MTR070028

MITRE TECHNICAL REPORT

Sensor Data & Analysis Framework (SDAF) Data Warehouse

February 2007

Eddy Cheung, E549
Stephan Nadeau, E145
Don Landing, E145
Mark Munson, E543
Jennifer Casper E547

Sponsor: The MITRE Corp.
Dept. No.: E145 **Project No.:** 03MSR002-A7

MITRE
Center for Air Force C2 Systems
Bedford, Massachusetts

MITRE Department
and Project Approval:

Donald Landing

Project Approval:

Stephan Nadeau

Abstract

The Sensor Data & Analysis Framework (SDAF) Data Warehouse is part of the SDAF project. As more and more sensors are producing volumes of data regarding objects that change with respect to location and time, evaluating this stream of information in a timely fashion requires the integration of current and historical data. The SDAF research project seeks to investigate and understand various approaches to integrating streamed and historical sensor data to support spatio-temporal queries. The SDAF Data Warehouse (SDAF DW) effort experiments with different techniques to store and organize the historical sensor data efficiently in a persistent data store.

In this paper, we will discuss the finding that to support spatial queries of objects relating to location and time, it is best to partition the data by date/time. There are two advantages to the approach. First, partition elimination keeps the number of partitions to search for a query to a minimum. Second, as the size of the database grows, the number of partitions to search for the same query remains the same, thereby keeping the response time relatively constant.

KEYWORDS: SDAF, Sensor Data and Analysis Framework, Moving Target indicator, MTI, SMTI, GMTI, Data Warehouse

Table of Contents

List of Figures

List of Tables

1 Overview

As Phase I of the experiment to store spatio-temporal data efficiently, Ground Moving Target Indicator (GMTI) data in the NATOEX format is used. NATOEX is a GMTI data format widely used in sensors, as well as trackers and visualization tools. A new GMTI standard gaining popularity is STANAG 4607. The format is scalable to allow all types of radar systems to use the format and tailor the data flow to the capabilities of the sensor and the available communications channels. Smaller systems can use the basic capabilities of the format to transmit only moving target reports. Larger, more capable systems can use the same format for the moving target reports, and provide high range resolution data, and other products of extended processing of the radar returns.

The GMTI data collected by a sensor for one mission (between four to eight hours) is contained in a binary file. The one or more mission data files generated per day are loaded into a database. In order to accommodate the dual demands of continuously ingesting new data into a database, and at the same time making the database available for query operations, the solution is to organize the database tables into partitions.

The information of most interest in the database is the location and detection time of each moving target. To best support spatial queries of the kind “select all the targets within distance X of location Y between time 1 and time 2”, the decision is to partition the tables by date/time. We also partition the same data by mission for comparison.

In this paper, we will detail the steps in setting up the SDAF DW. We will also discuss the reasoning behind how the data is organized. We will show the same data organized differently can produce dramatic improvement in query response time.

2 GMTI Data

The GMTI data in NATOEX format consists of variable length packets. Each packet has a 128-byte Header Segment plus zero or more Moving Target Indicator (MTI) Segments (Figure 2-1). The maximum packet size cannot be more than 1472 bytes.

Header Segment (128 bytes)
Zero or more MTI Segments (Maximum of (1472-128=1344) Bytes)

Figure 2-1 NATOEX Packet Structure

The Header Segment (Figure 2-2) contains the type of message, the number of subsequent MTI segments in the packet and other information.

Field	**Field Name**	**Subfield Name**	**Subfield Type**
H1	Message Indicator	Message Flags	4-bit flags
		Message Type	12-bit enumeration
H2	Radar Mode		8-bit enumeration
H3	Scan Flags		8-bit enumeration
H4	Resolution	Range Resolution	16-bit integer
		Cross-Range Resolution	16-bit integer
H5	Label		8x8-bit ASCII characters
H6	Sequence Number		16-bit unsigned integer
H7	Target or Imagery Packet Count		16-bit unsigned integer
H8	Scan Area (Rectangular Format)	Point 1 - X	32-bit floating point
		Point 1 - Y	32-bit floating point
		Point 1 - Z	32-bit floating point
		Point 2 - X	32-bit floating point
		Point 2 - Y	32-bit floating point

		Point 2 - Z	32-bit floating point
		Point 3 - X	32-bit floating point
		Point 3 - Y	32-bit floating point
		Point 3 - Z	32-bit floating point
	Scan Area (Polar and Aim Point Formats)	Min Range	32-bit floating point
		Max Range	32-bit floating point
		Start Azimuth	32-bit floating point
		Stop Azimuth	32-bit floating point
		Aim Point - X	32-bit floating point
		Aim Point - Y	32-bit floating point
		Aim Point - Z	32-bit floating point
		Dummy	32-bit floating point
		Dummy	32-bit floating point
H9	Scan Number		16-bit unsigned integer
H10	Service Request Number		16-bit unsigned integer
H11	Sensor Platform Time Stamp		64-bit unsigned integer
H12	Data Time Stamp		64-bit unsigned integer
H13	Sensor Platform ID	Platform Type	8-bit enumeration
		Platform Track Number	24-bit unsigned integer
H14	Sensor Platform Position	X	32-bit floating point
		Y	32-bit floating point
		Z	32-bit floating point
H15	Sensor Platform Velocity	V_X	32-bit floating point
		V_Y	32-bit floating point
		V_Z	32-bit floating point
H16	Sensor Platform Heading		32-bit floating point
H17	Topocentric Origin	Latitude	32-bit floating point

		Longitude	32-bit floating point
		Elevation	32-bit floating point
H18	Protocol Version Number		16-bit unsigned integer
H19	Byte Count		16-bit unsigned integer
H20	Sending Platform ID	Platform Type	8-bit enumeration
		Platform Track Number	24-bit unsigned integer

Figure 2-2 NATOEX Header Segment

Each MTI Segment (Figure 2-3) specifies information such as location, speed, and heading for one target. No more than 42 target reports can be contained in a single packet.

