Optimized DSO Data Activation Using Massive Parallel Processing in SAP NetWeaver BW 7.3 on IBM DB2 for Linux, UNIX, and Windows

Applies to:

SAP NetWeaver BW 7.3. For more information, visit the EDW homepage.

Summary

SAP NetWeaver BW 7.3 can process the activation of new data in standard DataStore objects using massive parallel processing (MPP) on IBM DB2 for Linux, UNIX, and Windows (in the following referred to as DB2 for LUW) databases that use the Database Partitioning Feature (DPF). With the MPP-optimized implementation, the runtime of data activation can be considerably reduced. This document describes both the standard procedure for DataStore object data activation on non-MPP databases, as well as the MPP-optimized procedure and its implementation on DB2 for LUW. Recommendations for MPP-optimized data activation on DB2 for LUW are provided.

Author: Brigitte Bläser (IBM)

Company: IBM Germany Research & Development GmbH

Created on: 21 November 2011

Author Bio

Brigitte Bläser is a senior software engineer at IBM Deutschland Research & Development GmbH in Böblingen. She is a member of the joint IBM/SAP development team that enables SAP applications for DB2 for Linux, UNIX and Windows and is responsible for integrating DB2 for LUW solutions into SAP NetWeaver BW.

Table of Contents

Introduction

The activation of new data in standard DataStore objects (DSOs) is a central but time-consuming SAP NetWeaver BW ETL operation. Before SAP NetWeaver BW 7.3, this operation was not optimized to exploit massive parallel processing (MPP).

As of SAP NetWeaver BW 7.3, the activation of data in standard DataStore objects exploits MPP on database platforms that offer this feature. In this new implementation (in the following referred to as MPP-optimized data activation), the data activation mainly takes place in the database where several large transactions are executed that process the data to be activated in parallel. In many cases, no data has to be sent to the SAP application server. IBM DB2 for Linux, UNIX, and Windows (in the following referred to as DB2 for LUW) supports massive parallel processing with the Database Partitioning Feature (DPF). SAP NetWeaver BW 7.3 offers a DB2 native implementation of MPP-optimized data activation which significantly reduces the runtime especially for DSOs that are distributed over a large number of DB2 database partitions.

This document presents an overview of MPP-optimized DSO data activation and its implementation on DB2 for LUW. We first briefly describe the standard DSOs, illustrate their data activation in an example, and show the standard procedure for DSO data activation in SAP NetWeaver BW releases prior to SAP NetWeaver BW 7.3. Then we outline the MPP-optimized data activation and describe when it is used, how logs of the SQL statements that are executed during MPP-optimized data activation can be collected, and how MPP-optimized data activation can be disabled. Then we introduce the native DB2 for LUW implementation of MPP-optimized data activation. We provide recommendations for its usage and test results that show the performance improvements that can be achieved.

Standard DataStore Objects in SAP NetWeaver BW

In SAP NetWeaver BW, DataStore objects (DSOs) are used for storing operational data at a detailed level and for tracking changes to the data. SAP provides three types of DSOs: standard DSOs, transactional DSOs, and write-optimized DSOs. Data activation is an operation on standard DSOs only. In the database, standard DSOs consist of three tables:

- The activation queue table
- The active table
- The change log table

New data from external systems or PSA is first loaded into the activation queue table. Data in the activation queue table is not yet visible in SAP NetWeaver BW, but has to be activated first.

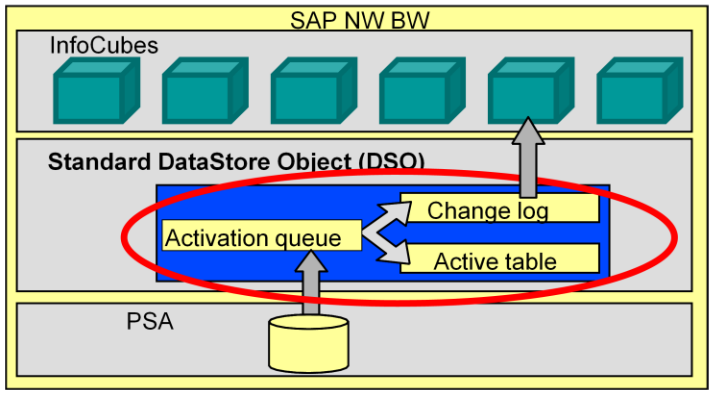


Figure 1: Standard DSO in SAP NetWeaver BW

Data Activation in Standard DataStore Objects

The data activation process updates the data in the active table and writes the before-images and after-images of the data to the change log. The change log serves two purposes:

- It enables rollback of data activation.
- It allows for delta updates from the DSO into other data targets.

The data activation is determined by the processing instructions that are provided for each new data record in the *RECORDMODE* field and by the aggregation behavior that is defined for the data fields of the DSO.

Example:

The following example roughly illustrates the data activation of a new record and a record with an already existing semantic key:

DSO MYDSO contains the number of product items that a company sold per day. The semantic key fields are *PRODUCT* and *CALDAY*, and the only data field is *ITEMS_SOLD*. Therefore, the active table of the DSO has the following columns:

- PRODUCT
- CALDAY
- ITEMS_SOLD
- RECORDMODE

The activation queue table consists of the active table fields and the three technical key columns *REQUEST SID, DATAPAKID*, and *RECORD*. The change log table consists of the active table fields plus the three technical key fields *REQUEST, DATAPAKID*, and *RECORD*.

The following figure shows the existing data in the active table and the new data to be activated in the activation queue table:

Active table

PRODUCT	CALDAY	ITEMS_SOLD	RECORDMODE
DEF	20101201	5	

Activation queue table

SID	DATAPAKID	RECORD	PRODUCT	CALDAY	ITEMS_SOLD	RECORDMODE
10	1	1	ABC	20101201	15	
10	1	2	DEF	20101201	8	

Figure 2: Example - Activation of Data in a Standard DSO

Data activation works as follows:

1. The record for product *ABC* and calendar day *20101201* in the activation queue table is new. There is no record in the active table for these semantic key values. Therefore, the record is inserted into the active table. In addition, an after-image record is inserted into the change log table.

   The following figure shows the contents of the active table and the change log table after the record has been activated:

   Active table:

PRODUCT	CALDAY	ITEMS_SOLD	RECORDMODE
DEF	20101201	5	
ABC	20101201	15	

   Change log table:

REQUEST	DATAPAKID	RECORD	PRODUCT	CALDAY	ITEMS_SOLD	RECORDMODE
R10	1	1	ABC	20101201	15	

   Figure 3: Example - Activation of Data in a Standard DSO: Insertion of New Data

2. A record for product *DEF* and calendar day *20101201* already exists in the active table. The existing record is updated by the new data as specified in the aggregation behavior. For example, the aggregation behavior might define that *ITEMS_SOLD* in the active table is overwritten with the new value. The active table is updated accordingly, and two records that represent the before-image and the after image of the data activation are inserted into the change log table.

   After the record has been activated, the active table and the change log table look as follows:

   Active table:

PRODUCT	CALDAY	ITEMS_SOLD	RECORDMODE
DEF	20101201	8	
ABC	20101201	15	

   Change log table:

REQUEST	DATAPAKID	RECORD	PRODUCT	CALDAY	ITEMS_SOLD	RECORDMODE
R10	1	1	ABC	20101201	15	
R10	1	2	DEF	20101201	-5	X
R10	1	3	DEF	20101201	8	

   Figure 4: Example - Activation of Data in a Standard DSO: Update of Existing Data

   Note that '*X*' in RECORDMODE indicates that the record is a before -mage record.

