SAS® Scalable Performance Data Server® 4.3

Scalability Solution for SAS Dynamic Cluster Tables
Introduction

SAS Scalable Performance Data Server (SAS SPD Server) is designed to meet the storage and performance needs for processing large amounts of SAS data. As the size of data grows, the demands for processing the data quickly increase, and the storage architecture must change to keep pace with business needs.

Hundreds of sites worldwide use the SAS SPD Server, which hosts data warehouses in excess of 30 terabytes. SAS SPD Server provides high-performance data storage for customers from industries such as banking, credit card, insurance, telecommunications, health care, pharmaceutical, transportation, and brokerage and government agencies.

This paper provides an overview of the dynamic cluster tables feature of SAS Scalable Performance Data Server 4.3. Earlier releases of SAS SPD Server supported two types of clustered data tables: time-based partitioning and partition by value (in an experimental form). Dynamic cluster tables enable both the partitioning of data based on criteria in the data and parallel loading of the cluster tables. They also enhance the manageability of tables. New table options, as well as SQL planner enhancements, have been added to take advantage of these new capabilities and to improve query performance.

It is assumed that the reader has a basic understanding of SAS SPD Server tables and the concepts of data partitioning, symmetric multiprocessing, and disk configuration for I/O scalability. SAS SPD Server performs best with multiple CPUs and parallel I/O, and the hardware must be set up correctly to obtain the maximum benefit.

Cluster Tables

SAS SPD Server provides a virtual table structure, which is called a clustered data table (Figure 1). A cluster contains a number of slots, each of which contains a SAS SPD Server table. The clustered data table uses a layer of metadata to manage the slots.
The virtual table structure in SAS SPD Server offers flexible storage that enables users to organize tables based on values (including SAS date, time, or datetime values) that are contained in numeric columns. Introduced in SAS SPD Server 4.3, this new type of organization is called a dynamic cluster table. Dynamic cluster tables enable parallel loading of tables and selective removal of data from tables, making management of large warehouses easier. These unique capabilities provide organizational features and performance benefits that traditional SAS SPD Server tables cannot provide.

**Dynamic Cluster Table Loading Benefits**

The power of dynamic cluster tables derives from their ability to load and process data in parallel. Dynamic cluster tables provide the flexibility to add new data to or to remove historical data from the clustered table by working with the metadata for the cluster without affecting underlying tables. This capability reduces the time needed to complete the job. Additionally, a complete refreshing of a dynamic cluster table requires a fraction of the disk space that would be needed otherwise, and the refresh process can be divided into parallel jobs that complete sooner. All of these benefits are achieved using simple SPD Server operator interface procedure (PROC SPDO) commands to create and alter a cluster. Such operations run in seconds. These features of dynamic cluster tables reduce the downtime of the table for maintenance and improve the availability of the warehouse.

**Commands for Creating and Undoing a Cluster**

Clusters are simple structures, and creating or undoing a cluster only takes seconds. The two most basic commands are CLUSTER CREATE and CLUSTER UNDO. A third command, ADD, is discussed later, and the syntax for a fourth command, LIST, also is presented in a later section. The commands are executed using the SPDO procedure.

The CLUSTER CREATE command requires three options:

- the name of the cluster table that will be created
- a list of SAS SPD Server tables that will be included in the cluster
- the number of slots (for member tables) the cluster will have.
The following example shows the syntax for an SPDO procedure with a CLUSTER CREATE command:

```sql
PROC SPDO LIBRARY=domain-name;
  SET ACLUSER user-name;
  CLUSTER CREATE cluster-table-name
    MEM = SPD-Server-table1
    MEM = SPD-Server-table2
    MEM = SPD-Server-table3
    MEM = SPD-Server-table4
    MEM = SPD-Server-table5
    MEM = SPD-Server-table6
    MEM = SPD-Server-table7
    MEM = SPD-Server-table8
    MEM = SPD-Server-table9
    MEM = SPD-Server-table10
    MEM = SPD-Server-table11
    MEM = SPD-Server-table12
    MAXSLOT=24
QUIT;
```

When the CLUSTER CREATE command is run, it initiates the creation of a new layer of metadata, which contains information about the SAS SPD Server tables that are included in the cluster. The SAS SPD Server tables listed in the command are hidden from direct access by users, making the data accessible through the cluster only. Underneath the top layer of metadata, nothing is changed. Each table still contains the same columns, rows, and indexes. The tables in a cluster can be seen in the CONTENTS procedure output. Because the tables are virtually concatenated into a single entity using metadata, the sorted-by flag for the cluster is turned off.

