**Hierarchy**

[Содержание](#Содержание)

# [Consuming BW Parent Child Hierarchy in HANA – A workaround](#Consuming_BW_Parent_Child_Hierarchy)

# [BW Hiearchy in HANA Modelling](#BW_Hiearchy_in_HANA_Modelling)

# [Generating Hierarchies](#Generating_Hierarchies)

[The Navigation Functions](#The_Navigation_Functions)

[The Aggregate Functions](#The_Aggregate_Functions)

## [SAP HANA SQL extension for generating a tree hierarchy](#SQL_extension_for_generating_a_tree_h)

### [SAP HANA Level Hierarchy](#SAP_HANA_Level_Hierarchy)

# [New Hierarchy SQL enablement with Calculation Views in SAP HANA 1.0 SPS 10](#Hierarchy_SQLenablement_with_CV)

# [Query Hierarchy Data in SAP HANA Database using SQLScript](#Query_Hierarchy_Data_using_SQL_script)

[Not Assigned Members enabled hierarchies with SAP HANA Calculation Views](#Not_Assigned_Members_enabled_ierarchies)

# Consuming BW Parent Child Hierarchy in HANA – A workaround

<https://blogs.sap.com/2013/08/06/consuming-bw-parent-child-hierarchy-in-hana-a-workaround/>

We need to perform a self-join on this H table in order to get the relationship

select

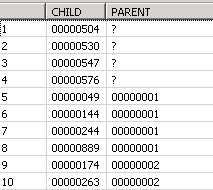
c.NODENAME as Child, p.NODENAME as Parent

from "SAPABAP1"."/BIC/HBCGI0000" as c

left outer join "SAPABAP1"."/BIC/HBCGI0000" as p

on p.nodeid = c.parentid

order by p.NODENAME



The ? or null value signifies that the node does not have any parent and it is the top most node of the hierarchy.

*Rem*

Whenever you create a parent child hierarchy in HANA and try to consume it in front end then all the nodes should have *one parent*. The *topmost node* should have null as parent. Also, *the chain* of Parent Child *should not break* while creating Hierarchies.

Now, this data needs to be pushed into a persisted table which then, can be utilized in a Calculation View to create a Parent Child hierarchy.

If you do not wish to persist the data into a table and would like to execute the query on the fly when user runs the report, then you can create a script based calculation view using the same SQL. This calculation view will again be used inside the final calculation view.

# BW Hiearchy in HANA Modelling

<https://blogs.sap.com/2015/12/16/bw-hiearchy-in-hana-modelling/>

[Содержание](#Содержание)

**Level Based Hierarchy**

There is no automated process and no standard functions available to i*mport BW hierarchy in SAP HANA* and use it in reporting tools.

Flattening is based on tracing the parent of every node starting from leaf node. It is an unbalanced Hierarchy. Scripted Calculated view needs to be created to flatten the hierarchy.

select

zz.NODENAME, t\_cgi\_lv\_.txtlg as NODENAME\_T,

L1, t\_cgi\_lv1.txtlg as L1\_T,

L2, t\_cgi\_lv2.txtlg as L2\_T,

L3, t\_cgi\_lv3.txtlg as L3\_T,

L4, t\_cgi\_lv4.txtlg as L4\_T,

L5, t\_cgi\_lv5.txtlg as L5\_T

from ( select

F.NODENAME,

(CASE F.TLEVEL

WHEN 6 THEN H1.NODENAME

WHEN 5 THEN H2.NODENAME

WHEN 4 THEN H3.NODENAME

WHEN 3 THEN H4.NODENAME

WHEN 2 THEN H5.NODENAME

ELSE '' END) AS L1,

(CASE F.TLEVEL

WHEN 6 THEN H2.NODENAME

WHEN 5 THEN H3.NODENAME

WHEN 4 THEN H4.NODENAME

WHEN 3 THEN H5.NODENAME

ELSE '' END) AS L2,

(CASE F.TLEVEL

WHEN 6 THEN H3.NODENAME

WHEN 5 THEN H4.NODENAME

WHEN 4 THEN H5.NODENAME

ELSE '' END) AS L3,

(CASE F.TLEVEL

WHEN 6 THEN H4.NODENAME

WHEN 5 THEN H5.NODENAME

ELSE '' END) AS L4,

(CASE F.TLEVEL

WHEN 6 THEN H5.NODENAME

ELSE '' END) AS L5

FROM "/BIC/HBCGI0000" as F

LEFT OUTER JOIN "/BIC/HBCGI0000" H5 ON H5."NODEID" = F."PARENTID"

LEFT OUTER JOIN "/BIC/HBCGI0000" H4 ON H4."NODEID" = H5."PARENTID"

LEFT OUTER JOIN "/BIC/HBCGI0000" H3 ON H3."NODEID" = H4."PARENTID"

LEFT OUTER JOIN "/BIC/HBCGI0000" H2 ON H2."NODEID" = H3."PARENTID"

LEFT OUTER JOIN "/BIC/HBCGI0000" H1 ON H1."NODEID" = H2."PARENTID"

LEFT OUTER JOIN "/BIC/HBCGI0000" H0 ON H0."NODEID" = H1."PARENTID"

where F.IOBJNM = 'BCGI0000' and

F.OBJVERS = 'A'

) as zz

left outer join "/BIC/TBCGI0000" as t\_cgi\_lv1 on t\_cgi\_lv1."/BIC/BCGI0000" = zz.L1

left outer join "/BIC/TBCGI0000" as t\_cgi\_lv2 on t\_cgi\_lv2."/BIC/BCGI0000" = zz.L2

left outer join "/BIC/TBCGI0000" as t\_cgi\_lv3 on t\_cgi\_lv3."/BIC/BCGI0000" = zz.L3

left outer join "/BIC/TBCGI0000" as t\_cgi\_lv4 on t\_cgi\_lv4."/BIC/BCGI0000" = zz.L4

left outer join "/BIC/TBCGI0000" as t\_cgi\_lv5 on t\_cgi\_lv5."/BIC/BCGI0000" = zz.L5

left outer join "/BIC/TBCGI0000" as t\_cgi\_lv\_ on t\_cgi\_lv\_."/BIC/BCGI0000" = zz.nodename

SELECT TOP 1000 \* FROM "\_SYS\_BIC"."RetailAzs.KPI/CV\_HCGI\_LVLD/hier/hcgi\_lvld"

# Generating Hierarchies

[Содержание](#Содержание)

Currently, SAP HANA offers four hierarchy generator functions.

Most prominently /заметно/, the [***HIERARCHY*** Generator Function](https://help.sap.com/viewer/4f9859d273254e04af6ab3e9ea3af286/2.0.04/en-US/f29c70e984254a6f8df76ad84e78f123.html) covers the predominant case of generating *a hierarchy from a recursive parent-child source* data structure.

The [***HIERARCHY\_SPANTREE*** Generator Function](https://help.sap.com/viewer/4f9859d273254e04af6ab3e9ea3af286/2.0.04/en-US/6f17763614824969915e4329d1729047.html) uses the same source data format as the HIERARCHY function, but *generates* only a *minimal spanning tree*, which is useful *when the source* data *contains many multiple parent edges*.

The [***HIERARCHY\_TEMPORAL*** Generator Function](https://help.sap.com/viewer/4f9859d273254e04af6ab3e9ea3af286/2.0.04/en-US/0be52ba7971740feab5c0a330fced1d1.html) *operates on parent-child source* data *containing an additional validity interval*. This generates a hierarchy for a specific time interval.

The [***HIERARCHY\_LEVELED*** Generator Function](https://help.sap.com/viewer/4f9859d273254e04af6ab3e9ea3af286/2.0.04/en-US/5720ec4044ea4d47955a3dd21272a47b.html) generates an analogous output structure *based on source* data *where individual source columns correspond to hierarchy levels* and source rows define paths from a root to a leaf node.

The [***HIERARCHY\_COMPOSITE\_ID*** Scalar Function](https://help.sap.com/viewer/4f9859d273254e04af6ab3e9ea3af286/2.0.04/en-US/3a6f2a354e1c4dbcb4a80e7024c85481.html) facilitates working *with node identifiers consisting of multiple components*.

# HIERARCHY Generator Function

Generates a hierarchy based on recursive parent-child source data.