Field	**Field Name**	**Subfield Name**	**Subfield Type**
M1	Target Location	X	32-bit floating point
		Y	32-bit floating point
		Z	32-bit floating point
M2	Target Radial Velocity		32-bit floating point
M3	Target Radar Cross Section		32-bit floating point
M4	Target Classification		8-bit enumeration
M5	Radial Velocity Quality		8-bit unsigned integer
M6	Count		16-bit unsigned integer
M7	Truth Tag	Count	8-bit unsigned integer
		Application	8-bit unsigned integer
		Entity	16-bit unsigned integer
M8	Error	Range Error	16-bit unsigned integer
		Cross-Range Error	16-bit unsigned integer

Figure 2-3 NATOEX MTI Segment

3 Database Schemas

Of all the information in a NATOEX packet, the target location and detection time in the MTI segment are of most interest. The spatio-temporal nature of the data requires a database management system that supports and facilitates the storage, retrieval, update, and query of spatial features in a database. Oracle 10g Release 2 has matured spatial features and can handle large amounts of data. It is the best available choice as the Database Management System (DBMS) for the SDAF DW effort.

The format of the NATOEX data lends itself very nicely to a database schema with three major tables: MISSION, MESSAGE, and DOT. The MISSION table contains the mission ID, start time, end time, etc. The MESSAGE table contains packet header information. The DOT table contains the target information. The geometric description (latitude, longitude) of the spatial object (target) will be stored in a single row, in a single column of object type SDO_GEOMETRY in the DOT table.

A table containing a spatial object data requires a spatial index for efficient access to the data. This is because Oracle Spatial uses a two-tier query model to resolve spatial queries. The primary filter permits fast selection of candidate rows to pass along to the secondary filter. The secondary filter applies exact selection criteria to produce the result set. Oracle uses the spatial index to implement the primary filter. In addition, Oracle does not require the use of both primary and secondary filters. In some cases, just using the primary filter is sufficient to produce the desired results.

The SDAF DW effort benefited from existing work establishing a Forensic GMTI Data Warehouse (GMTI DW). The GMTI DW project stores GMTI data collected by disparate sensors to support forensic analysis of intelligence and sensor data. Using the data and a set of tools, the analyst can identify and annotate tracks, reconstruct an event, isolate and identify patterns of behavior, etc. SDAF DW leveraged the GMTI DW database schemas (Figure 3-1) and software tools in order to speed up development time and prevent duplication of effort. The SDAF DW effort started with Version 3 of the GMTI DW schemas and software. We updated to Version 4 when the GMTI DW was upgraded. The SDAF DW Version 4 (SDAF DWv4) database schemas vary slightly from the GMTI DW Version 4 (GMTI DWv4) database schemas (Table 3-1). The only major difference is how to organize data into partitions.

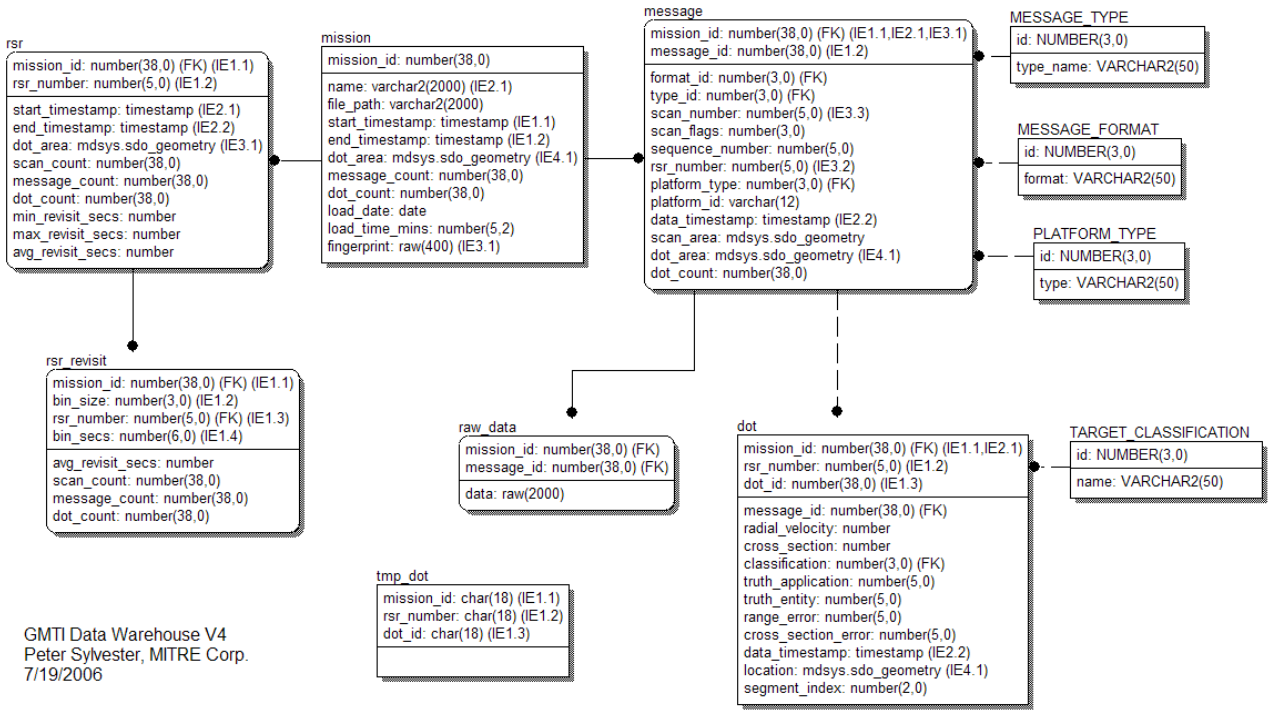

Figure 3-1 GMTI DWv4 Database Schemas

Tables	GMTI DWv4	SDAF DWv4
MISSION		
MESSAGE		
DOT		LAT and LON columns are added to hold latitude and longitude values of the target
RAW_DATA	A table to hold NATOEX binary data	As the most important information of the NATOEX packets is stored in the database as columns, we decided there is no need to store the binary data

Table 3-1 Differences between GMTI DWv4 and SDAF DWv4 Database Schemas

3.1 GMTI DWv4 Database Tables

The GMTI DWv4 database consists of six major tables (Appendix A). The MISSION table contains information about each mission. The MESSAGE table contains header information of NATOEX packets. The DOT table contains information about ground moving targets. The Radar Service Request (RSR) and RSR_REVISIT tables contain target data information associated with RSR. The RAW_DATA table contains the original binary NATOEX data.

