Paper 2445-2014

# EMPOWERING SAS® USERS ON THE SAP HANA PLATFORM

Christoph Morgen, SAP AG, Germany

## ABSTRACT

This session provides an overview of how SAS® environments can be best integrated with SAP HANA. You'll learn what is different about SAP HANA and how SAS users can access and push-down their work, and thus start to benefit from the in-memory power of SAP HANA. We'll further explore how a SAS® Predictive Modeling environment will be even closer embedded in the SAP HANA platform.

## INTRODUCTION

The starting point for any SAS developer integrating SAS applications with a new database system in order to leverage its capabilities to the benefit of the SAS application is the respective SAS/ ACCESS interface. The same is true for unleashing the power of SAP HANA, where the SAS/ ACCESS to SAP HANA interface was introduced in 2013, providing a mature and rich foundation for any SAS and SAP HANA integration scenario.

The purpose of this paper is to provide broad overview of the current integration capabilities and to highlight and explore the SAP HANA specifics. Furthermore we will give a future outlook of what's coming soon with respect to SAS In-Database Analytics for SAP HANA.

### WHAT IS SAP HANA AND HOW IS SAP HANA DIFFERENT?

The SAP HANA platform implements a new approach to business data processing. In fact, it is much more than the traditional definition of a database and the in-memory implementation of SAP HANA is much more than a naïve caching of disk data structures in the main memory.

First and foremost, SAP HANA incorporates a full database management system (DBMS) with a standard SQL interface, transactional isolation and recovery as well as high availability. Additional functionality, such as freestyle search, text analysis or geospatial processing engine are implemented as SQL extensions.

SAP HANA is a by-design in-memory platform, leveraging the processing power of modern CPUs by achieving highest utilization ratios in different caching layers of the CPU. This outstanding IO-efficiency is achieved by leveraging compressed in-memory data stores along with compression-aware data processing operators. The columnar data storage along with dictionary encoding allows highly efficient data compression rates and eliminates the need for additional index structures in most cases. Compressed data can be loaded into CPU caches more quickly and operations such as scans and aggregations are accelerated because the operators are designed to process compressed data.

For massive parallelization on large multicore processors systems SAP HANA manages the SQL processing instructions into an optimized model that allows parallel execution and scales extremely well with the number of cores. The optimization includes parallel column operations as well as partitioning the data in sections for which the calculations can be executed in parallel. Naturally this includes partitioning of large tables and their processing across multiple nodes. Moreover SAP HANA's parallel-data-flow computing model extends beyond the SQL language with application specific logic and languages like SQLScript, MDX, planning operations or application functions. Specific compilers translate their distinct logic into a common representation called a "calculation model." Such a SAP HANA calculation model can be regarded as a directed acyclic data flow graph, which implicitly enables massive parallelism of the processing and assures the use the optimal database operators and the best performance.

The developer environment SAP HANA Studio provides a graphical modeler tool to visually design SAS HANA Information Views as a design-time graphical representation of the calculation models run-time objects. Developers can thus build on in-memory tables or referenced views as data source inputs and use different operations (join, aggregation, projection and so on) on top of them and so easily design custom data flows. This approach allows the design of virtual cubes, virtual data marts or even most complex and highly stacked virtual data models on top of compressed in-memory data while assuring the best performance and avoiding to store physical data aggregates. This approach is found for example with SAP BW on HANA generated views, SAP Business Suite extensions of SAP HANA Live Views or custom SAP HANA implementations.

In order to best leverage the power of the SAP HANA platform, you will want to make sure that the heavy data lifting is pushed down for processing from your SAS environment to SAP HANA and that you can leverage SAP HANA specific capabilities like processing SAP HANA Information Views.

