501-EMD-001

EDGRS Database Performance Study

White Paper

October 2003

Prepared Under Contract NAS5-03098

RESPONSIBLE AUTHOR

Siwei Xu /s/	10/29/03
Siwei Xu, Software Engineer ECS Maintenance and Development Project	Date

RESPONSIBLE OFFICE

Arthur Cohen /s/	10/29/03
Arthur Cohen, Custom Code Maintenance ECS Maintenance and Development Project	Date

Raytheon Company
Upper Marlboro, Maryland

Abstract

Goddard Space Flight Center uses a reporting tool called EDGRS to measure the Ingest rate of its science products. The reporting tool also queries the Inventory metadata tables for update events. This later query was identified as a performance issue. This paper identifies the problem query and suggests an option for improving its performance.

Keywords: EDGRS, SDSRV, performance, database, event, update

This page intentionally left blank.

Contents

This page intentionally left blank.

1. Introduction

1.1 Purpose

Provide a recommendation for improving the performance of the EDGRS query.

The EDGRS query is used to identify granules recently updated within SDSRV. In addition to EDGRS, there are other functions within ECS that query for recent SDSRV granule updates. The new enhancement described in this paper could improve performance for both EDGRS and these other ECS functions.

1.2 Organization

This paper is organized as follows: Section 1, Introduction; Section 2, Current Performance Issue; Section 3, The New Enhancement.

Questions regarding technical information contained within this Paper should be addressed to the following EMD contacts:

- EMD Contacts
  - Siwei Xu, Software Engineer, 301-925-0882, sxu@eos.hitc.com

Questions concerning distribution or control of this document should be addressed to:

Data Management Office
The EMD Project Office
Raytheon Systems Company
1616 McCormick Drive
Upper Marlboro, MD 20774-5301

This page intentionally left blank.

2. References

The following ECS documentation apply:

Release 6B Science Data Server Database Design and Schema Specifications for the ECS Project.

The ECS document numbers are:

311cd62401s1
311cd62401s3a
311cd62401s3b
311cd62401s4
311cd62401sA

This page intentionally left blank.

3. Current Performance Issue

3.1 Current Performance Issue

The current mechanism for identifying recent updates to SDSRV granules is to query the lastUpdate column within the DsMdGranules table. However, this column does not provide any information on what part of the granule was updated. In addition, queries on the lastUpdate column typically require a table scan because the column is not indexed. Indexing the column would be inefficient because it is updated frequently.

Two problems that had to be resolved included: finding an efficient way to identify recent granule updates and to identify which part of the granule was updated.

Specifically, the EDGRS script contains a query similar to:

```
SELECT  "dbID"                 = Rtrim(Ltrim(convert(varchar(19) ,dbID))),
        "ShortName"            = Rtrim(Ltrim(convert (varchar(9),ShortName))),
        "SizeMBECSDataGranule" = Rtrim(Ltrim(convert(varchar(25),
                                 convert(numeric(25,7),SizeMBECSDataGranule)))),
        "BeginningDateTime"    = Rtrim(Ltrim(convert
                                       (varchar(26),BeginningDateTime))),
        "EndingDateTime"       = Rtrim(Ltrim(convert
                                       (varchar(26),EndingDateTime))),
        "insertTime"           = Rtrim(Ltrim(convert (varchar(26),insertTime))),
        "ProductionDateTime"   = Rtrim(Ltrim(convert
                                       (varchar(26),ProductionDateTime))),
        "lastUpdate"           = Rtrim(Ltrim(convert (varchar(26),lastUpdate))),
        "LocalGranuleID"       = Rtrim(Ltrim(convert
                                       (varchar(100),LocalGranuleID))),
        "VersionID"            = Rtrim(Ltrim(convert (varchar(5),VersionID))),
        "deleteEffectiveDate"  = Rtrim(Ltrim(convert
                                       (varchar(26),deleteEffectiveDate))),
        "DeleteFromArchive"    = Rtrim(Ltrim(convert
                                       (varchar(5),DeleteFromArchive)))
FROM    DsMdGranules
WHERE   lastUpdate >= DATEADD(DAY, -15, getdate())
```

This query selects information about granules that were updated within the past 15 days.