DSO Data Activation before SAP NetWeaver BW 7.3

In SAP NetWeaver BW 7.0, Enhancement Package 1 for SAP NetWeaver BW 7.0, Enhancement Package 2 for SAP NetWeaver BW 7.0, and lower releases, DSO data activation was implemented as follows:

The data activation is parallelized in the SAP application server. You can configure the number of parallel ABAP processes to be used for the activation and the size of the data packages that each activation process uses. The default number of parallel processes is three whereas one process reads from the activation queue table and feeds the data into two processes that activate the data. Common data package sizes are between 20,000 and 100,000 records.

Each data activation process receives one data package, processes it, and then pauses until it receives the next data package from the process that reads from the activation queue table. The records in each data package are processed as follows:

1. During processing of the records, the following four ABAP-internal tables are filled:
   - A table for records to be inserted into the active table
   - A table for records with which the active table is to be updated
   - A table for records that are to be deleted from the active table
   - A table for the before-images and the after-images of the records that are to be inserted into the change log table
2. For each record, the system checks whether the semantic key already exists in the active data. This check is done with a SELECT statement that retrieves the record from the active table if the key already exists.
3. If the key does not exist, the new record is added to the ABAP-internal table for records to be inserted. The before-image is inserted into the change log ABAP-internal table.
4. If the key exists and the new record contains a deletion instruction, the record is added to the ABAP-internal table for records to be deleted. The before-image is inserted into the change log ABAP-internal table.
5. If the key exists and is not to be deleted, the new non-key column values are calculated from the existing and the new record, and the result is added to the ABAP-internal table for records to be updated. The before-image and the after-image are inserted into the change log ABAP-internal table.
6. When all records have been processed, the ABAP-internal tables are materialized in the database.

This procedure does not benefit from the massive parallel processing capabilities of DB2 for LUW. The data is read and processed in rather small transactions, and massive parallel processing does not take place.

Note: In SAP NetWeaver BW 7.3, the standard DSO data activation procedure has also been optimized. The selection of single records from the active table is replaced by a FOR-ALL-ENTRIES statement. With this statement, the records are processed in small packages the size of which is determined by the SAP profile parameter rsdb/max_blocking_factor. Although this is more efficient than the selection of single records, it still does not explore the possibilities of massive parallel processing as does the new implementation described in the next section.

MPP-Optimized DSO Data Activation in SAP NetWeaver BW 7.3

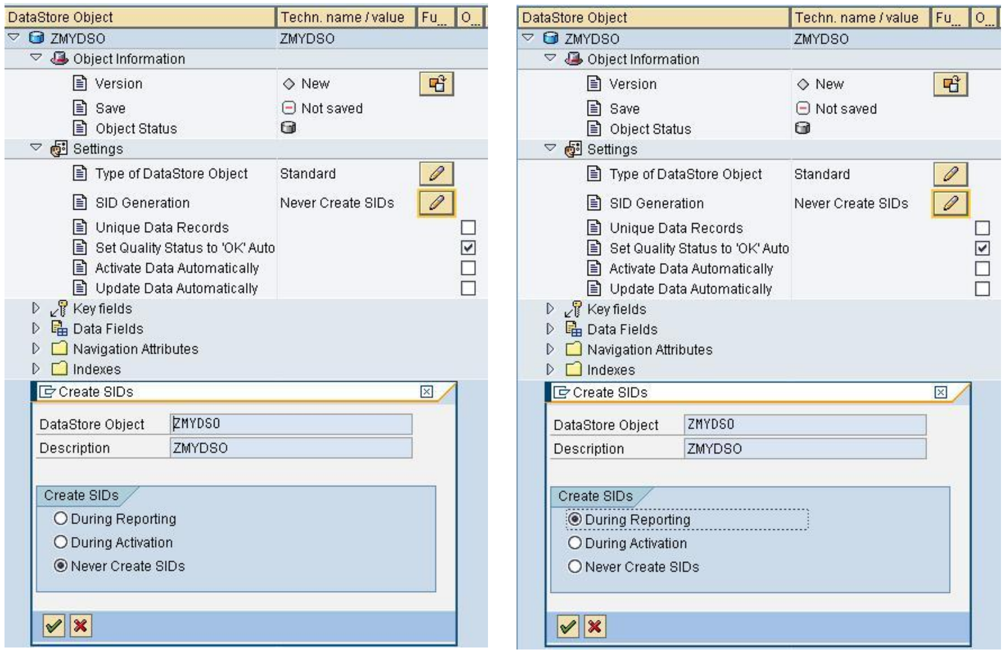
Overview

As of SAP NetWeaver BW 7.3, the activation of data in standard DataStore objects uses massive parallel processing (MPP) on database level. MPP means that SQL statements process data to be activated in parallel on all the database partitions where the DataStore object resides. Instead of reading the data into the SAP application server and parallelizing the data activation there, SQL mass INSERT, UPDATE, MERGE, and DELETE statements, which process all the data to be activated, are executed in parallel directly in the database engine. This works well for database platforms that support massive parallel processing, such as DB2 for LUW with the Database Partitioning Feature (DPF).

A standard DataStore object is eligible for MPP-optimized data activation if the following requirements are met in BW transaction RSA1:

- In the *Create SIDs* dialog window, either the *Never create SIDs* or the *During Reporting* option must be selected.
- The *Unique Data Records* flag must not be set in the settings of the DataStore object.

If these requirements are met and the database platform supports MPP-optimized data activation, new data of the DSO is activated with the MPP-optimized data activation.



DataStore Object	Techn. name / value	Fu...	O...
ZMYDSO	ZMYDSO		
Object Information			
Version	New		
Save	Not saved		
Object Status			
Settings			
Type of DataStore Object	Standard		
SID Generation	Never Create SIDs		
Unique Data Records			☐
Set Quality Status to 'OK' Auto			☑
Activate Data Automatically			☐
Update Data Automatically			☐
Key fields			
Data Fields			
Navigation Attributes			
Indexes			

Create SIDs

DataStore Object: ZMYDSO
Description: ZMYDSO

Create SIDs
- ○ During Reporting
- ○ During Activation
- ◉ Never Create SIDs

DataStore Object	Techn. name / value	Fu...	O...
ZMYDSO	ZMYDSO		
Object Information			
Version	New		
Save	Not saved		
Object Status			
Settings			
Type of DataStore Object	Standard		
SID Generation	Never Create SIDs		
Unique Data Records			☐
Set Quality Status to 'OK' Auto			☑
Activate Data Automatically			☐
Update Data Automatically			☐
Key fields			
Data Fields			
Navigation Attributes			
Indexes			

Create SIDs

DataStore Object: ZMYDSO
Description: ZMYDSO

Create SIDs
- ◉ During Reporting
- ○ During Activation
- ○ Never Create SIDs

Figure 5: DSO Settings for MPP-Optimized Data Activation

Even if MPP-optimized data activation is possible for a DSO, there can still be records in the data to be activated that have to be processed with the standard data activation procedure. These are records with specific processing instructions in RECORDMODE and records whose keys occur several times in the data to be activated. These records are identified in the first step of MPP-optimized data activation (PREPARE) and are processed using the standard data activation before the remaining records are processed using the MPP-optimized data activation procedure.

For DSOs that are eligible for MPP-optimized data activation, an additional table is created in the database during the activation of the DSO. This table is called outer-join table. The outer-join table follows the naming convention of the DSO active table and the DSO activation queue table, and uses the suffix *60*.