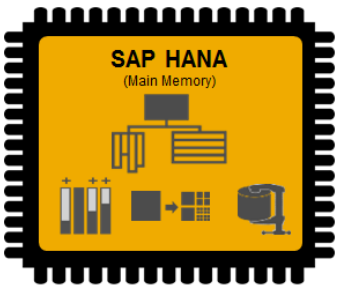


**Figure 1. SAP HANA In-Memory Platform**

### INTEGRATION OVERVIEW

For SAS Business Analytics customer environments the SAS/ACCESS to HANA module is first key enabling technology to work with SAP HANA, this implies SAS BI-, SAS Advanced Analytics- as well as SAS Data Integration scenarios or custom SAS Applications. The interface is a mature and sophisticated SAS/ACCESS Engine including the latest flavors of SAS SQL pass-through capabilities and the implicit use of SAP HANA load utilities. In fact the SAS Data Integration Server has already been certified as an integrated ETL application for SAP HANA.

While SAS/ACCESS to HANA interface provides matching capabilities to other SAS database interfaces, it additionally enables SAS users to consume SAP HANA Information Views for a maximum in performance when working with SAP HANA. SAP HANA Information Views maybe serviced as pre-defined by SAP HANA-based applications or be utilized to specifically pre-model data intensive data preparation flows inside SAP HANA. Data flow architectures may be re-thought and optimized in a SAS and SAP HANA environment accordingly.

Specifically for advanced analytics applications like SAS Enterprise Miner the SAP HANA integration will reach out to enable SAS In-Database Scoring in SAP HANA and will enable SAS High-Performance Data Mining scenarios with SAP HANA.

## GETTING STARTED WITH SAS/ACCESS TO HANA

### SAS/ACCESS LIBNAME ENGINE

With a SAS/ACCESS libname statement, your SAP HANA stored data is ready to consume with your usual SAS toolbox of DATA Step and PROCs almost as if you were working with ordinary SAS DATA sets. The libname engine for SAP HANA **SASIOHNA** translates the data access commands and sometimes even procedural logic into SQL request, which get passed implicitly to SAP HANA, causing as much work to be done in the database as possible.

The libname statement for SAP HANA is not too different from others and will appear quite familiar to the SAS developer. Here is an example connecting to SAP HANA and executing a SAS procedure:

```
libname shine sasiohna  server="hanahost" instance=00 user=user password=pwd
                        schema=shine <libname-options>;

proc print data=shine.BusinessPartnerContacts;
  var first_name last_name email_address;
  where UPCASE(substr(First_Name,1,3)) = "SAS";
run;
```

In order to better follow what's happening behind the scenes, i.e. to determine the success or failure of implicit pass-through for a query, you must examine the SQL that the SAS/ACCESS engine submits to the SAP HANA by using tracing SAS system options:

```
OPTIONS SQL_IP_TRACE=ALL /* To start with */
        SASTRACE=',,,d' SASTRACELOC=SASLOG  NOSTSUFFIX /* For more details */;
```

With these settings activated, one can now investigate the efforts undertaken by SAS to push down a SQL-based data access request, including whenever possible where-clause filtering, column selections, sample options, SAS to SAP HANA function mapping in filtering expressions and more.

For the given example from above, note from the trace information in the SAS log how the SAS function UPCASE is translated to the SAP HANA function UPPER in order to not break the push-down effort:

```
SAPHANA_2: Prepared: on connection 2
SELECT  "FIRST_NAME", "LAST_NAME", "EMAIL_ADDRESS"
        FROM "SHINE"."BusinessPartnerContacts"
        WHERE  (UPPER(SUBSTRING( "FIRST_NAME", 1, 3)) = 'SAS' )
SAPHANA_3: Executed: on connection 2
Prepared statement SAPHANA_2
```

A helpful option to be used with the SAS/ACCESS libname statement for HANA will be the option `PRESERVE_TAB_NAMES`, especially when working with SAP HANA object names in mixed case and special characters in names.

```
libname shine sasiohna  server="hanahost" instance=00 user=user password=pwd
                        schema=shine PRESERVE_TAB_NAMES=YES;
```

Along with emulating SAS I/O requests as SQL pushed to SAP HANA, the interface is also handling data type conversion between SAP HANA and SAS data types, as well as doing its best even to parallelize the query request. Of course depending on the individual use case, but in general large data transfers shall be avoided and the filtering and aggregation logic let be taken care off by your in-memory database SAP HANA.