The second command, CLUSTER UNDO, removes the cluster metadata. When the metadata layer is removed, you can view the tables in the cluster as normal SAS SPD Server tables. The syntax for the CLUSTER UNDO command is:

```sql
PROC SPDO library=domain-name;
  CLUSTER UNDO cluster-table-name;
QUIT;
```

**Cluster Table Rules**

Cluster tables are crucial structures in large warehouses and data marts; therefore, certain rules must be followed. Users who create cluster tables must have control privileges for the access control list (ACL) so they can alter ACLs for the tables or in a domain.

Cluster tables are read-only structures. You cannot perform updates, inserts, deletions, and modifications of rows on data that is stored in a dynamic cluster. In addition, you are not allowed to create indexes, to append rows, or to change column labels and formats. To work with individual tables or rows in the cluster, you must use the UNDO command to remove the structure.

You can only perform four functions on a dynamic cluster table: creating a cluster, undoing a cluster, adding tables, and listing member tables. To perform any function other than these four, you must first undo the cluster with the CLUSTER UNDO command. Then, any changes must be made to all individual tables in the cluster. Once changes are made, you can re-create the cluster using the CLUSTER CREATE command.
All member SAS SPD Server tables in a cluster must be identical; that is, they must have the same definitions for columns names, column widths, table indexes, formats on the columns, partition size, compression ON/OFF, and so forth. You cannot change table names, column names, column formats, indexes, or anything else about any member table, including the cluster name.

The owner of all member tables must be the same. To create a cluster, the owner of the cluster must be the same user that owns the member table. For example, suppose USER1 creates a table, and USER2 also creates a table. These tables cannot be members of the same cluster, regardless of what ACL permissions are granted on the tables.

Unique Indexes

Dynamic clusters support the use of unique indexes across the cluster table. To implement a unique index across the cluster table, the index on each member must be unique. In addition, the values in each unique index for a member table must contain values that are unique across all member tables.

When member tables are combined into a cluster or when a member table is added to a cluster, the application checks values for uniqueness across the entire cluster. If the values are not unique, the process of adding or combining member tables will halt. The process will also halt if you add a member or members to an existing cluster with unique indexes and the application determines that the index contains values that exist in one of the cluster indexes.

Cluster Table Syntax

The following syntax examples illustrate the four cluster functions:

**Example 1: Syntax for Creating a Cluster**

```sas
PROC SPDO LIBRARY=&domain;
SET ACLUSER user-name;
CLUSTER CREATE sales_hist
   MEM = sales200401
   MEM = sales200402
   MEM = sales200403
   MEM = sales200404
   MEM = sales200405
   MEM = sales200406
   MEM = sales200407
   MEM = sales200408
   MEM = sales200409
   MEM = sales200410
   MEM = sales200411
   MEM = sales200412
   MAXSLOT=24;
QUIT;
```
Example 2: Syntax for Undoing a Cluster

```
PROC SPDO LIBRARY=domain-name;
   SET ACUSER user-name;
   CLUSTER UNDO sales_hist;
QUIT;
```

Example 3: Syntax for Adding Tables

```
PROC SPDO LIBRARY=domain-name;
   SET ACUSER user-name;
   CLUSTER ADD sales_hist
       MEM = sales200501
       MEM = sales200502
       MEM = sales200503
       MEM = sales200504
       MEM = sales200505
       MEM = sales200506;
QUIT;
```

Example 4: Syntax for Listing Sales History Data

```
PROC SPDO LIBRARY=domain-name;
   SET ACUSER user-name;
   CLUSTER LIST sales_hist;
QUIT;
```

Load in Parallel

A dynamic cluster table comprises SAS SPD Server tables that are called member tables. A clustered table is loaded in parallel by loading each member table using separate, concurrent jobs to load each portion. Either SAS/MP CONNECT software or SAS Data Integration Studio can create and manage the overall process. When all of the individual load jobs are complete, the software issues a simple command to create the cluster. Then, the dynamic cluster table can be accessed as a single virtual table.