The outer-join table is used for storing the result of a left outer join of the data to be activated in the activation queue table with the already existing data in the active table. The left outer join determines the following for each record to be activated:

- Whether the semantic key already exists in the active table

- The data field values in the active table (if the semantic key already exists)
- Whether the record is to be activated with the standard or the MPP-optimized data activation procedure

The outer-join table contains the following columns:

- The technical key fields *SID, DATAPAKID, RECORD* from the activation queue table
- The field *ACTIVATN_METHOD* that contains the information whether to use the standard data activation (ACTIVATN_METHOD = 2) or the MPP-optimized data activation (ACTIVATN_METHOD = 1)
- The semantic key fields of the DSO
- Active data fields for storing the active data field values for semantic keys that already exist or NULL values
- Modified data fields for storing the new data field values
- The *RECORDMODE* fields from the active and the new data which determine how the record is to be processed

Two views are defined on the outer-join table. The first one selects all records that must be activated with the standard data activation procedure. These are the records containing ACTIVATN_METHOD = 2. In the following, this view is called non-MPP view. The second view selects all records that can be activated with the MPP-optimized data activation procedure. These are the records containing ACTIVATN_METHOD = 1. In the following, this view is called MPP view.

The following table shows the names of the tables created for a DSO called *ZMYDSO* that fulfills the requirements for MPP-optimized data activation:

Name of DSO	ZMYDSO
Name of DSO activation queue table	/BIC/AZMYDSO40
Name of DSO active table	/BIC/AZMYDSO00
Name of DSO outer-join table	/BIC/AZMYDSO60
Non-MPP view of outer-join table	/BIC/AZMYDSO62
MPP view of outer-join table	/BIC/AZMYDSO61
Name of DSO change log table	/BIC/B0…

Table 1: Database Tables and Views of Standard DSOs Supporting MPP-Optimized Data Activation

MPP-Optimized DSO Data Activation Procedure

MPP-optimized data activation consists of the following steps:

1. PREPARE: To determine which keys in the data to be activated already exist in the active table, a left outer join between the activation queue table and the active table is executed. The result is stored in the outer-join table. The SQL statement executed for this step also determines the records that can be processed by MPP-optimized data activation and the records that need to be activated by standard DSO data activation. This information is stored in the *ACTIVATN_METHOD* field. All the steps that follow select from the outer-join table only. The activation queue table no longer needs to be read.

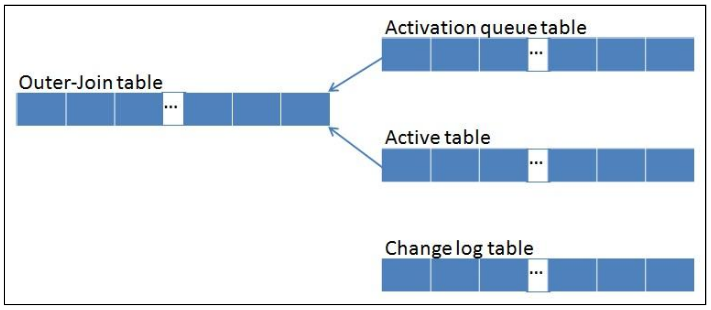


Figure 6: Filling of the Outer-Join Table

2. The records that need to be activated by standard DSO data activation are read from the outer-join table, transferred to the application server, and processed with the standard data activation procedure as described in the previous chapter.

3. All remaining records are processed in large database transactions that are implemented using the following methods:

   - INSERT: Records with keys that do not yet exist in the active table are read from the MPP view of the outer-join table and inserted into the active table. The after-images of these records are inserted into the change log table.

   - UPDATE: Records with keys that already exist in the active table are read from the MPP view of the outer-join table, the new values are calculated based on the aggregation behavior, the old and the new values, and the existing records in the active table are updated. Before-images and after-images of these records are inserted into the change log table.

   - DELETE: The keys of records that already exist and that are to be deleted are read from the MPP view of the outer-join table and deleted from the active table. The before-images of these records are inserted into the change log table.

   - MERGE: Alternatively to INSERT, UPDATE, and DELETE, a MERGE can be used to perform all or part of these operations in one step and to insert all before-images and after-images into the change log table.

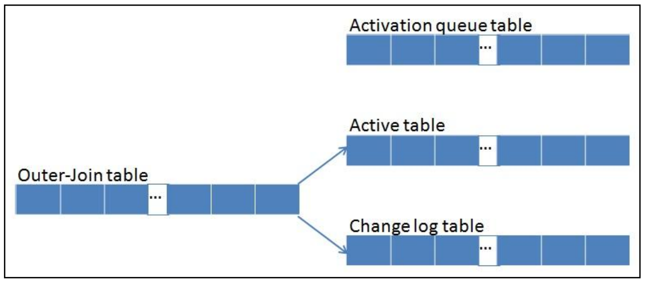


Figure 7: Filling of the Active Table and the Change Log Table from the Outer-Join Table

For each method (except PREPARE) that is called during the data activation, there is a corresponding method for rollback that is called when a request that was activated with MPP-optimized data activation is rolled back. These rollback methods are called for requests that were activated with MPP-optimized data activation only. Requests that were activated with the standard data activation procedure are rolled back with the standard data rollback procedure. The rollback methods for MPP-optimized data activation execute large database transactions to undo the changes to the active table that were made during the data activation. The following rollback methods are called during rollback:

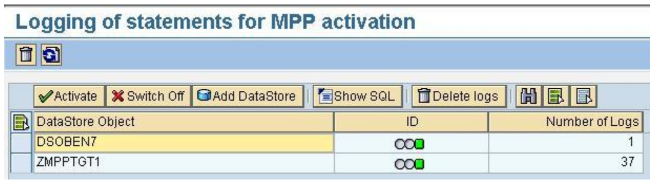
- ROLLBACK_INSERT: Deletes the records that were inserted during the activation of a request from the active table.
- ROLLBACK_UPDATE: Sets back the values of the data fields in the active table to the values before the activation of the request that is rolled back.
- ROLLBACK_DELETE: Re-inserts the records that were deleted during the MPP-optimized data activation into the active table.
- ROLLBACK_MERGE: Undoes all changes that were made to the active table during MPP-optimized data activation in one step.

After the active table has been processed, the request that was added to the change log table during the MPP-optimized data activation is deleted.

Tracing the SQL Statements Executed During MPP-optimized Data Activation

To log the SQL statements that are executed during the MPP-optimized data activation and to view the collected SQL logs, you perform the following steps:

1. In transaction SE38, run report RSODSO_MPP_SQL_LOG. The *Logging of statements for MPP activation* screen appears.

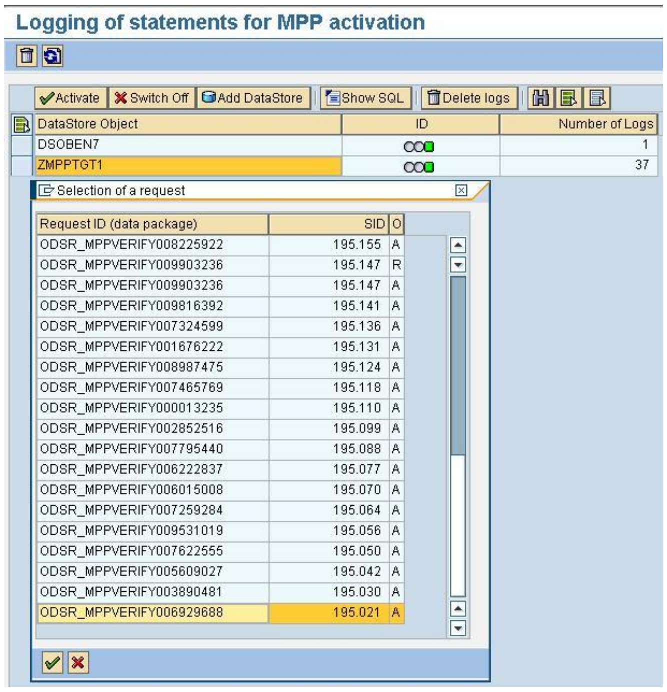


DataStore Object	ID	Number of Logs
DSOBEN7	○○🟩	1
ZMPPTGT1	○○🟩	37

Figure 8: Logging of SQL Statements for MPP-Optimized Data Activation

The DSOs for which logging is activated are displayed together with the number of SQL logs that have been collected.

