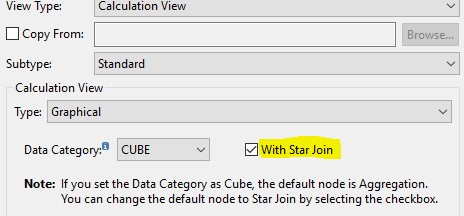
Consolidated best practices for HANA modeling that we witnessed for better performance.

* Try using Star-join instead of joining multiple nodes of master data/dimensions to a fact table of measures.
* Try to use Calculation view for all the modeling and design that involves complex calculations and data massaging.

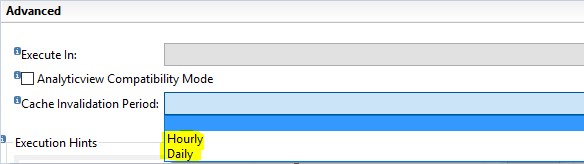
For master data/attribute view types use – Dimension

For mulit-dimensional purpose/OLAP scenarios – CUBE

* Even attribute views/analytic view with calculated columns & measures/logic will be passed to calculation engine of HANA view.
* If the design involves many joins of master data-dimensions with measures (fact tables columns) use Star-join for the type. This has better performance rather than making joins at multiple nodes



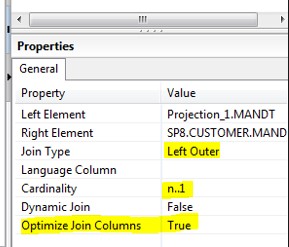
* Setting the Cache invalidation period for views/models that doesn’t have frequent changes on underlying data will give you quicker results and hold the cached result for an hour or day



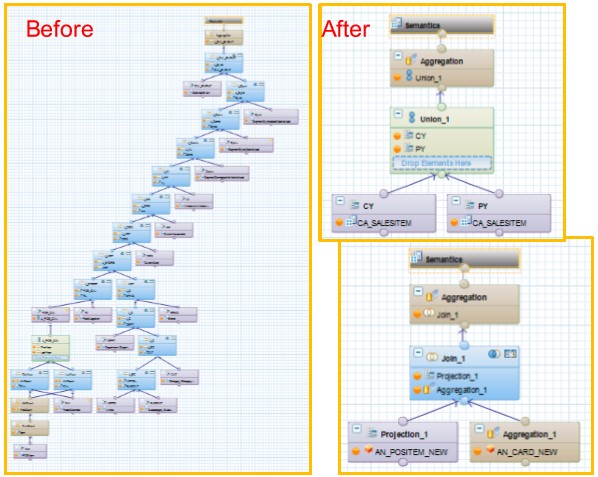
Suggestions on Joins

* If possible try to utilize the same node with calculated columns for different conditions and project with union nodes to combine several conditions/segregated projections of same node. (Avoid projecting same node/information to different segments)

If joins are required try to use Left outer/right outer with cardinality (n:1 or 1:1) – if you are sure about the driving tables, as this would enhance the performance and option of selecting “Optimize join = TRUE.



Also, consider multiple join nodes throughout the design to analytic view and then join those analytic views in the calculation view. Avoid single data source (or node) feeding multiple nodes in calculation view.



* Also, minimize the number of nodes and joins as much as possible and try to use union instead.
* For better query performance and quicker retrieval of results try to avoid filtering/joining on calculated columns and do not mix up scripted views and graphical calculation views in a single view.
* Alternately CE\_CALC functions or script can be considered for joining scripted views or several graphical calculation views.

*Always follow – HANA Development Points*

* Use left outer joins maximum.
* Specify cardinality in joins (n:1 or 1:1) – only if sure
* Set optimize join = true (only if cardinality is set like above)
* Use table functions instead of scripted calculation views.
* Execute in SQL-engine (for HANA live)
* Reduce the data set as early as possible. Use design time filters at the lowest level
* Input Parameter: Placeholders part of the models Can be used in Calculation. Can accept multiple values and can be derived from table or store procedure
* Ensure Variables (where clause) is pushed to the lowest level. Confirm using Visualization Plan
* Use analytical privilege and SQL analytical privilege (SP10) to filter data
* Avoid script-based calculation view, WHERE clause will not be pushed down.
* Using Filter is better than using Inner Join to limit the dataset.
* Avoid filters on calculated column (consider materializing these columns).
* One of the best practice in HANA modeling is to define joins on columns with either INTEGER or BIGINT as data types.

