BUILDING AN OPERATIONAL DATA STORE FOR A DIRECT MARKETING APPLICATION SYSTEM

A Thesis
Presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science

by
Chad Smith
March 2009
COMMITTEE MEMBERSHIP

TITLE: BUILDING AN OPERATIONAL DATA STORE FOR A DIRECT MARKETING APPLICATION SYSTEM

AUTHOR: Chad Smith

DATE SUBMITTED: March 2009

COMMITTEE CHAIR: Alexander Dekhtyar, Ph.D.

COMMITTEE MEMBER: Hasmik Gharibyan, Ph.D.

COMMITTEE MEMBER: Clark Turner, J.D., Ph.D.
ABSTRACT

BUILDING AN OPERATIONAL DATA STORE FOR A DIRECT MARKETING APPLICATION SYSTEM

by

Chad Smith

An operational data store (ODS) can be generally described as an architectural construct that is both similar and different in design and purpose to a data warehouse. It is similar in that it is subject-oriented and integrated from the various systems and sources that feed it. However, it is unique in that the data is volatile and updated on a relatively regular basis, it holds little to no archival data, and the data is kept at a detailed level with little summarization.

The purpose of this thesis is to provide an overview of an ODS and the data that it holds, as well as to describe the author’s participation in a corporate project to build an ODS used to process and score customer information for a direct marketing application. This application gathers customer information from point-of-sale transactions and online orders.

The author’s role on the project was to develop, test, and implement solutions concerning database performance, as well to provide database analysis and support throughout the project lifecycle. This thesis will provide an overview of the system’s functionality and the individual contributions the author made to the overall solution.
ACKNOWLEDGEMENTS

I would like to thank the following people, who have been a tremendous help to me during my time at Cal Poly and throughout the process of writing this thesis:

- Dr. Laurian Chirica. Thank you for first stoking my interest in databases and helping me develop this thesis topic.
- Dr. Alexander Dekhtyar. Thank you for graciously guiding me through the writing process and providing invaluable feedback throughout.
- Dr. Clark Turner and Dr. Hasmik Gharibyan. Thank you for contributing your time by sitting on my committee and helping me through this process.
- Diane Nott. You have shown unbelievable patience with my never-ending questions and requests. Thank you so much for all of your help!
- James Feld, Ed Bruns, Aaron Abitia, and Jason Holt. You have all contributed to my professional growth at Cal Poly in more ways than you know. Thank you!

I would also like to thank my business colleagues who shall remain anonymous, but have been no less instrumental in helping me with this thesis and my continuing education in the “real world”.

To my family and friends: Thank you for your undying love and support throughout this entire process. Thank you also for your gentle reminders that I had not yet graduated until my thesis was done.
Finally, to my wife, Kara: You have had faith in me every step of the way. Thank you for bestowing God’s love and grace upon me when I needed it most. You have seen me through all of this, and it belongs to you as well. I love you.
# TABLE OF CONTENTS

**List of Tables**

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>xi</td>
</tr>
</tbody>
</table>

**List of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>xii</td>
</tr>
</tbody>
</table>

1 Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>4</td>
</tr>
<tr>
<td>1.3.1</td>
<td>5</td>
</tr>
<tr>
<td>1.3.2</td>
<td>5</td>
</tr>
<tr>
<td>1.3.3</td>
<td>7</td>
</tr>
<tr>
<td>1.4</td>
<td>8</td>
</tr>
</tbody>
</table>

2 Background

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>11</td>
</tr>
<tr>
<td>2.2.1</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2</td>
<td>13</td>
</tr>
<tr>
<td>2.2.2.1</td>
<td>13</td>
</tr>
<tr>
<td>2.2.2.2</td>
<td>14</td>
</tr>
<tr>
<td>2.2.2.3</td>
<td>15</td>
</tr>
<tr>
<td>2.2.3</td>
<td>15</td>
</tr>
<tr>
<td>2.2.4</td>
<td>16</td>
</tr>
</tbody>
</table>
3 Solution

3.1 Historical Context

3.2 Architecture

  3.2.1 Inbound External Vendor File Transfer to B2B Server
  3.2.2 ODS File Pull from B2B Server
  3.2.3 Transfer of Internal Files
  3.2.4 Export of Data from Enterprise Data Warehouse (EDW)
  3.2.5 De-Duplication and Load into Staging Tables
  3.2.6 Population of the ODS Core Tables
  3.2.7 Population of the Outbound Interface Tables
  3.2.8 ADM Data Pull from Outbound Interface Tables
  3.2.9 ADM Load
  3.2.10 ADM Schema Flip
  3.2.11 RAR Data Pull from Outbound Interface Tables
  3.2.12 RAR Scoring Update to the ODS
  3.2.13 Analyst Access
  3.2.14 Developer Access
  3.2.15 File Extract for EDW Transaction Filter
  3.2.16 File Extract for COA Processing
  3.2.17 Application Campaign Extract
  3.2.18 B2B File Transfer to Vendor
  3.2.19 EDW Transaction Filter File Transfer
  3.2.20 Customer De-Duplication
3.2.21  Merged Customer Data Load  
3.2.22  Application Server Data Access  
3.2.23  Web Server Communication with Application Server  
3.2.24  Business User Access to Web Pages  
3.2.25  Remote Admin Server  
3.2.26  Campaign Definition/Control Group Load into ODS  
3.2.27  RAR Ad-Hoc Score Updates to the ADM  