**Separating Hierarchy Model From Navigation**

Often multiple queries will be executed using the same hierarchy generator function result set. Therefore, it is recommended to either materialize the hierarchy generator function output into a temporary table or define a view over it. Materializing the hierarchy generator function output into a temporary table guarantees consistent and stable navigation results for the entire lifecycle of the temporary table and completely eliminates the hierarchy generation cost for subsequent queries

CREATE LOCAL TEMPORARY COLUMN TABLE **#h\_demo** AS (

SELECT \*

FROM HIERARCHY (SOURCE t\_demo SIBLING ORDER BY ord )

ORDER BY hierarchy\_rank );

SELECT hierarchy\_rank, parent\_id, node\_id FROM #h\_demo;

| **HIERARCHY\_RANK** | **PARENT\_ID** | **NODE\_ID** |
| --- | --- | --- |
| 1 |  | A1 |
| 2 | A1 | B1 |
| 3 | B1 | C1 |
| 4 | B1 | C2 |
| 5 | A1 | B2 |
| 6 | B2 | C3 |
| 7 | C3 | D1 |
| 8 | C3 | D2 |
| 9 | B2 | C4 |
| 10 | C4 | D3 |

Materializing a hierarchy into a temporary or persistent table cuts all connections between the hierarchy and its sources. If a query should return the most current state of the data, it is recommended to create a view over a hierarchy generator function, which guarantees that hierarchy navigation results always correctly reflect the current transactional view of the source data. If a hierarchy source is fully deterministic - that means that multiple query executions over the same data set always return the same result/, the hierarchy generator function output will be cached automatically. Caching avoids the overhead of recalculating the hierarchy if the sources do not change between subsequent navigations and at the same time guarantees a transactionally correct result

CREATE VIEW **h\_demo** AS

SELECT \*

FROM HIERARCHY ( SOURCE t\_demo SIBLING ORDER BY ord )

ORDER BY hierarchy\_rank;

SELECT hierarchy\_rank, parent\_id, node\_id

FROM h\_demo;

| **HIERARCHY\_RANK** | **PARENT\_ID** | **NODE\_ID** |
| --- | --- | --- |
| 1 |  | A1 |
| 2 | A1 | B1 |
| 3 | B1 | C1 |
| 4 | B1 | C2 |
| 5 | A1 | B2 |
| 6 | B2 | C3 |
| 7 | C3 | D1 |
| 8 | C3 | D2 |
| 9 | B2 | C4 |
| 10 | C4 | D3 |

Some of the hierarchy navigation functions can also work on partial subtrees of a hierarchy. To demonstrate their use, the view H\_DEMO\_B2 only contains the complete subtree under node B2.

CREATE VIEW subtree\_B2 AS

SELECT

hierarchy\_rank,

hierarchy\_tree\_size,

hierarchy\_parent\_rank,

hierarchy\_level,

hierarchy\_is\_cycle,

hierarchy\_is\_orphan,

parent\_id,

node\_id,

ord,

amount

FROM

HIERARCHY\_DESCENDANTS( SOURCE h\_demo START WHERE node\_id = 'B2');

SELECT hierarchy\_rank, parent\_id, node\_id

FROM subtree\_B2 ORDER BY hierarchy\_rank;

| **HIERARCHY\_RANK** | **PARENT\_ID** | **NODE\_ID** |
| --- | --- | --- |
| 5 | A1 | B2 |
| 6 | B2 | C3 |
| 7 | C3 | D1 |
| 8 | C3 | D2 |
| 9 | B2 | C4 |
| 10 | C4 | D3 |

In an analog way, the view H\_DEMO\_ERR is defined as

CREATE VIEW h\_demo\_err AS

SELECT \*

FROM HIERARCHY ( SOURCE t\_demo\_err SIBLING ORDER BY node\_id )

ORDER BY hierarchy\_rank;

SELECT hierarchy\_rank, parent\_id, node\_id

FROM h\_demo;

| **HIERARCHY\_RANK** | **PARENT\_ID** | **NODE\_ID** |
| --- | --- | --- |
| 1 |  | A1 |
| 2 | A1 | B1 |
| 3 | B1 | C1 |
| 4 | B1 | C2 |
| 5 | A1 | B2 |
| 6 | B2 | C3 |
| 7 | C3 | D1 |
| 8 | C3 | D2 |
| 9 | B2 | C4 |
| 10 | C4 | D3 |

**Caution**

Please be aware that hierarchy navigation functions consuming views over hierarchies as a source (or hierarchy generation function results directly) require fully deterministic sources for correct operation, including a stable sort order of sibling nodes. Otherwise, it cannot be guaranteed that hierarchy ranks calculated within the source specification semantically match the hierarchy ranks calculated within in the START / START WHERE specification, leading to non-deterministic and wrong results. This also applies to hierarchy self-joins and similar operations.

For compatibility reasons, a strict enforcement of the <hierarchy\_genfunc\_order\_spec> is not possible. Therefore, a potential determinism issue is indicated by a warning:

Warning 654:

no row order on table set:

hierarchy generation function result is partially non-deterministic due to missing SIBLING ORDER BY clause or missing ORDER BY clause in SOURCE specification.

However, it cannot be detected if a SIBLING ORDER BY clause is present, but does not ensure a deterministic sibling sort order.

# HIERARCHY\_ANCESTORS\_AGGREGATE

# <http://sapabapcentral.blogspot.com/2020/01/abap-cds-recursive-association-using-hierarchy.html>

## **[SAP HANA SQL extension for generating a tree hierarchy](https://tmilinovic.wordpress.com/2019/04/27/sap-hana-sql-extension-for-generating-a-tree-hierarchy/)**

<https://tmilinovic.wordpress.com/2019/04/27/sap-hana-sql-extension-for-generating-a-tree-hierarchy/>

[Содержание](#Содержание)

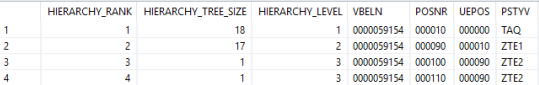
The multitude /множество/ of hierarchical data already exist in most real-world database installations but is not explicitly defined as hierarchical data.

Due to the limitations of the relational model and SQL, logic for hierarchy handling within these applications has mostly been written in ABAP and therefore runs within the application server.

**Adjacency List Model**

The simplest method of storing and maintaining a tree of this type is for each record to include a reference to its *parent* - you may see this referred to as an Adjacency List Model.

The table in the figure 1 represents a sales order - *VBELN* consisting of bill of materials - *BOM* /спецификация/. Field *POSNR* - item number uniquely identifies each part. The *hierarchical relationship* is established by a self-reference item number *UEPOS* - higher-lev.item associating each row with its respective parent row, the part containing it



A hierarchy can be ***leveled*** - *fixed number of levels* or ***recursive*** -*various number of levels*.  
In our case, delivery elements ZTE1 and ZTE2 represents recursive hierarchy that could have 1 or 2 levels. Representing a tree in this way allows us to easily write simple queries to access the nodes immediately adjacent to a given node /*above or below them in the hierarchy*/ but requires us to use *recursion* if we wish to query data spanning further levels of the tree.

**Recursive Common Table Expressions - RCTEs**

*The only standard hierarchy-handling tools* that SQL-based DBMSs today offer are *Recursive Common Table Expressions - RCTEs* as defined in SQL1999 standard.

As a generic mechanism, RCTEs have interesting uses far beyond traversing hierarchical data. On the other hand, the general problem with RCTEs is that the *recursive join must iterate over large subtrees* of the hierarchy, yielding large intermediate results, only to find that there are almost no matching parts in these subtrees.

***SAP HANA hierarchy functions*** are native and fully integrated SQL extensions with focus on tree-like data topologies. Due to that limited scope, the offered hierarchy functions and algorithms can be optimized easier and can *offer an increased performance* *over generic graph algorithms*, but *may not be ideal* for *generic mesh-like topologies* in which case we can use SAP HANA Graph.

SAP *HANA supports three types of hierarchy functions* - *generator*, *navigation* and *scalar*.  
In this article I am focusing on HIERARCHY generator function that generates a hierarchy based on recursive parent-child source data.

The ***HIERARCHY*** generator function *generates a hierarchy* by deriving it *from a given adjacency-list-formatted source table*, which may be a table, a view, or the result of a subquery. Each row defines a parent-child relation and attributes of a node or edge that may become part of the output table, provided that they are reached during traversal.

The source query, should have a *node\_id* and a *parent\_id* column to identify the relation in the hierarchical data structure.

The *structure of such a hierarchy* is always the same, regardless of the original format of the source data, and *consists of an ordered list of* its *nodes*. The nodes themselves are represented by a minimal set of orthogonal hierarchy topology attributes plus a projection of the original source attributes.