### PROC SQL Implicit Pass-through

By using SQL you will naturally maximize your performance benefit expected when consuming data from a database with SAS. The SAS BASE language Procedure “Proc SQL” allows to use the database-neutral SAS SQL dialect (based on ANSI SQL2), where the SAS/ACCESS Engine technology again does their magic to implicitly translate the SAS SQL query into a native SAP HANA SQL request, which is pushed for processing to the database.

In fact many SAS Applications make heavy use of SQL processing like SAS BI Reporting or SAS Marketing Automation building on SAS Information Maps, which form a metadata structure for generating SQL-based queries. Other applications like SAS Enterprise Guide or SAS Data Integration Studio allow modeling of SQL-based processing including visual validation and indication tools (the latter) for the SQL push-down effort.

The following example shows Implicit PassThrough (IP) as the result of a collaborative effort between PROC SQL and SAS/ACCESS Libname Engine:

```
libname shine sasiohna server="hanahost" instance=00 schema=shine <options>;
proc sql;
  select tm.year, so.PartnerId, so.CreatedAt,
         STRIP(so.BillingStatus), sum(so.GrossAmount) as Revenue
  from shine.SalesOrder so, shine.TimeDim as tm
  where so.createdat=tm.date_sql
   and CreatedAt between "01JUN12"d and today()
   and put(BillingStatus, $billstats.)='Invoiced Only'
  group by tm.year, so.PartnerId, so.CreatedAt, so.BillingStatus;
quit;
```

```
SQL_IP_TRACE: passed down query:
select tm."YEAR", so."PartnerId", so."CreatedAt",
       TRIM(so."BillingStatus"), SUM(so."GrossAmount") as Revenue
from "SHINE"."SalesOrder" so, "SHINE"."TimeDim" tm
where (so."CreatedAt" = tm."DATE_SQL") and
      (so."CreatedAt" between  {d '2012-06-01' } and  {d '2014-02-15' } and
      (so."BillingStatus" = 'I'))
group by tm."YEAR", so."PartnerId", so."CreatedAt", so."BillingStatus"
ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.
SQL_IP_TRACE: The SELECT statement was passed to the DBMS.
```

As in the first example, a SAS function gets implicitly mapped to the SAP HANA function(STRIP to TRIM), but moreover the complete SQL query is pushed into the database for processing. If we take a closer look, we can notice the magic includes the resolution of SAS date constants and a datetime function to the respective SAP HANA syntax and values. The logic of using SAS formatted values with the SAS PUT function in the where clause is resolved and inverted to a filter expression based on the unformatted value using the so-called SAS unPUT optimization.

### PROC SQL Explicit Pass-through

In certain scenarios though, the use of implicit SQL generation may be regarded as too constrained, because for example specific SAP HANA native SQL language capabilities or language extensions shall be used, database stored procedures or table functions called. In fact it is a commonly used approach in SAS applications to get your optimized database SQL expression passed the SAS SQL parser.

For such scenarios, SAS developers can make use of the explicit SQL pass-through concept of Proc SQL. In either the construct of `select <> from connection to sasiohna (<HANA SQL>)` or the `(<HANA SQL>) execute by sasiohna;` call, SAP HANA native SQL calls can be issued from a SAS program.

Here is an example of an explicit pass-through SQL request to SAP HANA, leveraging SAP HANA fuzzy text search capabilities for filtering your data:

```
proc sql;
connect to sasiohna as myHANA (server="hanahost" instance=00 <options>);
select * from connection to myHANA
(Select "Category", "Text" as "ProductName",  sum("GrossAmount") as "GrossAmount"
    from "MDXC"."Products" P, "MDXC"."texts" T, "MDXC"."SalesOrderItem" SO
    where P."NameId"=T."TextId" and P."ProductId"=SO."ProductId"
          and T."Language"='E'
          and contains("Text", 'Notebuch', Fuzzy(0.7))
    Group by  "Category", "Text");
disconnect from myHANA;
quit;
```

In SAS Application development scenarios this offers a great opportunity to control the generation of very specific but native SAP HANA SQL requests and thus assures the execution of application logic inside the SAP HANA database. With adding the flexibility of the SAS Macro language and programming, SAS Application can be enabled to generate really dynamic SAP HANA SQL requests.

