Integrating MicroStrategy 9.3.1 with SAP HANA

This document provides an overview of SAP HANA 1.0 SP5 and how MicroStrategy integrates with SAP HANA. To see integration of older SAP HANA service packs, refer to TN42052: Integrating MicroStrategy 9.3.0 with SAP HANA
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>About MicroStrategy</td>
<td>4</td>
</tr>
<tr>
<td>About SAP HANA</td>
<td>4</td>
</tr>
<tr>
<td>SAP HANA and MicroStrategy Certification</td>
<td>5</td>
</tr>
<tr>
<td>MicroStrategy Certification Status</td>
<td>5</td>
</tr>
<tr>
<td>Protocols and Connectivity</td>
<td>5</td>
</tr>
<tr>
<td>How to connect to SAP HANA from MicroStrategy</td>
<td>5</td>
</tr>
<tr>
<td>How does MicroStrategy connect to SAP NetWeaver BW</td>
<td>7</td>
</tr>
<tr>
<td>Standalone SAP HANA</td>
<td>7</td>
</tr>
<tr>
<td>SAP HANA for SAP Netweaver BW</td>
<td>7</td>
</tr>
<tr>
<td>About SAP HANA for SAP Netweaver BW</td>
<td>8</td>
</tr>
<tr>
<td>Integrating with SAP HANA Database Objects</td>
<td>8</td>
</tr>
<tr>
<td>Tables</td>
<td>9</td>
</tr>
<tr>
<td>True Temporary Table</td>
<td>10</td>
</tr>
<tr>
<td>MicroStrategy Warehouse Catalog: Available Tables</td>
<td>10</td>
</tr>
<tr>
<td>Information Views</td>
<td>11</td>
</tr>
<tr>
<td>Attribute Views</td>
<td>11</td>
</tr>
<tr>
<td>Analytical Views</td>
<td>12</td>
</tr>
<tr>
<td>Calculation Views</td>
<td>13</td>
</tr>
<tr>
<td>Information Views Naming Convention</td>
<td>15</td>
</tr>
<tr>
<td>How to import Information Views</td>
<td>16</td>
</tr>
<tr>
<td>Input Parameters and Variables</td>
<td>18</td>
</tr>
<tr>
<td>Information Views Building Blocks</td>
<td>23</td>
</tr>
<tr>
<td>Stored Procedures</td>
<td>23</td>
</tr>
<tr>
<td>Hierarchies</td>
<td>23</td>
</tr>
<tr>
<td>Security</td>
<td>23</td>
</tr>
<tr>
<td>User Management</td>
<td>23</td>
</tr>
<tr>
<td>Integration into Single Sign-On Environments</td>
<td>24</td>
</tr>
<tr>
<td>Using Kerberos Authentication with MicroStrategy and SAP HANA</td>
<td>24</td>
</tr>
<tr>
<td>VLDB Reference</td>
<td>25</td>
</tr>
<tr>
<td>Overview of MicroStrategy Architecture</td>
<td>25</td>
</tr>
</tbody>
</table>
Introduction
This paper provides a brief overview of the MicroStrategy architecture and explains how this architecture takes advantage of the technology and business intelligence functionality of SAP HANA. MicroStrategy offers an unsurpassed range of analytical functionality, which is further optimized when integrated with SAP HANA’s broad list of database features that improve data warehousing performance.

About MicroStrategy
MicroStrategy is a full featured BI platform, delivering all 5 styles of BI in a fully integrated architecture. The MicroStrategy platform provides VLDB drivers for all supported RDBMS platforms to generate optimized SQL that takes advantage of database specific functionality. The full set of VLDB properties is documented in the MicroStrategy System Administration Guide. Settings that are most relevant to SAP HANA 1.0 are discussed below.

About SAP HANA
SAP HANA is a game-changing, real-time platform for analytics and applications. While simplifying the IT stack, it provides powerful features like: significant processing speed, the ability to handle big data, predictive capabilities and text mining capabilities.

SAP HANA stands for High Performance Analytical Appliance. SAP HANA is an in-memory and columnar or row-based database management system (DBMS) at heart. It comes in the form of an appliance (only SAP hardware partners can configure SAP HANA software), and its architecture is based on Massive Parallel Processing (MPP) capabilities to perform queries and calculations against compressed data resident in RAM rather than on disk, reducing almost all overhead related to reading/writing data onto hard drives.

Key SAP HANA features:

- **Available as Appliance**: software licensing + SAP partner hardware
- **Massively Parallel Processing (MPP) architecture**: data is partitioned within and across nodes (for parallelization of queries)
- **Columnar and row-based storage**: optimized for both write and read operations (row, column or hybrid stores)
- **In-memory database**: OLAP and calculation operations done in-memory (no disk I/O, except for logging and backup)
- **Data compression**: up to 8x compression (in memory); RLE helps do quick aggregations...
- **Support of application-specific business objects** (e.g. OLAP cubes) and logic (domain-specific function libraries like R) **directly in the database engine**.
SAP HANA and MicroStrategy Certification

MicroStrategy Certification Status
SAP HANA 1.0 SP2 – SP4 is certified as warehouse platform with the MicroStrategy 9.3.0, 9.3.1 and 9.4 releases.

SAP HANA 1.0 SP5 is certified as warehouse platform with MicroStrategy 9.3.1 and 9.4

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Recommended ODBC Driver</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 64-bit</td>
<td>32-bit HDBODC32 Driver 1.00.56.49638</td>
<td>Certified: WH</td>
</tr>
<tr>
<td>Linux</td>
<td>32-bit HDBODC32 Driver 1.00.56.49638</td>
<td>Certified: WH</td>
</tr>
</tbody>
</table>

This certification is carried out by MicroStrategy, to understand how MicroStrategy architecture takes advantage of the technology advances and business intelligence functionality of SAP HANA.

Starting 9.3.1, SAP certifies MicroStrategy as a supported analytical platform.

Protocols and Connectivity

How to connect to SAP HANA from MicroStrategy
SAP HANA supports both the SQL and MDX query languages. MicroStrategy only supports connecting to SAP HANA through ODBC. MicroStrategy uses SAP HANA 32-bit ODBC drivers to connect.

Starting from MicroStrategy 9.3, database connectivity object has been added. After creating a DSN to SAP HANA, go to Desktop and in the Database Instance, you will see SAP HANA VLDB object.

In MicroStrategy release 9.4, both the VLDB objects are automatically added in the list of connectors.
Once you have the Database Instance created, you can load all the tables through the Warehouse Catalog.

For information on how to connect to SAP HANA through Linux, please refer to **TN40923**: **How to configure the SAP HANA 1.0 SP2 ODBC driver for connectivity to a SAP HANA database for the MicroStrategy Intelligence Server 9.3.0 on Linux.**

SAP HANA SP5 is supported starting 9.3.1. The new *SAP HANA 1.x SP5 VLDB* object supports new functions that have been added to SAP HANA SP5. Contact MicroStrategy Technical Support to obtain the new PDS file.

Starting
How does MicroStrategy connect to SAP NetWeaver BW

Based on available literature on SAP HANA, there are three ways of taking advantage of SAP BW infocubes or DSOs in SAP HANA:

Standalone SAP HANA

Create an ETL process (using tools like SAP Landscape Transformation or SAP Data Services) to move data from SAP BW to SAP HANA. This is apparently a way to work with SAP BW when using the standalone version of SAP HANA.

There are considerations for this approach:
- MicroStrategy can use SAP HANA as any other relational database, removing all limitations that are present when using the BAPI/MDX interface.
- A new ETL process is required, including initial transfer and incremental data refreshes.

SAP HANA for SAP Netweaver BW

Optimized SAP BW

Use SAP HANA for SAP Netweaver BW as the underlying database on which SAP Netweaver BW runs. In this case, all existent infocubes have the option to be optimized for SAP HANA (infocube underlying tables are flattened, removing the dimension tables and using the master data directly). Under this configuration, SAP BW continues being the data provider, with a SAP HANA database backend; MicroStrategy would connect to SAP BW through the Java Connector and the BAPI interfaces and issue MDX against SAP BW (not SAP HANA).

A few considerations of this scenario:
- Some of the performance improvements offered by SAP HANA can be assimilated immediately without significant changes to the current reporting models in SAP BW (Infocubes and DSOs would need to explicitly be optimized for SAP HANA usage). Additional infocube flattening/modeling is required to maximize performance improvements.
- As mentioned before, the reporting logic still relies in SAP BW; the SAP OLAP BAPI is still the interface to retrieve/query data from the MicroStrategy standpoint, implying some of the limitations inherent of this interface (including performance, scalability and functional limitations when used within MicroStrategy).
- Finally, there is also a minimum version requirement for SAP BW to integrate with SAP HANA.

SAP BW InfoObjects as Information Views

As of SAP HANA for SAP BW SPS 5, SAP BW infocubes and DSOs optimized for SAP HANA can be exposed as information views. MicroStrategy, in this case, would use the usual ODBC connectivity (SQL-based) to SAP HANA in order to consume all SAP BW models as SAP HANA models, leveraging both, SAP HANA optimized in-memory data retrieval and all MicroStrategy functional capabilities.