The *HIERARCHY* generator function can be used *wherever a table reference is allowed* - in particular, a *FROM* clause. Its result is a *temporary table* containing the data from the source table plus an additional *NODE* column. The function is evaluated

1. *by first* - *self-joining the source table* in order *to derive a parent-child relation* representing the edges,
2. *then* - building a *temporary hierarchy* representation from that,
3. and *finally* - producing the corresponding *NODE* column.

The ***START WHERE*** subclause can be used to *restrict the hierarchy* to *only the nodes* that are reachable from any node satisfying start condition.

The ***SEARCH BY*** subclause can be used to specify a desired sibling order; if omitted, siblings are ordered arbitrarily.

Most *legacy* /устаревших/ applications rely entirely on views *featuring* /содержащие/ HIERARCHY expressions on top of adjacency lists, thus avoiding any schema changes. Hence, generating hierarchy is, at least conceptually, used on each view evaluation; though it *may be elided* /можно игнорировать/ often in practice, since HANA employs view caching.

A *hierarchy-aware joins* work on the join-optimized *NODE column*, while *RCTEs* must necessarily work on the *key column*. Because of that, a *hierarchy-aware join does not need to enumerate whole subtrees* to find matching nodes; thus, the predicate can be pushed down to the table scans and only parts that meet the filter condition participate in the join in the first place. That makes a hierarchy-aware join so fast that its execution is still much faster than the RCTE even if we always generate a temporary hierarchy representation prior to executing the query.

**Example**  
The following SQLScript anonymous block uses hierarchy function for finding all delivery items

**DO**

**BEGIN**

main\_dlv\_items =

**select** vbap.vbeln **as** vbeln, vbap.posnr, vbap.uepos, vbap.pstyv, vbap.vrkme,

mska.kalab, mska.lgort, mska.charg, mska. Sobkz

**from** vbap

***inner******join*** vbak **on** vbak.mandt=vbap.mandt **and** vbak.vbeln=vbap.vbeln

***left******join*** mska **on** mska.mandt=vbap.mandt **and** mska.vbeln=vbap.vbeln **and**

mska.posnr=vbap.posnr **and** mska.kalab=vbap.kwmeng **and** mska.werks=’1000′ **and**

mska.kalab>0  
 **where** vbap.mandt=’250′ **and** vbap.pstyv **in** (‘TAQ’,’TAN’,’ZTE1′,’ZTE2′)**and**

vbak.auart=’ZOR’ **and** vbak.netwr=0 **and** vbak.kunnr=’0000055555′;

**SELECT** hierarchy\_rank, hierarchy\_ tree\_size, hierarchy\_level,

vbeln, posnr, uepos, pstyv

**FROM *HIERARCHY***

(**SOURCE** (**select** main.vbeln||main.uepos **as** *parent\_id*,

main.vbeln||main.posnr **as** *node\_id*, main.pstyv, main.vbeln, main.posnr, main.uepos,

main.kalab, main.lgort

**from** :main\_dlv\_items **as** main

)

**START WHERE** main.uepos=’000000′ **and** main.kalab > 0

**SIBLING** **ORDER** **BY** node\_id

);

**END**;

### **ABAP SQL- Grouping&Windowing, Hierarchies, Geo spatial Data type**

<http://karthikkennedys.blogspot.com/2019/11/abap-sql-grouping-hierarchies-geo.html>

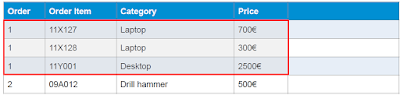
In this post we will look into windowing and hierarchy concepts in SAP ABAP SQL for HANA.  
But, before do check my previous blog on [ABAP SQL for HANA](https://karthikkennedys.blogspot.com/2019/11/abap-sql-for-sap-hana.html) covering why ABAP SQL, Inline declarations, Host variables, and CTE's.

### **Windowing**

Windowing *is dividing data sets into subsets*. You can introduce ordering on your subsets and use it for access navigation,aggregate information over a subset without the need for grouping

Example

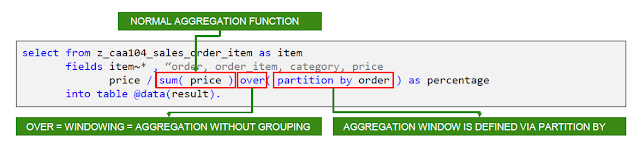
let's consider the below data model,



Now, windowing allows us to segregate based on key elements and do aggregation over the same as below,

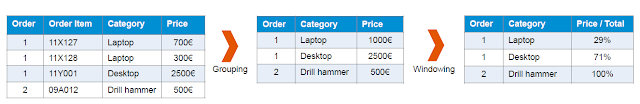


The sample code for the above window function  will be

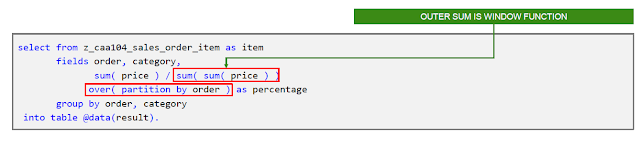


Generally, *windowing concept* in code comes into play, after grouping - *similar items are grouped* and *the window function* created to have *aggregates over them*.

**A sample** **Grouping and Windowing scenario**



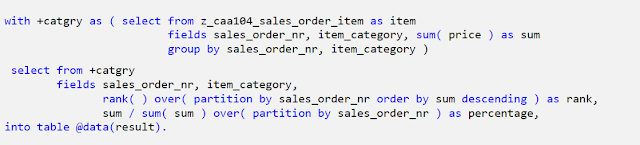
The ABAP sql to achieve above will be as follows:



Similarly, Windowing allows certain special operations, some of them were as follows

* ALL CLASSICAL AGGREGATE FUNCTIONS
* *RANK* in a window, may contain gaps, if some rows are equal
* *DENSE\_RANK* as RANK but, without gaps
* *ROW\_NUMBER* - numbering of each row
* *LEAD*- access to a subsequent line in a window
* *LAG*- access to a prior line in a window

Classical example of using rank in the above data model explained.



### Hierarchies:

Now, lets dive into hierarchies:

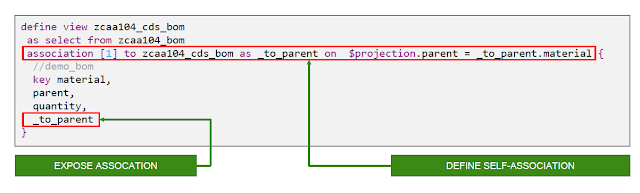
This won't be a new concept for people working with CDS views. Its been around a while.

Hierarchies as the name defines represents arranging the data sets with self-associations into a tree-model. Henceforth, allows Easily follow relations in your data set over arbitrary association steps, aggregate information along hierarchical relations and work on sub trees.

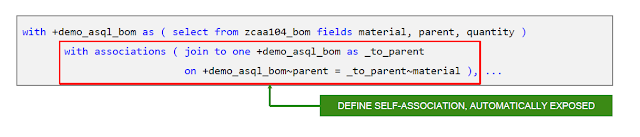
Lets proceed by taking the simple example below:

|  |
| --- |
|  |
| Hierarchy |

Now, the hierarchy source will be

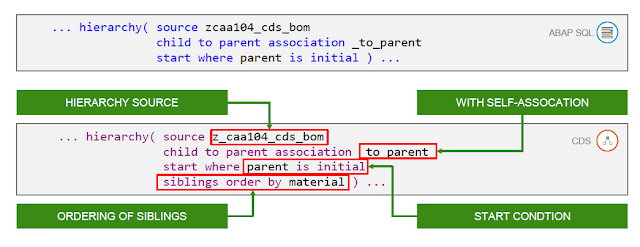
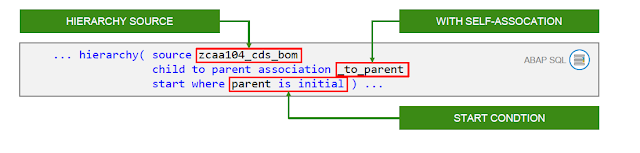


**Hierarchy self association,**

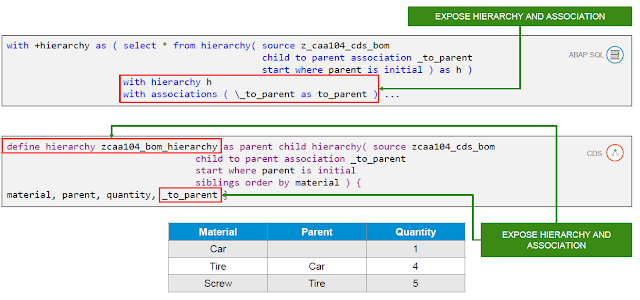


**Hierarchy Definition:**

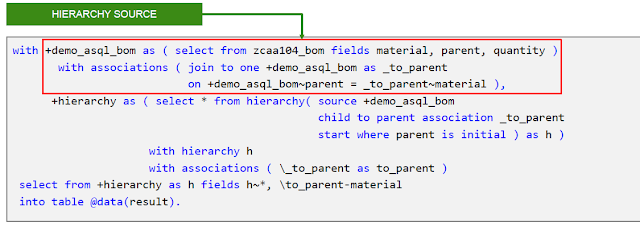
The definition starts with the hierarchy source, with association, and start condition.