It is not recommended to have joins on calculated columns/fields with NVARCHAR or DECIMAL as data types, might create performance issues

**2nd Article**

**1.) ENVIRONMENT SETUP:**

**PACKAGE:**

* + Create one top-level package for all of the customer’s content (if acceptable to the customer).
    - Create a sub-package for the content you will deploy in the project.
      * Multiple sub-packages may be created, if required or desired to organize a large amount of content.
      * Create all content models under this package or the appropriate sub-packages
      * Create separate sub-packages for each group of related content that will be deployed together as a set (e.g., by project).
  + Create one top-level package called “dev” (or “development”) for work-in-progress
  + Create a sub-package for each developer
  + Create all Analytical and Calculation views under these play-area packages.
  + Once the development is complete (including unit testing), these views can be copied/moved to the main package for the project when it is ready to migrate between environments.
  + **Always take backups (Exporting Content ) of entire content of a project.**
  + **Import only your content instead of Complete Project content  and restore only your stuff** —  (This will avoid over writing others stuff when you import)

* **Optional** – Create a top-level package called “test” or “qa”
  + The structure under this package should match that of the top-level customer package for your content to be deployed.
  + This allows you to have code in testing before committing it to the deployment package.
  + You may create multiple top-level test packages for complex projects with multiple workstreams (this is not generally recommended unless necessary).
* Use HANA repository check in/out
  + SAP HANA Development perspective (introduced in SP05) should be used to track version of objects (instead of developer mode export /import)

**DELIVERY UNIT:**

* + Create a delivery unit for each group of related content that you will deploy for the project.
    - This will be used for promoting your content from DEV to QA and PROD
    - In general, the delivery units should match the sub-packages under the top-level package for the customer.
  + Assign your deployment packages (under the top-level customer package) to the appropriate delivery units.

**DELETING PACKAGES:**

A package cannot be deleted when it is not empty.

* Delete all the underlying objects before deleting the package. This includes sub-packages.
* Note that some objects are hidden in the Modeler perspective.  Use the HANA Development Perspective to see all of the associated objects (also you can change preferences to see hidden objects)

**2.) DATABASE TABLES/SQLs**

* **Schema**
  + Plan your Schema layout before you start your project.
  + Create generic names to Schemas (for ex : SLTECC, DS\_SCHEMA , CUSTOM\_SCHEMA etc.) .If possible keep same schema names in all environments. This will help migration go smoothly from one environment to other environment. If schema names are different you might have to adjust schemas when you migrate to new environments.
  + All tables replicated through SLT should be in the same schema.  However, do not create custom tables or tables loaded through DS in the same schema as the SLT tables.
  + All tables created/loaded through Data Services should be in a separate schema from SLT tables.
  + Staging tables for ETL should be in a separate staging schema.
  + Custom tables or static tables should be in a separate schema.
* **Table Creation**
  + All tables to be used in content models should be created as COLUMN TABLES in HANA for best performance.  (Note: Data Services and SLT create column tables by default).
  + It is recommended to always provide a comment/description on the table and on each column for clarity and documentation purposes.
* **Security**
  + You must grant SELECT privileges on the database schemas or tables to \_SYS\_REPO in order to use the tables in content models

          Syntax:

GRANT SELECT ON SCHEMA “<SCHEMA\_NAME>” to \_SYS\_REPO WITH GRANT OPTION;

* Grant DML commands to different schema

Syntax:

GRANT SELECT/UPDATE/DELETE  on tables/procedure to  <schema\_owner> ;

* Grant individual table access to different schema

Syntax:

**GRANT SELECT ON “<SCHEMA\_NAME>”.”<TABLE>” to “<SCHEMA\_NAME>” WITH GRANT OPTION;**

**AUTO-GENERATED COLUMNS (Concat Attributes)**

Auto generated columns are new columns automatically populated based on a calculation using the physical columns in the same table. These have some advantages and disadvantages.