There are considerations for this scenario:
- Some of the performance improvements offered by SAP HANA can be assimilated immediately without significant changes to the current reporting models in SAP BW.
- Moreover, MicroStrategy could read these models in a relational way through SAP HANA’s ODBC driver and SQL; all OLAP BAPI interface-specific limitations are not present.
- This approach requires optimizing current SAP BW infocubes and DSOs for SAP HANA. There is a minimum revision requirement on SAP HANA (SPS5). There are also minimum version requirements for SAP BW to integrate with SAP HANA.
- While SAP BW authorizations with respect to data access are imported as SAP HANA analytic privileges, there may be additional administrative overhead to maintain these privileges in two system, BW and SAP HANA.

**About SAP HANA for SAP Netweaver BW**

SAP NetWeaver BW can be accelerated by using SAP HANA as its underlying database. In this scenario, SAP NetWeaver BW functions, such as those of SAP NetWeaver BW Accelerator software and new SAP HANA–optimized information providers (objects within a database that act as data providers within a query definition), are performed within SAP HANA to benefit from its in-memory and calculation engine functionality.

SAP BW integrates extraction of data from SAP sources, data management, and OLAP analysis, and provides predefined Business Content, in the form of packaged extractors, multidimensional models (InfoCubes) and reports.

However, by connecting directly to SAP HANA, users can take advantage of additional features that are not available when connecting through MDX sources.

- Users can use the metric editor to customize metrics, there is no Metric Editor for MDX Metrics.
- When connecting directly to SAP HANA, users can create standalone filters, standalone templates and security filters.
- Users are not limited to creating prompts only in Desktop. Prompts cannot be created in Web against MDX sources.
- When directly connecting through SAP HANA, one can modify attribute expression. When connecting through SAP BW, users must use expressions from MDX cube.
- Users can create custom drill maps.
- Users can create entire reports using Freeform MDX.
- When connecting directly to SAP HANA, users are not limited to drilling within the cube. Users can also drill to template.

**Integrating with SAP HANA Database Objects**

SAP HANA is a relational database management system providing many data structures that are commonly used in other relational database management systems, such as schemas, tables, procedures, views, etc.

Additionally SAP HANA offers data structures supporting OLAP analysis at the database engine level, including business logic/modeling, such as Attributes, Measures, Hierarchies, Calculated columns, Information Views, etc.
MicroStrategy implements relational OLAP (ROLAP) providing a logical data model in its metadata that map to relational data structures. To take advantage of the OLAP data structures offered by SAP HANA MicroStrategy performs a mapping of SAP HANA objects to objects in the MicroStrategy metadata.

Many of the SAP HANA OLAP data structures are exposed as **table structures through the ODBC interface**, making them suitable for ROLAP analysis. See Appendix 2: SAP HANA Architecture and Modeling Basics for more information on SAP HANA’s architecture.

Below is the summary of how each SAP HANA object is modeled in MicroStrategy:

<table>
<thead>
<tr>
<th>SAP HANA Object</th>
<th>MicroStrategy Object</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute View</td>
<td>Table object</td>
<td></td>
</tr>
<tr>
<td>Analytical View</td>
<td>Table object</td>
<td>Table-level setting automatically set: “Trust Table Key” is unchecked to enable the use of “group by” on queries against Analytical views.</td>
</tr>
<tr>
<td>Calculation View</td>
<td>Table object</td>
<td></td>
</tr>
<tr>
<td>Table (row- or column-based)</td>
<td>Table object</td>
<td></td>
</tr>
<tr>
<td>Schema (similar to the “database” concept of SQL Server)</td>
<td>Table Prefix</td>
<td>Tables present in multiple schemas have a different prefix.</td>
</tr>
<tr>
<td>View</td>
<td>Table Object</td>
<td></td>
</tr>
</tbody>
</table>

For the specific features within the definition of the different information views, particularly, these are modeled in MicroStrategy in the following way:

<table>
<thead>
<tr>
<th>SAP HANA modeling elements</th>
<th>MicroStrategy Object</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Column</td>
<td></td>
</tr>
<tr>
<td>Measures</td>
<td>Column</td>
<td></td>
</tr>
<tr>
<td>Calculated Columns</td>
<td>Column</td>
<td></td>
</tr>
<tr>
<td>Hierarchies</td>
<td>n/a</td>
<td>Currently not supported</td>
</tr>
<tr>
<td>Variables</td>
<td>n/a</td>
<td>Currently not supported</td>
</tr>
<tr>
<td>Input Parameters</td>
<td>Table-level prompts</td>
<td>Allow entering of parameter values at run time</td>
</tr>
</tbody>
</table>

In the following sections will discuss the detailed integration of SAP HANA objects.

**Tables**

SAP HANA offers two storage options for tables, columnar and row-based, that are optimized for different use cases. SAP recommends adhering to the following modeling principles in SAP HANA:

- Use columnar tables for analytical applications (including SAP BW, MicroStrategy)
- Use row-based tables for transactional applications (like SAP ERP)

The choice of storage option is transparent to MicroStrategy and both are fully supported. MicroStrategy recommends the use of columnar table structures for fact and lookup tables for optimal performance.
MicroStrategy differentiates two sets of tables as part of its analytical workload. The first are made up of a permanent set of base tables that are created and populated by a DBA. These generally consist of fact and lookup tables as required to implement the desired logical data model for the analytical application.

The second is a set of intermediate tables that is generated by MicroStrategy's Multi-pass SQL and is required to answer analytical questions that cannot be answered with a single SQL query block. MicroStrategy offers significant flexibility in how these intermediate tables are defined to optimally leverage all features of the RDBMS.

**True Temporary Table**
MicroStrategy uses the True temporary table feature as the default option to implement multi-pass SQL in SAP HANA.

By default, MicroStrategy creates explicit True temporary tables to implement multi-pass SQL. SAP HANA stores all the data in memory, so there is low overhead cost for creating true temporary table and provides the best level of performance. In addition, temporary tables do not incur logging to the transaction log when the NO LOG option is used. True temporary table can cover some scenarios that derived tables cannot. The details will be covered in VLDB section under Derived Table’s.

**MicroStrategy Warehouse Catalog: Available Tables**
By default, the Warehouse Catalog SQL in MicroStrategy Desktop returns regular tables present ONLY in the logged-in user’s schema (schema is the same as user name) and all SAP HANA information views. These are the “tables” that can be added to the set of MicroStrategy project tables. Nonetheless, there are a couple of scenarios when the default catalog SQL can be modified to show or hide some of the default tables shown in the MicroStrategy Warehouse Catalog:

1. Show tables in schemas that are different from user’s default schema
2. Hide “auxiliary” views created for each information view

The default Catalog SQL can be seen and modified from MicroStrategy Desktop (Desktop > Schema > Warehouse Catalog > Tools > Options > Catalog-Read Settings > Settings...):

```
SELECT DISTINCT SCHEMA_NAME as NAME_SPACE, TABLE_NAME as TAB_NAME,
    TABLE_TYPE as TAB_TYPE
FROM SYSTABLES
WHERE SCHEMA_NAME = '#LOGIN_NAME#'
UNION
    SELECT DISTINCT SCHEMA_NAME as NAME_SPACE, VIEW_NAME as TAB_NAME,
    VIEW_TYPE as TAB_TYPE
FROM SYS.VIEWS
WHERE SCHEMA_NAME = '#LOGIN_NAME#' OR SCHEMA_NAME = '_SYS_BIC'
```

**Show Tables in Additional Schemas**
By default, the warehouse catalog does not show all tables in multiple schemas because these would be treated as the same object if they are named equally. If two schemas in SAP HANA have two equally
named tables, MicroStrategy would allow adding both to the set of project tables, but the equally named tables would technically be the same logical table. The table would have only one prefix.

Base tables in SAP HANA in multiple schemas need to have distinct names to be used in MicroStrategy.

To avoid the above situation, we do not show by default tables present in other database schemas (schema being the equivalent to a database in SQL Server or a user in Oracle, generally speaking). Nonetheless, when tables in specific schemas need to be added directly to the MicroStrategy project tables, the catalog SQL can be modified. In the example below, the default Catalog SQL was modified to be able to read tables in the “MICROSTRATEGY_TUTORIAL” schema as well (highlighted text):

```
SELECT DISTINCT SCHEMA_NAME as NAME_SPACE, TABLE_NAME as TAB_NAME, TABLE_TYPE as TAB_TYPE
FROM SYS.TABLES
WHERE SCHEMA_NAME IN ('#LOGIN_NAME#', 'MICROSTRATEGY_TUTORIAL')
UNION
SELECT DISTINCT SCHEMA_NAME as NAME_SPACE, VIEW_NAME as TAB_NAME, VIEW_TYPE as TAB_TYPE
FROM SYS.VIEWS
WHERE (SCHEMA_NAME = '#LOGIN_NAME#' OR SCHEMA_NAME = '_SYS_BIC') AND VIEW_TYPE NOT IN ('HIERARCHY')
```

**Information Views**

Information views are the objects that allow “business logic modeling.” Tables are used to build “abstraction layers”, or information views. Traditional database views expose a SQL query (a set of columns from a set of joined tables) as a single table. Information views are similar to these traditional database views in how they work. No data materialization is done (meaning, no new data copy/aggregation is necessarily done). Moreover, instead of just being a SQL query exposed as a table, information views extend the traditional database view concept, allowing the definition of more complex joins of many tables or other information views, and the modeling of business logic.

MicroStrategy does not see any difference between physical tables and views when retrieve from the MicroStrategy Warehouse Catalog.

