Disclaimer

This presentation outlines our general product direction and should not be relied on in making a purchase decision. This presentation is not subject to your license agreement or any other agreement with SAP.

SAP has no obligation to pursue any course of business outlined in this presentation or to develop or release any functionality mentioned in this presentation. This presentation and SAP’s strategy and possible future developments are subject to change and may be changed by SAP at any time for any reason without notice.

This document is provided without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. SAP assumes no responsibility for errors or omissions in this document, except if such damages were caused by SAP intentionally or grossly negligent.
Agenda

Introduction
Series Data Basics
Creating Series Data Tables
Built-in Functions for Series Data
Series Data Analytic Functions
Series Data Storage
Horizontal Aggregation
Introduction
Series Data: What is it?

**Series Data** is a sequence of data points, typically consisting of successive measurements made over a time interval.

Examples of series data: ocean tides, counts of sunspots, the daily closing value of the Dow Jones Industrial Average.

- Used in statistics, signal processing, pattern recognition, econometrics, mathematical finance, weather forecasting, earthquake prediction, electroencephalography, control engineering, astronomy, communications engineering, and largely in any domain of applied science and engineering which involves temporal measurements.
- While measurements are typically made over a time interval, in some cases there is no temporal component to the dataset.
  - Thus, while this topic is often referred to as *Time Series*, we use the term *Series Data* as it is more inclusive.

Where is Series Data Used? For example, in…

- **Finance**
- **Retail**
- **Predictive Maintenance**
- **Customer Energy Management**
- **Utilities**
Series Data Basics
**Example: Electrical Utility with Smart Meters in Homes**

**Meters measure and upload electricity consumption for each of 10 million homes**

Consumption measured every 15 minutes – 96 measures/day per home, 960 million / day

We would like to analyze data in many ways:

- Identify factors such as temperature with the highest influence on consumption
- Group by different geographic hierarchies (room, building, city, county, province)
- Summarize at different time resolutions (15 min, hour, day, month)
- Detect errors (find and correct missing or anomalous values)
- Forecast future consumption based on history, weather

The use case introduces several requirements:

- Efficient (time/space) storage for multivariate series with both equidistant and non-equidistant time
- Flexible query interface
- Time zone conversions (allow both local time and UTC; DST) and ability to deal with calendars (holidays, …)
Operations on series data

Basics:

Detect and correct errors or anomalies
Outlier detection, missing value replacement, editing

Query processing
Select, project, join
Grouping across series (e.g., group by province)

“Horizontal aggregation”: hourly to daily measurements

Series analysis
Similarity, regression, smoothing, binning, sampling, …
Series Data: Concepts

- Logically one series per profile with two values (Consumption and Temperature)
- Generally we want to think of operations as working on entire series, for example:
  - Add two specific series together (e.g., add the values in corresponding positions)
  - Take a “slice” out of a series (between two times)
  - Compute the dot product of two series

| Time | 09:00 | 09:15 | 09:30 | 09:45 | 10:00 | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Profile 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| KWh   | 5    | 3     | 1     | 5     | 7     | ...
| °C    | 20   | 21    | 21    | 22    | 23    | ...
| **Profile N** |     |       |       |       |       |       |
| KWh   | 2    | 1     | 4     | 9999  |     | ...
| °C    | 11   | 12    | 9     | 14    |     | ...
The natural SQL representation for series data is a “tall/skinny table”

### Series table contains zero or more series
- Series identified by a “series key” (e.g., ProfileID)
- Period columns define the intervals (e.g., Time)
  - These can be TIMESTAMP or even INT (e.g., year)
- One or more value columns hold the values of the series
  - Value columns support scalar types (DOUBLE, INT, NCHAR, …)