Syntax :

ALTER table “<SCHEMA>”.”<TABLE>” ADD (ZZYEAR NVARCHAR (4) GENERATED ALWAYS AS ‘2014′);

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Calculations are pushed down to the  Database (instead of models) | The code is not visible and can’t be transported (unless table definitions are created in the repository) |
| Increases performance | The code to create the columns needs to be manually run across environments and needs maintenance |
|  | Migration of Models to diff environments will fail if the virtual columns or Auto Generated columns are not there |

**SQLs**

While using data preview on Models

|  |  |
| --- | --- |
| **DOs** | **Don’ts** |
| Apply a filter condition:  Select \* from View Where < Condition> | SELECT \* From View |
| Select Count( column) from View | Select Count(\*) from view |

Use following conversion functions while migration from other databases (Oracle/Teradata and etc.) to HANA.

* DECODE function (Oracle )  to CASE statement in HANA
* CONVERT function  to CAST  statement in HANA
* TRUNC function  to CAST statement in HANA
* SYSDATE, SYSTIME or SYSTTIMESTAMP to CURRENT\_DATE, CURRENT\_TIME or CURRENT\_TIMESTAMP statements in HANA
* Converting BLOB/CLOB Data Types (Oracle ) to HANA Data types :

BLOB/ CLOB fields cannot be used HANA models. However, you can create a generated column selecting first 5000 characters (or anything else you’d like) to report on the data stored in the BLOB/CLOB field.

String functions like SUBSTRING works fine on NCLOB field. For other BLOB type fields, you can cast it to VARCHAR before using string functions.

**Substring** (**cast**( BTEST **as** **VARCHAR** (5000)),1,10)

* Concat() Functions in Models are very expensive and the create Virtual columns (auto-generated column) for concatenated text strings  will make more sense.

 Check following link for some additional info.

<http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/b0f2d8c9-f321-3110-41bb-dc8e8e14dc9a?QuickLink=index&overridelayout=true&58815282151702>

**3.) NAMING CONVENTIONS**

There is no hard and fast rule for HANA Model Naming convention.  However, we have followed couple of scenarios and you can pick which ever naming convention you like. We are also providing link from SAP on this topic.

* **Select a consistent naming conventions throughout the project**
* **Name every element in CAPITAL LETTERS.**
* **Give meaningful business names to all columns which are exposed to users.** Keep the name as short as possible (preferably under 15 to 20 chars)

|  |  |  |
| --- | --- | --- |
|  | **Option 1** | **Option II** |
| **ATTRIBUTE VIEWS** | **DIM\_DATE\_AT** | **AT\_DATE** |
| * **Column Names** | Give business names.  If column is participating in Key for  ex : SITE\_KEY | Same |
| * **Column descriptions** | **SITE\_DESC** | The element used in Label Column (label mapping) should be renamed as  <ATTRIBUTE>.descriptions (e.g. REGION.descriptions) |
| **ANALYTICAL VIEWS** | **FACT\_SALES\_AV** | **AN\_SALES** |
| * **Data foundation columns** | If ECC you could leave technical names  or suffix with \_FACT | Same |
| * **Restricted Columns** | If exposed to user give a Business Name otherwise RM\_.  Ex : **SOLD\_AMT\_US**  (Exposed to user )  **RM\_SOLD\_AMT\_US** (used for internal calculations ) | same |
| * **Calculated Columns** | Business Name of the measure | same |
| **CALCULATION VIEWS** | **FACT\_SALES\_CV** | **CA\_SALES** |
| * **Restricted Columns** | If exposed to user give a Business Name otherwise RM\_.  Ex : **SALE\_AMT**  (Exposed to user )  **CM\_SALE\_AMT** (used for internal calculations ) | Same |
| * **Calculated Columns** | Business Name of the measure | Same |
| **ANALYTICAL PRIVILEGES** | **AP\_RESTRICTION\_AT** (On ATTR View)  **AP\_RESTRICTION\_AV** ( On Analytical View)  **AP\_RESTRICTION\_CV** (On Calc view) | Same |
| **HIERARCHY** | **HI\_<BUSINESS\_NAME>\_PC** (for Parent Child hierarchy) **HI\_<BUSINESS\_NAME>\_LV** (for Label based hierarchy). | Same |
| **INPUT PARAMETERS** | **IP\_PARAMETER\_NAME** | same |
| **VARIABLES** | **VA\_DATE** | Same |
| **PROCEDURES** | **SP\_PROCEDURENAME** | same |
| **CUSTOM TABLES** | **ZT\_TABLENAME** | ZT\_tablename |