There are three types of Information views (for more details, refer to the SAP HANA development guide):

**Attribute Views**

Attribute views are used to model an entity based on the relationships between attribute data contained in multiple source tables. For instance, a customer dimension view could be created from the join of the LU_CUSTOMER, LU_CUST_CITY, LU_CUST_GENDER, LU_CUST_EDUCATION, et... tables. These are exposed, nonetheless, as one single view with all customer dimension columns.
Sample geography attribute view: expose the join of employee, region, call center, etc. lookup tables as one unified view.

Best Practices around using Attribute views

- If more joins are defined in an attribute view, performance could be ‘impacted’ (which might be negligible or not depending on the model, the amount of data, or the actual number of tables that need to be joined to fulfill a specific query). For more info, refer to the SAP HANA development guide
- Create Attribute views based on the business need. For example, if a material entity requires material type and other type of information for most business domains, it makes sense to create a holistic abstraction model of material that includes material and other material information lookup ‘tables’.

Analytical Views

Analytical views provide access to data in a logical structure that supports OLAP analysis. They are used for analytical purposes where reading operations in mass data is required. According to SAP analytical views have a very high performance on SELECT and are well optimized.

In general, analytic views are used to model data that includes measures (or facts). An analytical view definition could consist, for instance, of a central fact table, ORDER_HEADER, joined with the customer dimension attribute view, other lookup tables (such as LU_EMPLOYEE) or any additional attribute views. The end result is a view that is exposed as a “de-normalized” ORDER_HEADER table with all the dimensional information attached to the fact table. This is almost equivalent to a cube (and therefore the OLAP analysis reference for analytical views) that has all dimensional and factual information; to get relevant information around the central fact table, joins with already considered lookup tables are not necessary in the SQL, as these are factored in the analytical view definition already.
Sample order header analytical view: expose the join of ORDER_HEADER fact table with other attribute views as one unified view.

**Best Practices around using Analytical views**

- Analytical views are the best in terms of performance of all modeling views to aggregate fact data. These may be potentially the most commonly used modeling views by customers (to aggregate metric data, as opposed to calculation view).

**Calculation Views**

Calculation Views are used to provide composites of other views. They can be used in the same way as analytic views. In general, calculation views allow modeling more complex calculations that involve some kind of data processing (whether joining, uniting or even doing statistical analysis) on one or several data structures (tables or other information views). Calculation views can have layers of calculation logic, can include measures sourced from multiple source tables and can include advanced SQL logic. The data foundation of the calculation view can include any combination of tables, column views, attribute views and analytic views. It is possible to create joins, unions, projections, and aggregation levels on the sources.

There are several ways to create calculation views. The screenshot below shows a graphical way of creating calculation views. Additionally, these may be defined using SQLScript, SAP HANA’s own SQL extension used also to create stored procedures, or other languages (e.g. integration with R).

Overall, calculation view are superior to other views in the sense that since calculation views are built on top of OLAP engine and/or join engine, calculation views can achieve complex calculations that cannot be achieved using the Attribute Views and Analytics Views. This also makes them slightly less performant.
Sample calculation view: create a view with metric data for current year and last year. Take one analytical view; filter data (projection) for current year. Do the same for last year’s data. Join (join or union) the data of the two filtered/projected structures into one single structure, and expose current and last year as ‘CY’ or ‘LY’ values, respectively, in a new calculated column. The end result is a “denormalized” view of the fact data containing all dimensional data. Instead of having a column with years 2010 (the current year), 2009 (last year) and all other years (2008, 2007, etc), it is a filtered view of only the current and last year; it has a column with ‘CY’ and ‘LY’, representing 2009 and 2010.

**Best Practices around using Calculation views**

- Use Calculation views to do advance processing of data: joining/uniting multiple facts/analytical views, incorporating external data processing (R scripts for example), incorporate stored procedures for data processing (use analytics libraries, e.g.), etc.
- **Calculation views are more costly in terms of performance due to additional processing by the calculation engine;** moreover, data travels from the underlying views to the different processing steps in the calculation engine. Overall, calculation engine in SAP HANA adds overhead (which might be negligible or not depending on the model and amount of data).
- Design calculation views by enabling the transfer of the least amount of data between the calculation nodes (for example, do first a filtered projection before doing a union).
- When designing calculation views, use UNION operation over JOIN. Join is more costly in terms of performance.

Below is a chart on usage, pros and cons of each view type from the SAP HANA documentation [3]
Information Views Naming Convention

Information views (attribute, analytical and calculation views) are modeled and organized within packages, similar to how classes in Java are organized in packages. This allows SAP HANA to define permissions on who can modify/create/delete content objects (like information views) on a per-package basis; object/configuration management within the SAP HANA environment is also done based on packages. When the views are “Activated” for consumption, these are published or made available in the _SYS_BIC schema with a name consisting of the package where the information view was defined along with its name.

For example, the attribute view “AV_CUSTOMER” was defined in the package “mstr.tutorial” ...
When the attribute view is activated/published, it appears under the _SYS_BIC schema as `mstr.tutorial/AV_CUSTOMER`. This is the actual table name MicroStrategy reads:

How tables are seen from SAP HANA Studio

Views seen from MicroStrategy

**How to import Information Views**

Even though MicroStrategy treats views no different than tables, tables and views appear differently in MicroStrategy. Views in MicroStrategy appear with the name of their package and tables with just the name of the tables. You can see the difference below in the way views and tables appear in MicroStrategy.
Along with importing the views in the MicroStrategy Warehouse Catalog, you also need to import the prefix _SYS_BIC so that the MicroStrategy SQL engine can generate proper SQL against the views. To import the prefix, right click on the view in the warehouse catalog and select Import Prefix.

It is possible to configure the Warehouse Catalog to automatically import table prefixes (In the Warehouse Catalog, choose Tools > Options...):
Activating this option seem to have an effect on the warehouse catalog loading time; therefore, consider using this option only if it is absolutely necessary.

Hide Auxiliary Information Views
When activating information views in SAP HANA, several additional auxiliary views are created in the system besides the main view that is queried. These auxiliary views contain, for example, information related to hierarchies. For reporting purposes, these do not provide much value and sometimes make it difficult to identify the actual information views on which reporting will be done. The catalog SQL can be modified to avoid displaying these “auxiliary” views from the Warehouse catalog as shown in this example (highlighted text):

```
SELECT DISTINCT SCHEMA_NAME as NAME_SPACE, TABLE_NAME as TAB_NAME, TABLE_TYPE as TAB_TYPE
FROM SYS.TABLES
WHERE SCHEMA_NAME = '#LOGIN_NAME#'
UNION
SELECT DISTINCT SCHEMA_NAME as NAME_SPACE, VIEW_NAME as TAB_NAME, VIEW_TYPE as TAB_TYPE
FROM SYS.VIEWS
WHERE (SCHEMA_NAME = '#LOGIN_NAME#' OR SCHEMA_NAME = '_SYS_BIC') AND VIEW_TYPE NOT IN ('HIERARCHY')
```

Input Parameters and Variables
SAP HANA has the capability to parameterize modeling views. In other words, in order to execute certain SQL on top of a modeling view, if the view has input parameters defined in it, these must be provided with an answer/value before a query can run. These are “prompts” defined at the view level. There are two types of “prompts:” Variables and Input Parameters.

**Variables** are used to filter data in a view based on a column when a query executes. The filtering, in SQL, translates to a condition in the WHERE clause. From HANA Studio, for example, running a query (data preview option) on a view with variables results in a prompt dialog; after providing answers, the qualification is included in the WHERE clause of the SQL statement. Variables are logically equivalent to prompted filters in MicroStrategy.

In other words, the variable values are passed to the HANA engine via the WHERE clause of the SQL statement. Variables are not known by the engine, only by the clients.

How Variables show up to users in HANA Studio (HANA’s client software):
How these variables are rendered in the HANA Studio generated SQL:

```
SELECT ...
FROM "_SYS_BIC"."mstr.tutorial/CL_VARIABLES_MLINE"
WHERE ("YEAR_ID" IN ('2010')) AND
  ("MONTH_ID" BETWEEN ('200901') and
   ('200911') ) AND ("DAY_DATE" > ('2009-01-08') )
GROUP BY ...
```

**Input Parameters** are used as a means to provide data used as input for internal calculations defined in the view. This data can be used to resolve calculated columns, for currency conversion (i.e. return data in EURO or USD), or for other operations. Input parameters, from HANA Studio, are rendered in the same way as variables when executing a query on a view with input parameters (using Data Preview option): a dialog appears so that users provide answers to prompts. In the SQL, these answers are included in special clauses in the FROM clause of the SELECT statement.

As opposed to variables, an input parameter defines an internal parameterization of the view. This means the SAP HANA engines need to know it and uses the parameter value during the execution (e.g. to calculate a formula for a calculated measure). The parameter value is passed to the engine via the PLACEHOLDER clause of the SQL statement.