The scalability of a parallel load is dependent on the scalability of the hardware. The first area that must scale is the I/O of the server. Parallel loading requires multiple, concurrent instances of writing to the disk since each member table is being loaded simultaneously. If the I/O hardware does not scale appropriately, the loading process becomes a bottleneck and degrades performance.

Another area that requires scalability is the number of CPUs on the server. As an example, consider index creation, which is a CPU-intensive process. SAS SPD Server is capable of creating multiple indexes on the same table in parallel. As long as the system has enough CPU processing power, parallel index creation in SAS SPD Server is highly scalable, and it requires only a single pass through the table to read the data. Even a single index creation can benefit from CPU scalability because the creation process for each index is multithreaded and distributing the threads across multiple CPUs improves performance greatly. Multiple job runs to create indexes in parallel will become bound unless the server has enough cycles to handle the load.
Add New Member Tables

SAS SPD Server enables a new member table to be added to an existing cluster in a matter of seconds. When a cluster is created, the required option MAXSLOT sets the maximum number of slots (member tables) for the cluster table. For example, if a cluster is created with MAXSLOT=36, then up to 36 tables can be part of the cluster. If the existing cluster has 24 slots in use, then 12 more slots are available for tables to be added to the cluster.

However, you can increase the maximum number of slots for a cluster. For example, if all 36 slots in the cluster are filled but you need to add additional data, you can

1. undo the cluster using the CLUSTER UNDO command
2. create a new cluster (with the CLUSTER CREATE command)
3. set a higher MAXSLOT value for the new cluster.

This latest release of SAS SPD Server makes it easier to extend a clustered data table in order to add new data. You can independently create the table you want to add to the cluster, then test and verify the table before you add it.

Regardless of the size of the new table, snapping a new table into a dynamic cluster takes only seconds. Data that already exists in the cluster is not affected by this process.

Remove Existing Member Tables

You can only remove existing member tables when the cluster structure is removed (using the CLUSTER UNDO command). The tables then become standard SAS SPD Server tables that you can archive, copy to other domains, or delete. Once you remove any unwanted member tables, you simply re-create the cluster using the CREATE CLUSTER command.

Perform Maintenance in Parallel

Once the member tables are unclustered, multiple processes can run in parallel on individual SAS SPD Server tables. For example, while one process loads data into a new table, another process can refresh existing tables individually to remove deleted rows and to coalesce space within the tables. Simultaneously, the application can delete tables to free space while leaving other tables unchanged. When all processes are finished on the individual SAS SPD Server tables, you can quickly re-create the dynamic cluster table.
Refresh a Table with Limited-Space

Complete refreshing of a table is often done to

• recapture space from rows that have been deleted in the table
• update existing rows
• reorder data to optimize performance.

However, a complete refresh process on a table can temporarily use twice the disk space of the table itself. With such tables, this issue of space can limit the update process of a warehouse or data mart. When server space is scarce, the amount of data that can be refreshed at any given time may be limited. This behavior can cause an increase in the load or in the Refresh window, or it can make refreshing of tables impossible.

However, you can avoid space issues on the disk by undoing the cluster into the original tables, and refreshing the much smaller, individual tables. Using this method, the maximum space requirement is only twice the size of the largest slot in the cluster.

For example, if a dynamic cluster table has 12 equally sized slots, a complete refresh process will only use temporary disk space equal to 1/12th of the entire cluster size. To perform this kind of refresh process, you need to

1. uncluster the tables
2. use the free space to refresh one of the individual SAS SPD Server tables
3. back up the refreshed section of the cluster table
4. delete the old SAS SPD Server table to create more free space
5. repeat steps 1-4 until all individual tables have been refreshed
6. re-create the cluster table.

If the server has enough free space available to refresh more than one SAS SPD Server table at a time, then the refresh process can be done in parallel on multiple tables.

Refresh Multiple Tables with Limited-Space

Often a data mart or warehouse has several tables that need to be refreshed concurrently on a regular schedule (monthly, weekly, and so on). When space limits the number of these tables that can be refreshed concurrently, you can use dynamic cluster tables as a method to enable that process.

For example, suppose you have 12 equally sized tables to refresh, but the server only has enough space to refresh three of the tables at a time using a standard refresh. The amount of time it will take to complete a standard refresh process on all of these tables will be equal to the
amount of time it takes to refresh 3 tables concurrently by 4 groups of 3. If each group requires 2 hours, the total run time will take 8 hours. On robust and scalable hardware, much of the server’s resources could be idle while refreshing only 3 tables, and much of the free space allocated for the refresh process would not be in immediate use.