**4.) MODELING**

* Before You Begin
  + Check for Key columns for any Null Values at table level.
  + The columns that are part of the joins in the HANA models should not contain any NULL values (resolve null values through ETL or SLT jobs before starting modeling).
* General
  + Create all expected Attribute Views first.
    - These will be used later in creating analytic views and calculation views.
    - An attribute view can be used in multiple analytic views or calculation views
    - To the extent possible, design your attribute views as common components that can be used in multiple models to reduce maintenance effort.
  + Decision tree for modeling (in order of performance)
    - Analytic View –>Attribute View –>Graphical Calc View –>Calc View (CE function)–> Calc View (SQL)
  + **Create views Step by step.** 
    - What it means is create your views step by step. Verify each step before moving on to the next step.
    - For example in creating an Analytic View: Create the data foundation first and activate it and see the data. If your data set is big use a filter where you could get 10-25 rows or less to validate the model. Next add an attribute view join and activate and check data. It might be slow process but once you are done modeling it will be done. The advantage by doing this is you can find any data or join issues at every level.

**4.1) ATTRIBUTE VIEWS:**

* You must define at least one Key Attribute ( Ex : SITE\_KEY) on an attribute view. This is typically the column(s) that will be used to join the view to other tables/content.
* Expose only required columns used in reports and hierarchies.  Do not create columns for everything “just because it’s there”.
* Give meaningful business names for all exposed attributes.
* In generally, try to make sure all attribute names in the View are unique.  There should be no duplicate Attributes in other Attribute views.( Same attribute column should not be in two Attribute Views, in general)
* Avoid calculated columns (example To\_date(), concat(), To\_char() and etc. )  in Attribute views.
  + Calculated columns create an implicit calc view wrapper around the analytic view and will impact performance.
  + Consider replacing them with materialized columns (or auto-generated columns) in the underlying table.
  + Alternate create them in Calculated Views.
* Level based hierarchies’ work in most of the reporting tools.  Parent child hierarchies works only in Analysis OLAP, Analysis Office or Excel.
* Check performance of attribute views.  Queries on Attribute views should respond in seconds otherwise it will impact overall performance of the views that use it.
* For Description columns the element used in **Label Column** (label mapping) should be renamed as <ATTRIBUTE>.descriptions (e.g. REGION.description)

NOTE : There are some extra steps you need to do if you are creating Calc views based on Analytical views with Attribute views consists of “column.description” . In Calc views it converts column.description to column\_description and this has to be remapped with column.description in XML and re-import and activate Calc views.

### 4.2) ANALYTICAL VIEWS

* Star Schema Design is possible with analytical views.
* You can define measures only from **one fact table**, even if you have multiple fact tables in your data foundation.
* Use design time filters/input parameters to limit the dataset (if possible).
* Use joins on integer/date key columns (if possible)
* Create restricted measures by using Dimensions from attribute or from Fact keys from data foundation.
* Use restricted measures (where possible). This performs better in terms resource consumption (memory and/or CPU time) vs. calculated measures
* Avoid creating Calculated Attributes in AVs. For example To\_date (), concat () , To\_char() and etc. Consider moving them to calculation view or push them to database layer (materialized or virtual column). Calculated attributes are computed in the context of Calc Engine (even though they are in Analytic View), hence there will be data transfer between engines and hence lower performance. Keeping all the calculations in the context of OLAP engine will give the best performance.
* After activating AV check in \_SYS\_BIC if it is generating AV/olap  (it is called wrapper calc view) This means data is moving between the engines. This needs to be avoided.
* Avoid using “calculation before aggregation” on BIG data sets  (this is very intensive in terms of CPU time) . Consider moving these calculation to database layer (materialized column or virtual columns)
* If you have a design time filter in the Attribute view and if this Attribute view is joined to the data foundation using **referential join,** the join to the attribute view will be always executed (even if no attribute is selected from Attribute View). Watch out for this while you are modeling.
* Be careful using Referential joins since this can lead to inconsistent results if referential integrity on both sides of the join is not assured.  If you are not sure, use a Left Outer Join or an Inner Join, as appropriate, for consistent results across queries employing different columns from the model.
* Use Temporal Join (introduced in HANA SP05) for modeling slowly changing dimension. Only **referential join** and **date & integer** data types are supported.
* Avoid Compound joins .This may not possible always but watch out for any performance issues.
* If your model have many joins you could also deploy on Join engine to get a best performance (possible only OLAP engine specific functions like Temporal Join is not used)
* Use Input parameters to calculate measures based on user input
* Use variables to further restrict the dataset for better performance