How input parameters show up to users in SAP HANA Studio (HANA’s client software) when doing a data preview on a view with parameters:
How these input parameters are rendered in the SAP HANA Studio generated SQL:

```
SELECT ...
FROM "_SYS_BIC"."mstr.tutorial/CL_INPUT_PARAM"
  ('PLACEHOLDER' = ('$$INPUT_DATE_DAT$$', '2012-09-23'),
   'PLACEHOLDER' = ('$$INPUT_STATIC_LIST$$', 'true'),
   'PLACEHOLDER' = ('$$INPUT_ATTRIBUTE$$', '2010'),
   'PLACEHOLDER' = ('$$INPUT_BLANK_TEXT$$', 'hello'),
   'PLACEHOLDER' = ('$$INPUT_BLANK_INT$$', '10'),
   'PLACEHOLDER' = ('$$INPUT_DATE_NUM$$', '2012-09-17'))
GROUP BY ...
```

Summary: Input Parameters and Variables

<table>
<thead>
<tr>
<th>Variables</th>
<th>Input Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special SQL</td>
<td>No (WHERE clause)</td>
</tr>
<tr>
<td>HANA Engine Awareness</td>
<td>No</td>
</tr>
</tbody>
</table>

Sample use cases for input parameters are:

- Currency conversion. Several parts of the currency conversion can be parameterized: source and target currency, type of exchange rate and date of exchange rate.
- Creation of Transformation Metrics from SAP HANA. Instead of creating transformation metrics in MSTR, model these from SAP HANA through calculation views.

MicroStrategy 9.3.1 adds support for input parameters in information views.

In 9.3.0, MicroStrategy could make use of input parameters only through freeform SQL reports. Prompt placeholders can be put in the SQL in the input parameters clause.
Starting with MicroStrategy 9.3.1, input parameters are imported by MicroStrategy as table-level prompts for any information view that supports input parameters (Analytical and Calculation views).

By default, MicroStrategy creates value prompts for each imported input parameter (since parameters are answered by providing only one value, whether it is numeric, date, text, etc.). Input parameter properties, such as default answer value, whether it is mandatory or not, are imported as part of the prompt definition as well. Moreover, these value prompts created by default can be replaced with other types of prompts, such as attribute element prompts, to allow users to choose values instead of making them entering plain values.

When a report uses an attribute or metric modeled on top of the parameterized table (an information view with input parameters), these table-level prompts will appear as regular MicroStrategy prompts to end users. MicroStrategy will also generate the appropriate SQL constructs in order to pass the parameter values on to SAP HANA’s information views.

Among the enhancements made to the product to enable input parameter support are the following:

- Warehouse Catalog allows previewing the “Parameters…” defined for an input parameter.
- Table editor allows viewing, modifying and replacing default prompts for each of the input parameters defined for the table being edited; there is a new tab in the table editor called “Parameters.”
- SQL sent to SAP HANA when querying information views with parameters will add the special input parameter SQL constructs with the end user answers.
Variables as of now are not supported in MicroStrategy. An enhancement request has been logged to enable their support. Please contact MicroStrategy Technical Support for update on this enhancement.

**Information Views Building Blocks**

Attribute, analytical and calculation views all have the same basic elements in their definition; these elements, in SAP HANA’s terminology, are explained below.

**Attributes and measures**

In essence, attributes and measures represent individual columns. They form the content data that is used for modeling. Attributes represent the descriptive data like city and country; measures represent quantifiable data like revenue and quantity sold. Measures provide settings that enable currency or unit conversions, among other things in the SAP HANA model.

- **Calculated columns** (similar to compound metrics or formula-based attribute forms) allow defining new “derived” columns based on a formula, other columns or input parameters.
- **Calculated attributes/measures** are exactly the same as calculated columns. The difference relies on the view where they are created. Calculated attributes, as of SPS5, appear only in calculation views; the difference is that calculated attributes offer two input expressions: one for the ID of the attribute and another for the description. Calculated columns simply represent one additional column.

These can be exposed to MicroStrategy when created attributes or metrics during schema creation.

**Stored Procedures**

SAP HANA support stored procedures using a language called SQL Script. It also allows other languages like R. These procedures can be embedded in calculation views. Stored procedures have the benefit to manipulate and query database information along with the flexibility of programming to handle more complex tasks. These new stored procedures can then be integrated with MicroStrategy via pre/post statements in a report, called from freeform SQL reports, or anywhere else that custom SQL is allowed.

**Hierarchies**

They are not used in SQL/ODBC interfaces; these are primarily used when accessing SAP HANA from Excel with the HANA ODBO (OLE DB for OLAP) driver. They define relationships between columns in a view (e.g. ACCOUNTID column is child of PARENT_ACCOUNTID column, to represent recursive hierarchies in a table, for example). Currently Hierarchies in SAP HANA cannot be exposed through MicroStrategy.

**Security**

**User Management**

All users who want to access the SAP HANA database must be explicitly created in the database beforehand. A user who is connected using the external authentication provider but is not known to the database cannot access the database. Once a user has been set for the database in SAP HANA, this is the user MicroStrategy is going to use to connect through Database Instance.

When accessing the SAP HANA database using a client interface, so currently MicroStrategy will connect through ODBC, any access to data must be backed by corresponding privileges. Different schemes are
implemented. On a higher level, this concept provides authorization for the data contained in the
database when it is accessed using MicroStrategy. In the SAP HANA database system, the regular SQL
authorization concept is implemented.

For each SQL statement type (for example, SELECT, UPDATE, and CALL), a corresponding privilege exists
that the executing user needs to have. Additionally, objects in the database (such as tables, views, or
stored procedures) have an owner who can access the objects and grant privileges for them. In order to
access tables and views in SAP HANA from MicroStrategy, proper privileges should be provided. For
more information on what privileges should be granted please access SAP HANA Security Guide.

Integration into Single Sign-On Environments
SAP HANA supports the Kerberos protocol for single sign-on. It has been tested with Windows Active
Directory Domain Kerberos implementation and MIT Kerberos network authentication protocol. The
ODBC database client support Kerberos.

Using Kerberos Authentication with MicroStrategy and SAP HANA
Kerberos Authentication to SAP HANA through MicroStrategy has been post certified in 9.3.1 on
Windows. In order to setup Kerberos to be used with SAP HANA and MicroStrategy, you will need to do
the following:

1. Setup Kerberos (Integrated) authentication for the MicroStrategy Intelligence Server 9.x on
Microsoft Windows operating systems – TN19580.
2. Log into the client machine using the Kerberos user defined in SAP HANA.
3. Ensure the database login in the Database Instance is using Windows authentication.
4. To use database passthrough, make sure to select the Warehouse Authentication type to Kerberos in the Project Configuration. For more information on how to how does warehouse passthrough authentication and connection mapping work together when running reports/documents in MicroStrategy Desktop 9.x refer to TN38290.

**VLDB Reference**

**Overview of MicroStrategy Architecture**

**MicroStrategy Technology Philosophy: Leverage the Power of the RDBMS**

The MicroStrategy architecture has its roots in the principles of Relational OLAP (ROLAP). A ROLAP architecture provides OLAP functionality to the end user (e.g. multidimensional framework, slice-and-dice interaction, drilling, etc.), but uses a relational database to resolve queries and perform calculations, rather than using a specialized proprietary multidimensional database.

While the virtues of ROLAP are fully extolled elsewhere, this is an important distinction for this article for two reasons. First, MicroStrategy applications have been able to achieve industry-leading data scale because of the reliance on the processing power and data management capabilities of the underlying RDBMS. The system design principle of partnering with the RDBMS, through features discussed in this document, is the cornerstone of achieving such scalability. Second, this focus on integration with the RDBMS allows a business intelligence system to tap into a powerful dynamic over time: technical advances in RDBMS technology are seamlessly accessible to a MicroStrategy-based BI system. The net
benefit to the customer is a BI system that increases in value when either MicroStrategy or SAP HANA adds features to their products.

**Model-based Dynamic SQL Generation**
Within the MicroStrategy Intelligence Server lies the MicroStrategy SQL Engine, this is responsible for generating SQL for all requests to the data warehouse. Hence, to the RDBMS, a MicroStrategy application is an SQL-based application, in many ways like any other SQL application accessing SAP HANA.

**Schema Abstraction**
The SQL Engine performs its work based on a metadata model defined to the system. Note that the MicroStrategy metadata is not used to store joins or schema-type information, such as star or snowflake. Instead, the metadata model stores content information for each table indicating that it contains a set of particular facts and a set of particular attributes. When a report request is submitted, the Engine breaks the report down into the individual components (i.e. attributes and facts), then begins searching the model to determine which combination of tables will be necessary and efficient in resolving the request.

Schema abstraction of the database columns (into MicroStrategy attributes and facts) provides the flexibility necessary to allow applications to be created quickly without having to change the structure of the data model. MicroStrategy is able to support virtually any type of star, snowflake, or hybrid physical design, including transactional schemas. The business model defined in MicroStrategy is easily able to span multiple stars/snowflakes in a single application and even a single query. MicroStrategy supports dimensional models well, but does not require a dimensional model.

**Aggregate Awareness**
Query performance in many data warehouses is enhanced through the use of aggregate tables. Aggregate tables, which are also called summary tables, store pre-computed results of data allowing users to query from a summarized set of data rather than the detail level data that would be stored in the fact table. In many cases, use of aggregate tables will improve query performance by orders of magnitude.

MicroStrategy provides an aggregate-aware Engine. MicroStrategy has allowed transparent navigation of aggregate tables, directing queries to summary tables when they exist without the user having to specify to use the table. The MicroStrategy SQL engine determines use of aggregate tables transparently at query time.