The actual query within the EDGRS script uses the insertTime column to break this query into 2 parts, but the above query captures the nature of the problem without the added complexity. This analysis is also consistent with the goals identified through conversations with the EDGRS development team. The primary interest is in “re-versioned” granules and granules marked or unmarked for deletion.

This page intentionally left blank.

4. The New Enhancement

4.1 Capturing update event history

The new design for finding recent granule metadata updates begins with adding two new tables to the SDSRV database for capturing all the identified update events that can happen to the granule's metadata.

The tables are:

```
      DsMdGrEventDomain-   which defines the update events.
      DsMdGrEventHistory- which records the update event history.
```

The following are the table definitions:

```
   create table DsMdGrEventDomain (
              eventId         smallint        not null,
              eventName       varchar(50)     not null,
              constraint PK_DSMDGREVENTDOMAIN PRIMARY KEY (eventId)
   )

   create table DsMdGrEventHistory (
              dbID            ID              not null,
              eventTime       datetime        not null,
              eventId         smallint        not null,
              constraint FK_DSMDGREVENTHISTORY FOREIGN KEY(eventId)
                         REFERENCES dbo.DsMdGrEventDomain(eventId)
   )

   create clustered index xDsMdGrEventHistory on
          DsMdGrEventHistory(eventId, eventTime)
```

Note: The foreign key constraint between the DsMdGranules table and the new DsMdGrEventHistory table is omitted due to the purpose of preserving the event history when the actual granule is deleted from the DsMdGranules table. However, the constraint does get enforced when the rows are inserted in the triggers.

The DsMdGrEventDomain table will be populated immediately after creation with the following rows:

Note: eventId 1-20 are reserved for update events on DsMdMeasuredParameter.

```
INSERT DsMdGrEventDomain VALUES  (1, "Update DsMdMeasuredParameter QAMUT")
INSERT DsMdGrEventDomain VALUES  (2, "Update DsMdMeasuredParameter others")
```

Note: eventId 21-50 are reserved for update events on DsMdMeasuredParameter

```
INSERT DsMdGrEventDomain VALUES (21, "Update DsMdGranules VersionID")
INSERT DsMdGrEventDomain VALUES (22, "Update DsMdGranules deleteEffectiveDate")
INSERT DsMdGrEventDomain VALUES (23, "Update DsMdGranules DeleteFromArchive")
INSERT DsMdGrEventDomain VALUES (24, "Update DsMdGranules processingHistoryId")
INSERT DsMdGrEventDomain VALUES (25, "Update DsMdGranules others")
```

Note: eventId 51-60 are reserved for update events on DsMdFileStorage

```
INSERT DsMdGrEventDomain VALUES (51, "Update DsMdFileStorage checkSum")
INSERT DsMdGrEventDomain VALUES (52, "Update DsMdFileStorage fileSize")
INSERT DsMdGrEventDomain VALUES (53, "Update DsMdFileStorage others")
```

Note: eventId are reserved for each of the following tables

```
INSERT DsMdGrEventDomain VALUES (61, "Update DsMdAncillaryInput")
INSERT DsMdGrEventDomain VALUES (71, "Update DsMdInputGranule")
INSERT DsMdGrEventDomain VALUES (81, "Update DsMdBrowseGranuleXref")
INSERT DsMdGrEventDomain VALUES (91, "Update DsMdGrSensorCharacteristics")
INSERT DsMdGrEventDomain VALUES (101, "Update DsMdGrStorageMedium")
INSERT DsMdGrEventDomain VALUES (111, "Update DsMdGrVerticalSpatialDomain")
INSERT DsMdGrEventDomain VALUES (121, "Update DsMdGranuleAnalysisXref")
INSERT DsMdGrEventDomain VALUES (131, "Update DsMdGranuleCampaignXref")
INSERT DsMdGrEventDomain VALUES (141, "Update DsMdGranuleLocality")
INSERT DsMdGrEventDomain VALUES (151, "Update DsMdGranuleReview")
INSERT DsMdGrEventDomain VALUES (161, "Update DsMdGranuleVersions")
INSERT DsMdGrEventDomain VALUES (171, "Update DsMdOrbitCalcSpatialDomain")
INSERT DsMdGrEventDomain VALUES (181, "Update DsMdOrbitCalculatedSpatial")
INSERT DsMdGrEventDomain VALUES (191, "Update DsMdQaGranuleXref")
INSERT DsMdGrEventDomain VALUES (201, "Update DsMdXAR")
INSERT DsMdGrEventDomain VALUES (211, "Update DsMdUninterpretedData")
```