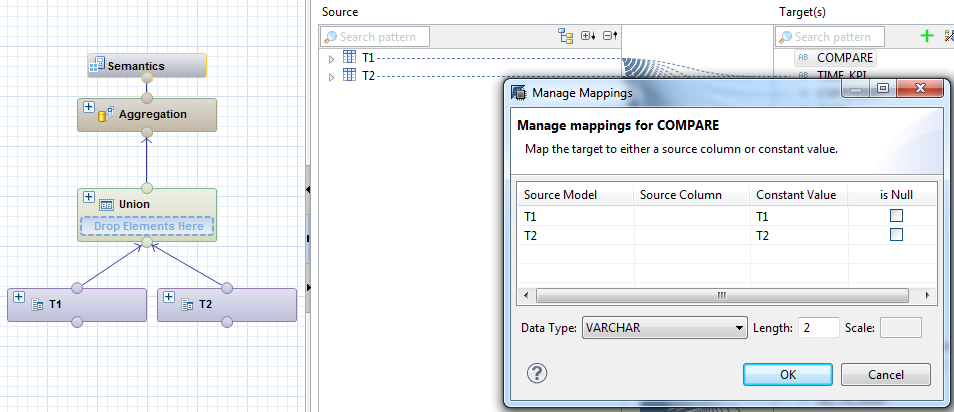
**DISTINCT COUNTS:**

Distinct counts in any large databases are challenging .If you run count distinct on OLAP view on a large fact tables with a high number of distinct values consider using the note.

1941113 – Influence standard behavior of OLAP queries in a distributed HANA installation.

**4.3) CALCULATION VIEWS**

* All views/tables should be used with a projection node. Projection nodes improve performance by narrowing the data set (columns).
* Further optimization can be done by applying filters at projection nodes.
* Avoid using JOIN nodes in calculation view. Consider replacing them with UNION nodes (where possible). Alternately consider pushing these joins to Analytic views.
* While using Unions make sure there will be no null values in Measure columns otherwise Union Operation will chock.
* While Unioning measures in Union Node do manage mappings and supply 0 values for null measures columns.
* Use Input Parameters/Variables to restrict the dataset within the Calc View.  Filters should be applied as early as possible in the data flow.If you are create a Calc View that unions multiple Calc views (or Sub Calc views), use Constant mapping in Union Node. This will improve the performance as the query will only fetch the results related to the constant value of the sub Calc view- bypassing other values for a particular sub Calc view underneath the union for execution in reporting if needed. (See the picture below)

[](https://blogs.sap.com/wp-content/uploads/2014/03/pic1sbt_418509.png)

* Calculated columns should be defined here
  + For example: **midstr (string(“ERDAT”), strlen(string(“ERDAT”))-9,4)**
  + **Concat (String 1, String2)**
* Calculated Columns are calculated before aggregation.
* Hierarchies (if any) need to be redefined in the Calculation view. Hierarchies defined in the Attribute Views are not visible in the Calculation View.
* Labels of Attribute (Label Column) & Description of Measures from underlying Attribute/ Analytic or Calculation views are not visible in the Calculation View.  These need to be re-mapped.
* Enable “Multidimensional reporting” (introduced in SP06) should be used.
  + If the view was created in SP05 or earlier revision, this can be enabled by clicking ‘Switch to Aggregation” in the context menu of the projection node (just below the “semantic” node)
* **STAR JOINS IN CALC VIEWS:**
* Introduced in SP07 .You can select multiple measures from multiple facts tables in one Star model to support 3rd Normal Form design.
* Check the following URL for more on Star Joins in Calculation views.[**http://scn.sap.com/community/services/blog/2014/02/13/hana-sp07–star-join**](https://blogs.sap.com/?p=102022)
* **4.3.1) SCRIPTED CALCULATION VIEWS**
  + DO NOT mix the functions of SQL Script and CE Functions in a composite model
  + (For **example** : do not code one Calculation View with SQL Script and use it inside another Calculation View with CE Functions). The performance will be very bad.
  + Try to expose ONLY required fields in the CE\_PROJECTIONS
  + Pass on all parameters as required to the lowest level ( this applies to Graphical or Script based Views ) .
  + Reduce the size of the data set as early as possible in the data flow.  This can be done by applying static filters or applying dynamic filters using parameters at the lowest level
  + Minimize the size of any intermediate result sets in complex processing.  Do not include any extra attribute columns that may increase the rows of aggregated results being passed between nodes in the data flow.
  + Avoid extremely large intermediate result sets, even if they will be reduced or aggregated before the final result is returned to the client.  These can have high memory requirements.
  + Perform calculations at the last node possible in the data flow (ideally done in final aggregation or projection).  This will reduce the amount of data rows on which you are performing the calculations also reduce the number of columns being transferred from one node to another.
  + In some situations, you can use a union and then aggregate the results instead of using a join for better performance.
  + Reduce Complexity of SQL Statements: Break up a complex SQL statement into many simpler ones. This makes a SQLScript procedure easier to comprehend.
  + Identify Common Sub-Expressions: If you split a complex query into logical sub queries it can help the optimizer to identify common sub expressions and to derive more efficient execution plans.
  + Understand the Costs of Statements: Employ the explain plan facility to investigate the performance impact of different SQL queries used in scripts.
  + Reduce Dependencies: As SQLScript is translated into a dataflow graph, and independent paths in this graph can be executed in parallel, reducing dependencies enables better parallelism, and thus better performance.