<table>
<thead>
<tr>
<th>Profile</th>
<th>Time</th>
<th>KWh</th>
<th>°C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile 1</td>
<td>09:00</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>09:15</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>09:30</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>09:45</td>
<td>5</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>10:00</td>
<td>7</td>
<td>23</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Profile N</td>
<td>09:15</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>09:30</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>09:45</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>10:00</td>
<td>9999</td>
<td>14</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2014 SAP SE or an SAP affiliate company. All rights reserved.
The natural representation for series data is difficult for many other DBMSs

**Storage overhead is enormous**
Each measurement might require only 1-2 bytes
Row header, series key and timestamps can exceed measurement by a factor of 8-10
Time to load, export and truncate data can be very long (due to high number of records)

**Simple operations are hard to express**
Relational systems intentionally ignore ordering in most operations
For example, linear approximation can be expressed but it is incredibly awkward

**Performance does not match specialized implementations**
Consider: GROUP BY year (ts)

**SAP HANA’s column-oriented in-memory structures have been extended to provide efficient processing for series data in a natural representation that works well with existing business data!**
Key Aspects of Series Data in SAP HANA

- Preserve all data intact
  - HANA Series data uses no lossy compression; all values originally inserted are accessible for auditing/regulatory
- Support very high volumes of data using effective compression techniques
- Support both equidistant and non-equidistant data
  - Often, source data will be non-equidistant; it will then be “snapped” to an equidistant “grid” for analysis, model fitting, etc.
- Allow time series manipulation, cleaning, and analytic operations to be expressed naturally in SQL while maintaining high performance
Series Data in SAP HANA: What We Provide

**Table Creation**
- Series property aspect of tables

**Built-in Functions**
- Special SQL functions for *working with* series data

**Analytic Functions**
- Special SQL functions for *analyzing* series data

**Storage**
- Advanced techniques for storing equidistant data using dictionary encoding
  - Can store large datasets efficiently

**Horizontal Aggregation**
- Extension of OLAP engine – efficient and fast handling of horizontal aggregation
Creating Series Data Tables
Creating a Series Data Table: Special Descriptors

1. SERIES_KEY – list of columns identifying a series in the table
2. PERIOD_TYPE – the data type of the period columns (Datetime or any metric type)
3. IS_INSTANT – measurements represent points in time, not periods
4. PERIOD_COLUMNS – one or two timestamps bounding the period for the measurement
5. IS_EQUIDISTANT – true if the delta between adjacent times is a multiple of a constant
   1. INCREMENT_BY – the delta between adjacent times (if no missing elements intervene)
   2. IS_MISSING_ALLOWED – can timestamps be omitted in the middle of a range
6. MIN_VALUE / MAX_VALUE – range for the periods
7. CALENDAR – a table defining “factory day” or valid timestamps
8. Value columns – the measurement columns in the table
Creating a Series Data Table: Example

```
CREATE COLUMN TABLE DailyWeather(
    station_id    varchar(3) not null references WeatherStation,
    date          date not null,
    maxtemp       decimal(3,1),
    primary key(station_id,date)
) SERIES (
    SERIES KEY(station_id)
    EQUIDISTANT INCREMENT BY 1 DAY MISSING ELEMENTS NOT ALLOWED
    PERIOD FOR SERIES (date,NULL)
)
```

The series property can be combined with other clauses (partitioning, logging, location,...)

Equidistant series tables use our new compression / storage options
Series Definition Syntax

<series_definition> ::= SERIES ( 
    [<series_key>] 
    [<series_equidistant_definition>] 
    [<series_minvalue>] [<series_maxvalue>] 
    <series_period> 
) 
<series_key> ::= SERIES KEY ( <series_key_column> [{,<series_key_column>...}])

<series_equidistant_definition> ::= 
    NOT EQUIDISTANT 
    | EQUIDISTANT INCREMENT BY <increment_by> [<missing_elements>]

<increment_by> ::= <real_const> | <interval_const> 
<missing_elements> ::= MISSING ELEMENTS [NOT] ALLOWED 