**Exposing Hierarchy:**



**Hierarchies -- Select:**



**Hierarchies -Accessor Functions:**

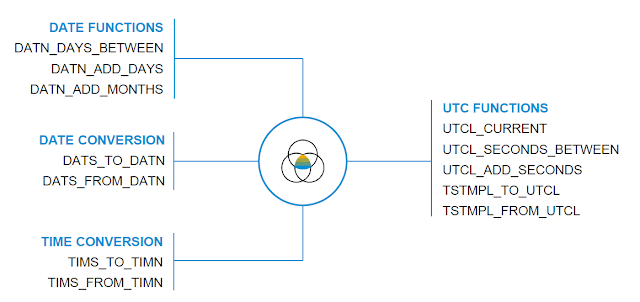
Now, to access data cells within hierarchy we have special accessor functions.

* HIERARCHY\_DESCENDANTS--Navigate through descendants
* HIERARCHY\_ANCESTORS--Navigate through ancestors
* HIERARCHY\_SIBLINGS--Navigate through siblings
* HIERARCHY\_DESCENDANTS\_AGGREGATE--Navigate through descendants with aggregation
* HIERARCHY\_ANCESTORS\_AGGREGATE--Navigate through ancestors with aggregation

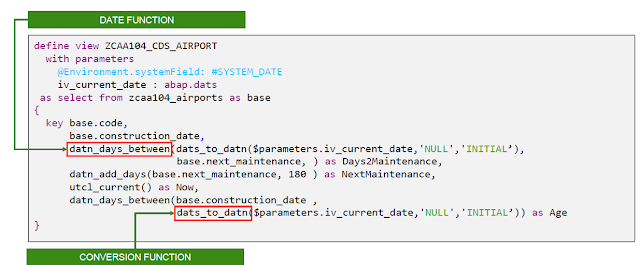
|  |
| --- |
|  |
| Hierarchy accessor function example |

**Built -in Functions:**

We know certain built-in functions for CDS views,.



Example using date function and conversion function,

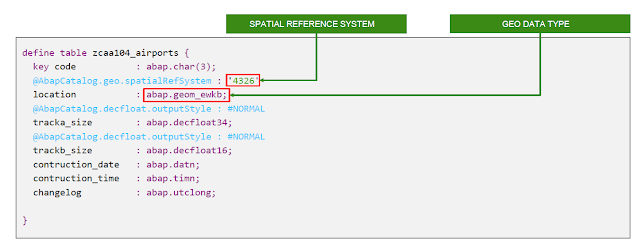


### Geo Type:

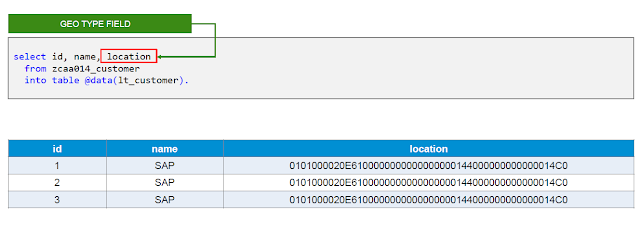
There is a new geo type in DDIC which allow usage in SAP HANA artefacts(DB Table, CDS View, SQL Query). NOGEO-specific semantics on ABAP-level. ABAP just allows dispatching of GEO data between consumers and SAPHANA.

**GEO-specific functionality in SAPHANAcan be used via AMDP**

Defining database table using Geo spatial data.



But, internally the geo-spatial data will be stored as type of coordinates.



**The Navigation Functions**

[Содержание](#Содержание)

The three navigation functions work on an existing hierarchy table and return a hierarchy table themselves

1. ***HIERARCHY\_DESCENDANTS***()
2. ***HIERARCHY\_ANCESTORS***()
3. ***HIERARCHY\_SIBLINGS***()

They use these parameters

* SOURCE – The HIERARCHY-Table.
* START – The start of the navigation. This can be multiple nodes.
* DISTANCE – To limit the results to a certain distance from the START. Not available for   
  HIERARCHY\_SIBLINGS.

***Demo - Hierarchy functions to fill the Parent /(grand-)children table***

do begin

lt\_hier = select \*

from *hierarchy*( source ( select nodeid as node\_id, parentid as parent\_id

from hier\_pc )

start where parentid is null ); *--root-node*

select \* from :lt\_hier; *--to display intermediate values*

lt\_tmp = select start\_id as nodeid, node\_id as childid

from *hierarchy\_descendants*

( source :lt\_hier

start ( select hierarchy\_rank as start\_rank, *--required name*

node\_id as start\_id

from :lt\_hier ) )

order by nodeid asc, childid asc ;

select \* from :lt\_tmp; *--to display the result*

**Algorithms to test**

***Lookup Parent/Child with hierarchy function***

create procedure ***hier\_lookup\_pc***(in iv\_nodeid int,

out et\_result *table*(nodeid int, nodename nvarchar(30) ) )

as begin

et\_result = select node\_id as nodeid, nodename

from *hierarchy\_descendants* ( source hier\_hf

start where node\_id = :iv\_nodeid );

end

**The Aggregate Functions**

[Содержание](#Содержание)

The hierarchy ... aggregation function provides optimized hierarchical aggregate capabilities. By reusing results of subordinate nodes, all aggregates can be calculated by one single linear index traversal.

The aggregate functions are parameterized like the navigation functions, but you can also define aggregated columns, a join to a fact table and specify some extra features, like subtotals.

1. ***HIERARCHY\_DESCENDANTS\_AGGREGATE***()
2. ***HIERARCHY\_ANCESTORS\_AGGREGATE***()

WITH h AS (

SELECT \* FROM *HIERARCHY*(

SOURCE (select zz.nodeid as node\_id, zz.parentid as parent\_id

from "SAPABAP1"."/BIC/HBCGI0000" as zz

)

START WHERE zz.tlevel >= 0

SIBLING ORDER BY node\_id

)

)

SELECT

node\_id,

path

FROM HIERARCHY\_ANCESTORS\_AGGREGATE (

SOURCE h

MEASURES ( string\_agg(node\_id, '/') AS path ) )

ORDER BY node\_id;

WITH h AS (

SELECT

h\_cgi.HIERARCHY\_RANK, h\_cgi.HIERARCHY\_TREE\_SIZE, h\_cgi.HIERARCHY\_PARENT\_RANK,

h\_cgi.HIERARCHY\_ROOT\_RANK, h\_cgi.HIERARCHY\_LEVEL, h\_cgi.HIERARCHY\_IS\_CYCLE,

h\_cgi.HIERARCHY\_IS\_ORPHAN, h\_cgi.NODE\_ID, h\_cgi.PARENT\_ID,

h\_cgi.TLEVEL,

sls."/BIC/BEMIT000", sls.calyear, sls.calmonth2,

--sum(sls."/BIC/BSSLS000") as "/BIC/BSSLS000"

sls."/BIC/BSSLS000"

FROM HIERARCHY(SOURCE (select nodeid as node\_id, parentid as parent\_id, tlevel from "SAPABAP1"."/BIC/HBCGI0000" as h\_bw\_cgi)

START WHERE h\_bw\_cgi.tlevel >= 0 --and sls.calyear = 2021 and sls."/BIC/BEMIT000" = 020

SIBLING ORDER BY node\_id

) as h\_cgi

left outer join "SAPABAP1"."/BIC/ABDNOM0002" as nom

on nom."/BIC/BCGI0000" = h\_cgi.node\_id

left outer join "SAPABAP1"."/BIC/ABDSLS0002" as sls

on sls."/BIC/BNOM0000" = nom."/BIC/BNOM0000"

where sls."/BIC/BEMIT000" = 020 and sls.calyear = 2021 and sls.calmonth2 = 11

--group by

-- h\_cgi.HIERARCHY\_RANK, h\_cgi.HIERARCHY\_TREE\_SIZE, h\_cgi.HIERARCHY\_PARENT\_RANK,

-- h\_cgi.HIERARCHY\_ROOT\_RANK, h\_cgi.HIERARCHY\_LEVEL, h\_cgi.HIERARCHY\_IS\_CYCLE,

-- h\_cgi.HIERARCHY\_IS\_ORPHAN, h\_cgi.NODE\_ID, h\_cgi.PARENT\_ID,

-- h\_cgi.TLEVEL,

-- sls."/BIC/BEMIT000", sls.calyear, sls.calmonth2

)

SELECT

node\_id,

path

FROM HIERARCHY\_ANCESTORS\_AGGREGATE (

SOURCE h

MEASURES ( string\_agg(node\_id, '/') AS path ) )

ORDER BY

node\_id;



"EKIMENKO"."RetailAzs::CV\_GDS\_WITH\_CGI\_H"

### **SAP HANA Level Hierarchy**

[Содержание](#Содержание)

<http://kabilsapworld.blogspot.com/2017/09/what-is-hierarchy-hierarchies-helps.html>

**Level hierarchies** consist of one or more *levels of aggregation*. Attributes roll up to the next higher level in a many-to-one relationship and members at this higher level roll up into the next higher level, and so on until they reach the highest level.