**TESTING :** Do unit testing for each model and create test cases document.This will help to create a deliverable document to client.

* **4.4 )  HANA Smart Data Access :**

Please refer following document for SDA Modeling Recommendations :

[**http://scn.sap.com/docs/DOC-73484**](http://scn.sap.com/docs/DOC-73484)

**5.) SECURITY**

* + Do not use the out of the box “SYSTEM” user to manage day to day database administration
  + Create Delegated Administrators to do day to day job and use System user if necessary.
  + Create Security Model ( Which user can see what data )
  + USERS/ROLES will be created ONLY by Delegated SYSTEM Administrator Users.
  + Use the design time role instead of the runtime role ( through Create role/user SQL statement)
  + Use design time roles to create security.
  + Creating User will also create a Schema by default.
  + By Default SYSTEM do not have access to others users Schemas. System user or any other user can see data only when Schema Users Grants access to Users and System Users.
  + Create a Security model (which user can see what content) before you implement Hana data level security.
  + Consider using row level security.
  + HANA Out of the box the COLUMN Level security feature is not there but this need to be handled in Modeling by creating one base model  with Secured Columns and create another view with hiding all the secured columns.
  + HANA security is implemented with Analytical privileges (APs) and APs will improve the performance as it applies filters to models.
  + Consider using Dynamic Analytic Privileges (introduced in SP05) whenever possible. This will simplify the process.
  + Always create roles and assign required privileges to role and assign Roles to users.
  + Do not grant \_SYS\_BI\_CP\_ALL privilege to any role/user.
  + Create a development project to store your design time objects in a shared repository.
  + Design security using the design time roles.
  + Design time role is portable to another SAP HANA Landscape through life cycle manager.

**6.) PERFORMANCE**

* Create all calculations in Analytical or Calculation views. Avoid creating any calculations in Reporting layer (Universe & Front end tools).
* Limit output columns using Projection nodes
* Consider partitioning large tables to get better performance
  + Max 2B records per table (or table partition) and max 1000 partitions per table.
  + For More details check following link

[**https://cookbook.experiencesaphana.com/bw/operating-bw-on-hana/hana-database-administration/system-configuration/partitioning-and-distribution-large-tables/**](https://cookbook.experiencesaphana.com/bw/operating-bw-on-hana/hana-database-administration/system-configuration/partitioning-and-distribution-large-tables/)

* Do not create working tables in different schemas. This will create security problems on ownerships.Instead of that create a separate schema and create all working tables and use it in your Modeling.
* Avoid composite primary keys whenever possible. Composite primary key creates additional indexes on the table, which will take additional space and hits the performance. If you have to use it be aware of this fact.
* If possible avoid Joins on Character columns.
* Analyze the performance of the Query/Models using Explain Plan and Visualization Plan
* Identify the long running queries by reviewing Performance tab to analyze system performance located under the Administration editor
* Hana Automatically handles Indexes on key columns. Create secondary index on non-key columns if it is absolutely necessary .Create indexes on non-primary key columns (with high cardinality) to enhance the performance of some queries using the index adviser.