Here is an example of what a SAS Macro-Language based version of the previous example may look like as part of a custom SAS application or a SAS Stored Process.

```
%MACRO DYNAMICHANASQL();
    proc sql;
    connect to sasiohna as myHANA (server="hanahost" instance=00 … );
    select * from connection to myHANA
     (Select "Category" "Text" as "PName",  sum("GrossAmount") AS "GrossAmount"
          from "MDXC"."Products" P, "MDXC"."texts" T, "MDXC"."SalesOrderItem" SO
                where P."NameId"=T."TextId"  and P."ProductId"=SO."ProductId"
                and T."Language"= %IF &USERLANG. EQ DE %THEN 'DE'; %ELSE 'E';
          and contains("Text", &fuzzyterm., Fuzzy(0.7))
                Group by  "Category", "Text");
    disconnect from myHANA;
    quit;
%MEND;

%let fuzzyterm='Notebuch';
%let USERLANG=EN;
%DYNAMICHANASQL();
```

In this simple example, the user-language (**&USERLANG.)** and the productname-string (**&fuzzyterm.**) are passed into the SAS program as a SAS Macro Variable, which an end-user may be have been prompted for in case of a SAS Stored Process.

## LOADING, UPDATEING AND CREATING SAP HANA TABLES FROM SAS

As a certified integrated ETL application for SAP HANA, the SAS Data Integration Server does implicitly support SAP HANA's bulk import routines. The SAS/ACCESS to HANA interface supports options for optimizing the performance of insert, updates and bulkloads like `INSERTBUFF`, `UPDATEBUFF` or `DBCOMMIT`.
In the following example 50000 rows will be inserted in 5 buffered insert sets of 10000 rows more efficiently using the SAS/ACCESS Data Set option `INSERTBUFF`:

```
libname HSANDBOX sasiohna server="hanahost" instance=00 schema=SANDBOX <options>;
 proc SQL;
    insert into  HSANDBOX.SalesOrder (insertbuff=10000)
    select  *  from WORK.SALESORDER (Obs=50000);
run;
```

```
SAPHANA_136: Prepared: on connection 3
INSERT INTO "SANDBOX"."SalesOrder"
("SalesOrderId","CreatedBy","CreatedAt","ChangedBy","ChangedAt","NoteId","Partner
Id","Currency","GrossAmount","NetAmount","TaxAmount
","LifecycleStatus","BillingStatus","DeliveryStatus")  VALUES ( ? , ? , ? , ? , ?
, ? , ? , ? , ? , ? , ? , ? , ? , ? , ? )
SAPHANA_137: Executed: on connection 3
Prepared statement SAPHANA_136
SAPHANA_138: Executed: on connection 3
Prepared statement SAPHANA_136
SAPHANA_139: Executed: on connection 3
Prepared statement SAPHANA_136
SAPHANA_140: Executed: on connection 3
Prepared statement SAPHANA_136
SAPHANA_141: Executed: on connection 3
Prepared statement SAPHANA_136
SAPHANA: COMMIT performed on connection 3.
NOTE: 50000 rows were inserted into HSANDBOX.SalesOrder.
```

Even faster inserts can be invoked using the SAS Procedure APPEND and the SAS/ACCESS Option BULKLOAD, which implicitly invokes SAP HANA's native bulk insert utilities behind the scene to maximize the load performance.

```
proc append
    base=HSANDBOX.SalesOrder  (bulkload=yes
                               BL_SFTP_HOST='hanahost' BL_SFTP_USER=uid)
    data=WORK.SALESORDER (Obs=50000) ;
run;
```

```
23         proc append  base= HSANDBOX.SalesOrder (bulkload=yes
24                                                 BL_SFTP_HOST='hanahost'
25                                                 BL_SFTP_USER=uid)
26         data=WORK.SALESORDER (Obs=50000) ;
27         run;
NOTE: Appending WORK.SALESORDER to HSANDBOX.SalesOrder.
NOTE: There were 50000 observations read from the data set WORK.SALESORDER.
NOTE: 50000 observations added.
NOTE: The data set HSANDBOX.SalesOrder has . observations and 14 variables.
SAPHANA_158: Executed: on connection 6
IMPORT FROM '/tmp/BL_SalesOrder_3.ctl'
SAPHANA: COMMIT performed on connection 6.
```