A hierarchy typically comprises of several levels, and you can *include a single level* in more than one hierarchy. A level hierarchy is rigid in nature, and you can access the root and child node in a defined order only.

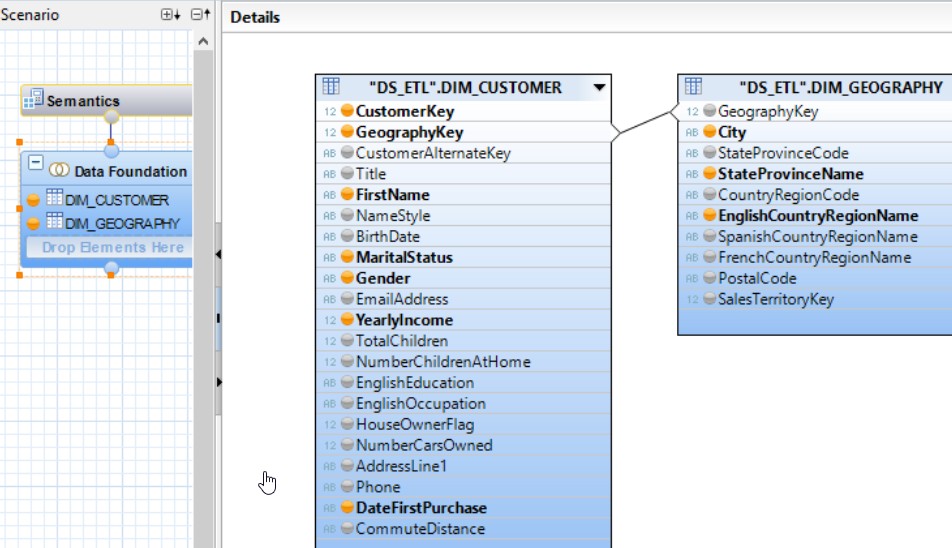
For example, a **time hierarchy** comprises of *levels* such as Fiscal *Year*, Fiscal *Quarter*, Fiscal *Month*, and so on.

## **Use Case**

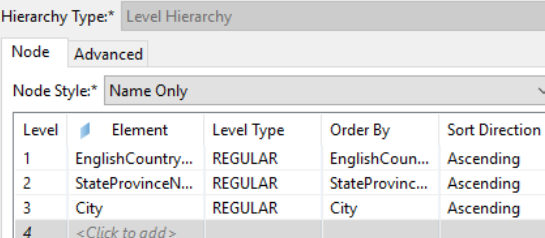
Customer wants to *analyze the sales revenue* by customer *country*, *state,* and *city.*

Now we will create a *level hierarchy* in our *attribute view* AT\_CUSTOMER\_GEOGRAPHY and access that using MS Excel to analyze the sales revenue.

1. Drag and Drop the required tables in a Data Foundation and make the link between them based on a relationship. Here, I used two tables - ***Dim\_Customer*** and ***Dim\_Geograpghy***. And join it by using *Geography\_Key* field



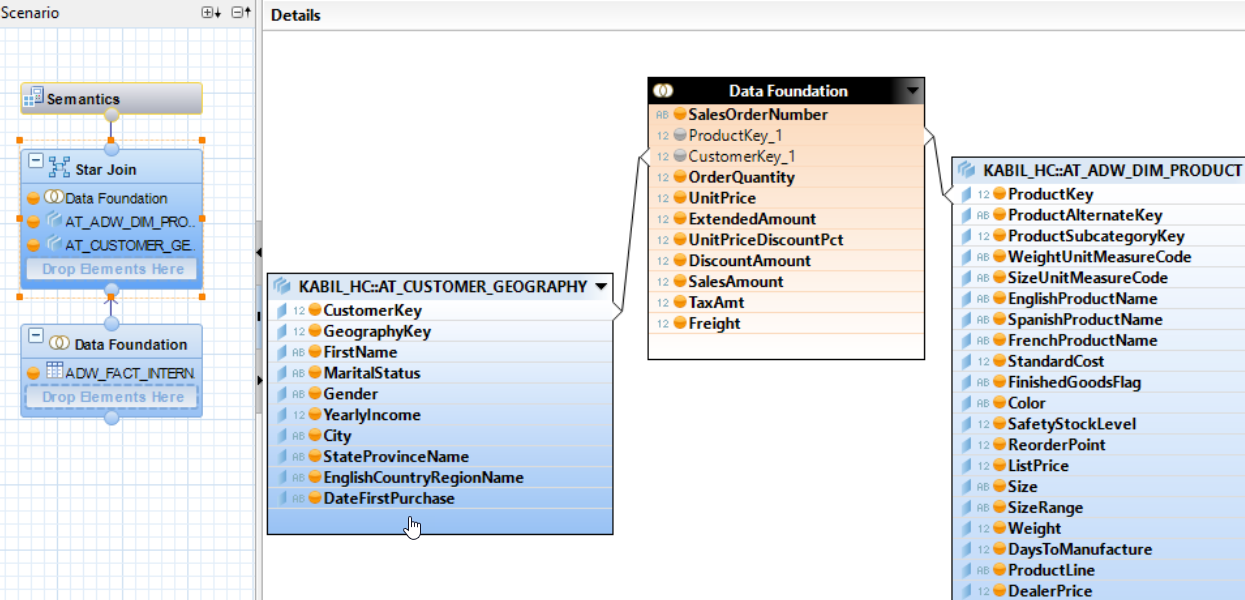
1. And now we create *level hierarchy* *HI\_CUS\_GEO* based on Geography location like *countr*y, *state,* *city*



1. Then Validate and Activate the view. And create another *attribute view* *AT\_ADW\_DM\_PRODUCTS* for Products - here I used Product table alone

## **To Create an *Analytic View* AN\_ADW\_FACT\_INTERNET\_SALES**

1. Drag and Drop the required tables in a Data Foundation and make the link between them based on a relationship. Here, I used two Attribute views - *AT\_CUSTOMER\_GEOGRAPHY* and *AT\_ADW\_DM\_PRODUCTS* which we are created above.
2. Drag and Drop the Fact table which contains measures in another Data Foundation. Here, I used *ADW\_FACT\_INTERNET\_SALES*. Then join this to the attributes based on the relationship



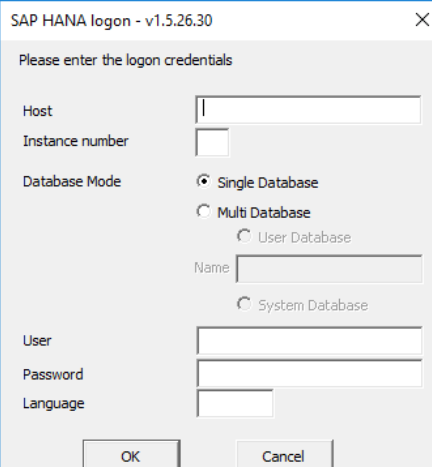
1. Then Validate and Activate the view.

## **Excel**

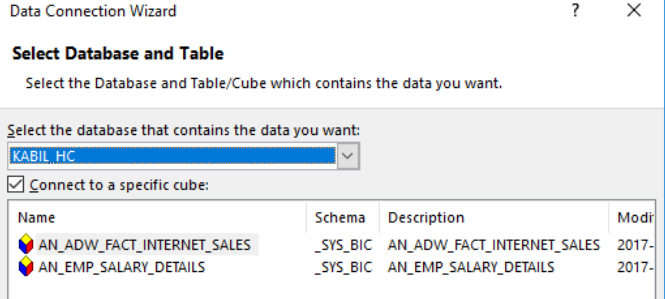
Open Excel and Click *Data* tab and select *From Other Sources* and Choose *From Data Connection Wizard*.