<series_minvalue> ::= NO MINVALUE | MINVALUE <constant_literal> 
<series_maxvalue> ::= NO MAXVALUE | MAXVALUE <constant_literal>
Built-in Functions for Series Data
Built-in Functions for Series Data

We add several built-in functions to SQL in SAP HANA that make it easier to work with series data:

1. SERIES_GENERATE – generate a complete series
2. SERIES_DISAGGREGATE – move from coarse units (e.g., day) to finer (e.g., hour)
3. SERIES_ROUND – convert a single value to a coarser resolution
4. SERIES_PERIOD_TO_ELEMENT – convert a timestamp in a series to its offset from the start
5. SERIES_ELEMENT_TO_PERIOD – convert an integer to the associated period

Details of each are shown in the following slides
SERIES_GENERATE Function Example

Generate all rows of a series between minvalue/maxvalue as a complete equidistant series:
SERIES_GENERATE(SERIES TABLE MySeriesTab, minvalue, maxvalue)
SERIES_GENERATE(‘INTERVAL 60 SECOND’, ‘1999-01-01 00:00:00’, ‘1999-01-01 01:00:00)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERATED_PERIOD_START</td>
<td>PERIOD_TYPE</td>
<td>the start of the period represented by this row; the period includes the period_start (i.e. it is a closed interval at the start)</td>
</tr>
<tr>
<td>GENERATED_PERIOD_END</td>
<td>PERIOD_TYPE</td>
<td>the end of the period represented by this row (open interval; the period represented by this row consists of all times greater than or equal to the start and strictly less than the end)</td>
</tr>
<tr>
<td>ELEMENT_NUMBER</td>
<td>BIGINT</td>
<td>the element number of this period within the generated series; equivalent to row_number</td>
</tr>
<tr>
<td>FRACTION_OF_MIN_MAX_RANGE</td>
<td>DOUBLE</td>
<td>the ratio of the length of this period to the length of all periods generated by this function</td>
</tr>
</tbody>
</table>
SERIES_DISAGGREGATE – 1-

SERIES_DISAGGREGATE_DECIMAL(source_increment_by => 10.0, generate_increment_by => 1.0, min_value => 0.0, max_value => 100.0)

SERIES_DISAGGREGATE(SERIES TABLE A.T1, SERIES TABLE A.T2, '2013-01-01', '2013-12-31')

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE_PERIOD_START</td>
<td>PERIOD_TYPE</td>
<td>the start of the source period that generated this row</td>
</tr>
<tr>
<td>SOURCE_PERIOD_END</td>
<td>PERIOD_TYPE</td>
<td>the end of the source period that generated this row</td>
</tr>
<tr>
<td>GENERATED_PERIOD_START</td>
<td>PERIOD_TYPE</td>
<td>the start of the period represented by this row; the period includes the period_start (i.e. it is a closed interval at the start)</td>
</tr>
<tr>
<td>GENERATED_PERIOD_END</td>
<td>PERIOD_TYPE</td>
<td>the end of the period represented by this row (open interval; the period represented by this row consists of all times greater than or equal to the start and strictly less than the end)</td>
</tr>
<tr>
<td>ELEMENT_NUMBER_IN_GENERATED_SERIES</td>
<td>BIGINT</td>
<td>element (row) number within the whole result set</td>
</tr>
<tr>
<td>ELEMENT_NUMBER_IN_SOURCE_PERIOD</td>
<td>BIGINT</td>
<td>element (row) number of this target row within its source interval</td>
</tr>
<tr>
<td>FRACTION_OF_SOURCE_PERIOD</td>
<td>DOUBLE</td>
<td>fraction of the length of the source period that this generated period covers</td>
</tr>
<tr>
<td>FRACTION_OF_MIN_MAX_RANGE</td>
<td>DOUBLE</td>
<td>fraction of the length of all generated periods that this generated period covers</td>
</tr>
</tbody>
</table>
### SERIES_DISAGGREGATE – 2 -

```sql
SELECT * FROM SERIES_DISAGGREGATE_SECONDDATE('INTERVAL 5 DAY', 'INTERVAL 24 HOUR', '2000-02-01', '2000-03-05');
```