If your scenario of loading data to SAP HANA also includes the actual creation of tables inside SAP HANA, there are a couple of additional considerations to take into account. We recall that SAP HANA is an in-memory database optimized for columnar data store and processing, hence the first and foremost is to make sure we create columnar tables!

Using explicit SQL pass-through provides the most explicit way of defining SAP HANA tables structures:

```
proc sql;
connect to sasiohna as myHANA (server="hanahost" instance=00 <options>);
 (Create column table <…>) execute by sasiohna;
disconnect from myHANA;
quit;
```

Frequently though as a SAS developer one tends to go the implicit path using the SAS/ACCESS Libname Engine. For that case, it is to note that the default setting for SAP HANA libname engine is however to create row-tables. This behavior can easily be controlled with the libname or data set option TABLE_TYPE=COLUMN.
In addition to that with the DBTYPE data set option it is possible to explicitly specify column type definition, instead of relying on the implicit type conversion from SAS column types and formats mapped to a default SAP HANA column format (varchar, doubles):

```
libname HSANDBOX sasiohna server="hanahost" instance=00 schema=SANDBOX …
                                                        TABLE_TYPE=COLUMN;
data HSANDBOX.class (dbtype=(AGE='INT' Profession='TEXT' Salary='DECIMAL(13,2)' )
                     insertbuff=100);
set sashelp.class;
…
run;
```

```
SAPHANA_273: Executed: on connection 6
CREATE COLUMN  TABLE "SANDBOX"."class" (Name varchar(8),gender varchar(1),
Age INT,Height double,Weight double,Profession TEXT,Salary DECIMAL(13,2))
SAPHANA: COMMIT performed on connection 6.
SAPHANA_275: Prepared: on connection 6
INSERT INTO "SANDBOX"."class" (Name,Sex,Age,Height,Weight,Profession,Salary)
VALUES ( ? , ? , ? , ? , ? , ? , ? )
SAPHANA_276: Executed: on connection 6
Prepared statement SAPHANA_275
SAPHANA: COMMIT performed on connection 6.
```

SAS provides a series of additional options for specific use cases, e.g. DBCREATE_TABLE_OPTS for appending additional clauses at the end of an SQL CREATE TABLE statement, like partitioning information. Or the libname option DBMSTEMP=YES, which enables the use to temporary database tables, potentially interesting where your SAS application may benefit from working with non-permanent tables.
For a more complete understanding, see SAS/ACCESS to HANA documentation, which can be found on the SAS Technical Support Web site at http://support.sas.com.

## LEVERAGING THE POWER OF SAP HANA WITH SAS/ACCESS TO HANA

While we have previously explored how SAS classically integrates with SAP HANA as a columnar data base, an even closer integration goes beyond that. The key approach to successfully unleash the power of SAP HANA's ultrafast in-memory aggregation capabilities is to make use of SAP HANA Information Views, which implicitly call on special operators designed for fastest calculations on top of your in-memory data.

SAP HANA Analytic- and Calculation Views can be designed as customer specific views using a graphical design tool within the SAP HANA development environment of SAP HANA Studio. Further they may have got deployed as SAP Business Suite on SAP HANA extensions, the so called SAP HANA Live-Views or have been generated from a SAP BW on HANA system.

SAP HANA Analytic Views can be regarded as cube-like virtual structures, combining fact tables with attributes and hierarchies from master data attribute views. Analytic Views store no aggregates and directly leverage SAP HANA's fastest on-the-fly mass-aggregation operations. They can be consumed by multidimensional reporting BI-clients using SQL or the multidimensional expression language (MDX).

