).

ABAP development on SAP HANA

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.

Structure of the book: Part I

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

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.

Deploying new technologies

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.

Sample data model

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.

How to use this book

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

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).

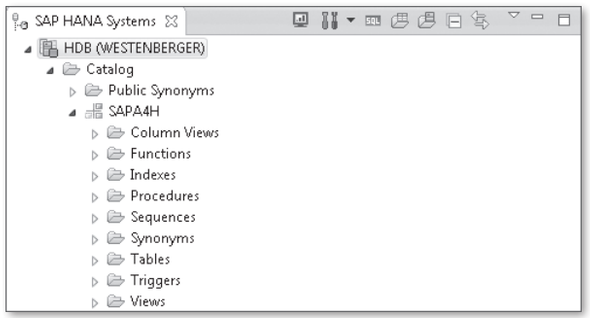


Figure 4.1 SQL Views and Column Views in the Database Catalog

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.

Reference example for this chapter

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.

View types

The following types of views will be discussed:

- *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

Join views

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.

Modeling concepts

When modeling attribute views, we differentiate between the following 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.

SQL examples

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
```

Text joins and referential joins

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.

[»] 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.

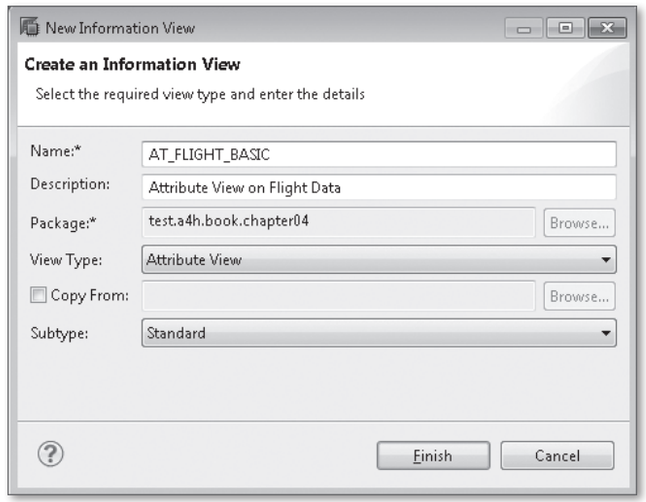


Figure 4.2 Creating an Attribute View

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.

Modeling editor

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.

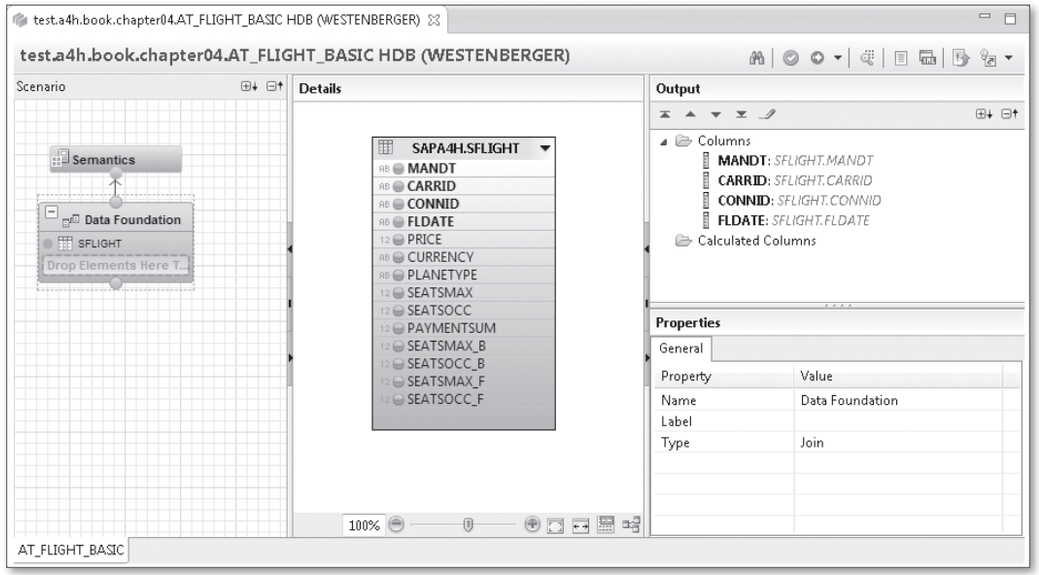


Figure 4.3 Definition of the Data Foundation

Defining metadata

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.

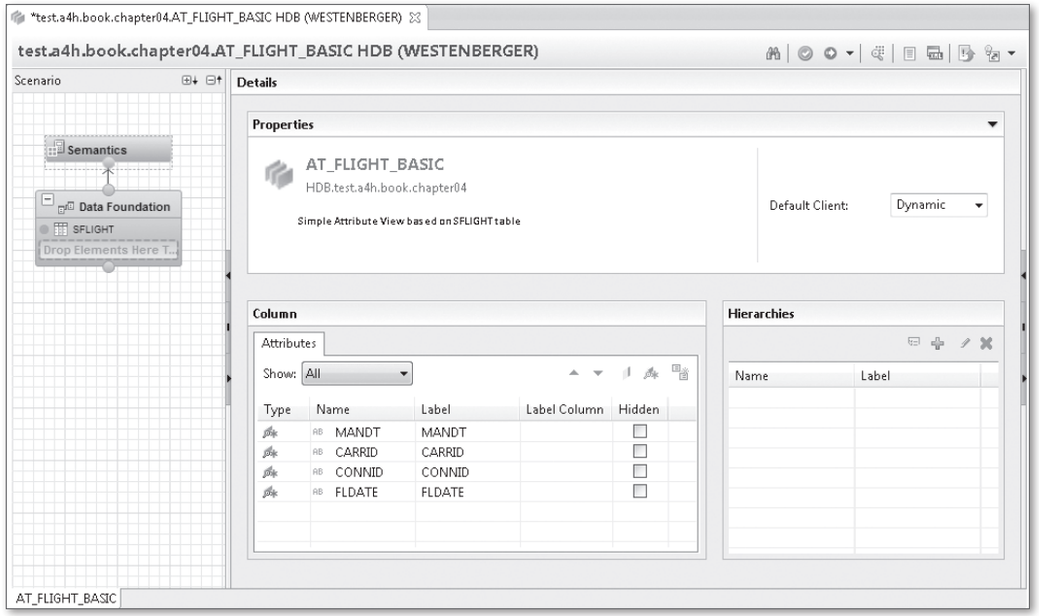


Figure 4.4 Further Semantic Configuration of the Attribute View

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.

