



SAP HANA Database – Partitioning and Distribution of Large Tables

■ SAP HANA Appliance Software SPS 03

Target Audience

- Consultants
- Administrators
- SAP Hardware Partner
- Others

Document version 1.0 – 2011-12-13

Typographic Conventions

Type Style	Description
<i>Example Text</i>	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
Example text	File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
Example text	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
EXAMPLE TEXT	Keys on the keyboard, for example, F2 or ENTER.

Icons





Icon	Description
	Caution
	Note or Important
	Example
	Recommendation or Tip

Table of Contents

1.	Introduction to Partitioning	1
1.1	Example	2
2.	Single-Level Partitioning.....	2
2.1	Hash	2
2.1.1	Hash Syntax.....	2
2.2	Round-robin	3
2.2.1	Round-robin Syntax.....	3
2.3	Range.....	3
2.3.1	Range Syntax.....	4
3.	Multi-Level Partitioning	4
3.1	Hash-Range.....	5
3.1.1	Syntax.....	6
3.2	Round-robin Range	6
3.2.1	Syntax.....	6
3.3	Hash Hash	6
3.3.1	Syntax.....	6
4.	Explicit Partition Handling	6
4.1	Syntax.....	7
5.	Moving Partitions.....	7
5.1	Syntax.....	7
6.	Split/Merge Operations.....	8
6.1	Syntax.....	9
6.2	Parallelism and Memory Consumption.....	10
7.	Delta Merge	10

1. Introduction to Partitioning

Using the partitioning feature of the SAP HANA database, tables can be partitioned horizontally into disjunctive sub-tables or “partitions” as they are also known.

Partitioning supports the creation of very large tables by decomposing them into smaller and more manageable pieces. Partitioning is transparent for most SQL queries and Data Manipulation Language (DML) statements. This means that there is no need to modify these statements to support partitioning.

Typical uses cases for partitioning:

1. Load balancing: Using partitioning, the individual partitions may be distributed over the landscape. This way, a query on a table is not processed by a single server but by all servers that host partitions that are relevant for processing.
2. Parallelization: Operations are parallelized by using several execution threads per table.
3. Partition pruning: Queries are analyzed to see if they match the given partition specification of a table. If a match is found, it is possible to determine the actual partitions that hold the data in question. Using this method the overall load on the system can be reduced and the response time is typically better.
4. Explicit partition handling: Applications may actively control partitions, for example by adding partitions that should hold the data for an upcoming month.

When considering using partitioning in your SAP HANA landscape, please bear in mind the following factors:

1. A non-partitioned table cannot store more than 2 billion rows. By using partitioning, this limit may overcome by distributing the rows to several partitions. Please note, each partition must not contain more than 2 billion rows.
2. The delta merge performance of the database is dependent on the size of the main index. If data is only being modified on some partitions (see Time-Based Partitioning below), there will be less partitions that need to be delta-merged and therefore the performance will be better.



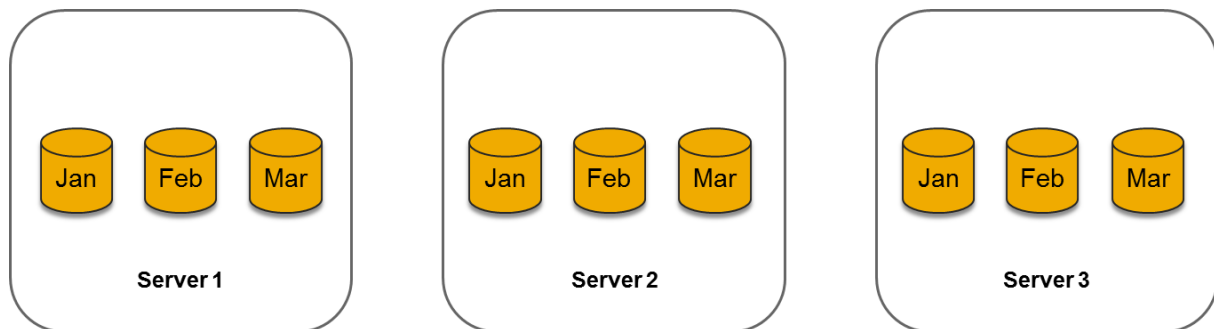
Partitioning is typically used in distributed landscapes. But it may also be beneficial for single-host systems.