**Multi-pass SQL**
One of the key elements to providing analytical sophistication in business intelligence applications is MicroStrategy’s ability to generate multi-pass SQL. Multi-pass SQL is required to answer analytical questions that cannot be answered with a single SQL query block. Examples of questions / scenarios that require multi-pass SQL include:
- Set qualification: “Show me sales by region over the last six months, but only for customers who purchased one of the 5 most popular products.”
- Split metrics: query returns sales data from a sales star schema and inventory data from an inventory star schema
- Metrics calculated at different levels of aggregation
- Metrics calculated with different filtering criteria
- Simulating outer joins on RDBMS platforms that do not support them natively
- Querying multiple tables due to application-level partitioning

Support for these scenarios, especially when combined together, provide a framework for significant analytic questions and value to the users of the system. One of the main optimizations the MicroStrategy SQL Engine makes is to generate SQL that performs these multi-pass queries as efficiently as possible.

**Default VLDB Settings for SAP HANA 1.0**

<table>
<thead>
<tr>
<th>VLDB Category</th>
<th>VLDB Property Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>Intermediate Table Type</td>
<td>True temporary table</td>
</tr>
<tr>
<td>Tables</td>
<td>Table Creation Type</td>
<td>explicit table</td>
</tr>
<tr>
<td>Indexing</td>
<td>Intermediate Table Index</td>
<td>Don’t create an index</td>
</tr>
<tr>
<td>Joins</td>
<td>Full Outer Join Support</td>
<td>Supported</td>
</tr>
<tr>
<td>Select/Insert</td>
<td>UNION multiple INSERT</td>
<td>Do NOT USE UNION</td>
</tr>
<tr>
<td>Select/Insert</td>
<td>Distinct / Group By Option</td>
<td>Use DISTINCT</td>
</tr>
<tr>
<td>Select/Insert</td>
<td>GROUP BY Non-ID Attribute</td>
<td>Use Group by</td>
</tr>
<tr>
<td>Query Optimizations</td>
<td>Sub Query Type</td>
<td>2: WHERE COL1 IN (SELECT s1.COL1) falling back to EXISTS ..</td>
</tr>
<tr>
<td>Query Optimizations</td>
<td>SQL Global Optimization</td>
<td>Level 4: Level 2 + Merge All Passes with Different Where</td>
</tr>
<tr>
<td>Query Optimizations</td>
<td>Set Operator Optimization</td>
<td>Enable set operator optimization</td>
</tr>
</tbody>
</table>

**Detailed Discussion of VLDB Settings**

<table>
<thead>
<tr>
<th>VLDB Category</th>
<th>VLDB Property Setting</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>Intermediate Table Type</td>
<td>True temporary table (default), Permanent table, Derived Tables, Temporary views</td>
</tr>
<tr>
<td>Tables</td>
<td>FallBack Table Type</td>
<td>Permanent table</td>
</tr>
<tr>
<td>Query Optimizations</td>
<td>Set Operator Optimization</td>
<td>Disable set operator optimization</td>
</tr>
</tbody>
</table>

**Intermediate Table Type**

**True temporary table**

One way to implement multi-pass SQL is to execute each pass (i.e. each query block) in a separate table. SAP HANA 1.0 SP2 stores all the data in memory, so there is low overhead cost for creating true temporary table and provides the best level of performance. In addition, temporary tables do not incur logging to the transaction log when the NO LOG option is used. Besides, True temporary table can cover some scenarios that derived tables cannot. The details will be covered in Derived Table’s section below.
CREATE LOCAL TEMPORARY TABLE #ZZMD00(
WJXBFS1 FLOAT) NO LOGGING

insert into #ZZMD00
select sum(a11.TOT_DOLLAR_SALES)  WJXBFS1
from "YR_CATEGORY_SLS" a11

select a12.CATEGORY_ID CATEGORY_ID,
a12.CATEGORY_DESC CATEGORY_DESC,
pa11.WJXBFS1  WJXBFS1
from #ZZMD00 pa11
cross join "LU_CATEGORY" a12

drop table #ZZMD00

**Derived table**

Rather than implement each pass in a separate table, Derived Table syntax allows the SQL Engine to
issue additional passes as query blocks in the FROM clause. Instead of issuing multiple SQL passes that
create intermediate tables, the SQL engine generates a single large pass of SQL. This can allow queries to
run faster since there is no CREATE TABLE or DROP TABLE statements to catalog, no corresponding locks
on the system tables, and no logging of records inserted into a physical table.

In theory, derived table syntax should result in query performance at least as good as when using
temporary tables. Customers may want to experiment with this option to determine if it is beneficial in
their specific environment.

```sql
select
a12.CATEGORY_ID CATEGORY_ID,
a12.CATEGORY_DESC CATEGORY_DESC,
pa11.WJXBFS1  WJXBFS1
from
(select
sum(a11.TOT_DOLLAR_SALES)  WJXBFS1
from "EAT_WH2"."YR_CATEGORY_SLS"
a11
)
pa11
cross join "EAT_WH2"."LUCATEGORY"a12
```

Note that not all reports are able to use derived tables. There are two primary scenarios in which
temporary tables must be used instead of derived tables:

- When a report uses a function supported in the MicroStrategy analytical engine that is not
  supported in SAP HANA 1.0 SP2 (e.g. many of the functions in the financial and statistical
  function packages). If these functions are used in intermediate calculations, the MicroStrategy
  analytical engine will perform calculations and then insert records back into the RDBMS for
  further processing. Inserting records back into SAP HANA requires a temporary table.

- When a report uses the MicroStrategy partitioning feature the SQL Engine uses a portion of the
  query in order to determine which partitions to use. The results are then used to construct the
  rest of the query. Because the full structure of the query is not known prior to execution, the
  SQL engine must use temporary tables to execute the query in multiple steps. Unfortunately, this
  feature is not available for use with SAP HANA but may be available in a future MicroStrategy release.

These situations do not cover 100% of the cases in which temporary tables must be used. The rest of the
cases are relatively obscure combinations of VLDB settings, such as certain combinations of Sub Query
Type plus outer join settings on metrics plus non-aggregatable metrics.

If the Intermediate Table Type is set to “Derived tables,” then either True temporary table should be
specified as the “fallback” table type should the specific report requirements not support the use of
derived table expressions. For each report, the SQL Engine will follow an “all or nothing” policy in
determining whether to use derived table syntax. If the entire report cannot be resolved in a single statement with derived tables, the SQL Engine will automatically revert to the Fallback Table Type syntax and not use derived table syntax at all for the report.

When using derived tables, the UNION Multiple INSERT setting should be setting to “Use Union.” This allows the engine to use “UNION” statements within inline views instead of multiple INSERT INTO statements. This setting is relevant for reports that use partitioning and consolidations.

### Permanent table
If Intermediate Table Type is set to Permanent Table, MicroStrategy will generate a “plain vanilla” CREATE TABLE statement for intermediate tables. This can be useful because other VLDB settings such as Table Qualifier, Table Space, etc. can be used to customize this standard CREATE TABLE syntax. One of the advantages is that columnar table can be introduced by adding “column” in Table Qualifier.
SAP does support Permanent table as intermediate table but MicroStrategy does not recommend using as it does not provide any performance improvement.

```sql
create column table ZZMD00
(  
  WJXBFS1 FLOAT
)

insert into ZZMD00
select  
  sum(a11.TOT_DOLLAR_SALES) WJXBFS1 
from  
  "EAT_WH2"."YRCATEGORY_SLS" a11

select  
  a12.CATEGORY_ID CATEGORY_ID,
  a12.CATEGORY_DESC CATEGORY_DESC,
  pa11.WJXBFS1 WJXBFS1
from  
  ZZMD00 pa11 
cross join"EAT_WH2"."LUCATEGORY"a12

drop table ZZMD00
```

### Temporary Views
If Intermediate Table Type is set to Temporary Views, MicroStrategy will generate a CREATE VIEW statement for intermediate tables.

SAP does support Temporary Views as intermediate table but MicroStrategy does not recommend using as it does not provide any performance improvement.

```sql
create view ZZMD00 (WJXBFS1) as
select  
  sum(a11.TOT_DOLLAR_SALES) WJXBFS1 
from  
  "EAT_WH2"."YRCATEGORY_SLS" a11

select  
  a12.CATEGORY_ID CATEGORY_ID,
  a12.CATEGORY_DESC CATEGORY_DESC,
  pa11.WJXBFS1 WJXBFS1
from  
  ZZMD00 pa11 
cross join"EAT_WH2"."LUCATEGORY"a12

drop view ZZMD00
```

Note that not all reports can use temporary views; the same restrictions apply as with derived tables. In the event that the SQL engine cannot construct the query using view SQL, it will fall back to the
Permanent table option as indicated in the Fallback Table. For each report, the SQL Engine will follow an “all or nothing” policy in determining whether to use view table syntax. If the entire report cannot be using temporary views, the SQL Engine will automatically revert to the Fallback Table Type syntax. When using temporary views, the UNION Multiple INSERT VLDB setting can be configured to “Use Union.” This allows the engine to use UNION statements when creating views instead of multiple CREATE VIEW statements. This setting is relevant for reports that use partitioning and consolidations.

**Table Creation Type**

When creating intermediate tables in SAP HANA 1.0 SP2, it is better to use “explicit” table creation. Though “implicit” table creation syntax is supported by SAP HANA there seems to be a limitation when one temp table is created based off the results of another. The error message corresponds to primary keys even though none of the tables use primary keys: “Cannot create temporary table with primary key”. This seems to be a defect with SAP HANA.