3.3  Data  
3.3.1  Customer Subject Area  
3.3.2  Other Subject Areas  

3.4  ODS Schemas  
3.4.1  Staging Schema  
3.4.2  Outbound Schema  
3.4.3  Core Schema  
3.4.4  Audit Schema  

3.5  Performance Enhancement Requirements  

4  Performance Enhancements  
4.1  Outbound Data Loads  
4.2  Table Partition Management  
4.3  Ad-Hoc Table Analysis  

5  Implementation  
5.1  Outbound Data Loads  
5.1.1  Placing Outbound Schema Tables in Read-Only Mode
LIST OF TABLES

2.1 Deciding how the data should be stored 11

3.1 Example ODS fact table 33

4.1 Steps in partition management process 43

4.2 Control table for ad-hoc table analysis 46

5.1 Sample test results for ODS outbound data loads 49

5.2 Sample test results for “rolling partition” process 52

5.3 Optimum values used in ad-hoc analysis control table 54
LIST OF FIGURES

2.1 Basic ODS architecture with multiple source data feeds 10
2.2 Example of a star schema 12
2.3 Inmon’s classification of ODS types, based on update frequency 13
2.4 Sample ETL architecture 17
3.1 Marketing application system architecture 20
3.2 Project ODS designed around the Customer subject area 30
4.1 Transportable Tablespace process 39
4.2 “Rolling Partitions” process 42
CHAPTER 1

Introduction

1.1 Overview

An operational data store, or ODS (see Appendix B for a list of acronyms used in this thesis), can be generally described as an architectural construct that is both similar and different in design and purpose to a data warehouse. When comparing it to a data warehouse, the ODS is similar in that it is subject-oriented, meaning that the ODS is designed based on the subject areas involved in the particular company using it. Some common subject areas may include Customers, Products, Orders, and Shipments. It is also similar to a data warehouse in that the data stored in the ODS is integrated from the various systems and sources that feed it.

However, because the data in an ODS is operational in nature, it follows that (1) the data is volatile and updated on a relatively regular basis, (2) there is little to no archival data found in the ODS, and (3) the data held within the ODS is kept at a detailed level with little summarization [5].

Recently, the author was involved in a corporate project at a certain company (hereafter, “the company”) to build an ODS for a direct marketing application system. This application gathers customer information from point-of-sale transactions and online orders. The application itself, before the advent of the proposed ODS, attempted to sort
through this customer purchase information and then market directly to the customer based on the specific products purchased and the frequency of the customer’s purchase activity. This was the problem to be solved by the design and implementation of an application-specific ODS; namely, to create a facility to analyze customer purchasing habits, score them appropriately, and communicate offers to each customer based on their purchases.

The purpose of this thesis is to describe the facility for customer information processing that was built to improve the efficiency of the company’s direct marketing campaigns, as well as to document the author’s contributions to the new system. The design, development and testing phases of this system took over a year to complete and required the dedicated efforts of a large project team including 15-20 development and testing engineers, three UNIX system engineers, and three Oracle database engineers (including the author).

Because of the size and complexity of the system, as well as the numerous contributions from many individuals, we cannot detail the entirety of the newly-developed ODS for the purposes of this thesis. Instead, the author will provide an overview of the system’s functionality and the individual contributions he made to the overall solution. These areas of contribution are centered on performance enhancement to the overall system and include:

- analysis and development of a process for heavy-volume data movement between the ODS and the application databases it feeds;
- development of stored procedures to manage the volume of data kept within large, partitioned tables;
designing a process to allow system users to perform ad-hoc table analysis as required for performance.

The ODS was built as a relational database using the Oracle 9i DBMS running on a UNIX operating system. Other components interacting with the ODS, which include (but not limited to) metadata and reporting databases, applications servers, and servers dedicated to ETL processing, also run on UNIX.

The next section provides a brief overview of the author’s individual contributions to the project ODS.

1.2 System Performance Enhancements

This thesis describes the enhancements the author was asked to design and implement to address three different performance issues. Of the individual solutions provided over the course of the project, the author chose to describe the following three enhancements because he believes they bring the greatest performance gains to the system. The performance issues are as follows:

1. **Outbound Data Loads** – one of the system’s data processing requirements involves transferring large quantities of data, up to half a terabyte, from the ODS to a proprietary application data mart and a reporting and analysis database repository. An automated process was required to move this data across systems on a daily basis and execute completely during non-business hours.

2. **Table Partition Management** – specific tables in the ODS were partitioned and designed to hold large volumes of data, up to 1.5 billion rows. In order to manage the size and storage requirements of these tables over a period of time, an
efficient process was required to keep only the needed data partitions, specified by date, and purge any others that fell outside of this date range. This process required monthly execution and only during non-business hours.

3. **Ad-Hoc Table Analysis** – due to the large data volumes of many of the ODS tables, instances can occur where the standard schema analysis job will not gather current statistics on some of these tables. Current statistics are required by Oracle’s cost-based optimizer in order to determine the best data access paths for a given query, thus improving query performance. To solve the issue, a new process was required to gather statistics at the table level instead of at the schema level. Furthermore, the process needed to be built so that selected ODS users could gather table statistics as required, at any time of day, without having to involve the DBA each time.