        Syntax:  CREATE INDEX <name> ON <table>.<column>

* Use the index adviser to find out for which tables and columns indexing would be most valuable. The indexAdvisor.py script is part of a SAP HANA system installation and runs from the command line. It is located in the $DIR\_INSTANCE/exe/python\_support directory.
* Indexing the primary key columns is usually sufficient because queries typically put filter conditions on primary key columns. When filter conditions are on non-key fields and tables have many records, creating an index on the non-primary key columns may improve the performance.
* There is a trade-off between indexing and memory consumption: While indexing non-primary key columns can make query execution faster, the downside is that memory consumption increases. The index adviser takes this trade-off into account: In dynamic mode, the index adviser looks for the tables and columns that are used most often. The higher the selectivity is, that is, the more different values are in the column, the higher are the performance gains from indexing the columns.
* To check whether there is an index for a column, you can see the system view M\_INDEXES.
* With SAP HANA, you do not need to perform any tuning to achieve high performance. In general, the SAP HANA default settings should be sufficient in almost any application scenario. Any modifications to the predefined system parameters should only be done after receiving explicit instruction from SAP Support.
* If two columns are frequently compared by queries, ensure the two columns have the same data type. For columns of different types, SAP HANA uses implicit type casting to enable comparison in HANA Models. However, implicit type casting has a negative effect on performance.

**7.) MIGRATION :**

If you want to transport the HANA content as you are used to do for the ABAP landscape and has e.g. process tools in top of the system transport landscape, then the recommended approach would be to use CTS+.

if the HANA landscape is completely independent and there are no requirements concerning process integration or coupling of other application artifacts, then you can use the HANA only transport tool that is part of HANA Lifecycle Management (known as HANA Application Lifecycle Manager), which is a HANA only tool.

Currently the limitation in both scenarios is, that you have to transport the complete Delivery Unit (DU) and cannot transport smaller granularities.

**SAP HANA Lifecycle Manager:**

The SAP HANA lifecycle manager (HLM) is a tool that enables flexible customizations of an existing SAP HANA system. There are three available working modes for the SAP HANA lifecycle manager:

1. Using SAP HANA studio

2. Using the command line

3. Using a standalone browser

**Using SAP HANA Lifecycle Manager through SAP HANA Studio:**

Keep in mind that, to work with the HLM, you need to make certain configuration settings for the SAP HANA studio. There are also certain browser restrictions. For more information, see

[**http://help.sap.com/hana/SAP\_HANA\_Update\_and\_Configuration\_Guide\_en.pdf**](http://help.sap.com/hana/SAP_HANA_Update_and_Configuration_Guide_en.pdf)

Keep in mind that Lifecycle Management perspective in the SAP HANA studio requires version of Java Virtual Machine equal or higher to 1.6.0\_12.

**Using SAP HANA Lifecycle Manager through Standalone Browser:**

Make sure you review the following browser requirements:

* For Microsoft Windows, you need Internet Explorer version 9 or above. If you are running Internet Explorer version 9, make sure that your browser is not running in compatibility mode with your SAP HANA host. You can check this in your browser by choosing Tools ->Compatibility View settings.
* For Linux, you need XULRunner 1.9.2 or above. We recommend that you install XULRunner 1.9.2 (or newer) separately, but if you have already installed Firefox 3.6 (or newer), it contains XULRunner 1.9.2.
* To use the tool from a standalone browser, call the following URL:   [https://](https://scn.sap.com/)<host>:1129/lmsl/HLM/<SID>/ui/?sid=<SID>.
  + - Make sure you use the fully qualified name of the SAP HANA system, such as myhost.sap.com (not just myhost).

**8.) COMMON ERRORS**

* **Error : “Cannot create column index” during activation.**
  + This issue occurs when you try to redeploy/reactive a model after you made minor changes (or imported a model from another system).
  + Delete the entry from the RUNTIME\_OBJECTS table and activate again
  + DELETE from “\_SYS\_REPO”.”RUNTIME\_OBJECTS” where OBJECT\_NAME LIKE ‘%CA\_MY\_CALC\_VIEW’;
  + You’ll see this issue, if you try to activate a model – while massive data load to underlying table is in progress, which locks the table for significant amount of time (say 10 – 30min or more).
  + However, this may not be an issue with typical SLT replication (not first time load). Typical SLT loads only locks the tables for few seconds, So SLT load may slow down the activation. But it does not error out the activation.