3.2 SDAF DWv4 Database Tables

The SDAF DWv4 database consists of five major tables (Appendix B). The MISSION table contains information about each mission. The MESSAGE table contains header information of NATOEX packets. The DOT table contains information about ground moving targets. The RSR and RSR_REVISIT tables contain target data information associated with RSR.

4 Partitioning of Database Tables

Partitioning enhances database performance and scalability. A partition is a smaller, more manageable piece of a table or index. Each partition of a table or index must have the same logical attributes, such as column names, data types, and constraints. However, different partitions can have separate physical attributes and tablespaces. Each row in a partitioned table unambiguously belongs to a single partition. The partition key is a set of from one to 16 columns that determines the partition for each row. Oracle uses the partition key to decide in which partition to put each row of data.

4.1 Benefits of Partitioning

The benefits of partitioning are many. Some of the more important ones are:

- Store data in different tablespaces on a partition-by-partition basis
- Store indexes in different tablespaces on a partition-by-partition basis

This allows the spread of I/O load associated with table or index accesses across multiple disk drives and/or controllers.

- Search multiple table or index partitions in parallel
- Eliminate partitions from consideration based on a partition key

The last bullet is one of the most important ways partitioning can enhance performance. Partition elimination is the automatic exclusion of partitions that will not be participating in a query. If a query includes a partition key as a predicate in the WHERE clause, Oracle will automatically route the query to the partition or partitions that are associated with the query, eliminating (and not searching) those partitions that will not have data included in the result set. Partition elimination significantly reduces the amount of data and index information searched to return results.

Another major reason for implementing partitioning is to accommodate the competing demands of making the database available for query operations, and at the same time allowing ingest of new data into the database on a continuous basis.

Ingesting data into a non-partitioned table requires the following steps:

1. Disable/delete current indexes
2. Ingest new data
3. Rebuild all the indexes

Depending on the size of the table, rebuilding all the indexes may take a long time. Regardless, the existing data is not available for access by users during rebuild time. With partitioning, on the other hand, only the partition with the newly inserted data is not available. All other data partitions are online and ready for query operations.

4.2 Spatial Partition Pruning

Spatial partition pruning is another technique to enhance performance. In this case, location values, such as latitude and longitude, determine how to group data into partitions. Spatial partition pruning is similar to partition elimination, but it is based on location and a partition key is not required on the input query line. At query time, an area-of-interest of the query is compared to the Minimum Bounding Rectangle (MBR) of each partition. If they do not overlap, spatial partition pruning will occur without searching the data associated with that partition.

4.3 Partitioning Schemes

For the GMTI DWv4, the major tables are partitioned on mission ID. This means all the data related to a mission is stored in one partition. For queries such as “retrieve information

for a particular mission or a list of missions" will be very fast because Oracle needs to search only one partition or only the relevant partitions, and eliminates all other ones.

For the SDAF DWv4, we decided to partition the MESSAGE and DOT tables on date/time. The data is stored in one partition if all the data related to a mission is collected within a single day. The data will be stored in multiple partitions if a mission spans over multiple days. We based our decision on the fact that NATOEX data deals with target location and time, and we feel most of the queries will be of the type "retrieve target information within X meters of a location Y between date/time 1 and 2". This type of query eliminates all partitions that fall outside of the date/time specified, thereby speeds up the query response time.

There are of course advantages and disadvantages to each approach, as listed below:

GMTI DWv4	**SDAF DWv4**
Fast response on queries based on mission ID	Fast response on queries based on date/time
A long mission that spans over multiple days will create a very large partition	Data for a mission that spans multiple days will be stored in multiple partitions
Data collected by different missions (sensors) on the same day will be stored in different partitions	Data collected by different missions (sensors) on the same day will be stored in one partition. This can be remedied by partitioning the data on sensor/date/time
In order to achieve reasonable response time in retrieving target information from the DOT table, the MISSION and/or RSR table must be queried first to manually eliminate missions (partitions) that are not needed in the result set. This can only be done through a program and/or PL/SQL procedure	Target information can be queried interactively using standard tools such as SQLPLUS with reasonable response time. Partition elimination is done by Oracle automatically
Need to create more indexes to support query operations	Other than the required spatial indexes, only primary key index is created for each table
Can add partitions interactively	Need to add partitions manually

Table 4-1 Pros/Cons of Partitioning Schemes

4.4 Data Loading Schemes

Another important distinction between the two partition schemes is, with the GMTI DWv4 method, incoming data is always loaded into a new partition. When data for a new mission arrives, the data loading software generates a new unique Oracle sequence number and assigns it to be the mission ID. The data is then loaded into a new partition using the sequence number as part of its name. Because by definition each mission is unique, data for each mission is loaded into its own partition.