Solutions to these performance issues are discussed beginning in Chapter 4.

### 1.3 Impact of Individual Contributions

The individual solutions the author was asked to design and implement for the company ODS had the following impact on the system:

- critical downstream data processing between the ODS and other databases in the overall system was enabled, automated, and executed during non-business hours in preparation for the following day’s database processing;
- structural management of high-volume ODS partitioned tables was enabled, automated, and executed during non-business hours to maintain consistent and manageable table definitions through the data archival and purge processes;
• statistics management capabilities for high-volume database tables were enabled for ad-hoc execution at the user-level at any time of day.

The impact of these contributions to ODS performance is discussed briefly below.

1.3.1 Outbound Data Loads

The original architectural design for the marketing application system called on using the Oracle utility SQL*Loader to move data from the ODS outbound tables into the application data mart for processing and execution, as well as into the reporting and analysis database repository. Conceptually, it was a viable solution and technically possible. However, as data volumes increase, this becomes an increasingly time-consuming process. Furthermore, when considering that the data volume in the ODS outbound schema can reach up to half a terabyte, this process breaks down and becomes impractical.

To solve the problem, the author was asked to design a solution that takes advantage of Oracle’s Transportable Tablespace technology. Once the entire end-to-end process was thoroughly tested and prepared for production, it consistently completed successfully for both of the destination databases in one hour or less.

1.3.2 Table Partition Management

Project requirements dictated that the data inside the ODS persist for at least 24 months. Once data becomes older than 24 months, it is then subject to being archived and purged from the database. However, not all customer records older than 24 months are
necessarily purged. As long as a customer is active (i.e. has made a purchase in the last 24 months), the record will stay in the ODS.

In most cases, a special process handles this data offload without having to make any structural changes to the core schema. However, in the case of partitioned tables, which are tables that are logically, and possibly physically, split into several smaller table structures, there needed to be a process that would keep the table structure with as few partitions as needed in order to enhance ETL and query performance. Since the partitioned tables in question are partitioned by date (one partition for each month of the year), a new partition would need to be created each month to handle all new records. However, the single-step process of adding table partitions each month does not address the cleanup and removal of the earliest month partition in the table (which would become the 25th table partition each time a new month partition is added to the end of the partition list). Additionally, since data in the 25th table partition may not necessarily be archived and purged, it might need to remain in the ODS. This eliminates the possibility of simply dropping the earliest table partition outright after the new month partition is added.

The solution to this problem called for a design that would retain old but “active” ODS records after the archive and purge process executed every month while removing obsolete partition structures from the table. This solution is based on Kimball’s concept of the rolling month, whereby “…when the last day of the month is reached, the current rolling month becomes the most recent member of the standard months in the time series and a new current rolling month is created [8].” Once the entire end-to-end process was thoroughly designed, tested, and prepared for production, it consistently completed successfully in between five to six hours. On average, the process, which consists of
partition management for four of the largest ODS tables, took one minute of clock time for every seven million records of data held within the tables collectively.

1.3.3 Ad-Hoc Table Analysis

For the ODS, since the freshness of data is determined by each batch cycle, we decided that schema-level database object analysis should be executed once each batch cycle was complete. (A schema is a collection of logically-related database objects. ODS schemas are discussed further in Section 3.4.) Because gathering object statistics is a resource-intensive operation, it dictated the following three actions: (1) analysis was to occur after business hours when the system was not as active; (2) only a database administrator could have access to run the analysis, thus preventing application analysts from triggering the job at inopportune times; and (3) statistics would only be gathered on those tables and associated indexes whose data volume had changed by at least 10% from the last analysis. This last point was decided so that tables containing data volumes that were relatively stable would be skipped.

However, for some of the larger tables in the ODS, this was not an acceptable solution. For example, more than one table in the ODS was estimated to hold over one billion records. This would mean that for each table at this size, data volumes could increase to any level up to, but just below, 100 million records and the table would still be skipped under the statistics analysis guidelines. This change in data volume, without gathering fresh statistics, could adversely affect query performance, as the cost-based optimizer would be basing its data retrieval path on outdated statistics. We decided that there needed to be a utility that could provide statistics analysis on these large tables.
independent of the 10% data volume change rule. We also decided that since there were only a few tables that would fall into this classification, the application analysts should have access to update statistics on only these tables as needed.

To solve this problem, the author developed a process to gather statistics for these high-volume tables at the table level instead of at the schema level. He designed the process so that selected ODS users could gather statistics on these tables as required without having to involve the DBA each time. Execution times varied based on the size of the table, the data sample size used to estimate statistics for the entire table, and the degree of parallelism used for each table.