Error : SAP DBTech JDBC: [2048]: column store error: <?xml version=”1.0″ encoding=”utf-8″?><createCubeResult version=”1.0″><status><message>Index name conflicts with existing index name</message><errorCode>2019</errorCode></status><details><warnings><detail><element>cubeSchema</element><code>46</code><message>Default language not set. Use ‘en'</message></detail></warnings></details></createCubeResult>

* As System User :

drop calculation scenario “\_SYS\_BIC”.”pkg/view”; **Ex:** drop calculation scenario  \_SYS\_BIC”.”ms/FACT\_WM\_AV”;

**Error :** Could not execute ‘select year,count(\*) FROM “\_SYS\_BIC”.”ms/FACT\_SHIPPING\_AV” group by year’  
SAP DBTech JDBC: [2048] (at 28): column store error: search table error:  [2712] Error executing physical plan: olap: merging multi value dicts is not implemented;BwPopJoin2Inwards pop17(MODELER:VTTPen.TKNUM to .VBELN),in executor::Executor in cube: \_SYS\_BIC:ms/FACT\_SHIPPING\_AV

* UPDATE “schema”.”table” merge delta index; **Ex:** UPDATE “SAP\_MODELER”.”VBAK” merge delta index;

**Error :**    Internal deployment of object failed;Repository: Encountered an error in repository runtime extension;Internal Error:Create Scenario: failed aCalcEngine.createScenario(): The following errors occured: An index already exists with the same name (2003)nnprinting XML <?xml version=”1.0″ encoding=”UTF-8″ standalone=”no”?><cubeSchema defaultLanguage=”EN” defaultSchema=”\_SYS\_BIC”      “select \* from sys.m\_ce\_calcscenarios;

* drop calculation scenario “”\_SYS\_BIC””.””emc.sd/CV\_BACKLOG\_UPD2″”  cascade;

**If you are not able to open any of the views :**

* Could be STUDIO Version problem .Upgrade Studio

**Not able to open any AN VIEW :**

* Underlying Key Columns in joins are changed. Open Xml file and map correct keys.

**Trouble Deleting Package:**

Check in \_SYS\_BI

* select \* from “\_SYS\_BI”.”BIMC\_ALL\_CUBES” where cube\_name = ‘FACT\_WM\_AV’;
* select \* from “\_SYS\_BI”.”BIMC\_DIMENSIONS” where COLUMN\_OBJECT  ='”\_SYS\_BIC”.”ms/FACT\_WM\_AV”‘
* select \* from “\_SYS\_BI”.”BIMC\_DIMENSIONS”

FROM  \_SYS\_REPO

* select \* from “\_SYS\_REPO”.”ACTIVE\_OBJECT” where Object\_name = ‘FACT\_WM\_AV’
* select \* from “\_SYS\_REPO”.”PACKAGE\_CATALOG” where PACKAGE\_ID =’ms’
* select \* from “\_SYS\_REPO”.”INACTIVE\_OBJECT” where Object\_name = ‘FACT\_WM\_AV’

Some other issues solution can be found in following doc.

[**http://scn.sap.com/docs/DOC-50291**](http://scn.sap.com/docs/DOC-50291)

**9.) Best Practices to Build Business Objects Universe against HANA Models.**

* Universes should always create against HANA Database information Models.
* Business logic need to be pushed to HANA models to get maximizing the query performance and also it is presentation layer agnostic. Every reporting tool is able to consume HANA models as is. For Example Webi, Explorer, Lumira, A-Office, XSengine, HTML5 apps and other 3rd party reporting tools.

**10.) Some Useful Links:**

YTD,MTD other calculations based on relative dates

<http://scn.sap.com/docs/DOC-50420>

Handling SCDs with SLT

<http://scn.sap.com/docs/DOC-45991>

Hana DB Install:

<http://scn.sap.com/docs/DOC-31036>

Row to Column & Column to Row Transformation in Modeling.

<http://scn.sap.com/docs/DOC-51791>

<http://scn.sap.com/docs/DOC-50541>

[How To: Dynamic Transposition in HANA](http://scn.sap.com/docs/DOC-48833)