2. To display the collected SQL logs, select a DSO and choose the *Show SQL* pushbutton. The *Selection of a request* dialog box appears displaying the IDs of the requests for which SQL logs have been collected.

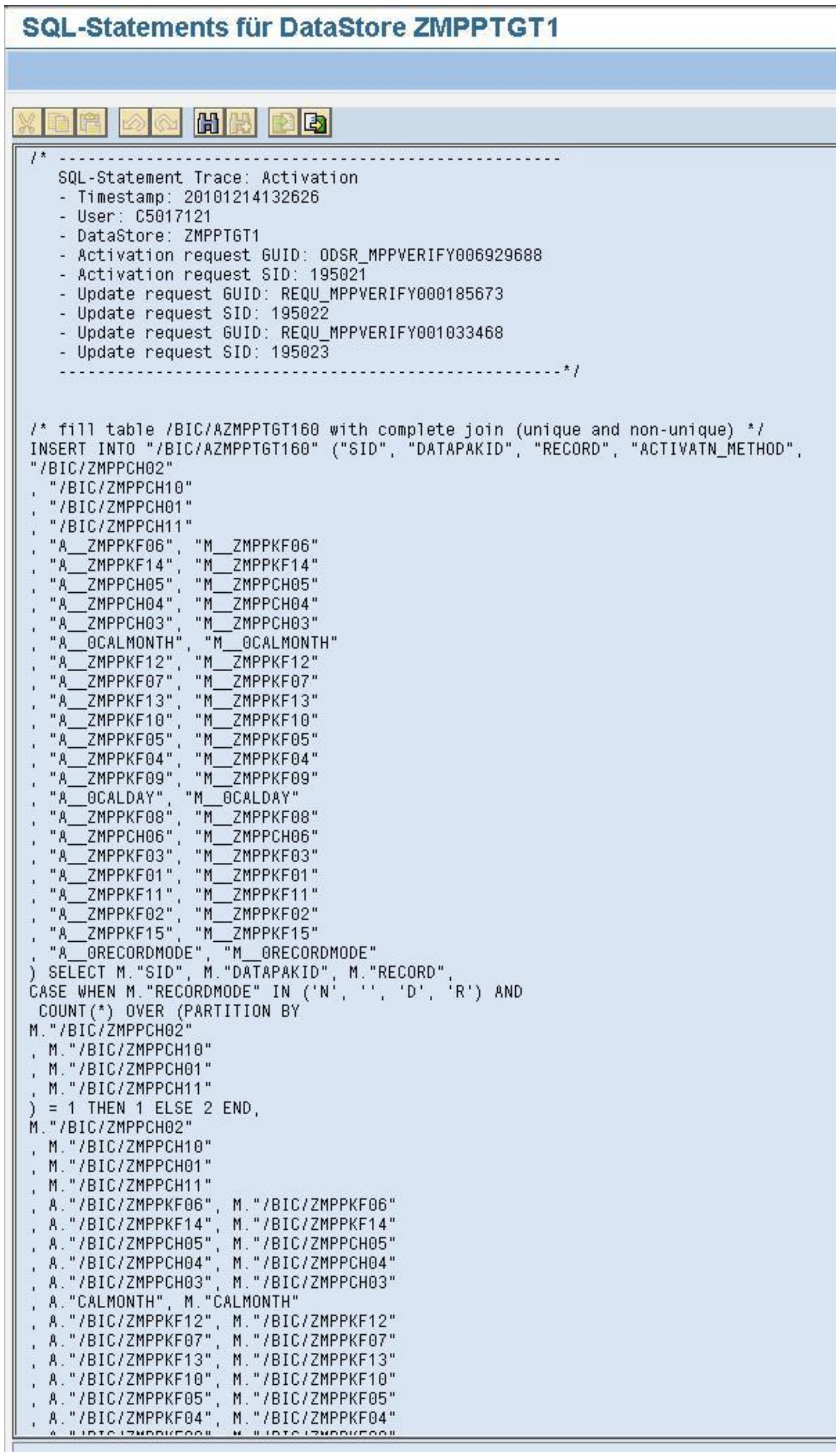


DataStore Object	ID	Number of Logs
DSOBEN7	○○🟩	1
ZMPPTGT1	○○🟩	37

Request ID (data package)	SID	O
ODSR_MPPVERIFY008225922	195.155	A
ODSR_MPPVERIFY009903236	195.147	R
ODSR_MPPVERIFY009903236	195.147	A
ODSR_MPPVERIFY009816392	195.141	A
ODSR_MPPVERIFY007324599	195.136	A
ODSR_MPPVERIFY001676222	195.131	A
ODSR_MPPVERIFY008987475	195.124	A
ODSR_MPPVERIFY007465769	195.118	A
ODSR_MPPVERIFY000013235	195.110	A
ODSR_MPPVERIFY002852516	195.099	A
ODSR_MPPVERIFY007795440	195.088	A
ODSR_MPPVERIFY006222837	195.077	A
ODSR_MPPVERIFY006015008	195.070	A
ODSR_MPPVERIFY007259284	195.064	A
ODSR_MPPVERIFY009531019	195.056	A
ODSR_MPPVERIFY007622555	195.050	A
ODSR_MPPVERIFY005609027	195.042	A
ODSR_MPPVERIFY003890481	195.030	A
ODSR_MPPVERIFY006929688	195.021	A

Figure 9: Overview of Requests Processed Using MPP-Optimized Data Activation

3. Double-click the request for which you want to display the SQL log. The SQL statements that were executed to activate the selected request are displayed in a new window.