1.4 Thesis Layout

Chapter 2 of this thesis provides background information on ODS implementations in general, including technical background found in the literature of the subject area. Chapter 3 covers the ODS solution developed during the project, including architectural information, database design, and data processing requirements. Chapter 4 discusses performance enhancements developed by the author to meet the ODS processing requirements. Chapter 5 discusses the implementation of the performance enhancements and the resulting effects on the ODS. Chapter 6 discusses conclusions drawn from the ODS implementation and the author’s individual contributions. Three appendices follow the text of the thesis and include (1) a description of the ODS project implementation phases, (2) a list of acronyms used in this thesis, and (3) step-by-step descriptions of the processes developed by the author to improve ODS performance.
CHAPTER 2

Background

The purpose of this chapter is to provide some background regarding major design characteristics of an operational data store (ODS) and to detail the purposes an ODS is designed to serve. Included in this background is a brief discussion of characteristics that are common across all ODS implementations, including the one the author helped build for the company.

2.1 Purpose of an ODS

According to Inmon, an ODS is a subject-oriented, integrated, volatile, current-valued, detailed-only collection of data in support of an organization’s need for up-to-the-second, operational, integrated, collective information [5]. In other words, an ODS is a collection of detailed data, derived from various external sources, that is operational in nature (as opposed to the summarized data typically found in a data warehouse) and used as a decision-support system for a company based on that company’s specific subject areas.

This type of database can be very useful when a company needs to run reports and make analytic decisions based on operational data. In some cases, the ODS can be merely a mirrored copy of an operational database (usually an online transaction processing, or OLTP, database). In this example, the ODS can be used strictly for reporting and data
Figure 2.1 Basic ODS architecture with multiple source data feeds

analysis, thus avoiding the need to exhaust the resources of the hardware hosting the OLTP database or of the database itself. No data integration or summarization during the load phase occurs in this case. However, once the source data reaches the ODS, any data manipulation required for decision support can then be executed without affecting the integrity of the core operational data. Paller is describing such a system when he says, “...in time, most organizations will migrate part or all of their data warehousing activities to computers dedicated to the data warehousing/data mining task. When operational data is mirrored on a separate computer for decision support, it is called an operational data store [1].” Paller [1] also captures the hardware requirements for this type of ODS in Table 2.1.

There are many cases, however, where an ODS is more complex than a simple mirror of an operational data source. In such a case, the ODS may be architecturally similar to a
data warehouse in that the database consists of data drawn from various, heterogeneous data sources. This type of ODS is not merely a mirror of another database, but closer to

<table>
<thead>
<tr>
<th>How busy are the operational computers?</th>
<th>Why the information is needed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>For real-time transaction-based decisions</td>
<td>Enterprise data warehouse and/or data mart</td>
</tr>
<tr>
<td>For pattern-based decisions</td>
<td></td>
</tr>
<tr>
<td>Virtual data warehouse</td>
<td></td>
</tr>
<tr>
<td>Operational data store (ODS)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1 Deciding how the data should be stored

Inmon’s definition of an ODS as an integrated data source consisting of collective information for decision support. Research on the viability of this type of ODS, for the purposes of handling and analyzing customer-centric data as part of the downstream flow to the data warehouse, is ongoing [7]. The project ODS discussed in this thesis consists of this type of architecture, which the author will elaborate on in the next chapter.

2.2 Common ODS Characteristics

2.2.1 Star Schema

Usually, updates to the data in an ODS occur infrequently and take place during batch cycles at a particular time of day when the ODS sees little user activity. Depending on the architecture and the nature of the data source feeds, it is technically possible to propagate
source data changes to the ODS as they occur. However, this can be very costly in terms of time and computing resources.

In contrast, the majority of user activities occurring in the ODS are read operations as queries. This requires that there is a high level of read performance in the ODS, which can be achieved using a star schema to join relational data.

![Figure 2.2 Example of a star schema](image)

The star schema is composed of “fact” and “dimension” tables. The fact table (see the Orders table in Figure 2.2) is the central table of the star design, and there can be more than one fact table in this design. The data contained inside the fact tables is considered to be the “measured” data of the business, such as sales transactions, which is loaded in from the external operational data sources [1]. The vast majority of data contained within an ODS is stored in the fact tables.

Surrounding each fact table are the dimension tables (see the non-Orders tables in Figure 2.2), which are pre-joined to the fact table by way of foreign key relationships. Typically, each dimension table has a fixed number of records (e.g. store locations, lists
of products, geographic locations). The data held within dimension tables is arranged according to the specific subject areas of the business. In this design, query performance is achieved through de-normalization. By de-normalizing the fact table as much as possible to combine all attributes of similar objects into a single table, the number of table joins required for a specific query can be significantly decreased, thus improving performance by facilitating a rapid return of the data result set.

### 2.2.2 Data Update Strategies

Inmon [5] describes three classes of the ODS, all of which are based on the frequency of data updates.

![Inmon's classification of ODS types, based on update frequency](image)

**Figure 2.3** Inmon’s classification of ODS types, based on update frequency

#### 2.2.2.1 Class I: Online Synchronous ODS

As mentioned earlier, it is technically possible to propagate changes made to the external source data into the ODS as these changes occur. This is considered one class of
ODS, whereby updates made to the source operational data are reflected in the ODS in a matter of seconds. Because the updates to the ODS occur so quickly and frequently, there is very little transformation of the data. The frequency of updates occurring in this ODS class requires a high level of computing resources, which can be very costly. An example of this type might be an ODS that is a mirror of an operational OLTP database. For instance, a Web site could use this type of ODS to query the number of hits its site has received at any given time during the day. Users would query this data stored in the ODS instead of directly querying the base system. Thus, the use of base system resources is avoided and its performance and response times are not impact by user queries and reports.