<table>
<thead>
<tr>
<th>SOURCE PERIOD START</th>
<th>SOURCE PERIOD END</th>
<th>GENERATED PERIOD START</th>
<th>GENERATED PERIOD END</th>
<th>ELEMENT NUMBER IN SOURCE PERIOD</th>
<th>ELEMENT NUMBER IN GENERATED SERIES</th>
<th>FRACTION OF SOURCE PERIOD</th>
<th>FRACTION OF MIN MAX RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feb-01</td>
<td>Feb-06</td>
<td>Feb-01</td>
<td>Feb-02</td>
<td>1</td>
<td>1</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-01</td>
<td>Feb-06</td>
<td>Feb-02</td>
<td>Feb-03</td>
<td>1</td>
<td>2</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-01</td>
<td>Feb-06</td>
<td>Feb-03</td>
<td>Feb-04</td>
<td>1</td>
<td>3</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-01</td>
<td>Feb-06</td>
<td>Feb-04</td>
<td>Feb-05</td>
<td>1</td>
<td>4</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-01</td>
<td>Feb-06</td>
<td>Feb-05</td>
<td>Feb-06</td>
<td>1</td>
<td>5</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-06</td>
<td>Feb-11</td>
<td>Feb-06</td>
<td>Feb-07</td>
<td>2</td>
<td>6</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-06</td>
<td>Feb-11</td>
<td>Feb-07</td>
<td>Feb-08</td>
<td>2</td>
<td>7</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-06</td>
<td>Feb-11</td>
<td>Feb-08</td>
<td>Feb-09</td>
<td>2</td>
<td>8</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-06</td>
<td>Feb-11</td>
<td>Feb-09</td>
<td>Feb-10</td>
<td>2</td>
<td>9</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-06</td>
<td>Feb-11</td>
<td>Feb-10</td>
<td>Feb-11</td>
<td>2</td>
<td>10</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-11</td>
<td>Feb-16</td>
<td>Feb-11</td>
<td>Feb-12</td>
<td>3</td>
<td>11</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-11</td>
<td>Feb-16</td>
<td>Feb-12</td>
<td>Feb-13</td>
<td>3</td>
<td>12</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-11</td>
<td>Feb-16</td>
<td>Feb-13</td>
<td>Feb-14</td>
<td>3</td>
<td>13</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>Feb-11</td>
<td>Feb-16</td>
<td>Feb-14</td>
<td>Feb-15</td>
<td>3</td>
<td>14</td>
<td>0.2</td>
<td>0.0333333333</td>
</tr>
</tbody>
</table>
SERIES_ROUND: Rounding to a coarser granularity

Round a value to a coarser granularity

- SERIES_ROUND( CURRENT_DATE, ‘INTERVAL 1 MONTH’ )
- SERIES_ROUND( CURRENT_DATE, SERIES TABLE MySchema.MySeriesTable )
- SERIES_ROUND( TO_INT(T.x), 10 )
- SERIES_ROUND( CURRENT_DATE, ‘INTERVAL 1 MONTH’, ROUND_DOWN )

The round_mode matches the ROUND() function
Parser actions convert a column reference to a constant string
SERIES_PERIOD_TO_ELEMENT and SERIES_ELEMENT_TO_PERIOD

Converting from period values to element number and vice versa

<series_element_func> ::= 
  SERIES_PERIOD_TO_ELEMENT( <period_expr>, <element_series> 
  [,<rounding_mode> [,<calendar_args>] ]) 
| SERIES_ELEMENT_TO_PERIOD( <element_expr>, <element_series> [,<calendar_args>] )