Partitioning is available for column store tables only.

1.1 Example

The following figure illustrates how a table is distributed over three servers with dedicated partitions for respective months.



2. Single-Level Partitioning

Rows can be distributed to partitions using different types of partitioning known as partition specifications. *Hash*, *range* and *round-robin* are offered as single-level partition specifications by the HANA database. Using the multi-level partitioning described in the next chapter these specifications can be stacked for advanced use cases.

2.1 Hash

Hash partitioning is used to equally distribute rows to partitions for load balancing and overcoming the 2 billion rows limitation. Usually it does not require in-depth knowledge of the actual content of a table for the implementation.

Each hash partition specification requires columns to be specified as partitioning columns. The actual values of these fields are used when the hash value is determined. If the table has a primary key, it is required that these columns are part of that key. This restriction comes with the advantage of a uniqueness check of the key which can be performed on the local server.

You can use as many partitioning columns as required to achieve a good variety of values for an equal distribution. Please also see the pruning chapter below for further information.

2.1.1 Hash Syntax

```
CREATE COLUMN TABLE mytab (a INT, b INT, c INT, PRIMARY KEY (a,b)) PARTITION BY HASH (a, b) PARTITIONS 4
```

- creates 4 partitions columns a and b
- determines the target partition based on the actual values in columns a and b
- at least one column has to be specified
- all columns have to be part of the primary key

```
CREATE COLUMN TABLE mytab (a INT, b INT, c INT, PRIMARY KEY (a,b)) PARTITION BY HASH (a, b)
PARTITIONS GET_NUM_SERVERS()
```

- Number of partitions is determined by the engine at runtime according to its configuration. It is recommended to use this function in scripts etc.

2.2 Round-robin

Round-robin is similar to *hash* partitioning as it is used for an equal distribution of rows to parts. When using this method it is not required to specify partitioning columns.

Hash partitioning is usually more beneficial than round-robin partitioning for the following reasons:

- The partitioning columns can be evaluated in a pruning step, therefore all partitions will be considered in searches and other database operations.
- Depending on the scenario it is possible that the data within semantically related tables resides on the same server. Some internal operations may then operate locally instead of retrieving data from a remote system.

2.2.1 Round-robin Syntax

```
CREATE COLUMN TABLE mytab (a INT, b INT, c INT)
PARTITION BY ROUNDOBIN PARTITIONS 4
```

- The table must not have primary keys

```
CREATE COLUMN TABLE mytab (a INT, b INT, c INT)
PARTITION BY ROUNDOBIN PARTITIONS GET_NUM_SERVERS()
```

- The number of partitions is determined by the engine at runtime according to its configuration. It is recommended to use this function in scripts or clients that may operate in various landscapes.

2.3 Range

Range partitioning can be used to create dedicated partitions for certain values or certain value ranges. Usually this requires in-depth knowledge of the values that are used / valid for the chosen partitioning column.

For example a *range* partitioning scheme can be chosen to create one partition per month of the year.

Applications may choose to use *range* partitioning to actively manage the partitioning of a table: partitions may be created or dropped individually. For example an application may create a partition for an upcoming month so that new data is inserted into that new partition.

Please be aware that *range* partitioning is not well-suited for load distribution. The multi-level partitioning specifications of the next chapter address this issue.

The *range* partition specification usually takes ranges of values to determine one partition, e.g. 1 to 10. It is also possible to define a partition for a single value. In this way a list partitioning, known in other database systems, can be emulated and also mixed with *range* partitioning.

When inserting or modifying rows, the target partition is determined by the defined ranges. If a value does not fit into one of these ranges, an error is raised. If this is not intended, it is possible to define a “rest partition” where all rows that do not match with any of the defined ranges will be inserted. Rest partitions can be created or dropped on-the-fly as desired.

Range partitioning is similar to *hash* partitioning in that the partitioning column has to be part of the primary key. *Range* partitioning also has restrictions on the data types that can be used. Only strings, integers and dates are allowed.