2.2.2.2 Class II: “Store-and-Forward” ODS

A second ODS class handles updates using the “store-and-forward” technique. In this ODS class, operational data updates are gathered in a separate location, independent of both the primary data sources and the ODS, and propagated to the ODS at one time at regular intervals. Since updates are not immediately reflected in the ODS, there is time for data integration and transformation before the ODS load, if this is required. The Class II ODS is less costly then the Class I version, but the trade-off is that the data does not have the currency capable in the Class I ODS. An example of this type might be an ODS used by an airline carrier which holds flight and baggage tracking data and is updated every two to three hours. An airline could use this ODS for reporting on-time trends and successful baggage transportation rates at various times during the day.
2.2.2.3 Class III: Batch Load ODS

The third ODS class receives its data updates asynchronously using batch processes. Here, updates to source data are often subjected to heavy integration and transformation. However, there is time for this extra processing. This is because the Class III ODS is loaded less frequently than the Class I or Class II versions, usually not more than once daily. This class of ODS would typically be employed when the currency requirements of the ODS data are not as critical as they would be for a Class I ODS. An example of this type might be an ODS used to run daily reports on store or warehouse inventory levels for a given business. The project ODS system, which will be discussed in the following chapters, could also be considered a Class III ODS.

2.2.3 Metadata Repository

The purpose of the metadata repository, and the reason it is crucial to the ODS (and data warehouse) environment, is that it describes the data held within the ODS. Because of the heavy integration and collection of data from various heterogeneous sources, the metadata repository is essential to help users understand what data is in the ODS and what it represents. It can be used to describe primary data sources and their contents, the objects contained within the ODS itself, and the transformations required to move the data from the primary sources into the ODS. It can also contain operational information such as a history of the migrated data, what organizations were responsible for the source data creation, what happened during the migration, what data has been purged or is scheduled to be purged, and who is using the data and how they are using it [2].
The metadata repository is built as a database separate from the ODS and is accessible to end-users. These users can use the repository to better understand the ODS data and to help them build queries and interpret the results. During the project, the metadata repository was built with proprietary objects defined by the marketing application vendor. However, the author will be able to describe the repository’s functionality with regards to the ODS and its purpose in the overall system.

2.2.4 Extract/Transform/Load

The extract/transform/load process (ETL) is used to move data into the ODS from external data sources and out of the ODS into the downstream flow, commonly the data warehouse and/or data mart. This is a necessary component of the ODS environment because data coming in from various heterogeneous sources will almost always come in different formats. Some examples include server flat files, XML data, or relational data with differing Data Definition Language (DDL) specifications. Thus, the purpose of ETL is to (1) extract data from the external data sources, (2) transform the source data into a format compliant with the ODS design, and (3) load the transformed source data into the ODS.
This three-step process is usually combined into a single exercise and handled by a utility independent of the ODS itself. The ODS built during the project utilizes a third-party ETL tool to handle data loads into the ODS, and to move data needed for historical purposes out of the ODS and into the enterprise data warehouse. This will be discussed in the following chapter.
CHAPTER 3

Solution

The purpose of this chapter is to describe the ODS solution developed at the company, which will include descriptions of the following:

- a brief background on why the solution was necessary
- an overview of the application system
- the type of data maintained in the ODS
- the multiple-schema ODS database design
- data processing requirements for performance enhancements

3.1 Historical Context

The motivation for this thesis is based on the time the author spent working with a technical project team on a corporate ODS implementation. His contributions to the project and lessons learned were possible due to the technical direction the company decided to take with its marketing application. Before the project began, the application was being hosted offsite by an external vendor. In viewing its overall strategic goals, in parallel with upgrading to the latest vendor release, the company decided to bring its marketing application in-house and expand on its direct marketing capabilities by building an ODS specifically for the application.
The major objective of the application is to increase revenues by driving both online and retail sales through effective, targeted marketing campaigns. The application is used to generate postal and email marketing campaigns. The project objective was to put customer information in a stable place, in a format that is flexible, scalable, and can support integrated marketing activities such as campaign management and customer analysis. The ODS was built to be used as a master repository of cleansed and de-duplicated customer data. In addition, it would be used to prepare all customer-related event data for efficient loading into the application data mart. Customer value and behavioral scores would be calculated in the ODS and loaded into the application data mart for use in campaign targeting.

3.2 Architecture

Figure 3.1 provides an architectural overview of the marketing application system, including the new ODS required for the storage and calculation of customer purchase data and campaign communications. In order to protect company confidentiality, any references to the company name or trademarks have been removed from the diagram. Also, any references to a third-party brand name or vendor have also been removed. This does not compromise the system functionality detailed in this diagram.
Figure 3.1 Marketing application system architecture
The following sections detail the data flow within the system and provide a point of reference for ODS functionality. The source of this information is a detailed design document created for this project by several members of the technical project team, including members of the database engineering team. Although there are several other areas of the system covered in the following steps, we will focus on the ODS following this section. These other areas are listed here to show that the ODS is not a system in itself, but instead a critical component of a larger computing system.

A brief description for each of the critical ODS data flow components is provided in the following sections.