**Explicit table syntax**

```sql
create table ZZSP01(
    YEAR_ID INTEGER,
    REGION_NBR INTEGER,
    WJXBFS1 DATE
);
insert into ZZSP01 ...
```

**Implicit table syntax**

```sql
create table ZZSP01 as {
    select ...
}
```

**Full Outer Join**

SAP HANA 1.0 SP2 supports FULL OUTER JOIN syntax between tables in the FROM clause. By default, MicroStrategy will use FULL OUTER JOIN when needed because in SAP HANA, queries that perform full outer joins between several tables perform better than when the Full Outer Join VLDB setting is set to No Support where the SQL Engine will generate SQL that simulates full outer joins by using UNION and left outer joins.

**Full Outer Join = Support**

```sql
select coalesce(pa11.CUSTOMER_ID, pa12.CUSTOMER_ID) CUSTOMER_ID,
    a13.CUST_LAST_NAME CUST_LAST_NAME,
    a13.CUST_FIRST_NAME CUST_FIRST_NAME,
    coalesce(pa11.MONTH_ID, pa12.MONTH_ID) MONTH_ID,
    a14.MONTH_DESC MONTH_DESC,
    pa11.WJXBFS1 WJXBFS1,
    pa12.WJXBFS1 WJXBFS2
from ZZMD00 pa11
full outer join ZZMD01 pa12
    on (pa11.CUSTOMER_ID = pa12.CUSTOMER_ID and pa11.MONTH_ID = pa12.MONTH_ID)
    join LU_CUSTOMER a13
    on (coalesce(pa11.CUSTOMER_ID, pa12.CUSTOMER_ID) = a13.CUSTOMER_ID)
    join LU_MONTH a14
    on (coalesce(pa11.MONTH_ID, pa12.MONTH_ID) = a14.MONTH_ID)) pa11
full outer join LU_YEAR a12
```
Full Outer Join = No Support

cREATE TABLE ZZOJ02 AS ( 
  SELECT pa11.MONTH_ID MONTH_ID, 
         pa11.CUSTOMER_ID CUSTOMER_ID 
  FROM ZZMD00 pa11) NO LOGGING;

INSERT INTO ZZOJ02 
SELECT pa11.MONTH_ID MONTH_ID, 
       pa11.CUSTOMER_ID CUSTOMER_ID 
FROM ZZMD01 pa11;

CREATE TABLE ZZOD03 AS ( 
  SELECT DISTINCT pa11.MONTH_ID MONTH_ID, 
             pa11.CUSTOMER_ID CUSTOMER_ID 
  FROM ZZOJ02 pa11) NO LOGGING;

SELECT pa11.CUSTOMER_ID CUSTOMER_ID, 
       a14.CUST_LAST_NAME CUST_LAST_NAME, 
       a14.CUST_FIRST_NAME CUST_FIRST_NAME, 
       pa11.MONTH_ID MONTH_ID, 
       a15.MONTH_DESC MONTH_DESC, 
       pa12.WJXBFS1 WJXBFS1, 
       pa13.WJXBFS1 WJXBFS2 
FROM ZZOD03 pa11 
LEFT OUTER JOIN ZZMD00 pa12 
  ON (pa11.CUSTOMER_ID = pa12.CUSTOMER_ID AND 
      pa11.MONTH_ID = pa12.MONTH_ID) 
LEFT OUTER JOIN ZZMD01 pa13 
  ON (pa11.CUSTOMER_ID = pa13.CUSTOMER_ID AND 
      pa11.MONTH_ID = pa13.MONTH_ID) 
JOIN LU_CUSTOMER a14 
  ON (pa11.CUSTOMER_ID = a14.CUSTOMER_ID) 
JOIN LU_MONTH a15 
  ON (pa11.MONTH_ID = a15.MONTH_ID)

Sub Query Type

There are many cases in which the SQL Engine will generate subqueries (i.e. query blocks in the WHERE clause):

- Reports that use Relationship Filters
- Reports that use “NOT IN” set qualification, e.g. AND NOT <metric_qualification> or AND NOT <relationship_filter>
- Reports that use Attribute qualification with M-M relationships, e.g. show Revenue by Category, filter on Catalog
- Reports that “raise the level” of a filter, e.g. dimensional metric at Region level, but qualify on Store
- Reports that use non-aggregatable metrics, e.g. inventory metrics
- Reports that use Dimensional extensions
- Reports that use Attribute to Attribute comparison in the filter
The default setting for Sub Query Type for SAP HANA 1.0 SP2 is Option 2—“Where col1 in (select s1.col1….) falling back to EXISTS (select * …) for multiple columns IN.” In most cases, the subquery expression will be generated with IN syntax.

```sql
select a11.REGION_ID, a13.REGION_NAME,
       sum(a11.TOT_DOLLAR_SALES) as WJXBFS1
from "STATE_SUBCATEG_REGION_SLS" a11
join "LU_SUBCATEG" a12
    on (a11.SUBCAT_ID = a12.SUBCAT_ID)
join "LU_REGION" a13
    on (a11.REGION_ID = a13.REGION_ID)
where (a12.CATEGORY_ID in (select ps21.CATEGORY_ID
                        from #ZZMQ01 ps21))
group by a11.REGION_ID,
        a13.REGION_NAME
```

If the subquery logically requires more than one column, then an expression using EXISTS syntax will be generated.

```sql
select a11.REGION_ID, a13.REGION_NAME,
       sum(a11.TOT_DOLLAR_SALES) as WJXBFS1
from "STATE_SUBCATEG_REGION_SLS" a11
join "LU_SUBCATEG" a12
    on (a11.SUBCAT_ID = a12.SUBCAT_ID)
join "LU_REGION" a13
    on (a11.REGION_ID = a13.REGION_ID)
where (exists (select *
              from #ZZTI3D3KLMMQ001 ps21
              where ps21.CATEGORY_ID = a12.CATEGORY_ID))
group by a11.REGION_ID,
        a13.REGION_NAME
```

The other settings are not likely to be advantageous with SAP HANA 1.0.

- **Option 0**—“WHERE EXISTS (select * …)”  
  IN performs better than EXISTS. This setting is useful for RDBMS platforms that do not support any of the other syntax.
- **Option 1**—“WHERE EXISTS (select col1, col2 ...)”  
  IN performs better than EXISTS. This setting is useful for RDBMS platforms that do not support other syntax and for which selecting column names performs better than select *.
- **Option 3**—“Where (col1, col2) in (Select s1.col1, s1.col2...)”  
  SAP HANA supports multiple columns in an IN subquery, but the performance does not show much improvement.
- **Option 4**—“Use temporary table falling back to [Option 0] for correlated subqueries”  
  IN performs better than EXISTS for. Option 6 is better than Option 4 for SAP HANA.
• Option 5 – “WHERE col1 IN (select s1.col1), falling back to [Option 1] for multiple columns” EXISTS (select * ...) performs better than EXISTS (select col1, col2), so Option 2 is a better choice than Option 5.

**SQL Global Optimization**

This setting can substantially reduce the number of SQL passes generated by MicroStrategy. In MicroStrategy 9, SQL Global Optimization reduces the total number of SQL passes with the following optimizations:

- Eliminates unused SQL passes, e.g. a temp table is created but not referenced in a later pass
- Reuses redundant SQL passes
  - E.g. exact same temp table is created multiple times → single temp table is created
- Combines SQL passes where the SELECT list is different
  - E.g. two temp tables have same FROM clause, same JOINs, same WHERE clause, same GROUP BY → SELECT lists are combined into single SELECT statement
- Combines SQL passes where the WHERE clause is different
  - E.g. two temp tables have same SELECT list, same FROM clause, same JOINs, same GROUP BY → predicates from the WHERE clause are moved into CASE statements in the SELECT list.

See the System Administration Guide for a complete description of the cases covered by this setting.

The default setting for SAP HANA 1.0 SP2 is to enable SQL Global Optimization at the highest level.

**Set Operator Optimization**

This setting is used to combine multiple subqueries into a single subquery using set operators (i.e. UNION, INTERSECT, EXCEPT). The default setting for SAP HANA 1.0 SP2 is to enable Set Operator Optimization.

```sql
SELECT a13.CATEGORY_ID, CATEGORY_ID,
       max(a14.CATEGORY_DESC) CATEGORY_DESC,
       sum(a11.TOT_UNIT_SALES) WJXBFS1
FROM ITEM_EMP_SLS a11
JOIN LU_ITEM a12 ON (a11.ITEM_ID = a12.ITEM_ID)
JOIN LU_SUBCATEG a13 ON (a12.SUBCAT_ID = a13.SUBCAT_ID)
JOIN LU_CATEGORY a14 ON (a13.CATEGORY_ID = a14.CATEGORY_ID)
WHERE (a11.EMP_ID) IN ((SELECT r11.EMP_ID
                        FROM ITEM_EMP_SLS r11
                        WHERE r11.ITEM_ID = 37)
                        EXCEPT (SELECT r11.EMP_ID
                                FROM ITEM_EMP_SLS r11
                                WHERE r11.ITEM_ID = 217)))
GROUP BY a13.CATEGORY_ID
```
Additional VLDB Settings
Many of the VLDB properties control string syntax used in SQL queries generated by the MicroStrategy SQL engine. MicroStrategy application developers can further optimize SQL for their specific SAP HANA environment using these string insertion settings. Possible locations for VLDB optimizations in the query structure are listed below.