A Pop-up will appear. And choose *Other/Advanced* and Click *Next*.

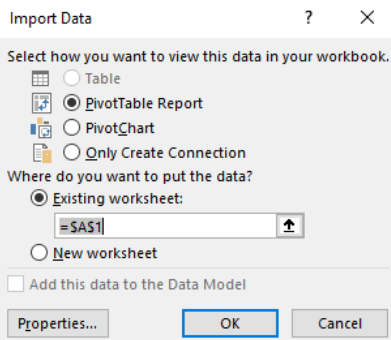
A Pop-up will appear *Data Link properties* select ***SAP HANA MDX Provider*** and Click Ok. A Pop-up window will appear as shown as below and provide required details

 and click ok.

A pop-up window will appear from this select your view which contains hierarchy. Here I choose AN\_ADW\_FACT\_INTERNET\_SALES

 and click Next.

Click Finish.

Another *Import Data* pop-up will appear to select how we want to view our data from this select *Pivot /сводная/ table Report*  and click Ok.

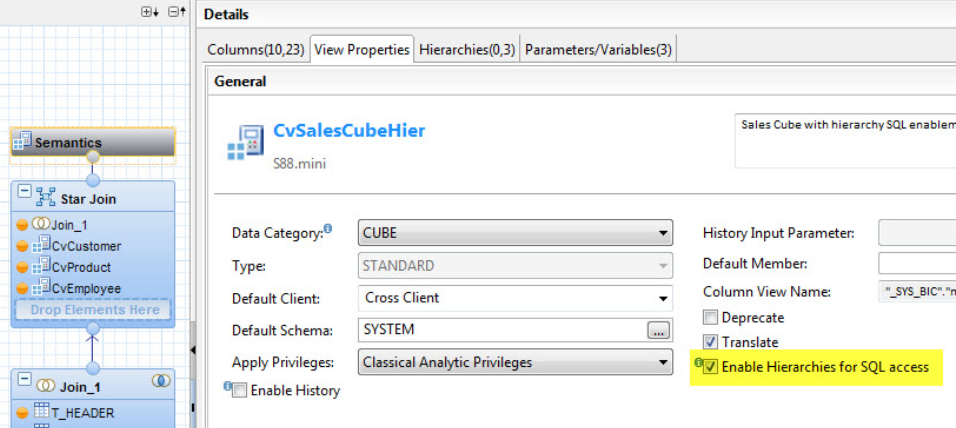
# New Hierarchy SQL enablement with Calculation Views in SAP HANA 1.0 SPS 10

[Содержание](#Содержание)

<https://blogs.sap.com/2015/07/09/new-hierarchy-sql-enablement-with-calculation-views-in-sap-hana-10-sps-10/>

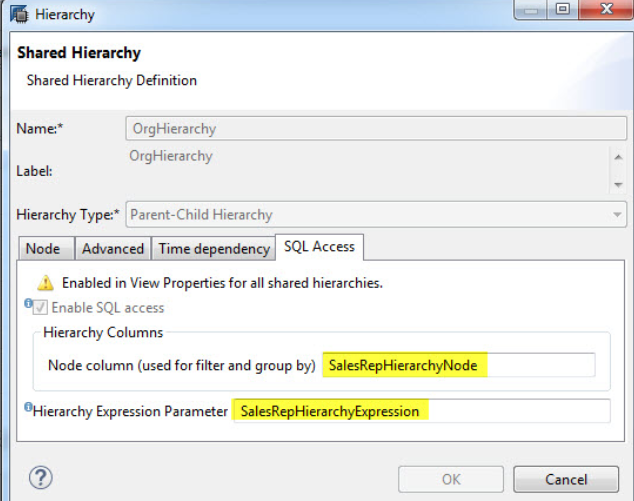
With SAP HANA SPS 10 CV provide a deeper integration of hierarchy objects and their exposure for usage within SQL. By leveraging the SQL integration of hierarchy objects, hierarchy-based *filters*, *aggregations* and hierarchy-driven *analytic privileges* are enabled.

## **Enabling hierarchies for SQL access**

Within the SAP HANA CV general properties, there is a new checkbox ***Enable Hierarchies for SQL access*** 

With that *hierarchy views from shared dimensions* used in a Star Join CV are enabled for SQL access.

Once activated, in the open hierarchy dialog for shared hierarchies /available in the semantic node of star join calculation views/, the new *SQL Access* tab is activated and proposes names for a new ***Node Column*** and a ***Hierarchy Expression Parameter*** for each shared hierarchy

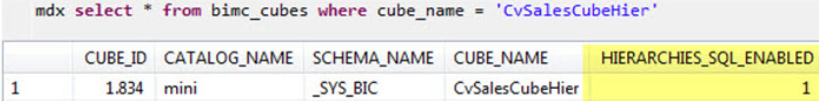
.

Alternatively to the general enablement of all shared hierarchies of a Star Join calculation view, hierarchies may be enabled individually too.

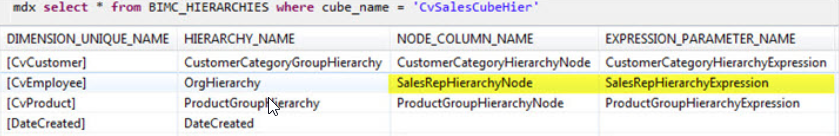
The ***hierarchy node column*** can be *referenced for filtering and aggregations* as the examples below will highlight.

The ***hierarchy expression parameter*** will be used to *leverage hierarchy-specific expressions in SQL queries against calculation views*. This capability will however only be fully enabled, once *hierarchy-expression* are documented in the SAP HANA SQL documentation.

The SAP HANA CV metadata tables indicate that SQL access is enabled in the flag *HIERARCHIES\_SQL\_ENABLED*of the analytic catalog view *BIMC\_CUBES*. With that, BI clients and query tools will be informed whether they can access the hierarchy data by querying the CV

.

Additionally, the node column and the expression parameter-names are documented in the view ***BIMC\_HIERARCHIES*** of the analytic catalog



**Note**

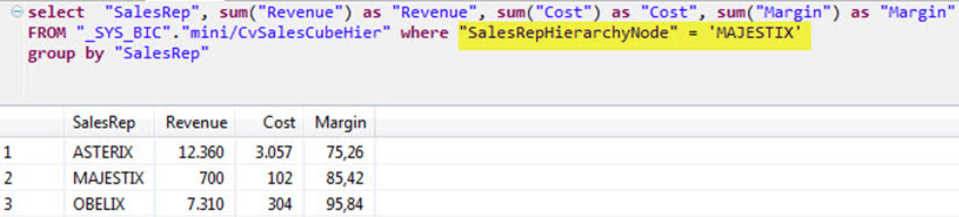
The analytic metadata queries prefixed with *MDX* <select from BIMC…> ensure consistent semantic information incl. for example localized descriptions.

## **Using hierarchies for filtering and aggregation within SQL queries**

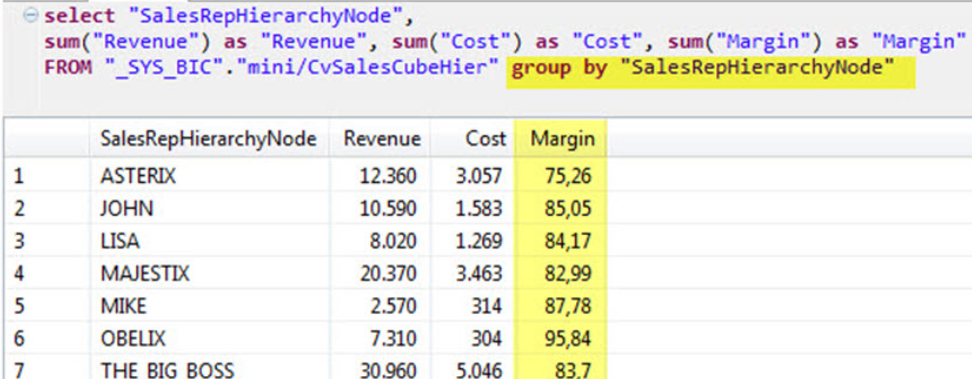
Now, let’s have a look at examples of how hierarchies enabled for SQL access can be leveraged in SQL queries.

Imagine the following hierarchy .