**3.2.1 Inbound External Vendor File Transfer to B2B Server**

The following ODS data sources are received via FTP from outside vendors:

a. customer address changes (COA)

b. e-marketing names push, customer preferences, email campaign results, catalogs, and email click responses

c. postal mail campaign results

d. credit card (CC) files

All of these transmitted files are transferred via FTP by the respective vendor. Files that contain customer names or address information (including email) arrive encrypted. Before any further processing, all encrypted files are decrypted.
3.2.2 ODS File Pull from B2B Server

Once the B2B server decrypts files sourced from outside vendors, the files are transferred via FTP from the B2B server to the ODS server.

3.2.3 Transfer of Internal Files

Data files are pushed to the ODS server via FTP from internal servers. These files include employee information to distinguish them from other regular customers. Files from internal source systems are transferred without encryption.

3.2.4 Export of Data from Enterprise Data Warehouse (EDW)

Point-of-sale information (POS), store & product information, Web demand & fulfillment, and Web traffic files are pulled onto the ODS server using EDW utilities which run on the ODS server.

3.2.5 De-Duplication and Load into Staging Tables

a. De-Duplicating Customer Information – any files that contain customer names and addresses or customer email information are passed to the de-duplication utility (DDU) to perform name and address standardization (i.e. Jim Smith, Jimmy Smith, and James Smith would become one person and one record, if the customer’s address is the same for all names). Once this is complete, the original source files are archived and replaced by the DDU output files for loading into the ODS.
b. ODS Staging Table Load – input data files are loaded into the ODS staging tables using the Oracle utility SQL*Loader.

3.2.6 Population of the ODS Core Tables

ETL is used to extract information from the staging tables and perform the necessary transformations to populate the core ODS tables.

3.2.7 Population of the Outbound Interface Tables

Once data in the ODS has been processed and scored for campaign targeting, ETL is used to populate the outbound interface tables with the information necessary to load the application data mart (ADM) and the reporting and analysis database repository (RAR). The outbound interface tables are purged daily of any information that both the ADM and RAR have already extracted.

3.2.8 ADM Data Pull from Outbound Interface Tables

All new information in the outbound interface tables is pulled into the ADM interface tables in the staging schema. Initially, this was to be done using SQL*Loader over the network. However, because of heavy data volumes in this schema, we had to come up with a more timely and efficient way to move the data to the ADM. For this step, the author designed a process to take advantage of Oracle’s Transportable Tablespace capabilities, which will be described later.
3.2.9 ADM Load

Once the ADM interface tables in the staging schema are fully populated, an internal application process is initiated to load the ADM application tables. Successful completion of this process is a prerequisite for “flipping” the load and view schemas.

3.2.10 ADM Schema Flip

A set of database views exists within the ADM through which the application performs all database processing. These views point to the active view schema, ensuring that data loads into the load schema do not affect application processing. After the load schema is fully populated, a script is executed to “flip” the load and view schemas. After this “flip” is complete, the old view schema becomes the new load schema and is available for loading.

3.2.11 RAR Data Pull from Outbound Interface Tables

Initially, this data pull was going to use scripts running on the RAR to bring data in over the network. However, because of heavy data volumes in this schema, we had to come up with a more timely and efficient way to move the data to the RAR. For this step, the author designed a process to take advantage of Oracle’s Transportable Tablespace capabilities, which will be described later.
3.2.12 RAR Scoring Update to the ODS

Periodically, RAR customer scores are updated for use in ADM campaign generation and reporting. These scores are generated within the RAR and written directly to a special ODS staging table.

a. RAR Scoring Push – a data load over the network is used to push the RAR updated scores into a special ODS staging table used just for the scoring feeds.

b. RAR Scoring ODS Load – an ETL process is invoked to load the RAR updated scores from the ODS staging table and update the scores in the ODS core schema.

3.2.13 Analyst Access

Business analysts can perform ad-hoc analysis on the RAR database using a client that is loaded on each analyst workstation.

3.2.14 Developer Access

Application developers can perform analysis on the RAR database using a client that is loaded on each developer workstation.

3.2.15 File Extract for EDW Transaction Filter

On a daily basis, ETL is used to generate a file of new, encrypted credit card numbers. This file represents an incremental addition to the master transaction filter file. This file is used by the EDW to filter POS transactions for inclusion in future extracts from the EDW into the ODS.
3.2.16 File Extract for COA Processing

On a quarterly basis, ETL is used to extract a complete list of all active customers along with the latest mailing address for each customer. This file is transferred via ETL directly to the B2B server. This information is subsequently transmitted by the B2B server to an external vendor for COA processing.

3.2.17 Application Campaign Extract

After a new campaign is generated within the application, a list of customers targeted by the campaign is created in the application metadata repository (AMR). The production application server then uses this data to generate a flat file with campaign details. This file is then pushed via FTP to the B2B server for encryption and transmittal to external e-Marketing or mailing vendors. This file is also pushed via FTP to the input data file staging area for loading into the ODS.

3.2.18 B2B File Transfer to Vendor

The B2B server encrypts all outbound files. These files are then pulled by the designated vendor system via FTP in order to execute the campaigns defined in the files.