VLDB String Insertion Settings when using Temporary Tables

```
[Report Pre Statements]

[Table Pre Statements]
create table <table_name>
as
select  <column_expressions>
from    <tables_and_joins>
where   <filter_expressions>
group by <column_expressions>
having  <column_expressions> [Insert Post String]

create [Index Qualifier] index [Index Prefix]<index_name> on <table_name>(column_name) [Index Post String]

[Table Post Statements]
select  <column_expressions>
from    <tables_and_joins>
where   <filter_expressions>
group by <column_expressions> [Select Post String] [Select Statement Post String]

[Report Post Statements]
drop table <table_name>

[Cleanup Post Statements]
```

VLDB String Insertion Settings when using Permanent Tables
When Intermediate Table Type is set to Permanent Tables, some additional string-valued settings are enabled so that the user can customize the syntax of CREATE TABLE statement. The SQL below shows the position of these VLDB settings.

```
[Report Pre Statements]

[Table Pre Statements]
create [Table Qualifier] table [Table Descriptor][Table Prefix]<table_name> [Table Option]
[Table Space] as
select <column_expressions>
from   <tables_and_joins>
where  <filter_expressions>
group by <column_expressions>
having <column_expressions> [Insert Post String]

CREATE INDEX ON <table_name>
```
Parallel SQL Execution

Parallel SQL Execution (PSE) is the ability to run some passes of a multi-pass SQL in parallel with the idea of improving the overall report execution time. Starting with MicroStrategy v9.3.0, each report that is executed in sequential mode may record projected improvement in execution time if this report were to run in parallel mode. This information can be used to turn on PSE Mode. PSE is typically useful for reports that source from multiple databases as the report SQL passes tend to be distributed across sources and there isn’t an increased query load on a single database due to passes being run in parallel. In addition to multisource reports, PSE can be considered for any critical or high visibility reports or for scheduled reports/cubes that are executed during off-peak hours when there load on the database tends to be low. It is important to consider the load on the database system, before turning on PSE for a large number of reports.

PSE VLDB Settings:

<table>
<thead>
<tr>
<th>Level</th>
<th>Category</th>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project, Report</td>
<td>Query Optimizations</td>
<td>Parallel Query Execution</td>
<td>Disable (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable</td>
</tr>
<tr>
<td>Project, Report</td>
<td>Query Optimizations</td>
<td>Parallel Query Execution</td>
<td>Disable (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable for multiple data source reports only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable for all reports that support it</td>
</tr>
<tr>
<td>Project</td>
<td>Query Optimizations</td>
<td>Maximum Parallel Queries Per Report</td>
<td>2 (default)</td>
</tr>
</tbody>
</table>

Datatype Mapping

MicroStrategy supports the following SAP HANA data types:

<table>
<thead>
<tr>
<th>SAP HANA Datatype</th>
<th>MSTR Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA TYPE</td>
<td>MSTR TYPE</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
</tbody>
</table>
### SAP HANA Functions Used by MicroStrategy

MicroStrategy makes use of SAP HANA functions listed below through built-in functions.

<table>
<thead>
<tr>
<th>SAP FUNCTION</th>
<th>MICROSTRATEGY PATTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>ABS(#0)</td>
</tr>
<tr>
<td>ACOS</td>
<td>ACOS(#0)</td>
</tr>
<tr>
<td>ASIN</td>
<td>ASIN(#0)</td>
</tr>
<tr>
<td>ATAN</td>
<td>ATAN(#0)</td>
</tr>
<tr>
<td>ATAN2</td>
<td>ATAN2(#1, #0)</td>
</tr>
<tr>
<td>CEILING</td>
<td>CEILING(#0)</td>
</tr>
<tr>
<td>CONCAT</td>
<td>CONCAT(#0,#1)</td>
</tr>
<tr>
<td>COS</td>
<td>COS(#0)</td>
</tr>
<tr>
<td>COSH</td>
<td>COSH(#0)</td>
</tr>
<tr>
<td>EXP</td>
<td>EXP(#0)</td>
</tr>
<tr>
<td>FLOOR</td>
<td>FLOOR(#0)</td>
</tr>
<tr>
<td>GREATEST</td>
<td>GREATEST (n1 [, n2]...)</td>
</tr>
<tr>
<td>LEAST</td>
<td>LEAST (n1 [, n2]...)</td>
</tr>
<tr>
<td>LN</td>
<td>LN(#0)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>LOG10</td>
<td>LOG(#0)</td>
</tr>
<tr>
<td>INT</td>
<td>FLOOR(#0)</td>
</tr>
<tr>
<td>INT2</td>
<td>FLOOR(#0)</td>
</tr>
<tr>
<td>VAR</td>
<td>VAR(#0)&lt;#*#&gt;</td>
</tr>
<tr>
<td>MOD</td>
<td>MOD(#0, #1)</td>
</tr>
<tr>
<td>POWER</td>
<td>POWER (b, e)</td>
</tr>
<tr>
<td>ROUND</td>
<td>ROUND(#0, 0)</td>
</tr>
<tr>
<td>ROUND2</td>
<td>ROUND(#0, #1)</td>
</tr>
<tr>
<td>SIN</td>
<td>SIN(#0)</td>
</tr>
<tr>
<td>SINH</td>
<td>SINH(#0)</td>
</tr>
<tr>
<td>SQRT</td>
<td>SQRT(#0)</td>
</tr>
<tr>
<td>TAN</td>
<td>TAN(#0)</td>
</tr>
<tr>
<td>TANH</td>
<td>TANH(#0)</td>
</tr>
<tr>
<td>LEFTSTR</td>
<td>LEFT(#0, #1)</td>
</tr>
<tr>
<td>LENGTH</td>
<td>LENGTH(#0)</td>
</tr>
<tr>
<td>LOWER</td>
<td>LOWER(#0)</td>
</tr>
<tr>
<td>LRTIM</td>
<td>LTRIM(#0)</td>
</tr>
<tr>
<td>POSITION</td>
<td>LOCATE(#1, #0)</td>
</tr>
<tr>
<td>RIGHTSTR</td>
<td>RIGHT(#0, #1)</td>
</tr>
<tr>
<td>RTRIM</td>
<td>RTRIM(#0)</td>
</tr>
<tr>
<td>SUBSTRING</td>
<td>SUBSTRING(#0, #1, #2)</td>
</tr>
<tr>
<td>UPPER</td>
<td>UPPER(#0)</td>
</tr>
<tr>
<td>Concat</td>
<td>CONCAT(#0, #1)</td>
</tr>
<tr>
<td>InitCap</td>
<td>(UPPER(SUBSTRING(#0, 1, 1)) + LOWER(SUBSTRING(#0, 2, Length(#0) - 1)))</td>
</tr>
<tr>
<td>Trim</td>
<td>LTRIM(RTRIM(#0))</td>
</tr>
<tr>
<td>ConcatBlank</td>
<td>(#0)&lt; + ' + #*#&gt;</td>
</tr>
<tr>
<td>ISNULL</td>
<td>IFNULL(#0, 0)</td>
</tr>
<tr>
<td>NULLIF</td>
<td>NULLIF(#0, 0)</td>
</tr>
<tr>
<td>CurrentDate</td>
<td>NOW()</td>
</tr>
<tr>
<td>DayOfMonth</td>
<td>DAYOFMONTH(#0)</td>
</tr>
<tr>
<td>DayOfWeek</td>
<td>WEEKDAY(#0)</td>
</tr>
<tr>
<td>DayOfYear</td>
<td>DAYOFYEAR(#0)</td>
</tr>
<tr>
<td>Week</td>
<td>TO_CHAR(#0,'WW')</td>
</tr>
<tr>
<td>Month</td>
<td>MONTH(#0)</td>
</tr>
<tr>
<td>Year</td>
<td>YEAR(#0)</td>
</tr>
<tr>
<td>CurrentDateTime</td>
<td>NOW()</td>
</tr>
<tr>
<td>CurrentTime</td>
<td>NOW()</td>
</tr>
<tr>
<td>Hour</td>
<td>HOUR(#0)</td>
</tr>
<tr>
<td>Minute</td>
<td>MINUTE(#0)</td>
</tr>
<tr>
<td>Second</td>
<td>SECOND(#0)</td>
</tr>
<tr>
<td>Date</td>
<td>TO_DATE(#0,'YYYY-MM-DD')</td>
</tr>
</tbody>
</table>
### DaysBetween

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAYS_BETWEEN(#0, #1)</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### AddDays

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_DAYS(#0, #1)</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### AddMonths

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_MONTHS(#0, #1)</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### MonthEndDate

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_DAYS((ADD_MONTHS(#0, 1)), -DAYOFMONTH((ADD_MONTHS(#0, 1))))</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### Millisecond

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>substring(cast(extract(second from #0) as char(10)), 4, 3)</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### MonthsBetween

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 * (EXTRACT(YEAR FROM #1) - EXTRACT(YEAR FROM #0)) + EXTRACT(MONTH FROM #1) - EXTRACT(MONTH FROM #0) - (CASE WHEN EXTRACT(DAY FROM #0) &gt; EXTRACT (DAY FROM #1) THEN 1 ELSE 0 END)</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### MonthStartDate

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_DAYS(#0,1) - DAYOFMONTH(#0)</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### Quarter

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO_CHAR (#0, 'Q')</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### YearEndDate