We can now reference the new *node column* as SalesRepHierarchyNode to filter the result set of our SalesRep-based hierarchy to the subtree of the filter specified



Furthermore, the *new node column* can be used within SQL aggregations in *group b*y clauses and thus additionally ensures the correct evaluation of calculated columns

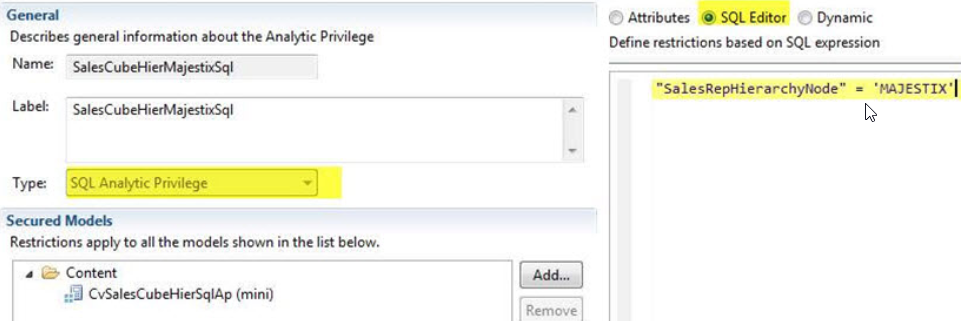


In the example shown the Revenue and Cost column contain the hierarchical aggregated values. The Revenue for MAJESTIX is the *sum* of the revenue of all its descendants /compare this with the Revenue in the filter query above that was grouped by SalesRep i.e. without *hierarchy aggregation*/.

## **Hierarchy-driven Analytic Privileges**

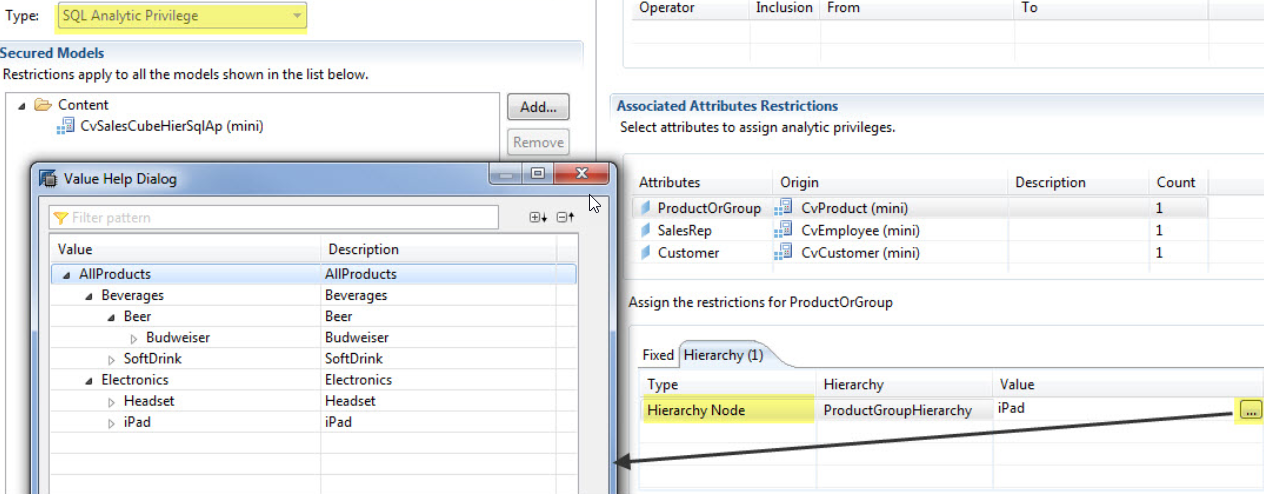
With SAP HANA SPS 10, in addition to the existing approach of creating classical analytic privileges, users can now create SQL-based analytic privileges as design-time objects, thus providing the flexibility to create analytic privileges restriction based on SQL expressions allowing a simpler way of implementing more complex filtering rules.

For Calculation View hierarchies, which have been enabled for SQL access as detailed out above, within the SQL expression editor of SQL-based Analytic Privileges, you can maintain a filter expression referencing the hierarchy node column



Because of using the node column this privilege will filter the subtree of the node at runtime and thus behave as a hierarchical analytic privilege.

With SAP HANA SPS 11 the hierarchical privileges can also be defined in the form based part of the editor. There you don’t have to write SQL and you even get a hierarchical value help



**Current Restrictions**

For parent-child hierarchies, compound hierarchies are not supported, i.e. the parent child hierarchies must only have a single attribute as parent and child.

**Further new hierarchy-related capabilities**

You find more information about new hierarchy capabilities in SAP HANA calculation views like ***time-dependent hierarchies*** and use of hierarchies with variable- and input parameter-definitions as part of the SAP HANA SPS 10 new capabilities blog /overview [here](https://blogs.saphana.com/2015/06/16/new-sap-hana-sps10/)/, specifically in the What’s new in SAP HANA Modeling presentation [here](http://de.slideshare.net/SAPTechnology/sap-hana-sps10-hana-modeling-1-49804522).

# Query Hierarchy Data in SAP HANA Database using SQLScript

[Содержание](#Содержание)

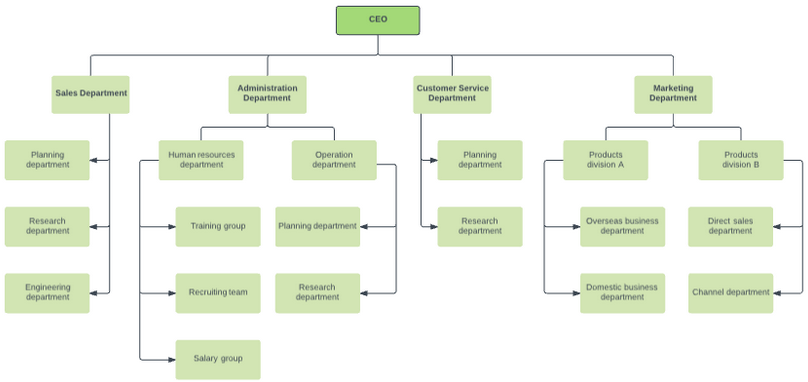
<https://www.kodyaz.com/sap-abap/query-hierarchy-data-using-sqlscript-on-hana-database.aspx>

With SAP HANA 2 SP02, SQLScript enhancements on HANA database enables SQL developers to query hierarchy data stored as *parent-child* relation in database tables easily using SQLScript *Hierarchy* function.

SQL Server provided a similar hierarchy function and enabled the use of [*recursive CTE queries* for hierarchy data](https://www.kodyaz.com/t-sql/sql-server-recursive-query-with-recursive-cte.aspx) for long time ago.

## **Hierarchy Data as Parent-Child Rows in SAP HANA Database**

The hierarchical data that I wanted to create in SAP HANA database using above SQL DML commands on parent-child table Organization is following organizational structure of a fictional company



I used *identity value* for the ID column, SQL developers can refer to [*SQLScript Identity Column*](https://www.kodyaz.com/sap-abap/hana-database-table-with-identity-column.aspx)for details.

create Column Table Organization (

  id int *generated by default as identity*(start with 1 increment by 1) not null,

unit varchar(100) Not Null,

parentID int

);

insert into Organization (unit, parentId) values ('CEO', NULL);

## **Query Hierarchy Data using SQLScript Hierarchy Function on HANA Database**

On a parent - child table, SQL programmers can easily use SQLScript function **Hierarchy** to resolve the hierarchical ranks, relations, levels of a node easily.

The source query, should have a ***node\_id*** and a ***parent\_id*** column to identify the relation in the hierarchical data structure.

select \* from *HIERARCHY* (

 source ( select id as *node\_id*,  parentid as *parent\_id*,  unit

from Organization

 )

)

The most important columns of the Hierarchy function output table is ***Hierarchy\_Level*** and ***Hierarchy\_Parent\_Rank***.

**Hierarchy\_Level** is showing the position of levels from top to down of the current node in the hierarchical structure.

## **List Parent Nodes or Ancestors of a Hierarchy Node on SAP HANA Database**

So if you are searching for the upper-level managers or upper organizational units in an organizational structure, which SQL query HANA database developers should use?