SQL-Statements für DataStore ZMPPTGT1

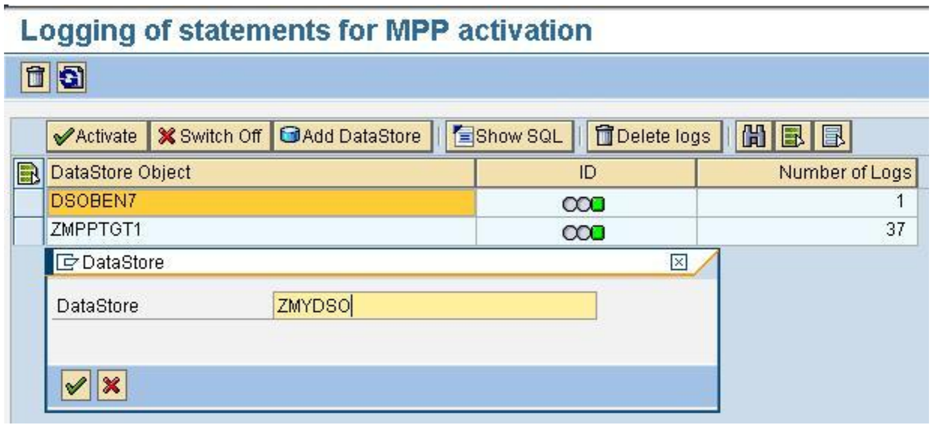
```
/* ---------------------------------------------------
   SQL-Statement Trace: Activation
   - Timestamp: 20101214132626
   - User: C5017121
   - DataStore: ZMPPTGT1
   - Activation request GUID: ODSR_MPPVERIFY006929688
   - Activation request SID: 195021
   - Update request GUID: REQU_MPPVERIFY000185673
   - Update request SID: 195022
   - Update request GUID: REQU_MPPVERIFY001033468
   - Update request SID: 195023
   ---------------------------------------------------*/


/* fill table /BIC/AZMPPTGT160 with complete join (unique and non-unique) */
INSERT INTO "/BIC/AZMPPTGT160" ("SID", "DATAPAKID", "RECORD", "ACTIVATN_METHOD",
"/BIC/ZMPPCH02"
, "/BIC/ZMPPCH10"
, "/BIC/ZMPPCH01"
, "/BIC/ZMPPCH11"
, "A__ZMPPKF06", "M__ZMPPKF06"
, "A__ZMPPKF14", "M__ZMPPKF14"
, "A__ZMPPCH05", "M__ZMPPCH05"
, "A__ZMPPCH04", "M__ZMPPCH04"
, "A__ZMPPCH03", "M__ZMPPCH03"
, "A__0CALMONTH", "M__0CALMONTH"
, "A__ZMPPKF12", "M__ZMPPKF12"
, "A__ZMPPKF07", "M__ZMPPKF07"
, "A__ZMPPKF13", "M__ZMPPKF13"
, "A__ZMPPKF10", "M__ZMPPKF10"
, "A__ZMPPKF05", "M__ZMPPKF05"
, "A__ZMPPKF04", "M__ZMPPKF04"
, "A__ZMPPKF09", "M__ZMPPKF09"
, "A__0CALDAY", "M__0CALDAY"
, "A__ZMPPKF08", "M__ZMPPKF08"
, "A__ZMPPCH06", "M__ZMPPCH06"
, "A__ZMPPKF03", "M__ZMPPKF03"
, "A__ZMPPKF01", "M__ZMPPKF01"
, "A__ZMPPKF11", "M__ZMPPKF11"
, "A__ZMPPKF02", "M__ZMPPKF02"
, "A__ZMPPKF15", "M__ZMPPKF15"
, "A__0RECORDMODE", "M__0RECORDMODE"
) SELECT M."SID", M."DATAPAKID", M."RECORD",
CASE WHEN M."RECORDMODE" IN ('N', '', 'D', 'R') AND
 COUNT(*) OVER (PARTITION BY
M."/BIC/ZMPPCH02"
, M."/BIC/ZMPPCH10"
, M."/BIC/ZMPPCH01"
, M."/BIC/ZMPPCH11"
) = 1 THEN 1 ELSE 2 END,
M."/BIC/ZMPPCH02"
, M."/BIC/ZMPPCH10"
, M."/BIC/ZMPPCH01"
, M."/BIC/ZMPPCH11"
, A."/BIC/ZMPPKF06", M."/BIC/ZMPPKF06"
, A."/BIC/ZMPPKF14", M."/BIC/ZMPPKF14"
, A."/BIC/ZMPPCH05", M."/BIC/ZMPPCH05"
, A."/BIC/ZMPPCH04", M."/BIC/ZMPPCH04"
, A."/BIC/ZMPPCH03", M."/BIC/ZMPPCH03"
, A."CALMONTH", M."CALMONTH"
, A."/BIC/ZMPPKF12", M."/BIC/ZMPPKF12"
, A."/BIC/ZMPPKF07", M."/BIC/ZMPPKF07"
, A."/BIC/ZMPPKF13", M."/BIC/ZMPPKF13"
, A."/BIC/ZMPPKF10", M."/BIC/ZMPPKF10"
, A."/BIC/ZMPPKF05", M."/BIC/ZMPPKF05"
, A."/BIC/ZMPPKF04", M."/BIC/ZMPPKF04"
```

Figure 10: SQL Statement Log

The SQL log contains all SQL statements that were executed during the MPP-optimized data activation. Below each statement, the runtime is displayed.

4. To add DSOs for which SQL logs are to be collected, choose the *Add DataStore* pushbutton. In the *DataStore* dialog box, enter the name of the DSO and choose the *Continue* pushbutton.



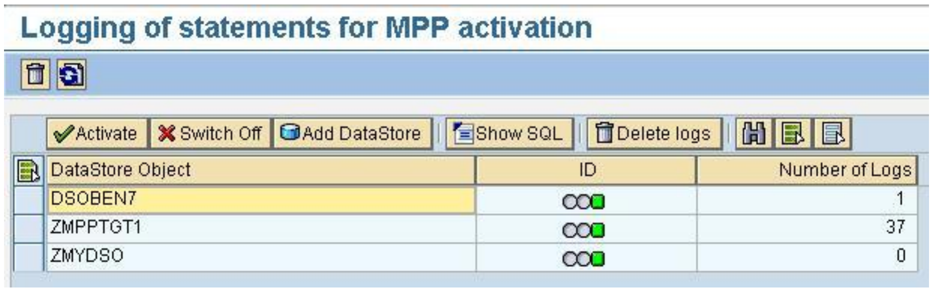
DataStore Object	ID	Number of Logs
DSOBEN7	○○🟩	1
ZMPPTGT1	○○🟩	37

DataStore

DataStore ZMYDSO

Figure 11: Enable SQL Statement Logging for a DSO

The DSO will be added to the list of DSOs for which SQL logs are collected.



DataStore Object	ID	Number of Logs
DSOBEN7	○○🟩	1
ZMPPTGT1	○○🟩	37
ZMYDSO	○○🟩	0

Figure 12: View of DSOs for Which SQL Statements Are Logged

Disabling MPP-optimized Data Activation

You can disable MPP-optimized data activation for all DSOs or selected DSOs manually in table *RSODSOMPPSWTCH*. Table *RSODSOMPPSWTCH* contains the following columns:

Column	Description
DATASTORE	Name of the DataStore object
PAR_ACT	Set to '*X*' if the standard procedure is to be used for data activation
MPP_ACT	Set to '*X*' if the MPP-optimized procedure is to be used for data activation
SQL_LOG	Set to '*X*' if the SQL statements that are executed during MPP-optimized data activation are to be logged

Table 2: Columns in Table RSODSOMPPSWTCH

To turn off MPP-optimized data activation for an individual DSO, add a new row to the table with the name of the DSO, set *PAR_ACT* to '*X*' and *MPP_ACT* and *SQL_LOG* to '*SPACE*'. If a row for the DSO already exists in the table, update the row accordingly.

To turn off MPP-optimized data activation for all DSOs, add the following row to table *RSODSOMPPSWTCH*:

Column	Description
DATASTORE	ALL
PAR_ACT	*X*
MPP_ACT	
SQL_LOG	

Note: You should not change the settings for DSOs in table RSODSOMPPSWTCH while the data activation for these DSO is still running. A data activation that is currently running or a data activation that has aborted must be completed with the activation procedure it has started with.

DB2 for LUW-Specific Implementation of MPP-optimized Data Activation

For DB2 for LUW, MPP-optimized data activation is supported for DSOs that are distributed over at least four database partitions. For DSOs that are distributed over less than four partitions, the standard data activation is used.

Implementation of the Outer-Join Table

The implementation of the new outer-join table in the database influences the performance of MPP-optimized data activation considerably. The distribution key that determines the distribution of the records in the table over the database partitions should be selected in a way that only a minimum amount of data needs to be sent between the database partitions during the execution of SQL statements for the MPP-optimized data activation. This can best be achieved by using the same distribution key columns as used for the active table, that is, the columns of the semantic key of the DSO. Selections from the outer-join table contain WHERE conditions on the ACTIVATN_METHOD column (only records flagged for MPP-optimized data activation are selected) and on the RECORDMODE values from the activation queue table (M_0RECORDMODE) and from the active table (A_0RECORDMODE). Since the number of records in a request can be large and there are only a few possible value combinations for these three columns, MDC can be applied efficiently. Thus, for DB2 for LUW, the outer-join table is implemented as follows:

- The distribution key consists of the semantic key fields of the DSO. If the active table and the outer-join table are created in the same tablespace or at least reside on the same database partitions, the data processed in the SQL statements is collocated.
- The table has no primary key index and no secondary indexes.
- MDC is used on the columns *ACTIVATN_METHOD*, A*_0RECORDMODE* (*RECORDMODE* value in active record), and *M_0RECORDMODE (RECORDMODE* value in record to be activated). These are the fields on which WHERE conditions are defined in the SQL statements that are executed during the data activation. These statements are processed efficiently by using the MDC block indexes to determine the rows in the result sets.