2.3.1 Range Syntax

```
CREATE COLUMN TABLE mytab (a INT, b INT, c INT, PRIMARY KEY (a,b))
  PARTITION BY RANGE (a)
    (PARTITION 1 <= VALUES < 5,
     PARTITION 5 <= VALUES < 20,
     PARTITION VALUE = 44,
     PARTITION OTHERS)
```

- Create partitions for ranges using <= VALUES < semantics
- Create partitions for single values using VALUE = semantics
- Create a rest partition for all values that do not match the other ranges using PARTITION OTHERS

3. Multi-Level Partitioning

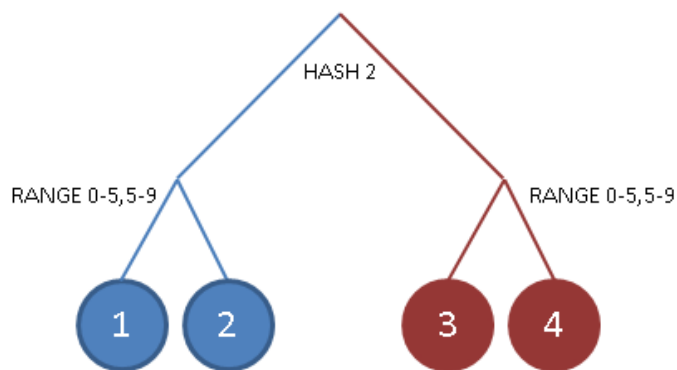
For some tables it is beneficial to partition by a column that is not part of the primary key. For example, if a date column is present it is desirable to leverage it in order to build partitions per month or year.

Hash and *range* partitioning have the restriction of only being able to use key columns as partitioning columns. This restriction can be overcome by using multi-level partitioning described in this chapter.

Multi-level partitioning is the technical implementation of time-based partitioning, this is where a date column is leveraged:

The performance of the delta merge is dependent on the size of the main index of a table. If data is inserted into a table over time and it also contains temporal information in its structure, for example a date, multi-level partitioning may be an ideal candidate to be used in this case. If the partitions containing old data are infrequently modified there is no need for a delta merge on these partitions: the delta merge is only required on new partitions, where new data is inserted. Therefore its run-time is constant over time as new partitions are being created and used.

The following illustration shows how a hash partitioning on the first level and a range partitioning the second level can be applied.



As mentioned above, there is a relaxation of the key column restriction. This applies only to the second level of partitioning (for *Hash Range* and *Hash Hash*). The reason why it is restricted for the first level is explained below:

Upon an insertion or update of a row, it is required to check the unique constraint of the primary key. If the primary key has to be checked on all partitions across the landscape, this will involve expensive remote calls. Therefore it is advantageous if only local partitions need to be checked. The concept of partition groups exists for this purpose. It allows inserts to occur whilst only requiring primary key checks on local partitions.

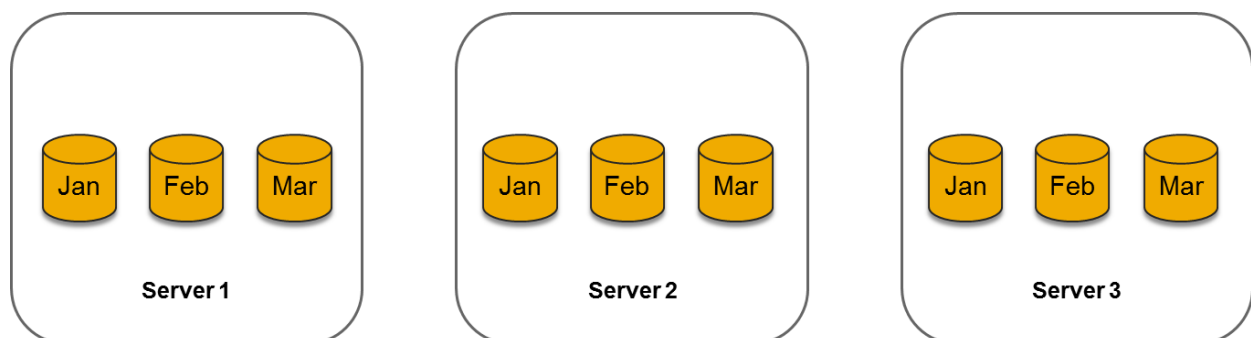
All corresponding parts of the second level form a group: Considering the illustration above, parts 1 and 2 and parts 3 and 4 each form a respective group. When a row is inserted into part 1, it is only required to check for uniqueness on parts 1 and 2.