In this situation, dynamic clusters can help speed the refresh process. Instead of running the process on the 12 standard SAS SPD Server tables, you can create 12 clustered data tables with 6 equal-sized slots each. By creating dynamic cluster tables, you end up with 72 slots across 12 cluster tables. The free space that previously could only handle a refresh process on 3 tables can now hold data for refreshing 18 slots concurrently. By breaking the job into 72 smaller jobs, all of the CPU and memory resources can now be used on the refresh process. Running these smaller jobs in parallel enables the system to continue processing one job while it waits for I/O on another job, which enables a much faster refresh process.

Create Restart Points

Managing and updating complex data stores often requires a divide-and-conquer approach. The method that SAS SPD Server uses to divide the load process is to create restart points. These are designated points along the load process where the process can be restarted from scratch if a problem occurs. These points should not be dependent upon any earlier step in the process.

For example, suppose you have a single job that takes 2 hours that you can split into 2 parts, and each part takes an hour. The first part runs to completion, but the second part fails. So only the second part needs to run again. Restart points enable you to run the individual part. By design, dynamic clusters build in these restart points that are not dependent on each other.

Conceptual Diagrams

Figure 2 illustrates an example in which you can use snap-in loading to add new tables to an existing cluster.

![Conceptual Diagram](image)

*Figure 2: Snap-In Loading*
Figure 3 illustrates how tables are added in seconds to a cluster using the ADD command.

![Figure 3: Using the ADD Command to Snap Tables into a Cluster](image)

Figure 4 illustrates loading a parallel table with a dynamic cluster.

![Figure 4: Partition Loading](image)

Figure 5 illustrates how unclustered SAS SPD Server tables are refreshed individually using available space.

![Figure 5: Limited-Space Table Refresh](image)
MINMAXVARLIST Option

MINMAXVARLIST is a new table option that is available in SAS SPD Server 4.3. The option enables you to specify a list of numeric columns during table creation. SAS SPD Server then stores the minimum and maximum values from the specified columns in the table metadata. These values are used by a new layer of WHERE planner optimization, which can greatly enhance performance.

In addition to the WHERE planner, the application has another layer called MIN/MAX planning. This layer determines whether the entire set of rows in the table are all true or all false based on the range of values that are stored in the table metadata and the values from the WHERE clause predicate that match the MIN/MAX columns. When the values are either all true or all false, this planning layer will bypass the normal WHERE-clause processing, which requires more resources. When the MIN/MAX planner cannot make a determination that the table rows are either all true or all false, the planner reverts to the second layer of WHERE-clause processing (which selects or filters rows by using available indexes or a full-table scan). The MIN/MAX planning layer is the highest level of WHERE-clause processing that is done by the server.

Table 1 uses a simple SAS SPD Server table to demonstrate the functionality of the MIN/MAX optimization. Consider a column in a SAS SPD Server table with a minimum value of 1 and a maximum value of 10. Based on the predicate from the WHERE clause, MIN/MAX planning returns different results, as shown in this table.

<table>
<thead>
<tr>
<th>Selection Criteria</th>
<th>Pre-Evaluation Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between 1 and 10</td>
<td>True</td>
</tr>
<tr>
<td>Less then or Equal 10</td>
<td>True</td>
</tr>
<tr>
<td>Greater then or equal 1</td>
<td>True</td>
</tr>
<tr>
<td>NOT GT 11</td>
<td>True</td>
</tr>
<tr>
<td>NOT LE 0</td>
<td>True</td>
</tr>
<tr>
<td>1 &lt;= column &lt;= 10</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Selection Criteria</th>
<th>Pre-Evaluation Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greater then 11</td>
<td>False</td>
</tr>
<tr>
<td>Less then 1</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Selection Criteria</th>
<th>Pre-Evaluation Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 5</td>
<td>Undetermined</td>
</tr>
<tr>
<td>Greater then 1</td>
<td>Undetermined</td>
</tr>
<tr>
<td>Less then 10</td>
<td>Undetermined</td>
</tr>
<tr>
<td>Greater then equal 5</td>
<td>Undetermined</td>
</tr>
<tr>
<td>1 &lt; column &lt;=10</td>
<td>Undetermined</td>
</tr>
<tr>
<td>Column in (value list)</td>
<td>Undetermined</td>
</tr>
</tbody>
</table>

*Table 1: MIN/MAX Planning with Minimum Value of 1/Maximum Value of 10*

In this example, for the selection criteria “=5” and “IN (value list)”, the planner cannot determine a value of True for ALL rows in the table because the column has a range of values of 1 through 10. The MIN/MAX processing has no information on what other values exist in the column. In this
case, where the MIN/MAX planning results are undetermined, the application passes the processing to a lower-level processing (WHERE-clause) that can handle the case.