These rows capture all the known metadata update Use Cases as well as many other possible use cases. The events that don't have identified use cases were added because the schema currently contains update triggers on these metadata tables, and it is possible that some DAAC's may have their own scripts for updating them. These update triggers currently cause the lastUpdate column within the DsMdGranules table to be modified.

As a future use, the design allows for additional events to be defined within each logical group by leaving gaps between the ranges for the logical groups. The range for each table is larger than the actual number of columns in the table. This allows each column to have it's own update eventId. A query can easily identify all updates to a table by specifying the appropriate range of consecutive eventIds or identify updates to specific columns within a table(s) by specifying a list of eventIds. The DAACs can insert new rows into the DsMdGrEventDomain table to capture events not included in the above list. In this case the DAAC will have to supply a mechanism for inserting the event into the DsMdGrEventHistory table.

For example:

```
INSERT DsMdGrEventDomain VALUES (6, "Update DsMdGranules DayNightFlag")
```

When the new event happens in a DUE script, the script needs to insert a row in the DsMdGrEventHistory table:

```
INSERT DsMdGrEventHistory VALUES (@dbID, getdate(), 6)
```

By introducing the two tables and having a composite index on eventId and eventTime the design provides an equivalent for an index on the DsMdGranules' lastUpdate column. These tables support a much more efficient way to determine all the granules that were updated during a specified time period for a given update event. Events will be stored in the

DsMdGrEventHistory for a configurable amount of time to allow each DAAC to specify what it considers a recent update. The DAACs need to be aware that the performance of the system will be related to the amount of event history maintained. A retention period of 2 weeks is recommended.

4.2 The new EDGRS query

The EDGRS query that is currently experiencing performance problems can take advantage of this new design by using the following query:

```
DECLARE @searchDate datetime
SELECT  @searchDate = DATEADD(DAY, -15, getdate())

SELECT  "dbID"                  = convert(varchar(19) ,dbID),
        "ShortName"             = convert (varchar(9),ShortName),
        "SizeMBECSDataGranule"  = convert(varchar(25),convert(numeric(25,7),
                                  SizeMBECSDataGranule)),
        "BeginningDateTime"     = convert (varchar(26),BeginningDateTime),
        "EndingDateTime"        = convert (varchar(26),EndingDateTime),
        "insertTime"            = convert (varchar(26),insertTime),
        "ProductionDateTime"    = convert (varchar(26),ProductionDateTime),
        "lastUpdate"            = convert (varchar(26),lastUpdate),
        "LocalGranuleID"        = convert (varchar(100),LocalGranuleID),
        "VersionID"             = convert (varchar(5),VersionID),
        "deleteEffectiveDate"   = convert (varchar(26),deleteEffectiveDate),
        "DeleteFromArchive"     = convert (varchar(5),DeleteFromArchive)
FROM    DsMdGranules
WHERE   dbID in (select distinct dbID
                 From DsMdGrEventHistory
                 Where eventId   between 21 and 22
                 AND   eventTime >= @searchDate)
```

Note: We introduced @searchDate so that dateadd() and getdate() functions only need to be computed once. We removed all the rtrim() and ltrim() function calls to increase performance and because they didn't seem to make any difference when we executed the query.