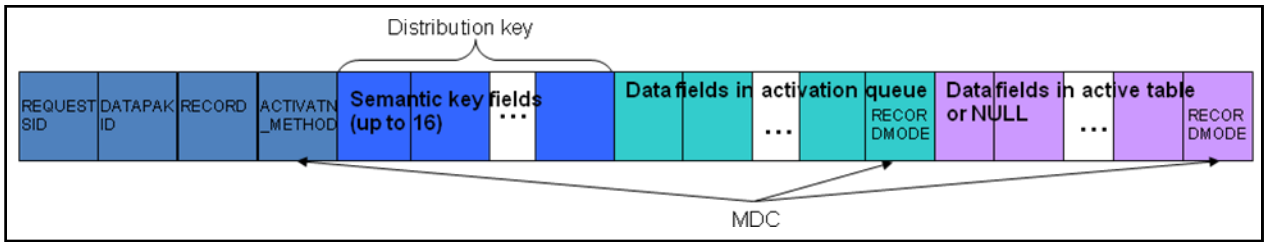

Distribution key
REQUEST SID
DATAPAK ID
RECORD
ACTIVATN _METHOD
Semantic key fields (up to 16)
...
Data fields in activation queue
...
RECOR DMODE
Data fields in active table or NULL
...
RECOR DMODE
MDC

Figure 13: Implementation of Outer-Join Table in DB2 for LUW

SQL Statements Executed in the MPP-optimized Data Activation Methods

The DB2 for LUW-specific implementation of MPP-optimized data activation consists of the methods PREPARE, INSERT, UPDATE, and DELETE. MERGE is not implemented. The SQL statements that are executed with these methods are created dynamically based on the structure of the DSO, the aggregation behavior defined for the data fields, and the size of the data packages to be written into the change log. This chapter outlines the SQL statements. The following identifiers and column names are used in the schematic description of the SQL statements:

Identifier or Column Name	Description
A_0RECORDMODE	Column in outer-join table: Indicates whether the semantic key of a record exists in the active table (set to '*X*' if semantic key exists, otherwise ' ' (SPACE))
M_0RECORDMODE	Column in outer-join table: Indicates the value of the *RECORDMODE* column from the activation queue table
ACTIVATN_METHOD	Column in outer-join table: Contains '*1'* for records that can be activated with MPP-optimized data activation and '*2'* for the other records
M_<data fields>	Columns in outer-join table: Indicate the data fields from the activation queue table
A_<data fields>	Columns in outer-join table: Indicate the data fields from the active table
<dpsize>	Number of records in a data package that are written to the change log table
<dpoffset>	Number of the first data package written to the change log table minus 1
<#rows affected on partitions 1 to n-1>	Number of records in the result set of the complex SQL statement on database partitions *1* to *n-1* (occurs in "CASE DBPARTITIONNUM(SID)" clauses where DBPARTITIONNUM(SID) corresponds to partition *n*)

Table 3: Identifiers and Column Names Used in Description of SQL Statements

PREPARE Method

With the PREPARE method, an INSERT-SELECT statement is executed that performs a left outer join on the data in the active table and on the data to be activated in the activation queue table and inserts the result into the outer-join table. Using a DB2 OLAP function (COUNT(*) over (PARTITION BY …)), this method also identifies which semantic keys in the data to be activated occur only once and which occur several times. Based on this information and on the processing instructions in RECORDMODE, it is determined whether the standard or the MPP-optimized data activation procedure is used for each record. This information is stored in column ACTIVATN_METHOD (*‘1’* for MPP-optimized data activation, *‘2’* for standard data activation). The statement has the following pattern:

```
INSERT INTO <Outer-Join table> ( ... )
SELECT
  AQ.SID, AQ.DATAPAKID, AQ.RECORD,
  AQ.<semantic key fields>,
  CASE
    WHEN AQ.RECORDMODE IN('N','D','R',' ') AND
         COUNT(*) OVER ( PARTITION BY <semantic key fields> ) = 1
    THEN 1 ELSE 2
  END CASE AS ACTIVATN_METHOD
  AQ.<data fields>, AQ.RECORDMODE
  ACT.<data fields>,
  CASE WHEN ACT.RECORDMODE IS NULL THEN ‘ ‘ ELSE ‘X’ END CASE,
  AQ.RECORDMODE
FROM
  <Activation Queue table> AS AQ
  LEFT OUTER JOIN <Active table> AS ACT
    ON AQ.<semantic key fields> = ACT.<semantic key fields>
```

Figure 14: SQL Statement for Filling the Outer-Join Table

The outer-join table is locked exclusively before the statement is executed. This is possible because only MPP-optimized data activation writes to the outer-join table.

If you are using DB2 V9.7 Fix Pack 3 or higher, the PREPARE method also identifies how many records on each database partition exist for each combination of ACTIVATN_METHOD, M_0RECORDMODE, and A_0RECORDMODE. This information is later used in the creation of the SQL statements for writing to the change log table.

```
SELECT
  DBPARTITIONNUM(SID), M__0RECORDMODE, A__0RECORDMODE, COUNT(*)
FROM
  <Outer-Join table MPP View>
GROUP BY
  DBPARTITIONNUM(SID), M__0RECORDMODE, A__0RECORDMODE
ORDER BY
  DBPARTITIONNUM(SID), M__0RECORDMODE, A__0RECORDMODE
```

Figure 15: SQL Statement for Counting the Number of Rows on Each Database Partition

INSERT Method

With the INSERT method, new records are inserted into the active table and after-image records are inserted into the change log table. The new records are marked with *‘N’* or *‘ ’* in the *M_0RECORDMODE* field and with *‘ ’* in the *A_0RECORDMODE* field.

The following is the SQL statement for writing to the active table:

```
INSERT INTO <Active table> ( ... )
SELECT
  <semantic key fields>,
  M_<data fields>,
  ‘ ‘ AS RECORDMODE
FROM
  <Outer-Join table MPP View>
WHERE
  M_0RECORDMODE IN ('N', ' ') AND
  A_0RECORDMODE = ‘ ‘
```

Figure 16: SQL Statement for Inserting into the Active Table

When writing to the change log table, the values for the technical key columns *DATAPAKID* and *RECORD* must be generated from the data package size and the data package offset that are provided as parameters. The data package offset is the number of the first data package to be written. The data package offset is provided as input parameter into the method. In order to create *DATAPAKID* and *RECORD* numbers, the records to be inserted into the change log need to be numbered. This is done with the DB2 OLAP function ROW_NUMBER() OVER (…). As of DB2 V9.7 Fix Pack 3, this function can be executed in parallel on the database partitions if called as follows:

ROW_NUMBER( ) OVER(PARTITION BY DBPARTITIONNUM(<column>))