3.2.19 EDW Transaction Filter File Transfer

The daily incremental transaction filter file is loaded into the EDW using utilities on the ODS server. This file is appended to the customer filter table used to determine which POS transactions should be extracted for the ODS.
3.2.20 Customer De-Duplication

After COA data files are received and processed, an ETL process is initiated to write a complete active customer file to the file system. This customer file includes the name and current postal address for every customer in the ODS. The customer file is processed by the DDU in order to generate a customer merge output file.

3.2.21 Merged Customer Data Load

The customer merge file created by the DDU is loaded into the ODS through the staging tables. Subsequent ETL causes duplicate customers to be merged within the ODS core tables. The completion of customer merge processing triggers a full data load into the ODS outbound interface tables.

3.2.22 Application Server Data Access

The application server accesses the active ADM data in the view schema, as well as metadata and operational information from the metadata repository.

3.2.23 Web Server Communication with Application Server

A Web server is used to communicate with the application server and transmit Web pages to business users accessing the application.
3.2.24 Business User Access to Web Pages

Business users access the application through a Web browser interface pointing to the Web server.

3.2.25 Remote Admin Server

The remote admin server is used to configure all metadata as well as configure the DDU standardization and de-duplication rules and make updates to the rules as necessary.

3.2.26 Campaign Definition/Control Group Load into ODS

On a daily basis, an ETL process is triggered to load new campaign definitions into the ODS. New campaign definitions are first loaded into the staging tables and are subsequently populated into the core ODS by another ETL job.

3.2.27 RAR Ad-Hoc Score Updates to the ADM

Ad-hoc scores are transferred directly from the RAR to the ADM and do not get stored in the ODS. These scores are pushed into special tables in the ADM staging schema.

3.3 Data

As mentioned in Chapter 2, by Inmon’s standards [1], the project ODS could be labeled a Class III ODS, due to the fact that data written to it occurs on a daily basis in a batch cycle. Although data coming in from external vendors may occur multiple times
during the day at different scheduled times, it will be held and cleansed (as discussed in
the previous section) before being loaded. This data, along with internal data, will be
loaded at one time during the day into the ODS staging area.

Like Inmon’s description of a standard, textbook ODS, the project ODS was designed
to store data based on subject areas. In this case, the central subject area is the Customer.
Around the Customer subject area are the peripheral subjects, such as Store, Product,
Sales, and Campaign subject areas.

3.3.1 Customer Subject Area

The Customer subject area consists of all the data that collectively describes the
customer. In this case, a customer is a known individual that has interacted with the
company and has implicitly or explicitly agreed to receive marketing information. An
individual is “known” if there is a way to contact that person through direct mail or
email. Unique customers are identified by full name and full address. If that information
is not available, a customer is defined by email address. If the source data does not
provide a name, an address, or an email address, then the customer is considered
“anonymous” and is not loaded into the ODS.

The source systems providing data are either Product- or Store-focused. As such, there
is little integration or consistency in the customer data. As a result, most of the data
forming the Customer entity is either created by the ODS based on business rules or is
 appended data from third-party sources. The major types of customer data collected by
the source systems include customer name and address, email address, credit card
account data, Web site browsing behavior, and customer preferences collected from in-
store or Web site submissions. Before the project to build an ODS for the application was launched, there was technically not a customer management system of record. Thus, the ODS was also designed to “create” customers based on business rules executed on the incoming source data.

![Diagram](image)

**Figure 3.2 Project ODS designed around the Customer subject area**

### 3.3.2 Other Subject Areas

The peripheral subject areas in the project ODS receive their data from other external data sources and are used as the basis for the derived customer information. For example, Product data is sourced entirely from a separate Web reporting database system. In this case, the Product subject area is integrated at the source and can be loaded directly into the ODS. The Store subject area is similar and is sourced entirely from the enterprise data warehouse where the data is already integrated.
In contrast, the Campaign subject area data originates from multiple sources. For example, the application creates the campaign definitions, but both the application and the mailing vendor provide information on campaign generating activities. Once the definitions are created, they are then integrated into the ODS. Once there, the ODS can also execute procedures to correlate campaigns with resulting sales activity.

3.4 ODS Schemas

When reviewing the literature and process documentation involved in building an ODS (either for an enterprise or a specific application), an assumption can be made that the ODS itself consists of a single schema that holds all of the data. However, due to the project requirements and the necessity to move large amounts of data into and out of the ODS, multiple schemas were required.

3.4.1 Staging Schema

The ODS staging tables are populated with data sourced from flat files and are used as the starting point for data processing within the ODS. These tables are designed to, as closely as possible, mirror the structure of the associated input date files to minimize the data load complexity. As such, the tables are “loosely” designed (i.e. no normalization or star representation is necessarily required) and do not adhere to a formal data model.

As stated earlier, data from the staging schema is loaded into the core ODS schema using an ETL utility. Data transformations are executed in this step in order to manipulate the data into the format that exists in the core schema. Because this area of the ODS is only a holding area for data as it comes in, no querying or analysis should occur in the
staging schema. Also, since the staging tables are used exclusively to stage the data loads for the core ODS schema, data in all tables in this area can be purged after each successful data load cycle.