A case does exist in which the MIN/MAX planning is able to determine that all rows have values of True for an equality expression that selects or filters the rows in a table. This case occurs when the minimum value and maximum value are equal; that is, there is one unique value in the table column. Table 2 shows this case with a table that has both a minimum and maximum value of 1.

<table>
<thead>
<tr>
<th>Selection Criteria</th>
<th>Pre-Evaluation Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 5</td>
<td>False</td>
</tr>
<tr>
<td>= 1</td>
<td>True</td>
</tr>
<tr>
<td>In (1)</td>
<td>True</td>
</tr>
<tr>
<td>In (1,2)</td>
<td>True</td>
</tr>
<tr>
<td>Between 0 and 2</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 2: MIN/MAX Planning with Minimum & Maximum Value of 1

MINMAXVARLIST Option and Cluster Tables

MIN/MAX processing may not seem particularly useful when applied to a single SAS SPD Server table because the planner is making a true/false determination for all the values in the table. However, when you combine MIN/MAX planning with the flexibility of dynamic cluster tables, powerful performance benefits emerge.

Suppose you have a dynamic cluster table named SALES_HIST_2004 that contains 12 equally sized slots, where each slot holds millions of rows. Each slot contains all of the records for a specific month based on the values in the column titled PROCESS_DTE RANGE. For example, all rows where the range for PROCESS_DTE is between 01JAN2004 and 31JAN2004 are in the slot SALES_HIST_012004 exclusively. Now, assume that there are an equal number of rows present for each day of the month.

You create each slot by using the MINMAXVARLIST option, which saves the minimum and maximum values for the column PROCESS_DTE. For example, suppose you have PROCESS_DTE records that range from 01JAN2004 through 31JAN2004. The MINMAXVARLIST option saves the values 01JAN2004 and 31JAN2004, respectively. This slot of the dynamic cluster table has one value for each day of the month, and it contains 31 distinct values in the PROCESS_DTE column. Table 3 shows the relationship between PROCESS_DTE and the slot tables.
Consider the following query run against SALES_HIST_2004:

```
select *
from SALES_HIST_2004
where PROCESS_DTE between '01JUL2004'd and '30SEP2004'd;
```

The planner recognizes that the MIN/MAX information is stored in the metadata for the column. It then performs a pre-evaluation on each individual slot of SALES_HIST_2004, identifying slots in the cluster that have values of True, False, or Undetermined. Table 4 shows the results for each slot of the dynamic cluster table.

<table>
<thead>
<tr>
<th>Slot Name</th>
<th>Pre-Evaluation Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALES_HIST_012004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_022004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_032004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_042004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_052004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_062004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_072004</td>
<td>True</td>
</tr>
<tr>
<td>SALES_HIST_082004</td>
<td>True</td>
</tr>
<tr>
<td>SALES_HIST_092004</td>
<td>True</td>
</tr>
<tr>
<td>SALES_HIST_102004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_112004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_122004</td>
<td>False</td>
</tr>
</tbody>
</table>

*Table 4: Results for Dynamic Cluster Table Slots*

Any table for which the MIN/MAX processing returns a value of False will be bypassed for further processing, because a result with a value of False means no row within that slot meets the selection criteria. The planner will also select all rows from the slots where the MIN/MAX value is True.
Similarly, consider the following SQL query where only a single PROCESS_DTE is selected instead of a range:

```sql
select *
from SALES_HIST_2004
where PROCESS_DTE='15SEP2004';
```

Table 5 shows the result of the MIN/MAX processing. Eleven out of 12 slots are eliminated using the MIN/MAX pre-evaluation. With 11 slots eliminated, the planner performs further processing on the one remaining slot, where it is undetermined how many, if any, rows meet the selection criteria. If an index exists for the PROCESS_DTE column, the WHERE processing will use that index to evaluate that portion of the predicate.