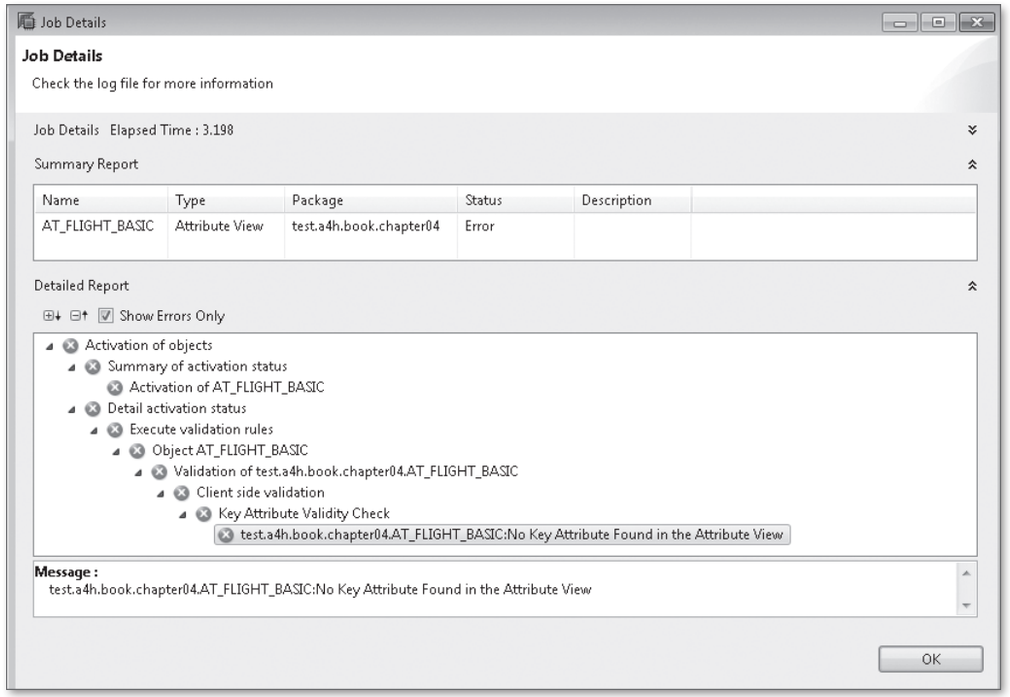


Figure 4.5 Example of an Activation Error

Client handling

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.

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.

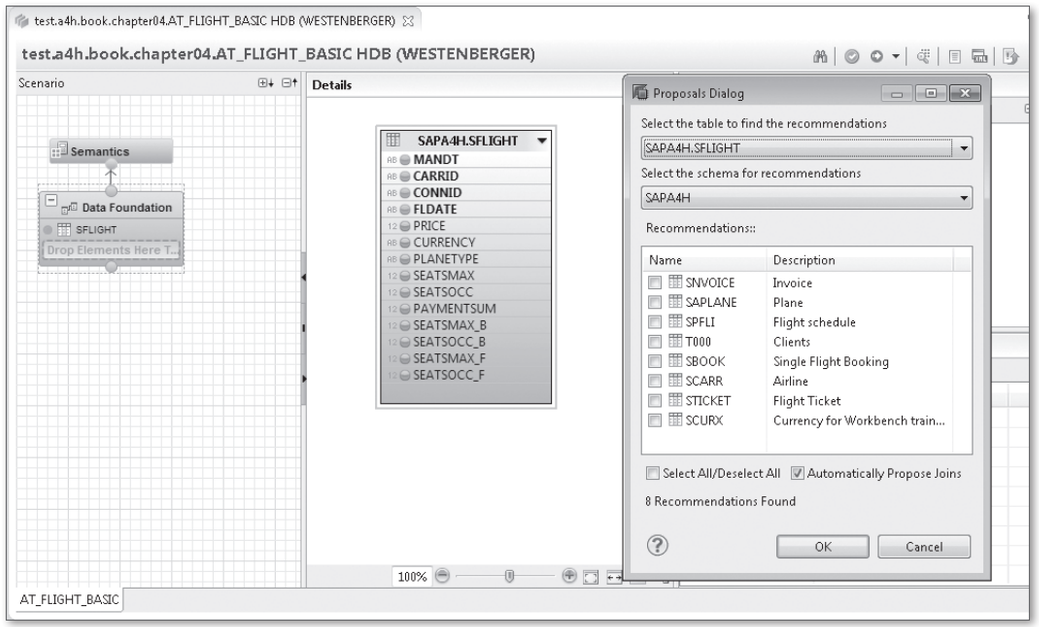


Figure 4.6 Proposed Values for Defining Joins

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.

Adding attributes

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.

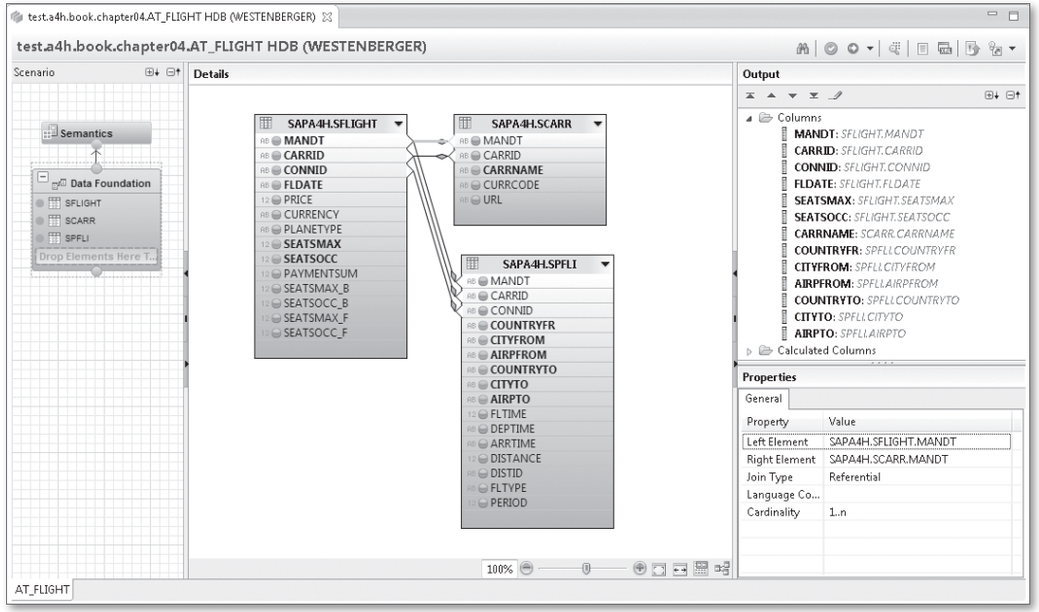
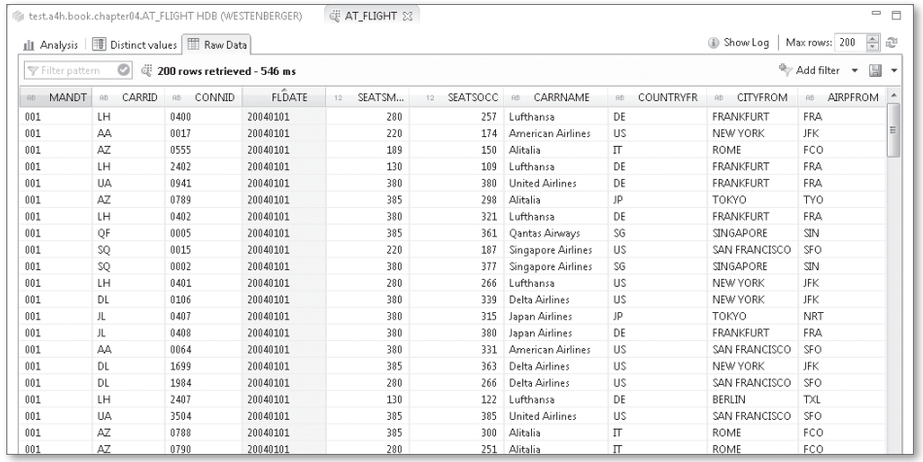


Figure 4.7 Attribute View Analogous to the DDIC View SFLIGHTS

Activate/test