With this function, the records in the result set on each database partition are numbered from *1* to *n*. In order to generate unique numbers for the complete result set, the total number of records in the result set on partitions *1* to *n-1* is added to the numbers generated on partition *n*. These numbers are taken from the output of the SELECT COUNT(*) statement that was executed in the PREPARE method. The following is the pattern for the SQL statement that inserts the after-images into the change log table:

```
INSERT INTO <Change Log table> ( ... )
SELECT '<requestID>',
  CASE DBPARTITIONNUM(SID)
    WHEN 1
    THEN SUBSTR(DIGITS(INT((ROW_NUMBER() OVER (PARTITION BY DBPARTITIONNUM(SID)) - 1) /
                           <dpsize>) + 1 + <dpoffset>),5)
    WHEN 2
    THEN SUBSTR(DIGITS(INT((ROW_NUMBER() OVER (PARTITION BY DBPARTITIONNUM(SID)) +
                            <#rows affected on partitions 1 to n-1>) /
                           <dpsize>) + 1 + <dpoffset>),5)
    ...
  END CASE,
  CASE DBPARTITIONNUM(SID)
    WHEN 1
    THEN MOD(ROW_NUMBER() OVER (PARTITION BY DBPARTITIONNUM(SID)) - 1,<dpsize>) + 1
    WHEN 2
    THEN MOD(ROW_NUMBER() OVER (PARTITION BY DBPARTITIONNUM(SID)) +
             <#rows affected on partitions 1 to n-1>,<dpsize>) + 1
    ...
  END CASE,
  <semantic key fields>,
  M_<data fields>,
  'N' AS RECORDMODE
FROM
  <Outer-Join table MPP view>
WHERE
  A_0RECORDMODE = ' ' AND M_0RECORDMODE IN ('N', ' ')
```

Figure 17: SQL Statement for Inserting After-Image Records into the Change Log Table

The functions SUBSTR and DIGITS convert the *DATAPAKID* number to a character field that corresponds to the data type of the *DATAPAKID* column in the change log table.

Both the active table and the change log table are locked in share mode before the execution of the INSERT-SELECT statements.

UPDATE Method

With the UPDATE method, records in the active table are updated and the before-images and after-images of these records are written to the change log. The records to be updated are marked with '*N*' or '*'*' in the *M_0RECORDMODE* field and with '*X*' in the *A_0RECORDMODE* field.

The following is the SQL statement for updating the active table:

```
MERGE INTO
  <Active table> AS T1
USING
  ( SELECT <semantic key fields>,
     max(M_<data fields>) as M_<data fields>,
     max(A_<data fields>) as A_<data fields>,
            max(M_0RECORDMODE) as M_0RECORDMODE,
   FROM <Outer-Join table MPP view>
    WHERE M_0RECORDMODE IN ('N', ' ')
      AND A_0RECORDMODE = 'X'
    GROUP BY <semantic key fields>
  ) AS T2
  ON ( T1.<semantic key fields> = T2.<semantic key fields> )
  WHEN MATCHED
    THEN UPDATE
      SET
       T1.<data fields> = T2.M_<data fields> + T2.A_<data fields>,
       T1.RECORDMODE = T2.M_0RECORDMODE
    ELSE IGNORE
```

Figure 18: SQL Statement for Updating the Active Table

Note that the update is done with a MERGE statement which is more efficient than the corresponding UPDATE statement.

In SAP NetWeaver BW, the before-image and after-image records that are written to the change log for an updated record in the active table must fulfill the following conditions:

- The before-image and after-image of an updated record must have the same *DATAPAKID*.
- The *RECORD* number of the after-image must immediately follow the *RECORD* number of the before-image (that is, *RECORD* number of after-image = *RECORD* number of before-image plus 1)

As of DB2 V9.7 Fix Pack 3, the most efficient way to achieve this is as follows:

```
INSERT INTO <change log table> ( ... )
        SELECT
          '<requestID>',
          CASE
            WHEN V.DBPN = 1
            THEN SUBSTR(DIGITS(INT((V.RN - 1) / <dpsize>) + 1 + <dpoffset>),5)
            WHEN V.DBPN = 2
            THEN SUBSTR(DIGITS(INT((V.RN - 1 + <#rows affected on partitions 1 to n-1>) /
                                   <dpsize>) + 1 + <dpoffset>),5)
            ...
          END CASE,
          CASE
            WHEN V.DBPN = 1
            THEN MOD(V.RN - 1,<dpsize>) + 1,
            WHEN V.DBPN = 2
            THEN MOD(V.RN - 1 + <#rows affected on partitions 1 to n-1>,<dpsize>) + 1,
            ...
          END CASE,
          V.<semantic key fields>,
          V.<data fields>
        FROM
          ( SELECT
              DBPARTITIONNUM(SID) AS DBPN,
              ROW_NUMBER() OVER (PARTITION BY DBPARTITIONNUM(SID)) AS RN,
              <semantic key fields>
              A_<data fields>,
              M_<data_fields>
            FROM
              <Outer-Join table MPP view>
            WHERE "A__0RECORDMODE" = 'X'
              AND "M__0RECORDMODE" IN ('N', '')
          ) AS A(A.DBPN, A.RN, A.<semantic key fields>, A_<data fields>, M_<data fields>),
          LATERAL (VALUES(A.DBPN, A.RN * 2 - 1, A.A_<data fields> [* -1], 'X'),
                         (A.DBPN, A.RN * 2, A.A_<data fields> + A.M_<data fields>, ' ')
                  )
          AS V(DBPN, RN, <semantic key fields>, <data fields>, RECORDMODE)
```

Figure 19: SQL Statement for Inserting Before–Image and After-Image Records into the Change Log Table

The SELECT clause inside the FROM clause selects the records to be updated from the outer-join table's MPP view. With the LATERAL construct, two records are created from each record (one record for the before-image and one record for the after-image). Record numbers are created such that the record number of the before-image is the record number of the after-image minus 1. In the SELECT clause that follows the INSERT clause, the values for *DATAPAKID* and *RECORD* are created from the generated record numbers.

DELETE Method

In the DELETE method, records with semantic keys to be deleted are deleted from the active table and before-image records are inserted into the change log table. The records with semantic keys to be deleted are the ones with value 'D' or 'R' in the M_0RECORDMODE field and 'X' in the A_0RECORDMODE field.

The deletion is executed with an SQL statement with the following pattern:

```
    DELETE FROM <Active table> AS T1
     WHERE EXISTS (
       SELECT 1
       FROM
         <Outer-Join table MPP view> AS T2
       WHERE
         T1.<semantic key fields> = T2.<semantic key fields> AND
         T2.A_0RECORDMODE = 'X' AND
         T2.M_0RECORDMODE IN ('D', 'R'))
```

Figure 20: SQL Statement for Deleting from the Active Table

The before-image records are inserted into the change log table using an SQL statement with the following pattern:

```
      INSERT INTO <Change Log table> ( ... )
      SELECT
        '<requestID>',
        CASE DBPARTITIONNUM(SID)
          WHEN 1
          THEN SUBSTR(DIGITS(INT((ROW_NUMBER() OVER (PARTITION BY DBPARTITIONNUM(SID)) - 1) /
                             <dpsize>) + 1 + <dpoffset>),5)
          WHEN 2
          THEN SUBSTR(DIGITS(INT((ROW_NUMBER() OVER (PARTITION BY DBPARTITIONNUM(SID)) +
                                 <#rows affected on partitions 1 to n-1>) /
                             <dpsize>) + 1 + <dpoffset>),5)
          ...
        END CASE,
        CASE DBPARTITIONNUM(SID)
          WHEN 1
          THEN MOD(ROW_NUMBER() OVER (PARTITION BY DBPARTITIONNUM(SID)) - 1,<dpsize>) + 1
          WHEN 2
          THEN MOD(ROW_NUMBER() OVER (PARTITION BY DBPARTITIONNUM(SID)) +
                   <#rows affected on partitions 1 to n-1>,<dpsize>) + 1
          ...
        END CASE,
        <semantic key fields>,
        A_<data fields> [* -1],
        'R'
      FROM
        <Outer-Join table MPP view>
      WHERE
        A_0RECORDMODE = ‘X’ AND M_0RECORDMODE IN ('D', 'R')
```

Figure 21: SQL Statement for the Insertion of Before-Image Records into the Change Log Table