With the SDAF DWv4 scheme, mission data from previous day may overlap data from current day, or data from different sensors may exist for the same day. The data loading software needs to make sure that it loads the data into the correct partition, and does not overwrite any existing data. The exact mechanism is a little more complicated, but involves the following steps for each method:

GMTI DWv4 loading steps:

1. Create temporary table
2. Load incoming data into the temporary table
3. Build all necessary indexes
4. Create new partition
5. Exchange data from temporary table into new partition
6. Destroy temporary table

SDAF DWv4 loading steps:

1. Create one or more temporary tables (if mission data spans more than one day)
2. If data exists in partitions that correspond to the temporary tables
    a. Exchange data from partitions into temporary tables
    b. Load incoming data into appropriate temporary tables
3. If data does NOT exist in partitions that correspond to the temporary tables
    a. Load incoming data into appropriate temporary tables
4. Build all necessary indexes
5. Exchange data from temporary tables into partitions
6. Destroy temporary table(s)

Even though the loading steps are more complicated for the SDAF DWv4, if most of the mission files do not contain many overlapping data, the loading times for both should be relatively similar.

5 Test Statistics

To see how different partitioning schemes would affect the performance of the databases, we used a test suite of five SQL queries containing various spatial functions. The first four queries return all the rows that interact with a given geometry object between two designated times. The last query returns all the rows that are within a specified distance from a location (in latitude/longitude), also between two time values. The test scripts (courtesy of Peter Sylvester) are in Appendix C.

5.1 Automatic Partition Elimination Test

We ran the test suite on each database as it was loaded with increasing amount of data. The assumption is the entire GMTI DWv4 will have to be searched to return the result set, because it is not partitioned by date/time. The time it takes will also grow as the size of the database increases. On the other hand, the SDAF DWv4 will minimize the number of partitions it needs to search by automatically eliminating all partitions that fall outside of the specified time values. In addition, the size of the database should not make a major difference in the response time. The reason is even as the database is increasingly loaded with more data, with automatic partition elimination provided by Oracle the number of partitions to search remains the same. The only extra cost is the time it takes to eliminate additional partitions. The following table shows the results.

Response Time			
Number of rows in DOT table	Test	GMTI	SDAF
51 millions	1	14 secs	6 secs
	2	13 secs	1 sec
	3	12 secs	1 sec
	4	14 secs	1 sec
	5	2 mins 4 secs	2 secs
90 millions	1	26 mins 20 secs	4 secs

	2	14 mins 48 secs	2 secs
	3	4 mins 25 secs	2 secs
	4	10 mins 44 secs	2 secs
	5	22 mins 7 secs	3 secs
116 millions	1	34 mins 14 secs	10 secs
	2	33 mins 55 secs	2 secs
	3	8 mins 48 secs	2 secs
	4	23 mins 39 secs	2 secs
	5	41 mins 40 secs	3 secs

Table 5-1 Automatic Partition Elimination Test

5.2 Manual Partition Elimination Test

One way to speed up the response time for the GMTI DWv4 is to do manual partition elimination. To do that, the five SQL queries had to be converted into PL/SQL procedures. Inside each procedure, the RSR table is queried first to get the mission_id's (i.e., partitions) containing data for the time values specified. A second query executes on only those partitions returned from the first query. We also added four more procedures to return increasing larger amount of data to see the effect of larger result sets have on the response time. For the SDAF DWv4, we added four more SQL queries to return more rows. The test scripts are in Appendix D.

Response Time (174 million rows in DOT table)			
Test	GMTI	SDAF	Rows Returned
1	39 secs	10 secs	8,880
2	23 secs	2 secs	8,880
3	15 secs	2 secs	1,319
4	15 secs	2 secs	8,663
5	26 secs	3 secs	15,170
1	35 secs	33 secs	110,485

2	29 secs	22 secs	110,485
3	20 secs	11 secs	15,838
4	21 secs	11 secs	107,743
5	33 secs	28 secs	190,698
1	2 mins 31 secs	1 mins 11 secs	215,925
2	2 mins 54 secs	36 secs	215,925
3	2 mins 42 secs	26 secs	30,896
4	4 mins 7 secs	24 secs	210,548
5	3 mins 10 secs	59 secs	373,529
1	2 mins 53 secs	1 mins 43 secs	411,291
2	2 mins 54 secs	2 mins 18 secs	411,291
3	3 mins 13 secs	1 mins 34 secs	57,236
4	2 mins 53 secs	2 mins 51 secs	401,041
5	3 mins 3 secs	2 mins 2 secs	709,621
1	2 mins 52 secs	2 mins 40 secs	530,029
2	2 mins 54 secs	2 mins 44 secs	530,029
3	2 mins 43 secs	1 mins 56 secs	72,240
4	2 mins 57 secs	2 mins 31 secs	516,968
5	3 mins 4 secs	3 mins 6 secs	908,943

Table 5-2 Manual Partition Elimination Test

6 Conclusion

The motivation for partitioning the SDAF DWv4 by date/time was to improve existing GMTI storage techniques to meet the demanding needs of time critical applications. It is based on the assumption that automatic partition elimination, an intrinsic capability provided by Oracle, should work well in providing good response time. The outcome of the testing

supported the assumption. The response time for the test suite of five spatial queries remains relatively the same as the database grew from 51 to 116 million rows. The response time for the SDAF DWv4 compared better than or equal to the GMTI DWv4 when the GMTI DWv4 included manual partition elimination and increased amount of data returned by each query.

In addition, the partition scheme of the SDAF DWv4 provides for more flexible and efficient query operations. Even though the loading steps are a little more complicated, the loading time for similar data should be comparable to the other scheme. The only restriction is that partitions have to be created ahead of time before data loading can take place. The SDAF DWv4 setup is also a good example of leveraging from another project to cut down on development time and effort, and at the same time deliver a quality product.

7 Acknowledgement

The SDAF DWv4 work borrowed heavily, both data schemas and software programs, from the Forensic GMTI DWv4 project. We would like to thank Peter Sylvester, William Dowling, and Curtis Brown for all their help.