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).



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.

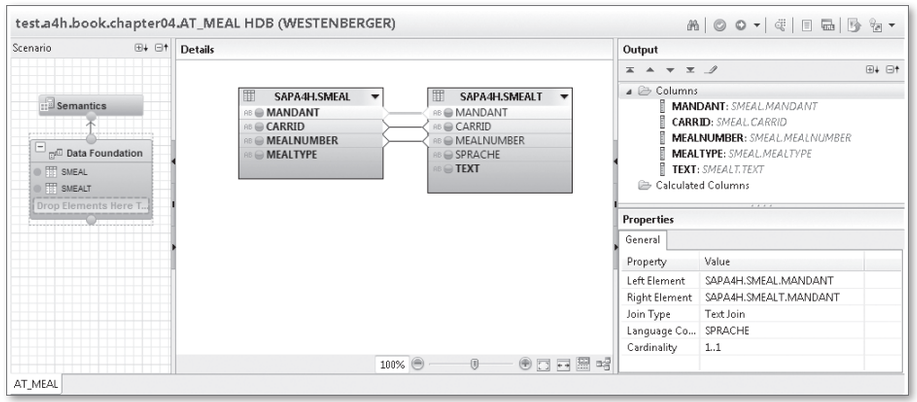


Figure 4.9 Using a Text Join

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).

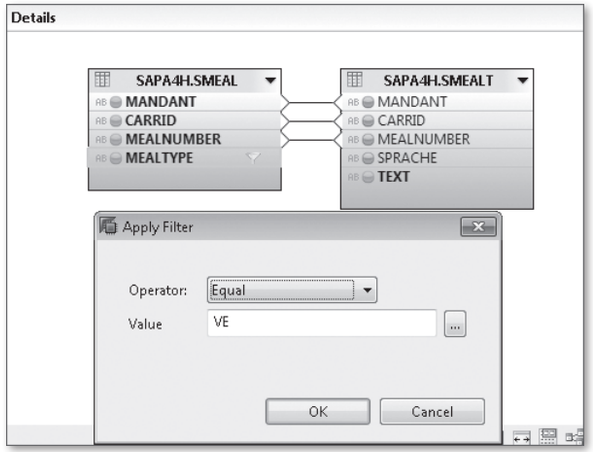
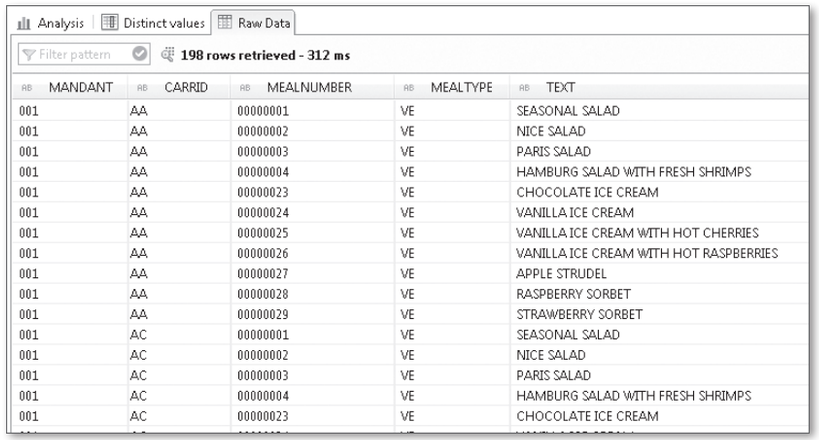


Figure 4.10 Filter for an Attribute

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).

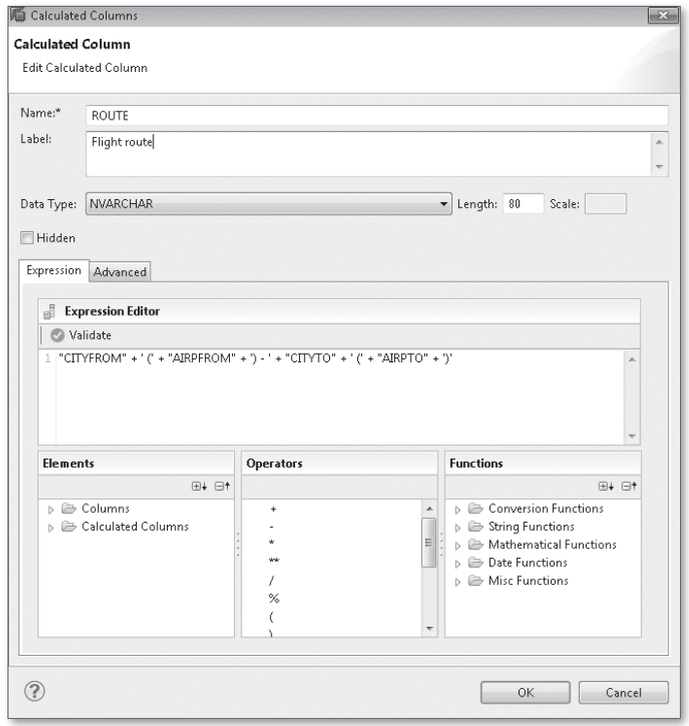


Figure 4.12 Definition of a Calculated Field

Defining expressions for calculations

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

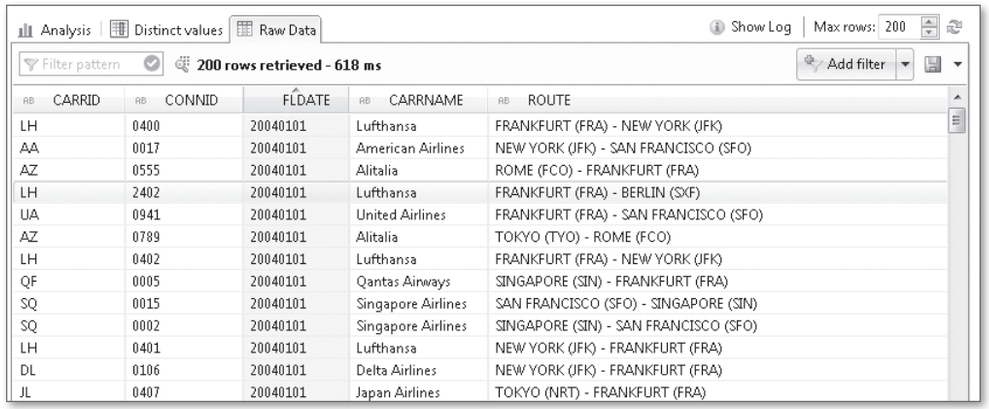