SAP HANA Calculation Views offer a more general way to model custom data flows. This includes multidimensional reporting scenarios, where e.g. multiple Analytic Views shall be combined to a multi-cube calculation view. Or calculation views are leveraged to model complex, multi-layered virtual data models based on normalized data structures.

Views flagged for multidimensional reporting, are surfaced to external clients through a specific SAP HANA schema, which the SAS/ACCESS to HANA interface can connect to and thus enable transparent access to SAP HANA Information Views.

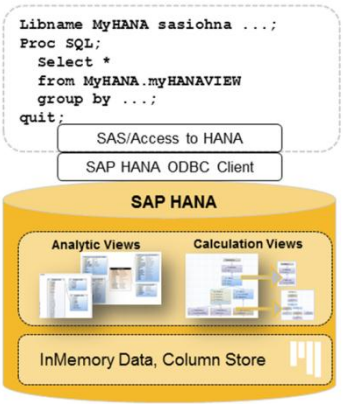


**Figure 2. SAP HANA Information Views**

## ACCESSING SAP HANA INFORMATION VIEWS

The SAP HANA database schema **_SYS_BIC** contains the collection of active SAP HANA Information Views and exposes these views according to the respective SAP HANA database privileges. The view names contain the content structure path of the SAP HANA repository-package before the actual object name including and separated by special characters, hence SAS name literals should be used as a reference.

SAP HANA Calculation Views can be directly consumed via a SAS/Access Libname Engine connection:

```
libname HSYSBIC sasiohna server="hanahost" instance=00 schema=_SYS_BIC …
                                                       PRESERVE_TAB_NAMES=YES ;
proc sql;
  select Year, ProductCAtegory, GrossAmountEUR
  from   HSYSBIC.'MDXC.dev/CV_SALESORDERS'n;
quit;
```

```
SAPHANA_2: Prepared: on connection 3
SELECT  "YEAR", "ProductCategory", "GrossAmountEUR"
      FROM "_SYS_BIC"."MDXC.dev/CV_SALESORDERS"

SAPHANA_3: Executed: on connection 3
Prepared statement SAPHANA_2.
```

**IMPORTANT NOTE**, although you query such a view via SQL and without specifying a group by-clause, the SAP HANA Information View still behaves like a cube, i.e. they will implicitly aggregate your request to the cardinality of columns specified in your select clause! This behavior can be observed to all SAP HANA views flagged for multidimensional reporting use by the SAP HANA developer.

| YEAR | ProductCategory | GrossAmountEUR |
|---|---|---|
| 2012 | Flat screens | 107958922.26 |
| 2012 | Graphic cards | 7165889.06 |
| 2012 | Handhelds | 45631138.12 |

Alternatively or additionally to accessing SAP HANA Information Views via the schema _SYS_BIC, public-schema or custom schema view- synonyms can be created to provision views in SAP HANA schemas along with other tables and schema content.

```
/* Leverage HANA Views - with View-Synonyms in User Schema */
/*CREATE SYNONYM SANDBOX.CV_SYN_SALESORDERS FOR "_SYS_BIC"."MDXC.dev/CV_SALESORDERS";*/
libname HSANDBOX sasiohna server="hanahost" instance=00 schema= SANDBOX …
                                                    PRESERVE_TAB_NAMES=YES ;
proc sql;
   select Year, ProductCAtegory, GrossAmountEUR
   from HSANDBOX.CV_SYN_SALESORDERS;
quit;
```

SAP HANA Analytic Views more rigorously behave like multidimensional cubes compared to Calculation Views, as they don't allow SQL queries with an unspecified column list like `select * from …;` Hence, they cannot be queried through the SAS/ACCESS Libname Engine connection, but require to use an specified explicit pass-through SQL select-expression:

```
/* Using explicit SQL Pass-Through Querying HANA Views */
proc sql;
connect to sasiohna as myHANA (server="hanahost" instance=00 …);
select * from connection to myHANA
 (select "Country", "ProductCategory", SUM("GrossAmount") AS "GrossAmount_SUM"
          from "_SYS_BIC"."MDXC.dev/ANV_SALESORDERS"
   where "Country" in ('DE', 'US')
   GROUP BY "Country", "ProductCategory"
   ORDER BY "Country" ASC, "ProductCategory" ASC);
disconnect from myHANA;
quit;
```

## ACCESSING SAP HANA VIEWS WITH VARIABLES AND PARAMETERS

SAP HANA Analytic and Calculation Views can contain SAP HANA variables or parameters. Variables and Input parameters in SAP HANA can be described as a type of input requests for values at query time.