Appendix A:

GMTI DWv4 Database Tables:

MISSION Table:

Name	Null?	Type
MISSION_ID	NOT NULL	NUMBER(38)
NAME	NOT NULL	VARCHAR2(2000)
FILE_PATH	NOT NULL	VARCHAR2(2000)
START_TIMESTAMP	NOT NULL	TIMESTAMP(6)
END_TIMESTAMP	NOT NULL	TIMESTAMP(6)
DOT_AREA		MDSYS.SDO_GEOMETRY
MESSAGE_COUNT	NOT NULL	NUMBER(38)
DOT_COUNT	NOT NULL	NUMBER(38)
LOAD_DATE	NOT NULL	DATE
LOAD_TIME_MINS	NOT NULL	NUMBER(5,2)
FINGERPRINT		RAW(400)

MESSAGE Table:

Name	Null?	Type
MISSION_ID	NOT NULL	NUMBER(38)
MESSAGE_ID	NOT NULL	NUMBER(38)
FORMAT_ID	NOT NULL	NUMBER(3)
TYPE_ID	NOT NULL	NUMBER(3)
SCAN_NUMBER	NOT NULL	NUMBER(5)
SCAN_FLAGS	NOT NULL	NUMBER(3)
SEQUENCE_NUMBER	NOT NULL	NUMBER(5)
RSR_NUMBER	NOT NULL	NUMBER(5)

PLATFORM_TYPE	NOT NULL	NUMBER(3)
PLATFORM_ID	NOT NULL	VARCHAR2(12)
DATA_TIMESTAMP	NOT NULL	TIMESTAMP(6)
SCAN_AREA	NOT NULL	MDSYS.SDO_GEOMETRY
DOT_AREA	NOT NULL	MDSYS.SDO_GEOMETRY
DOT_COUNT	NOT NULL	NUMBER(38)

DOT Table:

Name	**Null?**	**Type**
MISSION_ID	NOT NULL	NUMBER(38)
RSR_NUMBER	NOT NULL	NUMBER(5)
DOT_ID	NOT NULL	NUMBER(38)
MESSAGE_ID	NOT NULL	NUMBER(38)
RADIAL_VELOCITY	NOT NULL	NUMBER
CROSS_SECTION	NOT NULL	NUMBER
CLASSIFICATION	NOT NULL	NUMBER(3)
TRUTH_APPLICATION	NOT NULL	NUMBER(5)
TRUTH_ENTITY	NOT NULL	NUMBER(5)
RANGE_ERROR	NOT NULL	NUMBER(5)
CROSS_SECTION_ERROR	NOT NULL	NUMBER(5)
DATA_TIMESTAMP	NOT NULL	TIMESTAMP(6)
LOCATION	NOT NULL	MDSYS.SDO_GEOMETRY
SEGMENT_INDEX	NOT NULL	NUMBER(2)

RSR Table:

Name	**Null?**	**Type**
MISSION_ID	NOT NULL	NUMBER(38)

RSR_NUMBER	NOT NULL	NUMBER(5)
START_TIMESTAMP	NOT NULL	TIMESTAMP(6)
END_TIMESTAMP	NOT NULL	TIMESTAMP(6)
DOT_AREA	NOT NULL	MDSYS.SDO_GEOMETRY
SCAN_COUNT	NOT NULL	NUMBER(38)
MESSAGE_COUNT	NOT NULL	NUMBER(38)
DOT_COUNT	NOT NULL	NUMBER(38)
MIN_REVISIT_SECS	NOT NULL	NUMBER
MAX_REVISIT_SECS	NOT NULL	NUMBER
AVG_REVISIT_SECS	NOT NULL	NUMBER
STDDEV_REVISIT_SECS	NOT NULL	NUMBER

RSR_REVISIT Table:

Name	**Null?**	**Type**
MISSION_ID	NOT NULL	NUMBER(38)
RSR_NUMBER	NOT NULL	NUMBER(5)
BIN_SIZE	NOT NULL	NUMBER(3)
BIN_SECS	NOT NULL	NUMBER(6)
AVG_REVISIT_SECS	NOT NULL	NUMBER
SCAN_COUNT	NOT NULL	NUMBER(38)
MESSAGE_COUNT	NOT NULL	NUMBER(38)
DOT_COUNT	NOT NULL	NUMBER(38)

RAW_DATA Table:

Name	**Null?**	**Type**
MISSION_ID	NOT NULL	NUMBER(38)
MESSAGE_ID	NOT NULL	NUMBER(38)

DATA		RAW(2000)

Appendix B:

SDAF DWv4 Database Tables:

MISSION Table:

Name	Null?	Type
MISSION_ID	NOT NULL	NUMBER(38)
NAME	NOT NULL	VARCHAR2(2000)
FILE_PATH	NOT NULL	VARCHAR2(2000)
START_TIMESTAMP	NOT NULL	TIMESTAMP(6)
END_TIMESTAMP	NOT NULL	TIMESTAMP(6)
DOT_AREA		MDSYS.SDO_GEOMETRY
MESSAGE_COUNT	NOT NULL	NUMBER(38)
DOT_COUNT	NOT NULL	NUMBER(38)
LOAD_DATE	NOT NULL	DATE
LOAD_TIME_MINS	NOT NULL	NUMBER(5,2)
FINGERPRINT		RAW(400)

MESSAGE Table

Name	Null?	Type
MISSION_ID	NOT NULL	NUMBER(38)
MESSAGE_ID	NOT NULL	NUMBER(38)
FORMAT_ID	NOT NULL	NUMBER(3)
TYPE_ID	NOT NULL	NUMBER(3)
SCAN_NUMBER	NOT NULL	NUMBER(5)
SCAN_FLAGS	NOT NULL	NUMBER(3)
SEQUENCE_NUMBER	NOT NULL	NUMBER(5)

RSR_NUMBER	NOT NULL	NUMBER(5)
PLATFORM_TYPE	NOT NULL	NUMBER(3)
PLATFORM_ID	NOT NULL	VARCHAR2(12)
DATA_TIMESTAMP	NOT NULL	TIMESTAMP(6)
SCAN_AREA	NOT NULL	MDSYS.SDO_GEOMETRY
DOT_AREA	NOT NULL	MDSYS.SDO_GEOMETRY
DOT_COUNT	NOT NULL	NUMBER(38)