All parts of a partition group must reside on the same server. When using SQL commands to move partitions, please be aware that it is not possible to move individual parts of partition groups, only partition groups as a whole can be moved.

3.1 Hash-Range

Hash-range multi-level partitioning is most-typically used. It is implemented with *hash* on the first level for load balancing and *range* on the second level determining the time criterion.

The following illustration shows a typical usage scenario: The load is distributed via a *hash* partitioning to three server nodes. A *range* partitioning on the second level distributes the data to individual partitions per month.



3.1.1 Syntax

```
CREATE COLUMN TABLE mytab (a INT, b INT, c INT, PRIMARY KEY (a,b))
  PARTITION BY
    HASH (a, b) PARTITIONS 4,
    RANGE (c)
      (PARTITION 1 <= VALUES < 5,
       PARTITION 5 <= VALUES < 20)
```

3.2 Round-robin Range

Like *Hash Range* but with *Round-robin* on the first level.

3.2.1 Syntax

```
CREATE COLUMN TABLE mytab (a INT, b INT, c INT)
  PARTITION BY
    ROUNDROBIN PARTITIONS 4,
    RANGE (c)
      (PARTITION 1 <= VALUES < 5,
       PARTITION 5 <= VALUES < 20)
```

3.3 Hash Hash

Two-level Partitioning with *Hash* on both levels. The advantage is that the *Hash* on the second level may be defined on a non-key column.

3.3.1 Syntax

```
CREATE COLUMN TABLE mytab (a INT, b INT, c INT, PRIMARY KEY (a,b))
  PARTITION BY
    HASH (a, b) PARTITIONS 4,
    HASH (c) PARTITIONS 7
```

4. Explicit Partition Handling

For all partition specifications involving *Range*, it is possible to add additional ranges or to remove them at will. This causes partitions to be created or dropped as required by the ranges in use. In case of a multi-level partitioning, the operation desired will be applied to all nodes.

If a partition is created and if a rest partition exists, the rows of the rest partition that match the newly added range are moved into the new partition. If the rest partition is large be aware that this operation may take a long time: internally the split operation is executed which is discussed in chapter 0. If a rest partition does not exist, this operation will be fast as only a new partition is added to the catalog.

4.1 Syntax

```
ALTER TABLE mytab ADD PARTITION 100 <= VALUES < 200
```

```
ALTER TABLE mytab DROP PARTITION 100 <= VALUES < 200
```

It is also possible to create or drop a rest partition using the following syntax:

```
ALTER TABLE mytab ADD PARTITION OTHERS
```

```
ALTER TABLE mytab DROP PARTITION OTHERS
```

5. Moving Partitions

Partitions and partition groups can be moved to other servers. As mentioned before, when moving partition groups it is always only possible to move an entire group. However, in the case of single-level partitioning, each partition forms its own group. To see how partitions and groups relate to each other check the monitoring view `M_CS_PARTITIONS`. To see the current location of a partition, check `M_TABLE_LOCATIONS`.

5.1 Syntax

```
ALTER TABLE mytab MOVE PARTITION 1 TO '<host:port>'
```

where `port` is the port of the target index server and not the SQL port.

6. Split/Merge Operations

The partitioning of a table can either be determined upon creation or can also be changed later. The split/merge operations can be used to:

- transform a non-partitioned table into a partitioned table and vice versa
- change the partitioning of a table
 - change the partitioning specification e.g. from *Hash* to *Round-robin*
 - change the partitioning column
 - split partitions into more partitions
 - merge partitions into less partitions



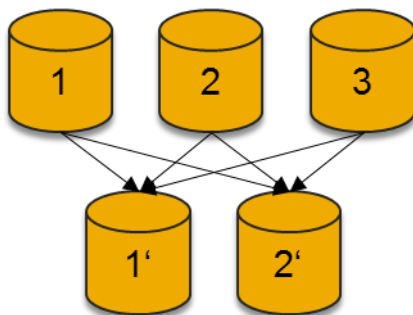
Be aware that the split/merge operation can be costly:

- long run time: may take up to several hours for huge tables
- relatively high memory consumption
- requires an exclusive lock (only selects are allowed)
- performs a delta merge in advance
- writes everything into the log, which is required for backup & recovery

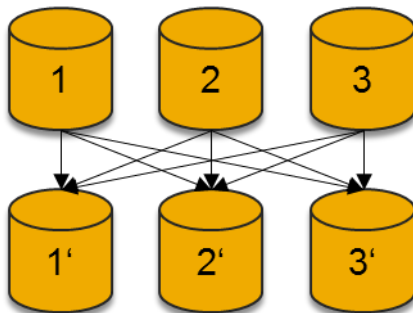
It is recommended to split tables before inserting mass data or while they are still small. If a table is not partitioned and reaches configurable absolute thresholds or a table grows a certain percentage per day, an alert is raised by the statistics server to inform the administrator.

There are three distinguishable types of re-partitioning:

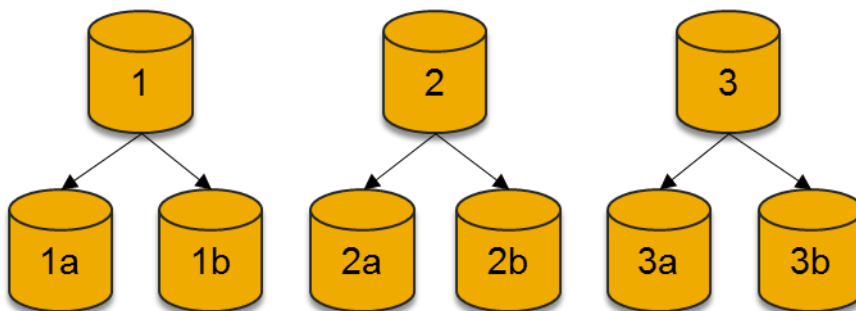
- 1) From n to m partitions where m is not a multiple/divider of n , for example from *HASH 3 X* to *HASH 2 X*.



- 2) From n to n partitions using a different partition specification or different partitioning columns, for example $\text{HASH } 3 \text{ X}$ to $\text{HASH } 3 \text{ Y}$.



- 3) From n to m partitions where m is a multiple/divider of n , for example $\text{HASH } 3 \text{ X}$ to $\text{HASH } 6 \text{ X}$.



For the first two cases, it is required that all source parts are located on the same host. Up to one thread per column is used to split/merge the table.

For the third case, it is not required to move all parts to a common server. Instead the split/merge request is broadcasted to each host where partitions reside. Up to one thread per column and source part is used. This type of split/merge operation is typically faster as it is always recommended to choose a multiple or divider of the source parts as number of target parts. This type of re-partitioning is called "parallel split/merge".

6.1 Syntax

`ALTER TABLE mytab PARTITION BY ...`

- This can be applied to non-partitioned tables and to partitioned tables.
- Depending on the type of the split/merge operation (see above) it may be necessary to move partitions beforehand.

`ALTER TABLE mytab MERGE PARTITIONS`

- Merge all parts of a partitioned table into a non-partitioned table.
- It is required that all source partitions reside on the same server.

6.2 Parallelism and Memory Consumption

Split/merge operations consume a high amount of memory. To reduce the memory consumption, it is possible to configure the amount of threads used.

The parameter `split_threads` in section `[partitioning]` in `indexserver.ini` can be set to change default for split/merge operations. If it is not set, 16 threads are used. In case of a parallel split/merge, the individual operations use a total of the configured number of threads per host. Each operation takes at least one thread.

If a table has a history index, it is possible to split the main and history index in parallel. Use parameter `split_history_parallel` in section `[partitioning]` in `indexserver.ini`. The default is "no".