SAS developers can match the concept to SAS Macro Variables prompts for a SAS Stored Process or filter-prompts with SAS Information Maps.

SAP HANA Variable are used as pre-defined filters to an attribute column of a view, hence they represent a runtime input request for values on how to restrict the data that will be queried. From a SAS developers point of view, the can be regarded as an additional where-clause to the SQL query.

SAP HANA Input Parameters enable dynamic view processing based on the input values provided. Input Parameters can be used to filter processed data at a distinct level of the view data flow. Moreover they are commonly used in calculated column expressions, dynamic unit or currency conversion calculations, dynamic hierarchies and more. In Calculation Views, input parameters can additionally be passed into script nodes for dynamic SQL-Script code generation.

The SAP HANA engines require values for input parameters to be passed along with query at runtime. The values can be passed to the engine via an `PLACEHOLDER` clause appended to the SAP HANA SQL statement. Input Parameters are referenced with leading and trailing pairs of "`$$`":

```
select ... from CV1( 'PLACEHOLDER' = ('$$var$$' = 'value')
```

The `PLACEHOLDER` clause can be passed to SAP HANA from SAS when SAS PROC SQL Explicit Pass-through is used, as it allows to specify any SAP HANA specific expression or clause.

In the following example a SAP HANA Information View is queried with passing values for the `IP_CutoffDate`, `IP_Discount_PCT` and `IP_Product_Category` Input Parameters and appended with the variable filtering clause for `Country`.

```
proc sql;
connect to sasiohna as myHANA (server="hanahost" instance=00 …);
select ProductCategory, GrossAmount_SUM,
       DiscountedGrossAmount_SUM format=12.2
from connection to myHANA
   (SELECT  "ProductCategory", SUM("GrossAmount") AS "GrossAmount_SUM",
            SUM("DiscountedGrossAmount") AS "DiscountedGrossAmount_SUM"
    FROM "_SYS_BIC"."MDXC.dev/ANV_SALESORDERS_III"
         ('PLACEHOLDER' = ('$$IP_CutoffDate$$', '2014-02-12 15:31:40'),
          'PLACEHOLDER' = ('$$IP_Discount_PCT$$', '0.9'),
          'PLACEHOLDER' = ('$$IP_Product_Category$$', 'Workstation ensemble'))
    WHERE ("Country" IN ('US') )
    GROUP BY "ProductCategory"
    ORDER BY "ProductCategory" ASC);
disconnect from myHANA;
quit;
```

More information about SAP HANA variables and parameters like default values, list of values, type, mandatory settings, etc. may be required for the SAS developer to pass the correct values to SAP HANA.

Therefore Metadata information about Variables and Parameters of SAP HANA Views is exposed to BI-clients through a set of tables in the _SYS_BI schema: BIMC_VARIABLE, BIMC_VARIABLE_VALUE, BIMC_VARIABLE_ASSIGNMENT, BIMC_VARIABLE_ODBO. The tables enable, for example, to retrieve the variables defined for a view and e.g. to create a SAS prompt so the end-user of the SAS Application can select or enter the correct values.