DOT Table

Name	Null?	Type
MISSION_ID	NOT NULL	NUMBER(38)
RSR_NUMBER	NOT NULL	NUMBER(5)
DOT_ID	NOT NULL	NUMBER(38)
MESSAGE_ID	NOT NULL	NUMBER(38)
RADIAL_VELOCITY	NOT NULL	NUMBER
CROSS_SECTION	NOT NULL	NUMBER
CLASSIFICATION	NOT NULL	NUMBER(3)
TRUTH_APPLICATION	NOT NULL	NUMBER(5)
TRUTH_ENTITY	NOT NULL	NUMBER(5)
RANGE_ERROR	NOT NULL	NUMBER(5)
CROSS_SECTION_ERROR	NOT NULL	NUMBER(5)
DATA_TIMESTAMP	NOT NULL	TIMESTAMP(6)
LOCATION	NOT NULL	MDSYS.SDO_GEOMETRY
SEGMENT_INDEX	NOT NULL	NUMBER(2)
LAT		FLOAT(126)
LON		FLOAT(126)

RSR Table:

Name	Null?	Type
MISSION_ID	NOT NULL	NUMBER(38)
RSR_NUMBER	NOT NULL	NUMBER(5)
START_TIMESTAMP	NOT NULL	TIMESTAMP(6)
END_TIMESTAMP	NOT NULL	TIMESTAMP(6)
DOT_AREA	NOT NULL	MDSYS.SDO_GEOMETRY
SCAN_COUNT	NOT NULL	NUMBER(38)
MESSAGE_COUNT	NOT NULL	NUMBER(38)
DOT_COUNT	NOT NULL	NUMBER(38)
MIN_REVISIT_SECS	NOT NULL	NUMBER
MAX_REVISIT_SECS	NOT NULL	NUMBER
AVG_REVISIT_SECS	NOT NULL	NUMBER
STDDEV_REVISIT_SECS	NOT NULL	NUMBER

RSR_REVISIT Table:

Name	Null?	Type
MISSION_ID	NOT NULL	NUMBER(38)
RSR_NUMBER	NOT NULL	NUMBER(5)
BIN_SIZE	NOT NULL	NUMBER(3)
BIN_SECS	NOT NULL	NUMBER(6)
AVG_REVISIT_SECS	NOT NULL	NUMBER
SCAN_COUNT	NOT NULL	NUMBER(38)
MESSAGE_COUNT	NOT NULL	NUMBER(38)
DOT_COUNT	NOT NULL	NUMBER(38)

Appendix C:

SQL Test 1:

```
select count(*) from
(
select d.message_id
from dot d
where
(d.data_timestamp>=to_date('27-FEB-06 07.34.05', 'dd-mon-yy hh24:mi:ss') and
d.data_timestamp<=to_date('27-FEB-06 13.00.27', 'dd-mon-yy hh24:mi:ss')) and
sdo_filter(d.location, SDO_geometry(
   2003,8307,NULL,
   SDO_elem_info_array(1,1003,3),
   SDO_ordinate_array(44.20,32.30, 44.30,32.365)) /* x_min,y_min,
x_max,y_max */
) = 'TRUE'
);
```

SQL Test 2:

```
select count(*) from
(
select d.message_id
from dot d
where
(d.data_timestamp>=to_date('27-FEB-06 07.34.05', 'dd-mon-yy hh24:mi:ss') and
d.data_timestamp<=to_date('27-FEB-06 13.00.27', 'dd-mon-yy hh24:mi:ss')) and
sdo_filter(d.location, SDO_geometry(
   2003,8307,NULL,
```

```
   SDO_elem_info_array(1,1003,1),
   SDO_ordinate_array(44.20,32.30, 44.20,32.365, 44.30,32.365,
44.30,32.30, 44.20,32.30))
) = 'TRUE'
);
```

SQL Test 3:

```
select count(*) from
(
select d.message_id
from dot d
where
(d.data_timestamp>=to_date('27-FEB-06 07.34.05', 'dd-mon-yy hh24:mi:ss') and
d.data_timestamp<=to_date('27-FEB-06 13.00.27', 'dd-mon-yy hh24:mi:ss')) and
sdo_relate(d.location, SDO_geometry(
   2003,8307,NULL,
   SDO_elem_info_array(1,1003,1),
   SDO_ordinate_array(44.20,32.30, 44.30,32.365)), 'mask=anyinteract'
) = 'TRUE'
);
```

SQL Test 4:

```
select count(*) from
(
select d.message_id
from dot d
where
(d.data_timestamp>=to_date('27-FEB-06 07.34.05', 'dd-mon-yy hh24:mi:ss') and
```

```
d.data_timestamp<=to_date('27-FEB-06 13.00.27', 'dd-mon-yy hh24:mi:ss')) and
sdo_relate(d.location, SDO_geometry(
   2003,8307,NULL,
   SDO_elem_info_array(1,1003,1),
   SDO_ordinate_array(44.20,32.30, 44.20,32.365, 44.30,32.365,
44.30,32.30, 44.20,32.30)), 'mask=anyinteract'
) = 'TRUE'
);
```

SQL Test 5:

```
select count(*) from
(
select d.message_id
from dot d
where
(d.data_timestamp>=to_date('27-FEB-06 07.34.05', 'dd-mon-yy hh24:mi:ss') and
d.data_timestamp<=to_date('27-FEB-06 13.00.27', 'dd-mon-yy hh24:mi:ss')) and
SDO_WITHIN_DISTANCE(d.location, SDO_geometry(
   2001,8307,NULL,
   SDO_elem_info_array(1,1,1),
   SDO_ordinate_array(44.20,32.30)), 'distance=10000'
) = 'TRUE'
);
```

Appendix D:

PL/SQL Test 1:

```
DECLARE

mission_id rsr.mission_id%TYPE;
rsr_number rsr.rsr_number%TYPE;
sql_stmt VARCHAR2(500);
loop_cnt NUMBER;
rs NUMBER;

CURSOR rsrCursor IS
  select mission_id
  from rsr
  where
  (start_timestamp>=to_timestamp('27-FEB-06 07.34.05.123456789', 'dd-mon-yy
hh24:mi:ss.ff') and
  start_timestamp<=to_timestamp('27-FEB-06 13.00.27.123456789', 'dd-mon-yy
hh24:mi:ss.ff')) or
  (end_timestamp>=to_timestamp('27-FEB-06 07.34.05.123456789', 'dd-mon-yy
hh24:mi:ss.ff') and
  end_timestamp<=to_timestamp('27-FEB-06 13.00.27.123456789', 'dd-mon-yy
hh24:mi:ss.ff')) order by mission_id;

BEGIN

  sql_stmt := 'select count(*) from dot where mission_id in (';
  loop_cnt := 0;

  OPEN rsrCursor;

  LOOP
    FETCH rsrCursor into mission_id;
    EXIT WHEN rsrCursor%NOTFOUND;

    IF loop_cnt = 0 THEN
      sql_stmt := sql_stmt || mission_id;
    ELSE
      sql_stmt := sql_stmt || ',' || mission_id;
    END IF;
```

```
   loop_cnt := loop_cnt + 1;

  END LOOP;

  CLOSE rsrCursor;

  sql_stmt := sql_stmt || ')' || ' and ' ||
    '(data_timestamp>=to_timestamp("27-FEB-06 07.34.05.123456789", "dd-mon-yy
hh24:mi:ss.ff") and ' ||
    'data_timestamp<=to_timestamp("27-FEB-06 13.00.27.123456789", "dd-mon-yy
hh24:mi:ss.ff")) and ' ||
    'sdo_filter(location, SDO_geometry(' ||
    '2003,8307,NULL,' ||
    'SDO_elem_info_array(1,1003,3),' ||
    'SDO_ordinate_array(44.20,32.30, 44.30,32.365))) = "TRUE"';

  DBMS_OUTPUT.PUT_LINE(sql_stmt);
  EXECUTE IMMEDIATE sql_stmt into rs;
  DBMS_OUTPUT.PUT_LINE(rs);

END;
/
```

PL/SQL Test 2:

```
DECLARE

mission_id rsr.mission_id%TYPE;
rsr_number rsr.rsr_number%TYPE;
sql_stmt VARCHAR2(500);
loop_cnt NUMBER;
rs NUMBER;

CURSOR rsrCursor IS
  select mission_id
  from rsr
  where
  (start_timestamp>=to_timestamp('27-FEB-06 07.34.05.123456789', 'dd-mon-yy
hh24:mi:ss.ff') and
  start_timestamp<=to_timestamp('04-MAR-06 13.00.27.123456789', 'dd-mon-yy
hh24:mi:ss.ff')) or
```

```
  (end_timestamp>=to_timestamp('27-FEB-06 07.34.05.123456789', 'dd-mon-yy
hh24:mi:ss.ff') and
  end_timestamp<=to_timestamp('04-MAR-06 13.00.27.123456789', 'dd-mon-yy
hh24:mi:ss.ff')) order by mission_id;

BEGIN

  sql_stmt := 'select count(*) from dot where mission_id in (';
  loop_cnt := 0;

  OPEN rsrCursor;

  LOOP
    FETCH rsrCursor into mission_id;
    EXIT WHEN rsrCursor%NOTFOUND;

    IF loop_cnt = 0 THEN
      sql_stmt := sql_stmt || mission_id;
    ELSE
      sql_stmt := sql_stmt || ',' || mission_id;
    END IF;

    loop_cnt := loop_cnt + 1;

  END LOOP;

  CLOSE rsrCursor;

  sql_stmt := sql_stmt || ')' || ' and ' ||
    '(data_timestamp>=to_timestamp("27-FEB-06 07.34.05.123456789", "dd-mon-yy
hh24:mi:ss.ff") and ' ||
    'data_timestamp<=to_timestamp("04-MAR-06 13.00.27.123456789", "dd-mon-yy
hh24:mi:ss.ff")) and ' ||
    'sdo_filter(location, SDO_geometry(' ||
    '2003,8307,NULL,' ||
    'SDO_elem_info_array(1,1003,3),' ||
    'SDO_ordinate_array(44.20,32.30, 44.30,32.365))) = "TRUE"';

  DBMS_OUTPUT.PUT_LINE(sql_stmt);
  EXECUTE IMMEDIATE sql_stmt into rs;
  DBMS_OUTPUT.PUT_LINE(rs);
```

```
END;
/
```

PL/SQL Test 3:

```
DECLARE

mission_id rsr.mission_id%TYPE;
rsr_number rsr.rsr_number%TYPE;
sql_stmt VARCHAR2(500);
loop_cnt NUMBER;
rs NUMBER;

CURSOR rsrCursor IS
  select mission_id
  from rsr
  where
  (start_timestamp>=to_timestamp('27-FEB-06 07.34.05.123456789', 'dd-mon-yy
hh24:mi:ss.ff') and
  start_timestamp<=to_timestamp('09-MAR-06 13.00.27.123456789', 'dd-mon-yy
hh24:mi:ss.ff')) or
  (end_timestamp>=to_timestamp('27-FEB-06 07.34.05.123456789', 'dd-mon-yy
hh24:mi:ss.ff') and
  end_timestamp<=to_timestamp('09-MAR-06 13.00.27.123456789', 'dd-mon-yy
hh24:mi:ss.ff')) order by mission_id;

BEGIN

  sql_stmt := 'select count(*) from dot where mission_id in (';
  loop_cnt := 0;

  OPEN rsrCursor;

  LOOP
    FETCH rsrCursor into mission_id;
    EXIT WHEN rsrCursor%NOTFOUND;

    IF loop_cnt = 0 THEN
      sql_stmt := sql_stmt || mission_id;
    ELSE
      sql_stmt := sql_stmt || ',' || mission_id;
    END IF;
```

```
   loop_cnt := loop_cnt + 1;

  END LOOP;

  CLOSE rsrCursor;

  sql_stmt := sql_stmt || ')' || ' and ' ||
    '(data_timestamp>=to_timestamp("27-FEB-06 07.34.05.123456789", "dd-mon-yy
hh24:mi:ss.ff") and ' ||
    'data_timestamp<=to_timestamp("09-MAR-06 13.00.27.123456789", "dd-mon-yy
hh24:mi:ss.ff")) and ' ||
    'sdo_filter(location, SDO_geometry(' ||
    '2003,8307,NULL,' ||
    'SDO_elem_info_array(1,1003,3),' ||
    'SDO_ordinate_array(44.20,32.30, 44.30,32.365))) = "TRUE"';

  DBMS_OUTPUT.PUT_LINE(sql_stmt);
  EXECUTE IMMEDIATE sql_stmt into rs;
  DBMS_OUTPUT.PUT_LINE(rs);

END;
/
```