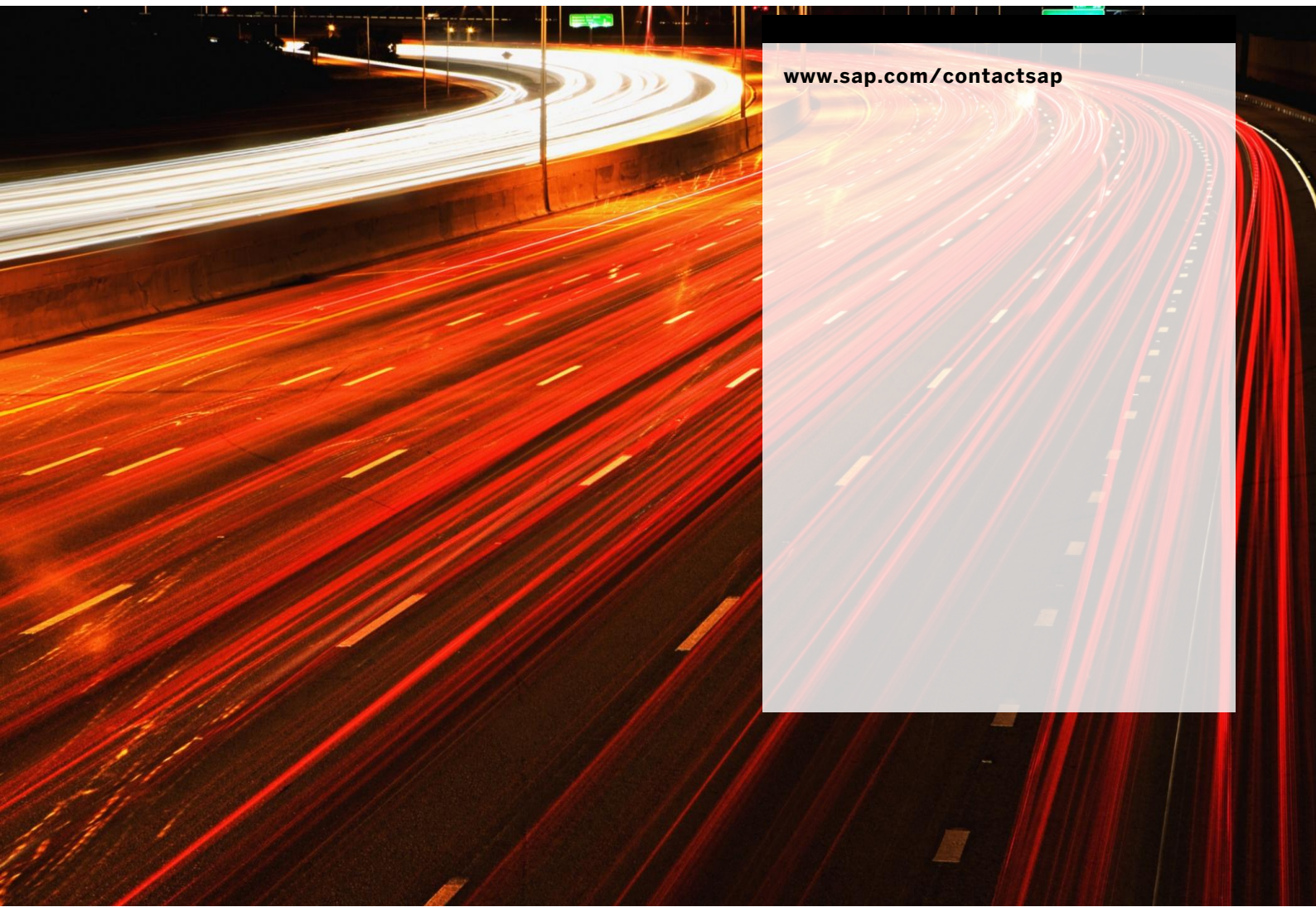
7. Delta Merge

The delta merge may operate in parallel on all available partitions. There are three parameters that control how the threading is handled. To configure the *delta merge* use the following parameters of section `[indexing]` in `hdbindexserver.ini`:

- 1) One thread per server (default): set `parallel_merge_location` to "yes".
- 2) Configurable amount of threads: set `parallel_merge_location` to "no" and `parallel_merge_part_threads` to the number of threads you wish to use. The default is 5.
- 3) One thread per part: set `parallel_merge_location` to "no" and `parallel_merge_part_threads` to "0". Please be aware that this may have a negative effect on the overall performance of the system if a table has a large number of partitions.



Be aware that a delta merge thread is not necessarily shown during merges on the master server as no dedicated thread per partition is started.



www.sap.com/contactsap



The Best-Run Businesses Run SAP™