6.3 SQL Statements Executed During Rollback

ROLLBACK_INSERT

The SQL statement, that is executed during the rollback of insertions into the active table deletes the records that were inserted into the active table:

```
      DELETE FROM <Active table> AS T1
      WHERE EXISTS
      (SELECT 1
       FROM <Outer-Join table MPP view> AS T2
       WHERE T1.<semantic key fields> = T2.<semantic key fields> AND
             T2.A_0RECORDMODE = ' ' AND T2.M_0RECORDMODE NOT IN ('D', 'R'))
```

Figure 22: SQL Statement for Rollback of Insertions into the Active Table

ROLLBACK_UPDATE

The SQL statement that is executed during the rollback of updates of the active table resets the data values of the updated records to the values they had before the update:

```
MERGE INTO <Active table> AS T1
USING
   ( SELECT <semantic key fields>,
            max(A_<data fields>) as A_<data fields>,
            ' ' as A_0RECORDMODE
     FROM  <Outer-Join table MPP view>
     WHERE M_0RECORDMODE NOT IN ('D', 'R')
       AND A_0RECORDMODE = 'X'
     GROUP BY <semantic key fields>
   ) AS T2
   ON ( T1.<semantic key fields> = T2.<semantic key fields> )
   WHEN MATCHED
     THEN UPDATE
       SET
        T1.<data fields> = T2.A_<data fields>,
        T1.RECORDMODE = T2.A_0RECORDMODE
     ELSE IGNORE
```

Figure 23: SQL Statement for the Rollback of Updates of the Active Table

ROLLBACK_DELETE

The following SQL statement that is executed during the rollback of deletions from the active table re-inserts the records that were deleted from the active table:

```
INSERT INTO <Active table>
SELECT
  <semantic key fields>, A_<data fields>, ' ' as RECORDMODE
FROM
  <Outer-Join table MPP view>
WHERE
  A_0RECORDMODE = 'X' AND   M_0RECORDMODE IN ('D', 'R')
```

Figure 24: SQL Statement for Rollback of Deletions from the Active Table

MPP-Optimized Data Activation Performance

MPP-optimized data activation has been tested extensively in an SAP Retail environment using the layered scalable architecture for POS data management. The results are published in the following whitepaper which is available on the SAP SDN:

SAP Enterprise Data Warehouse for Point of Sales Data Optimized for IBM DB2 for Linux, UNIX, and Windows on IBM Power Systems,
http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/b074b49a-a17f-2e10-8397-f2108ad4257f

These tests have shown that MPP-optimized data activation is approximately 2 to 2.5 times faster than standard data activation on SAP NetWeaver BW 7.30. The DSOs were distributed over seven database partitions and the size of the requests was about 16 million records. Note that - compared to SAP NetWeaver BW 7.0 and Enhancement Package 1 of SAP NetWeaver BW 7.0 - performance of standard data activation has already considerably improved.

Usage Recommendations

You should use at least IBM DB2 V9.7 for LUW Fix Pack 3 for SAP. This Fix Pack 3 for SAP contains an APAR that improves the performance of the SQL statements that insert records into the change log table.

Performance usually improves with the number of database partitions on which the DSO is located, provided enough CPU resources are available to process the SQL statements in parallel.

For MPP-optimized data activation, it is important that all DSO tables reside on the same database partitions. The easiest way to achieve this is to define the same data class for the activation queue table as for the other DSO tables. By using the same data class the tables are created in the same tablespace and are, by default, distributed over the same database partitions. If you want to create the activation queue table in a different tablespace, this tablespace should reside on the same database partitions as the tablespace for the other DSO tables.

You should apply SAP note 1515687. This note enables that the activation queue table of standard DSOs uses the same columns as distribution key for hash-partitioning as the active table and the outer-join table. You need to set the RSADMIN parameter DB6_DSO_TABLES_COLLOCATED to YES and re-activate the DSOs you want to change. This setting further reduces the amount of data that has to be transferred between database partitions during MPP-optimized data activation.

Applying DB2 data and index compression can considerably increase performance if I/O is the limiting factor. This is especially true for wide DSOs with many data fields and large requests to be activated.

Performance might vary depending on the available CPU resources. Due to parallel processing on the database partitions on which the DSO resides, one processor core or thread is occupied per database partition. If the system has limited CPU resources because another CPU-intensive workload is running in parallel, performance might be affected.

If the record length of the outer-join table of a DSO exceeds the maximum record length possible for the tablespace in which the table is to be created MPP-optimized data activation cannot be used for this DSO. In order to ensure that this is checked during the activation of the DSO you must implement SAP notes 1619782 and 1621978.

For information about how to proceed after an upgrade to SAP NetWeaver BW 7.3 from earlier SAP NetWeaver BW releases, see SAP note 1621978.

Log Space Consumption Considerations

Log space consumption depends on the number of records to be activated and on the record width. The minimum unit processed during data activation is one request. For large requests with 10 million records or more, log space consumption might be high.

The following is an estimate of the number of records inserted, updated, or deleted in the database transactions executed in the steps of MPP-optimized data activation:

The number of records inserted into the outer-join table during the PREPARE step corresponds to the size of the requests in the activation queue table to be activated. The record width is about twice the width of the DSO active table. Since the outer-join table uses MDC only and does not have a primary key index or secondary indexes, log space consumption for indexes is low.

The maximum number of records inserted into the active table during the INSERT step equals the size of the requests in the activation queue to be activated. This is the case if all semantic keys are new and all records can be activated using the MPP-optimized data activation procedure. In the same database transaction, the same number of records that are inserted into the active table are inserted into the change log table as after-images.

The maximum number of records updated in the active table during the UPDATE step equals the size of the requests in the activation queue to be activated. This is the case if all semantic keys already exist and all records can be activated using the MPP-optimized data activation procedure. In the same database transaction, twice the number of records that are updated in the active table are inserted into the change log table as before-images and after-images.

The maximum number of records deleted from the active table in the DELETE step is the size of the requests in the activation queue to be activated. This is the case if all semantic keys already exist and are to be deleted and all records can be activated with the MPP-optimized data activation procedure. In the same database transaction, the same number of records that are deleted from the active table are inserted into the change log table as before-images.

Note that log space consumption increases if you add indexes to the DSO in the Data Warehousing Workbench. The indexes, which are defined in the Data Warehousing Workbench, are created on the active table of the DSO in addition to the primary key index. This primary key index is defined on the semantic key columns of the DSO. The change log table only has the primary key index defined on the technical key fields REQUEST, DATAPAKID, and RECORD.

In general, applying DB2 data and index compression reduces log space consumption. By default, compression is activated for DSO tables including the outer-join table if you have a license for the DB2 Storage Optimization Feature.

Summary

IBM DB2 for Linux, UNIX, and Windows is an industry-leading database solution that delivers maximum performance, scalability and reliability with a wide range of operating systems and platforms. DB2 has been constantly optimized to support SAP environments over the last 10 years. SAP NetWeaver BW explores central DB2 warehousing features like DPF and MDC in reporting as well as in ETL processing already for a long time. MPP-optimized activation of data in standard DSOs is a new optimization of an important ETL operation in SAP NetWeaver BW 7.3. The native implementation of MPP-optimized data activation on DB2 for LUW boosts performance and thus helps to address the challenging business demands of fast data processing and efficient management of data in an Enterprise Data Warehouse (EDW) with a low total cost of ownership (TCO).

Related Content

SAP NetWeaver BW – What's New With SAP NetWeaver BW 7.3

SAP Enterprise Data Warehouse for Point of Sales Data Optimized for IBM DB2 for Linux, UNIX, and Windows on IBM Power Systems

https://websmp101.sap-ag.de/~sapidb/011000358700001420572010E.PDF

http://help.sap.com/nwbw73

IBM DB2 9.7 for Linux, UNIX and Windows Information Center

For more information, please visit the EDW homepage.

Disclaimer and Liability Notice