PL/SQL Test 4:

```
DECLARE

mission_id rsr.mission_id%TYPE;
rsr_number rsr.rsr_number%TYPE;
sql_stmt VARCHAR2(500);
loop_cnt NUMBER;
rs NUMBER;

CURSOR rsrCursor IS
  select mission_id
  from rsr
  where
  (start_timestamp>=to_timestamp('27-FEB-06 07.34.05.123456789', 'dd-mon-yy
hh24:mi:ss.ff') and
  start_timestamp<=to_timestamp('14-MAR-06 13.00.27.123456789', 'dd-mon-yy
hh24:mi:ss.ff')) or
```

```
  (end_timestamp>=to_timestamp('27-FEB-06 07.34.05.123456789', 'dd-mon-yy
hh24:mi:ss.ff') and
  end_timestamp<=to_timestamp('14-MAR-06 13.00.27.123456789', 'dd-mon-yy
hh24:mi:ss.ff')) order by mission_id;

BEGIN

  sql_stmt := 'select count(*) from dot where mission_id in (';
  loop_cnt := 0;

  OPEN rsrCursor;

  LOOP
    FETCH rsrCursor into mission_id;
    EXIT WHEN rsrCursor%NOTFOUND;

    IF loop_cnt = 0 THEN
      sql_stmt := sql_stmt || mission_id;
    ELSE
      sql_stmt := sql_stmt || ',' || mission_id;
    END IF;

    loop_cnt := loop_cnt + 1;

  END LOOP;

  CLOSE rsrCursor;

  sql_stmt := sql_stmt || ')' || ' and ' ||
    '(data_timestamp>=to_timestamp("27-FEB-06 07.34.05.123456789", "dd-mon-yy
hh24:mi:ss.ff") and ' ||
    'data_timestamp<=to_timestamp("14-MAR-06 13.00.27.123456789", "dd-mon-yy
hh24:mi:ss.ff")) and ' ||
    'sdo_filter(location, SDO_geometry(' ||
    '2003,8307,NULL,' ||
    'SDO_elem_info_array(1,1003,3),' ||
    'SDO_ordinate_array(44.20,32.30, 44.30,32.365))) = "TRUE"';

  DBMS_OUTPUT.PUT_LINE(sql_stmt);
  EXECUTE IMMEDIATE sql_stmt into rs;
  DBMS_OUTPUT.PUT_LINE(rs);
```

```
END;
/
```

PL/SQL Test 5:

```
DECLARE

mission_id rsr.mission_id%TYPE;
rsr_number rsr.rsr_number%TYPE;
sql_stmt VARCHAR2(500);
loop_cnt NUMBER;
rs NUMBER;

CURSOR rsrCursor IS
  select mission_id
  from rsr
  where
  (start_timestamp>=to_timestamp('27-FEB-06 07.34.05.123456789', 'dd-mon-yy
hh24:mi:ss.ff') and
  start_timestamp<=to_timestamp('19-MAR-06 13.00.27.123456789', 'dd-mon-yy
hh24:mi:ss.ff')) or
  (end_timestamp>=to_timestamp('27-FEB-06 07.34.05.123456789', 'dd-mon-yy
hh24:mi:ss.ff') and
  end_timestamp<=to_timestamp('19-MAR-06 13.00.27.123456789', 'dd-mon-yy
hh24:mi:ss.ff')) order by mission_id;

BEGIN

  sql_stmt := 'select count(*) from dot where mission_id in (';
  loop_cnt := 0;

  OPEN rsrCursor;

  LOOP
    FETCH rsrCursor into mission_id;
    EXIT WHEN rsrCursor%NOTFOUND;

    IF loop_cnt = 0 THEN
      sql_stmt := sql_stmt || mission_id;
    ELSE
      sql_stmt := sql_stmt || ',' || mission_id;
    END IF;
```

```
    loop_cnt := loop_cnt + 1;

  END LOOP;

  CLOSE rsrCursor;

  sql_stmt := sql_stmt || ')' || ' and ' ||
    '(data_timestamp>=to_timestamp("27-FEB-06 07.34.05.123456789", "dd-mon-yy
hh24:mi:ss.ff") and ' ||
    'data_timestamp<=to_timestamp("19-MAR-06 13.00.27.123456789", "dd-mon-yy
hh24:mi:ss.ff")) and ' ||
    'sdo_filter(location, SDO_geometry(' ||
    '2003,8307,NULL,' ||
    'SDO_elem_info_array(1,1003,3),' ||
    'SDO_ordinate_array(44.20,32.30, 44.30,32.365))) = "TRUE"';

  DBMS_OUTPUT.PUT_LINE(sql_stmt);
  EXECUTE IMMEDIATE sql_stmt into rs;
  DBMS_OUTPUT.PUT_LINE(rs);

END;
/
```