Attribute References and Constants in Expressions **[!]**

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.

Output of the calculated field

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.



CARRID	CONNID	FLDATE	CARRNAME	ROUTE
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 (`TDEVC`) 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).

Creating hierarchies

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).

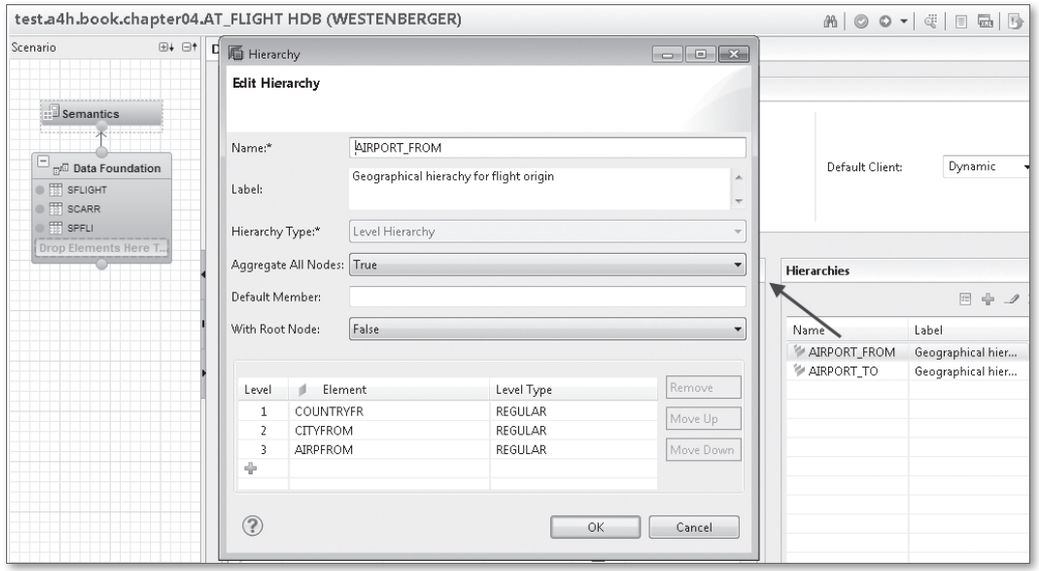


Figure 4.14 Hierarchy of an Attribute View

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.

Limitations of hierarchy support