• SERIES_PERIOD_TO_ELEMENT( CURRENT DATE, ‘INTERVAL 1 DAY’, ‘2001-01-01’)

© 2014 SAP SE or an SAP affiliate company. All rights reserved.
Series Data Analytic Functions
New Analytic SQL Functions for Series Data

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORR</td>
<td>Pearson product-moment correlation coefficient</td>
</tr>
<tr>
<td>CORR_SPEARMAN</td>
<td>Spearman rank correlation</td>
</tr>
<tr>
<td>LINEAR_APPROX</td>
<td>Replace NULL values by interpolating adjacent non-NULL values</td>
</tr>
<tr>
<td>MEDIAN</td>
<td>Compute median value</td>
</tr>
</tbody>
</table>
CORR: Computing correlation coefficients

```sql
select corr(nasdaq_rate, gold_rate) as corr_pearson,
       corr_spearman(nasdaq_rate, gold_rate) as corr_spearman
from NasdaqRateMonthly
     join GoldRateMonthly G
      on N.DATE=G.DATE;
```

<table>
<thead>
<tr>
<th>CORR_PEARSON</th>
<th>CORR_SPEARMAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.581707</td>
<td>0.510352</td>
</tr>
</tbody>
</table>

Nasdaq

Gold
LINEAR_APPROX: Column functions

Analyzing series data requires functions that consume an ordered column and return a column of the same size:
LINEAR_APPROX, outlier detection, binning, and exponential smoothing are examples

These are not window functions:
The order of rows in the evaluation is meaningful even with no frame definition
Window frame is UNBOUNDED / UNBOUNDED
Series Data Storage
Storage Problems with Typical Approaches: Timestamp Overhead

Typically, column tables use compression heavily
Dictionary encoding maps value → smaller ValueID
Index vectors are compressed (e.g., RLE)

Default techniques work poorly for series timestamps
Low duplication → dictionary is ineffective
Blocks of repeated ValueID unlikely → RLE, others ineffective

Equidistant time series are common
Dictionary encoding should exploit equidistance
Index vector compression should expect runs of value ids increasing linearly by one → LRLE compression
For an equidistant series, timestamps have a direct mapping to an “element number” in the series.

We can think of the dictionary as a function with \( O(1) \) parameters mapping timestamp \( \rightarrow \) ValueID.

Missing timestamps need special handling (not shown here).

SAP HANA requires that every ValueID have a matching row, so we cannot generate a sparse set of ValueIDs.

We do have a solution for this case.
Example: Series Data Table vs Regular Table

Series data table

```sql
create column table MyTabSeries(key int, ts timestamp, value int)
series(key(key),
      period for series(ts, null),
      equidistant increment by interval 60 second);

insert into MyTabSeries select 1, GENERATED_PERIOD_START, 7 from series_generate_timestamp('2010-01-01', '2011-01-01', 'INTERVAL 60 SECOND', null, null, null);
commit;
merge delta of MyTabSeries;
update MyTabSeries with parameters('optimize_compression'='force');

create column table MyTab(key int, ts timestamp, value int);
insert into MyTab select 1, GENERATED_PERIOD_START, 7 from series_generate_timestamp('2010-01-01', '2011-01-01', 'INTERVAL 60 SECOND', null, null, null);
commit;
merge delta of MyTab;
update MyTab with parameters('optimize_compression'='force');

select table_name, column_name, memory_size_in_total, sum(memory_size_in_total) over(partition by table_name) as tab_memory_size,
       uncompressed_size, sum(uncompressed_size) over(partition by table_name) as tab_uncompressed_size,
       compression_ratio_in_percentage as ratio, compression_type, "COUNT", dist
from a co_columns where table_name like 'MYTAB%';
```