In the following example the metadata tables are queried for variable information of the SAP HANA views `ANV_SALESORDERS_II` and `ANV_SALESORDERS_III`:

```
libname HSYSPARM sasiohna server="hanahost" instance=00 schema=_SYS_BI <options>;
proc sql;
    select * from HSYSPARM.BIMC_VARIABLE
    where CUBE_NAME in ('ANV_SALESORDERS_II', 'ANV_SALESORDERS_III');
quit;
```

Adding the SAS Macro Language on top, the dynamics of a parameterized SAP HANA Information View can be directly matched with the dynamics of SAS programming and enables SAS developers to build integrated SAS Applications leveraging the power of SAP HANA.

```
%let Discount='0.8';   /* SAS Macro-Variable declaration */
proc sql;
connect to sasiohna as myHANA (server="hanahost" instance=00 …);
select ... from connection to myHANA
   (SELECT ...  FROM ...
         ( 'PLACEHOLDER' =
          ('$$IP_Discount_PCT$$', &discount.) /* SAS Macro Variable reference */
          ...)
    ...);
disconnect from myHANA;
quit;
```

## CONCLUSION

### INTEGRATED PLATFORMS

For an optimized approach of a SAS and SAP HANA integrated application it is key to redesign classic data access and query patterns, pushing heavy data lifting or aggregation tasks to SAP HANA. This may involve moving data models from SAS to SAP HANA and preparing SAS HANA Information Views as ready for consumption data structures to SAS applications. SAS/ACCESS to HANA builds the foundational SAS component to enable this level of integration.

With specific reference to an optimized workflow in a SAS Predictive Modeling environment, data will likely be prepared in SAP HANA for fastest access to latest data updates and speed up of the analytic discovery and model development. However, once a champion predictive model has been built and chosen, and shall be applied

to production data or as part operational and SAP HANA-based application an even deeper integration approach has to be taken. Instead of pushing back the production data to the environment of the predictive model, it is best performed the other way around, enabling the SAS predictive model to be applied **inside** the SAP HANA environment.

## SAS PREDICTIVE MODELING INTEGRATION WITH SAP HANA

To facilitate this new level of integration with your SAP HANA environment, a new SAS bundle is under development to deliver SAS Enterprise Miner, SAS Model Manager, and SAS Scoring Accelerator specifically targeted for SAP HANA. This suite will allow you to get the most value out of your SAP HANA investment – leveraging the strengths of the SAP HANA infrastructure to provide optimal performance.

Coupled with the data access capabilities outlined above, you can also develop SAS predictive models and deploy them to run inside a SAP HANA. The SAS Predictive Modeling applications provide acknowledged capabilities to build models in a governed and self-documenting manner, while providing the ability to monitor the results over time. The SAP HANA platform, with its in-memory architecture, provides the infrastructure to support large volumes of data designed for analytics.

Organizations can now leverage a best of breed solution comprised of the robustness and flexibility of SAS analytics and the inherent scalability of the SAP HANA shared-nothing architecture. You can get started today with the SAS/ACCESS for SAP HANA and prepare for the new bundle, which should be available later in 2014.

---

## DISCLAIMER

The contents of this paper are the personal work and view of the author. All details and recommendations in this document have been compiled on a best efforts basis for informational purposes only, without representation or warranty of any kind. The content is considered to be accurate, however neither the author nor SAP AG shall be liable for errors or omissions with respect to the material.

## REFERENCES and RECOMMENDED READING


Plemmons, Howard (2009): Top Ten SAS® DBMS Performance Boosters for 2009, *Proceedings of the SAS® Global Forum 2009, available at* http://support.sas.com/resources/papers/proceedings09/309-2009.pdf

SAP AG (2012): “SAP HANA® Database for Next-Generation Business Applications and Real-Time Analytics” Available at http://www.sap.com/bin/sapcom/downloadasset.sap-hana-for-next-generation-business-applications-and-real-time-analytics-pdf.html

SAS Institute Inc. (2013): “SAS/ACCESS® 9.4 for Relational Databases: Reference, Fourth Edition”, SAS/ACCESS Interface to SAP HANA. Available at http://support.sas.com/documentation/


## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:


| | |
|---|---|
| Name: | Christoph Morgen |
| Enterprise: | SAP AG |
| Address: | Dietmar-Hopp-Allee 16 |
| ZIP City: | 69190 Walldorf, Germany |
| Work Phone: | +49 6227 742554 |
| E-mail: | Christoph.Morgen@sap.com |
| Linkedin: | http://www.linkedin.com/pub/christoph-morgen/17/57b/543 |