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).

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.

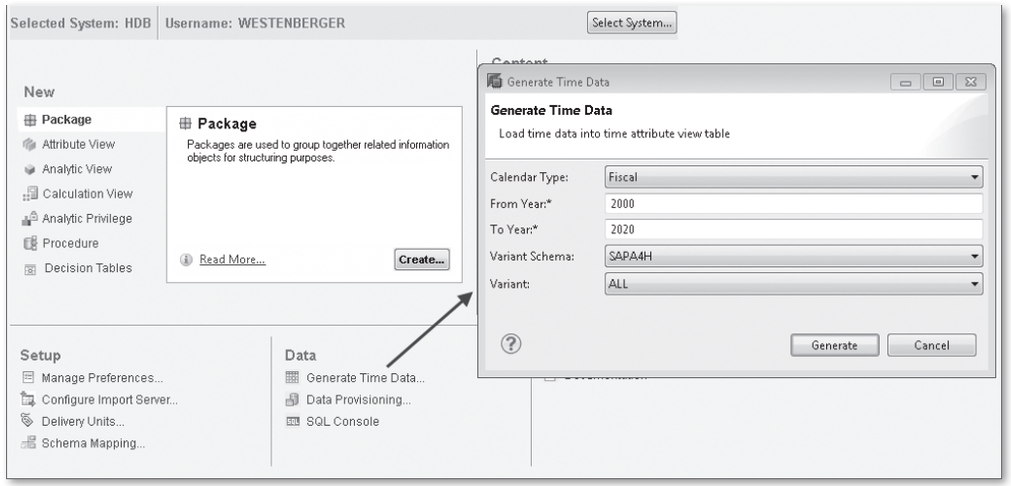


Figure 4.15 Generating the Data for the Fiscal Calendar

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 CALENDER_VARIANT.

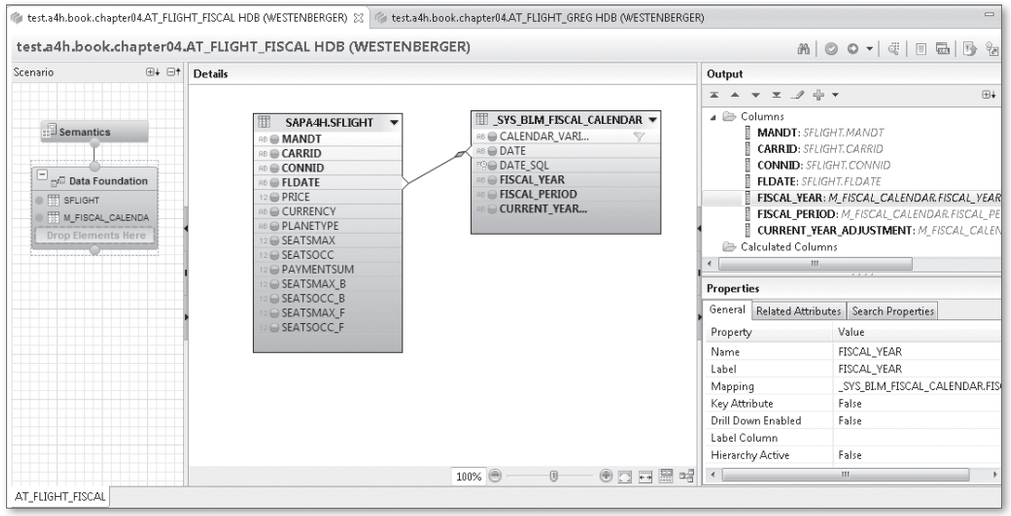


Figure 4.16 Determining the Fiscal Periods for Flight Data

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.

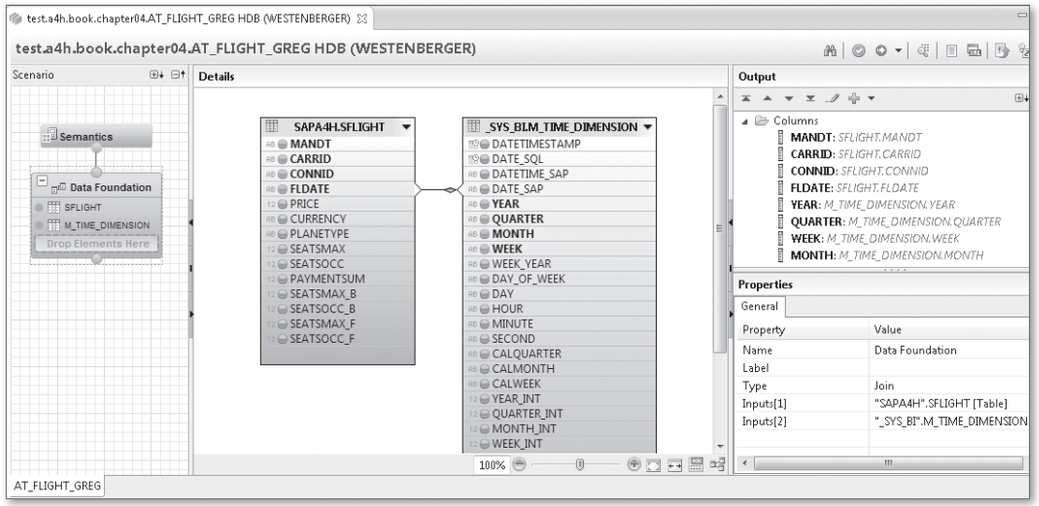


Figure 4.17 Determining the Quarter and Calendar Week

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.