EDGRS is currently using 15 days as the time range span. It's our understanding that the query is essentially run everyday. Having a query that covers 15 days could potentially generate 15 times the amount of work that would normally be needed. If this is used to guard against occasional gaps in running the script or script failures, then a different approach that tracks the last successful run date/time is recommended. One could use this plus some safety margin (which now can be much, much smaller than 15 days) to compute the time interval that needs to be covered.

A subtle new feature of this design is the ability to capture granules that are deleted during the query range. The current system queries the deleteEffectiveDate within the DsMdGranules table. This method can be inaccurate if the granule is physically deleted prior to running the EDGRS script. With the new design, by relaxing the integrity constraint between the DsMdGrEventHistory and the DsMdGranules tables, the records of the delete event are retained in the DsMdGrEvenHistory table even after the granules are physically deleted. To capture the dbIDs of the deleted granules, the above query needs to be modified to include an outer join on dbIDs so that it will select all the dbIDs from DsMdGrEventHistory table even if they don't exist in DsMdGranules table.

The following is the showplan for the new example query:

```
QUERY PLAN FOR STATEMENT 1 (at line 1).
    STEP 1
        The type of query is SELECT.
        FROM TABLE
            DsMdGrEventHistory   b
        Nested iteration.
        Using 2 Matching Index Scans
        Using Clustered Index.
        Index : xDsMdGrEventHistory
        Forward scan.
        Positioning by key.
        Keys are:
            eventId  ASC
            eventTime  ASC
        Using Clustered Index.
        Index : xDsMdGrEventHistory
        Forward scan.
        Positioning by key.
        Keys are:
            eventId  ASC
            eventTime  ASC
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            DsMdGranules    a
        Nested iteration.
        Index : PK_DSMDGRANULES
        Forward scan.
        Positioning by key.
        Keys are:
            dbID  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
        With LRU Buffer Replacement Strategy for index leaf pages.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
```

4.3 Maintenance Issue

The new DsMdGrEventDomain does not require any ongoing maintenance other than entering new events to capture. The new DsMdGrEventHistory table will need to be cleaned up periodically. This "clean-up" should be configured to be consistent with the running of the EDGRS script. The cleanup is configured as a "background" task within the EcDsScienceDataServer. This means the cleanup can only occur if the EcDsScienceDataServer is running. The EcDsScienceDataServer configuration file contains the following new parameters for controlling the cleanup:

ParameterName	Default Value
SDSRV_GREVENTHIST_DATARETENTIONDAYS	14
SDSRV_GREVENTHIST_BATCHDELETESIZE	500
SDSRV_GREVENTHIST_CLEANUPINTERVALSECS	3600

Hints:

Since the DsMdGrEventHistory table can have multiple rows for each granule that's updated, the system should be configured to remove events on a timely basis. The performance of the system is directly related to the number of rows in this table.

Since the data in this table changes all the time, we need to run update statistics on this table frequently.

4.4 Prototyping results

Prototyping a query to simulate the join between the DsMdGrEventHistory table and the DsMdGranules table was conducted in the PVC.

The following is a summary of the results.

“Num rows matching event" represents the number of rows in the DsMdEventHistory table that matched the query (and thus were joined to DsMdGranules). As a reference point it takes 240 minutes to table scan the 17 million row DsMdGranules table.

Num rows DsMdGranules	Num rows DsMdEventHistory	Num rows matching event	time(minutes)
17 million	1 million	400,000	24
17 million	1 million	1 million	61
17 million	2 million	100,000	6
17 million	2 million	200,000	12
17 million	2 million	400,000	23
17 million	2 million	2 million	110

The new design appears 10 times faster than the table scan when processing 400,000 events. As expected, the performance degrades as the number of matching events increases (looks to be linear). It's still 1/2 the table scan time with 2 million matching events. Note that the performance tends to remain stable as the number of events stored is increased (400,000 matches takes 24 minutes with 1 or 2 million total events).

In addition, the “cleanup” of events from the DsMdGrEventHistory table was also prototyped.

It took roughly 2 minutes to clean 200,000 rows out of 4 million rows in DsMdGrEventHistory table. Therefore it has very little impact on the update activities on this table.

This page intentionally left blank.