<table>
<thead>
<tr>
<th>Slot Name</th>
<th>Pre-Evaluation Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALES_HIST_012004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_022004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_032004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_042004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_052004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_062004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_072004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_082004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_092004</td>
<td>Undetermined</td>
</tr>
<tr>
<td>SALES_HIST_102004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_112004</td>
<td>False</td>
</tr>
<tr>
<td>SALES_HIST_122004</td>
<td>False</td>
</tr>
</tbody>
</table>

*Table 5: Results of MIN/MAX Processing*

This example illustrates how you can use MIN/MAX planning and the SAS SPD Server’s indexing capabilities together. If you expect many user queries that select single values for the column PROCESS_DTE, then it is appropriate to create an index. In the previous example, even though MIN/MAX planning eliminated 11 of the 12 slots, the planner still had a lot of work to do to further subset the millions of rows in the one slot that remains. However, with an index for the PROCESS_DTE column, the planner can efficiently select rows that match the 15SEP2004 date. Because September has 30 calendar days, you can estimate that one value of PROCESS_DTE will contain 1/30th of the rows in the slot. So you can see performance can be greatly enhanced by creating and using an index for queries where a single value is selected.

MIN/MAX planning also works when other columns have indexes and the WHERE clause has multiple predicates. For these cases, the planner first performs the MIN/MAX processing and eliminates any slots with a value of False. Then the WHERE planner uses the index or indexes to select or filter the rows of that table.

For example, suppose all slots have an index for the column STATE. The STATE column for each slot contains 50 values. The SAS SPD Server evaluates the following WHERE clause:

```sql
where PROCESS_DTE
      between '01JAN2004' and '31JAN2004'
and STATE='CT'
```
SAS SPD Server first eliminates slot tables by using MIN/MAX processing on the PROCESS_DATE column. Then, it uses the STATE index to select rows that match the second predicate in the WHERE clause. Similar to the previous example, the use of a secondary index greatly improves query performance because the SAS SPD Server can select a fraction of the rows from the member table.

**Conceptual Diagram**

Figure 6 shows a dynamic cluster with MIN/MAX processing on the 2004 member tables, where all of the tables are filtered except the January 2004 member.

![Dynamic Cluster with MIN/MAX on 2004 Member Tables](image)

**Cluster Security**

The section covers two aspects of security in the dynamic cluster model. The first aspect is the safety and integrity of underlying data that is contained in the cluster members. The safety and integrity of data are crucial because a cluster table often contains the entire historical view for a table.

To maintain safety and data integrity, the SAS SPD Server only enables users to perform the four functions mentioned earlier in “Cluster Table Rules”. As discussed previously, you can only create a cluster, undo a cluster, add tables, and list data. However, you CANNOT delete clusters. The main purpose of a cluster is to provide you with a safe tool for managing tables. Often, the underlying tables in a dynamic cluster are a continuous set of historical data. If users were allowed to delete clusters, they could delete all of the historical data in a table. So while you can remove individual members, you cannot delete the cluster and all of the historical data tied to it.

The second method of security in the SAS SPD Server is the use of access control lists (ACLs). ACLs that are applied to an individual SAS SPD Server table do not apply when that table becomes a member in a cluster table. Any ACL that existed before the table became part of a cluster will still apply when the cluster is undone. When a dynamic cluster table is created, the same SAS SPD Server security model that applies to a standard SAS SPD Server table also applies to a dynamic cluster table. To allow or restrict users from reading a cluster table, the creator of the cluster can apply restrictive ACLs to the cluster just as they could do for any other SAS SPD server table.
Conclusion

Dynamic cluster tables are designed to enhance and optimize the performance of operations on warehouses and data marts while consuming fewer disk resources. Clusters enhance performance in several ways:

- By creating virtual tables (clusters) from multiple SAS SPD Server tables, the SAS SPD Server can perform parallel loading that results in faster extract, transform, and load processes.

- Clusters enable you to refresh tables while using less space and administration.

- You can quickly and easily add incremental data or remove old data from a table, thereby reducing downtime for applications that are dependent on the warehouse or data mart.

- The inclusion of MIN/MAX metadata for columns of a table optimizes query planning. When teamed with scalable hardware, the SAS SPD Server’s dynamic cluster tables and MIN/MAX optimization improve the manageability, scalability, and performance of enterprise data stores.