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_MONTHS (ADD_DAYS (#0, 1 - (EXTRACT(DAY FROM #0))), 1 - (EXTRACT(MONTH FROM #0)))</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### YearStartDate

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_DAYS(ADD_MONTHS (#0, (13 - EXTRACT(MONTH FROM #0))), -EXTRACT(DAY FROM #0))</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### BANDING

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case when (#0=#2 and #2=#1+#3<em>floor((#2- #1)/#3) ) then floor(1.0</em>(#0- #1)/#3) +1 else 0 end</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### BANDINGC

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case when (#1=#2 and #0=#1) then #3 when (#0=#2) then #3 when (#0 between #1 and #2) then floor(1.0*#3*(#0- #1)/(#2- #1)+1) else 0 end</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### IF

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Case when #0 then #1 else #2 end)</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### GeoMean

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP(AVG(LN(#0)))</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### LOG

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG(#1, #0)</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### intersect

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#&lt;(##&gt;#</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### union

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#&lt;(##&gt;#</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### except

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#&lt;(##&gt;#</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### Quotient

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(CASE WHEN SIGN(#0/NULLIF(#1, 0)) =1 THEN FLOOR (#0/NULLIF(#1,0)) ELSE CEILING (#0/NULLIF(#1,0)) END)</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### Trunc

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(#0 AS INTEGER)</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### Acosh

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LN(#0+SQRT(#0-1)*SQRT(#0+1))</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### Asinh

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LN(#0+SQRT(POWER(#0,2)+1))</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### Atanh

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>((LN(1+#0)-LN(1- #0))/2</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

### Following window functions have been added to the SAP HANA 1.0 SP5 VLDB object.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank() over ([#P] [#O]) OLAPDBPATTERN=&quot;P:o</td>
<td>O:r&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lag(#0#lt;#</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lead(#0#lt;</td>
<td>Added to the SAP HANA 1.0 SP5 VL DB object.</td>
</tr>
</tbody>
</table>

## SAP HANA functions that MicroStrategy does not support out of the box

<table>
<thead>
<tr>
<th>Number functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITAND</td>
<td></td>
</tr>
<tr>
<td>COT</td>
<td></td>
</tr>
</tbody>
</table>

© MicroStrategy Incorporated, 2013
<table>
<thead>
<tr>
<th><strong>SIGN</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String Functions</strong></td>
</tr>
<tr>
<td>ASCII</td>
</tr>
<tr>
<td>CHAR</td>
</tr>
<tr>
<td>LOCATE</td>
</tr>
<tr>
<td>LPAD</td>
</tr>
<tr>
<td>NCHAR</td>
</tr>
<tr>
<td>REPLACE</td>
</tr>
<tr>
<td>RPAD</td>
</tr>
<tr>
<td>SUBSTR_AFTER</td>
</tr>
<tr>
<td>SUBSTR_BEFORE</td>
</tr>
<tr>
<td>UNICODE</td>
</tr>
<tr>
<td><strong>Date Time Functions</strong></td>
</tr>
<tr>
<td>ADD_SECONDS</td>
</tr>
<tr>
<td>DAYNAME</td>
</tr>
<tr>
<td>LAST_DAY</td>
</tr>
<tr>
<td>MONTHNAME</td>
</tr>
<tr>
<td>NEXT_DAY</td>
</tr>
<tr>
<td>SECOND_BETWEEN</td>
</tr>
<tr>
<td>CURRENT_UTCDATE</td>
</tr>
<tr>
<td>CURRENT_UTCETIME</td>
</tr>
<tr>
<td>CURRENT_UTCTIMESTAMP</td>
</tr>
<tr>
<td><strong>Miscellaneous Functions</strong></td>
</tr>
<tr>
<td>BINTOHEX</td>
</tr>
<tr>
<td>COALESCE</td>
</tr>
<tr>
<td>HASANYPRIVILEGES</td>
</tr>
<tr>
<td>HASSYSTEMPRIVILEGE</td>
</tr>
<tr>
<td>HEXTOBIN</td>
</tr>
<tr>
<td>ISAUTHORIZED</td>
</tr>
<tr>
<td>CURRENT_CONNECTION</td>
</tr>
<tr>
<td>CURRENT_SCHEMA</td>
</tr>
<tr>
<td>CURRENT_USER</td>
</tr>
<tr>
<td>GROUPING_ID</td>
</tr>
<tr>
<td>SESSION_CONTEXT</td>
</tr>
<tr>
<td>SYSUUID Function</td>
</tr>
</tbody>
</table>
Appendix 1: SAP HANA Versions, Use Cases and Editions

SAP HANA Versions and Editions

SAP HANA Versions
SAP HANA comes in several versions; versions indicate mainly how SAP HANA integrates with other SAP products, like SAP BW, for example.

There are two general versions of SAP HANA:

- Standalone (or SAP HANA 1.x)
- SAP HANA for SAP Netweaver BW (or SAP HANA 2.x)

The standalone version of SAP HANA takes the role of a regular database (with all the SAP HANA features, such as in-memory columnar stores), and it is used to build a traditional data warehouse solution.

The standalone version of SAP HANA is used as the base for several business domain-specific offerings from SAP BW that package ETL and data models in SAP HANA and reports to accelerate certain reporting scenarios (see below, SAP HANA Use Cases).

The SAP HANA for SAP Netweaver BW version is built purposely to integrate with BW (and also serve as a database in general). In other words, SAP HANA takes the role of the underlying SAP BW database, where DSO (Data Store Objects), infocube and other application object metadata and data is stored. These objects are stored more efficiently (infocubes, for example, do not have any dimension tables anymore, after optimizing them) and there is no need for aggregated cubes given SAP HANA’s columnar storage and data compression. Moreover, SAP BW (minimum SAP BW version required to have SAP HANA underneath) pushes down several operations down to SAP HANA instead of doing them at the application level.

There is potentially yet another SAP HANA version coming from SAP that would allow SAP’s ERP application to run on top of SAP HANA. The end objective is to allow both transactional and analytical (BW) applications to run on the same database, cutting down completely on the ETL overhead and allowing real-time reporting.

SAP HANA Editions
In addition to the different versions, there are editions of SAP HANA, each with different components included in the licensing. The difference among editions, for the most part, resides in the capability to extract, move or replicate data.

Platform Edition:

- For classical Extract, Transform and Load (ETL) development using SAP Data Services
- Best for non-SAP shops that want to accelerate any sources.
Enterprise Edition (and Enterprise Extended Edition, deprecated as of SAP HANA SPS5)

- Includes SAP ETL tools like SAP BO Data Services or SAP Landscape Transformations
- Supports trigger-based replication.
- Log-based replication
- Best for organization that have SAP ERP or an established BI environment.

**SAP HANA Use Cases**

The SAP HANA use cases vary by version, but these could be summarized as follows.

SAP HANA is used as a standalone database; in this case, there are ETL processes that move data into SAP HANA, and this data is consumed by multiple clients, either SAP applications (BICS and MDX) or other third-party clients like MicroStrategy through SQL for example.

SAP HANA is also used as an underlying database of other SAP products (among them SAP BW or in general SAP Business Suite). SAP application software is optimized (code changes) to reduce processing at the application level and push down as much logic to the database.
e.g. SAP BW -7.3 SP8- on HANA

e.g. SAP ERP 6.0, EHP6, CRM 7.0, EHP2, SCM 7.0 EHP2, Version for HANA
Appendix 2: SAP HANA Architecture and Modeling Basics

SAP HANA Architecture
SAP HANA architecture is made of multiple engines (JOIN engine, OLAP engine and CALCULATION engine), each of which is optimized to handle a specific data management task, along with multiple managers that handle also different functional aspects of the DBMS (security, backup, logging, etc.).


To note from the diagram, there are several interfaces in SAP HANA. Natively, there is SQL and MDX support. SAP HANA also supports a superset of SQL, SQLScript, which offers additional optimized functions for data processing; SQL Script is used for creation of stored procedures and is also used as a way to define complex database views in the form of Calculation Views.

Moreover, the different data processing engines are in-memory; the tables and structures are queried in-memory and all structures that handle these tables are optimized for in-memory processing.

SAP HANA Database Clients
SAP HANA comes with two different client products: SAP HANA Studio, SAP Information Composer. A third client can be Excel along with SAP HANA’s ODBO driver. SAP HANA Studio and all the low-level clients (ODBC/JDBC/ODBO drivers) can be downloaded (the Developer edition) from the SAP Community Network [here](http://sites.computer.org/debull/A12mar/hana.pdf).

A brief description of the clients is the following:

- **SAP HANA Studio** is a tool for administration and development; HANA Studio is analogous to the Microsoft SQL Server Management Studio.
  - Eclipse-based GUI
  - Main interface for administration, monitoring, development, modeling, etc.
  - This tool can be used to see tables and information views created in SAP HANA.
• **SAP HANA Information Composer** is a tool for power users; it is a web-based interface that allows users to “import” files (like excel files) into SAP HANA, and expose these as an information view.
  – Web-based SAP HANA modeling client
  – Data import and join with existing information views
• **Microsoft Excel** (using MDX and ODBO): Microsoft Excel, as part of its own native functionality, can connect to MDX providers using ODBO (OLE DB for OLAP) drivers. In this case, SAP HANA provides an ODBO driver (analogous to ODBC driver for SQL sources), so that Excel can query the different information views, exposed as multidimensional cubes.
References