SQLScript enhancements introduced SAP HANA database programmers the hierarchy function ***HIERARCHY\_ANCESTORS*** to query nodes in a hierarchical structure for parent nodes

select \* from *HIERARCHY\_ANCESTORS* (

 SOURCE HIERARCHY ( source ( select  id as node\_id, parentid as parent\_id, unit

   from Organization

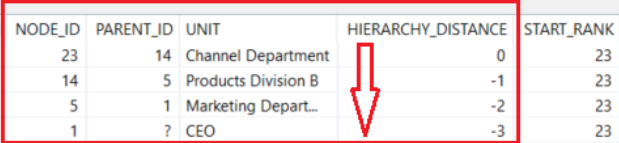
  )

)

 START WHERE node\_id = 23

)

Please note that the ***hierarchy\_distance*** column *shows the levels between two nodes*. *For parent nodes*, the distance is expressed in *negative values*



Let's combine our knowledge and list only the parent node names /unit descriptions/ in order starting from top to bottom seperated by *>* character

select *string\_agg*(unit, ' > ' order by *hierarchy\_rank*) as bottom2up

from *HIERARCHY\_ANCESTORS* (

 SOURCE HIERARCHY ( source ( select id as node\_id, parentid as parent\_id, unit

 from Organization

  )

  )

 START WHERE node\_id = 23

)

Here is the output - *CEO > Marketing Department > Products Division B > Channel Department*

## **Display Child Nodes of a Given Hierarchy Node using SQL**

Let's this time, display the child nodes of a given node in the hierarchy using SQLScript hierarchy functions on our sample HANA database table.

Assume business requirement needs *to see the child nodes of Marketing Department* which has the node ID value as 5.

This time we will use SQL ***HIERARCHY\_DESCENDANTS*** functions.

I want to create a string output variable with *indents* /'ɪndent делать отступ/ according to the node level compared to top most parent node *Marketing Department*

Simple SQL query is as follows

select \* from *HIERARCHY\_DESCENDANTS* (

 SOURCE HIERARCHY ( source ( select id as node\_id, parentid as parent\_id, unit

   from Organization

   )

  )

 START WHERE node\_id = 5

But for inserting indents showing visually the hierarchy between node, SQLScript programmers can modify the above SELECT statement as follows

select

*string\_agg*(LPAD('', HIERARCHY\_DISTANCE, CHAR(9)) || unit || BINTOSTR(HEXTOBIN('0D0A')),

   '' order by *hierarchy\_rank*) as topdown

from *HIERARCHY\_DESCENDANTS* (

 SOURCE HIERARCHY (

  source ( select id as node\_id, parentid as parent\_id, unit

   from Organization

 )

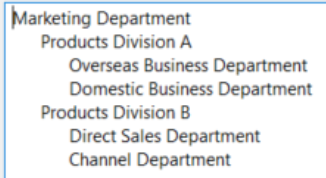
 )

  START WHERE node\_id = 5

)

I added TAB character which can be managed by using CHAR(9). The carriage return character can be managed by using BINTOSTR(HEXTOBIN('0D0A')).

The output of the above SQLScript code execution on tutorial sample hierarchy data will be as follows



## **Display Sibling Nodes on HANA Hierarchy Table using SQLScript**

SQL programmers can use another Hierarchy function named ***HIERARCHY\_SIBLINGS*** function to query a given hierarchical structure for the nodes sharing the same level or for siblings of a given parent-child node.

For example, if we want to get the same level nodes with *Marketing Department*

select node\_id, parent\_id, unit

from *HIERARCHY\_SIBLINGS* (

 SOURCE HIERARCHY ( source ( select id as node\_id, parentid as parent\_id, unit

   from Organization

 )

)

 START WHERE node\_id = 5

And the output will be just as we wanted to have



For more on Hierarchy function SQLScript database developers can refer to [official documentation](https://help.sap.com/viewer/4fe29514fd584807ac9f2a04f6754767/2.0.01/en-US/fc59f81a5c494f399cc2ff70b9c3b4c9.html).

In the reference documentation, SQL programmers developing on SAP HANA database will see two more hierarchy functions namely ***HIERARCHY\_LEVELED*** function and ***HIERARCHY\_COMPOSITE\_ID*** function.

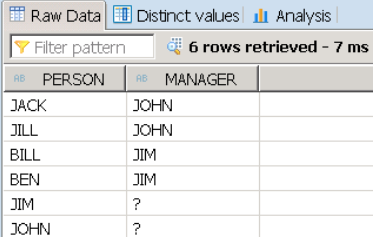
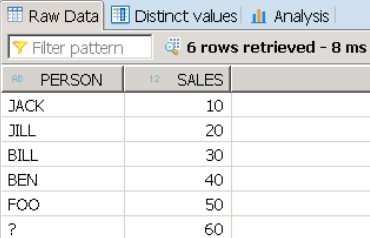
**Not Assigned Members enabled hierarchies with SAP HANA Calculation Views**

<https://learntips.net/unassigned-member-null-handling-hierarchy-in-calculation-view/>

[Содержание](#Содержание)

Often occurs that there will be *entries in fact table* for which corresponding master data doesn’t exist, for instance sales records for unregistered customers. In such cases when the two tables joined together with referential integrity, sales data for unregistered customers will be lost. Using the feature***Unassigned Member Null Handling*** in hierarchy, we can see the whole data in a hierarchical fashion without any loss of data wherein the *not assigned member*/*unregistered* customer details will be grouped under a node.

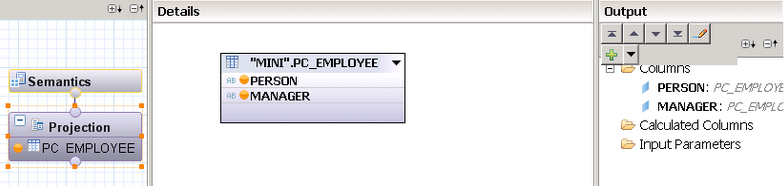
I am going to use below data tables PC\_EMPLOYEE and PC\_SALES. PC\_EMPLOYEE has records in a *parent child relationship*. And there are values in PC\_SALES table for which there is no corresponding master data exists in PC\_EMPLOYEE table /ex – FOO/

Now let us see it in a hierarchy with grouping all not assigned members together.

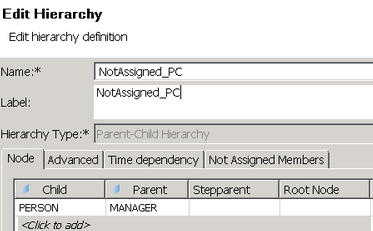
***Step1***

Create *dimensional* calculation view with PC\_EMPLOYEE table



***Step2***

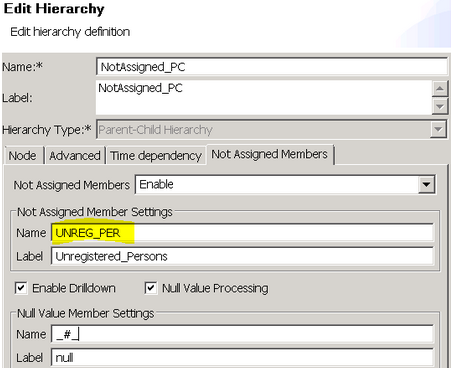
Create a *parent child hierarchy* with PERSON as child and MANAGER as parent



If your data is in levels, level hierarchy can be created. The feature is applicable for both level and parent child hierarchies.

***Step3***

Enable *Not Assigned Members* by giving a name for grouping all unassigned members as highlighted below



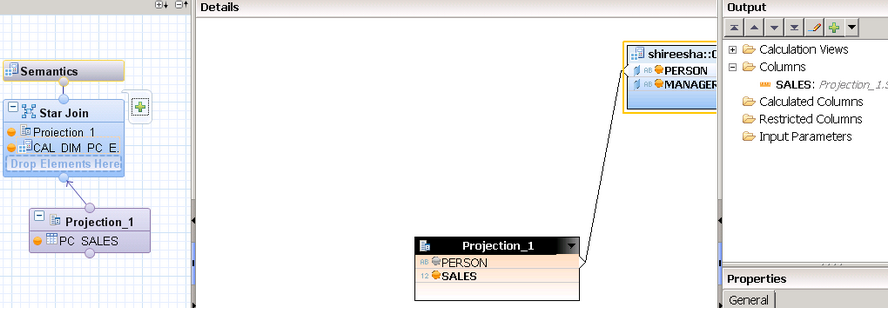
***Step4***

Once dimensional calculation view is activated, it generates a columnview “\_SYS\_BIC”.”shireesha/CAL\_DIM\_PC\_EMP/hier/NotAssigned\_PC” for hierarchy under \_SYS\_BIC where not assigned members will be ignored as below /Person FOO is ignored in the hierarchy/

To achieve the Unassigned member handling dimensional calculation view should be consumed in a star calculation view

***Step5***

Consume above dimensional calculation view in a *star* calculation view by using **PC\_SALES** as fact table



**Step6**

Once star calculation view is activated there will be two columnviews generated for hierarchy under *\_SYS\_BIC* columnviews folder, *one with hierarchy name* and *another with ending $NA*.