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>MEMORY_SIZE_IN_TOTAL</th>
<th>TAB_MEMORY_SIZE</th>
<th>UNCOMPRESSED_SIZE</th>
<th>TAB_UNCOMpressed_SIZE</th>
<th>RATIO</th>
<th>COMPRESSION_TYPE</th>
<th>COUNT</th>
<th>DISTINCT_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYTAB Series</td>
<td>KEY</td>
<td>4,828</td>
<td>14,656</td>
<td>2,107,490</td>
<td>8,425,780</td>
<td>0.23</td>
<td>RLE</td>
<td>525,600</td>
<td></td>
</tr>
<tr>
<td>2 MYTAB Series</td>
<td>TS</td>
<td>5,064</td>
<td>14,656</td>
<td>4,210,800</td>
<td>8,425,780</td>
<td>0.12</td>
<td>LINEAR RLE</td>
<td>525,600</td>
<td></td>
</tr>
<tr>
<td>3 MYTAB Series</td>
<td>VALUE</td>
<td>4,764</td>
<td>14,656</td>
<td>2,107,490</td>
<td>8,425,780</td>
<td>0.23</td>
<td>PREFixed</td>
<td>525,600</td>
<td></td>
</tr>
<tr>
<td>4 MYTAB</td>
<td>KEY</td>
<td>4,764</td>
<td>5,533,112</td>
<td>2,107,490</td>
<td>8,425,780</td>
<td>0.23</td>
<td>Prefixed</td>
<td>525,600</td>
<td></td>
</tr>
<tr>
<td>5 MYTAB</td>
<td>TS</td>
<td>5,523,584</td>
<td>5,533,112</td>
<td>4,210,800</td>
<td>8,425,780</td>
<td>131.18</td>
<td>DEFAULT</td>
<td>525,600</td>
<td></td>
</tr>
<tr>
<td>6 MYTAB</td>
<td>VALUE</td>
<td>4,764</td>
<td>5,533,112</td>
<td>2,107,490</td>
<td>8,425,780</td>
<td>0.23</td>
<td>PREFixed</td>
<td>525,600</td>
<td></td>
</tr>
</tbody>
</table>
Horizontal Aggregation
Efficient processing of “horizontal aggregation”

Horizontal aggregation is used to go from fine-grained (15min) to coarser (daily)

```
SELECT SERIES_ROUND(T.ts, 'INTERVAL 1 DAY') day, AVG(DT.value)
FROM T
GROUP BY SERIES_ROUND(T.ts, 'INTERVAL 1 DAY')
```

SAP HANA has an optimized OLAP engine for specific forms of GROUP BY query (star schema, group by dimensions)

Grouping by SERIES_ROUND() or other function previously prevented using the OLAP engine

Significantly faster (by orders of magnitude) to use the OLAP engine where feasible

OLAP engine support has been added for ROUND and SERIES_ROUND by extending and plumbing into existing “granulize” functionality

Previously this support was only available through internal interface (Python, ABAP)
How to find SAP HANA documentation on this topic?

- In addition to this learning material, you can find SAP HANA platform documentation on SAP Help Portal knowledge center at http://help.sap.com/hana_platform.
- The knowledge centers are structured according to the product lifecycle: installation, security, administration, development:

  **SAP HANA Platform SPS**
  - What’s New – Release Notes
  - Installation
  - Administration
  - Development
  - References

- Documentation sets for SAP HANA options can be found at http://help.sap.com/hana_options:
  - SAP HANA Options
    - SAP HANA Advanced Data Processing
    - SAP HANA Dynamic Tiering
    - SAP HANA Enterprise Information Management
    - SAP HANA Predictive
    - SAP HANA Real-Time Replication
    - SAP HANA Smart Data Streaming
    - SAP HANA Spatial
Thank you

Contact information

Ron Silberstein
SAP HANA Product Management

AskSAPHANA@sap.com
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. Please see http://global12.sap.com/corporate-en/legal/copyright/index.epx for additional trademark information and notices.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors.

National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP SE or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP SE or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE’s or its affiliated companies’ strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as of their dates, and they should not be relied upon in making purchasing decisions.