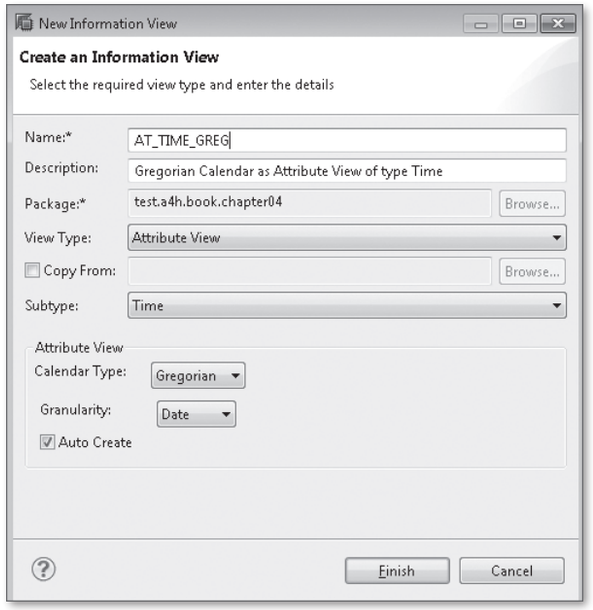


Figure 4.18 Attribute View for a Gregorian Calendar

4.1.6 Runtime Artifacts and SQL Access for Attribute Views

Addressing via SQL

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.

Column views

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.

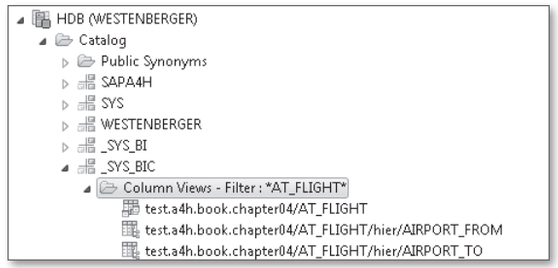


Figure 4.19 Column Views Generated for the Attribute View AT_FLIGHT

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

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

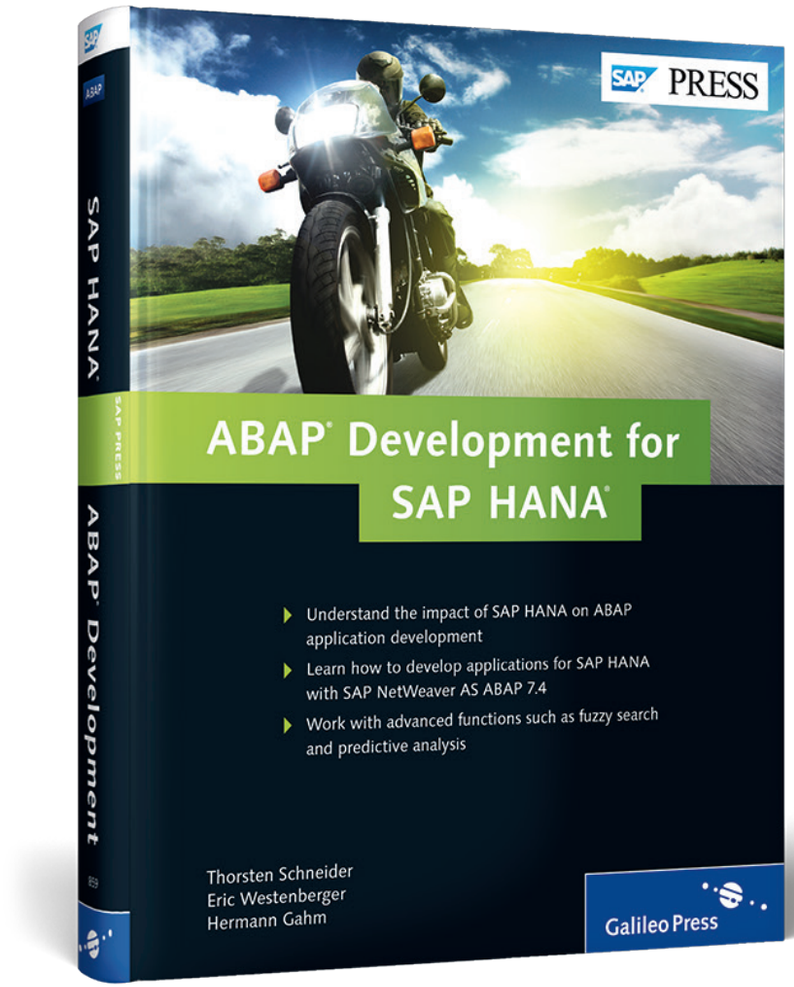
T

U

V

W

X



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.