### 3.4.2 Outbound Schema

The outbound tables were designed to provide the core ODS data in a structure that facilitates the population of the application data mart. These tables are also used to populate the reporting/analysis database in order to support analytic requirements. The tables were designed to, as closely as possible, mirror the structure of the associated data mart staging schema to minimize the data load complexity. As such, they were designed using the data mart staging schema as a basis, but the outbound schema in and of itself does not adhere to a formal data model. Because this area of the ODS is only a holding area for data prepared to be loaded into other data sources, essentially making it a type of staging area itself, no querying or analysis should occur here. Also, since the outbound tables are used exclusively to stage the data loads for the data mart and the reporting/analysis database, data in all tables in this area can be purged after each successful data load cycle.

### 3.4.3 Core Schema

The ODS core tables hold the data that is considered to be the system of record for customer information. The subject area data mentioned previously, after going through ETL transformation and scoring calculations during the load from staging, is contained here. The design approach chosen for the core tables was to use normalized structures in
third normal form, but to de-normalize where necessary to facilitate ETL processing and provide for query performance when necessary (i.e. fact table design). This is consistent with design best practices for a star schema in the ODS literature.

<table>
<thead>
<tr>
<th>SALES_TRANSACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>sales transaction key</td>
</tr>
<tr>
<td>store key (FK)</td>
</tr>
<tr>
<td>brand key (FK)</td>
</tr>
<tr>
<td>customer key (FK)</td>
</tr>
<tr>
<td>promotion code key (FK)</td>
</tr>
<tr>
<td>web promotion code key (FK)</td>
</tr>
<tr>
<td>customer order key (FK)</td>
</tr>
<tr>
<td>source customer key (FK)</td>
</tr>
<tr>
<td>generic source key (FK)</td>
</tr>
<tr>
<td>store number</td>
</tr>
<tr>
<td>register number</td>
</tr>
<tr>
<td>sales date</td>
</tr>
<tr>
<td>transaction number</td>
</tr>
<tr>
<td>transaction time</td>
</tr>
<tr>
<td>transaction type code</td>
</tr>
<tr>
<td>total quantity</td>
</tr>
<tr>
<td>total sales amount</td>
</tr>
<tr>
<td>taxable amount</td>
</tr>
<tr>
<td>sales tax amount</td>
</tr>
<tr>
<td>sales tax state code</td>
</tr>
<tr>
<td>shipping amount</td>
</tr>
<tr>
<td>sales transaction total amount</td>
</tr>
<tr>
<td>sales channel type</td>
</tr>
<tr>
<td>customer CC holder indicator</td>
</tr>
<tr>
<td>customer barcode number</td>
</tr>
<tr>
<td>invoice number</td>
</tr>
<tr>
<td>retail type code</td>
</tr>
<tr>
<td>first purchase within brand indicator</td>
</tr>
<tr>
<td>first purchase within brand channel indicator</td>
</tr>
<tr>
<td>CC tender indicator</td>
</tr>
<tr>
<td>gift card tender indicator</td>
</tr>
<tr>
<td>tender type code</td>
</tr>
<tr>
<td>discount amount</td>
</tr>
<tr>
<td>insert date</td>
</tr>
<tr>
<td>update date</td>
</tr>
</tbody>
</table>

Table 3.1 Example ODS fact table

As the main feeder system to downstream end-user applications, the core schema possesses the full set of data that must be delivered to both the application data mart and the reporting/analysis database. As data is completely loaded into the core tables, it becomes available to push to the ODS outbound schema where the other system databases can retrieve the data.
The data load into the core schema occurs on a daily basis, and the data persists until its timeframe threshold is exceeded. In some cases, data must remain in the ODS core for a period of 24 months. This requirement has a definite impact on schema design. Because the amount of data can grow to be very large in the core schema (i.e. capacity is defined in terms of terabytes), the star schema design was chosen with de-normalization of the fact tables. Complete normalization of the schema with this amount of data would result in numerous, unnecessary table joins, leading to poor ETL and query performance.

3.4.4 Audit Schema

The audit schema, separate from those schemas that at one point or another hold the main customer data, represents a set of tables used for logging status and performance information for any running process in the ODS. For the purposes of this project, the primary function of the tables in this schema is to hold data created during the end-to-end ETL process (reference data flow steps in Section 3.1). Records stored in these tables reflect information gathered during ETL executions, including the run times of each step to monitor performance, as well as any records that failed to be loaded due to format errors or violations of business constraints or referential integrity.

3.5 Performance Enhancement Requirements

Due to the data retention policy stating that active customer records within the ODS must remain for a period of 24 months, it followed that heavy data volumes within particular tables would exist. It was estimated that several of the core tables would contain millions or hundreds of millions of rows.
This heavy data volume presented challenges in fulfilling certain ODS data processing requirements. One such challenge was in how to transport this volume of data (up to half a terabyte) from the ODS to the application data mart and the reporting/analysis database, each of which would be hosted on their own server separate from the ODS server. **On a daily basis, data from the ODS outbound schema needed to be transported to each of these databases during non-business hours, which was from 5 p.m. to 8 a.m. the following morning.**

Another challenge we faced was in maintaining these heavy data volumes in partitioned tables, while at the same time managing the size and structure of the tables. We chose to use partitioned tables where data volumes were especially large (hundreds of millions of rows) in order to minimize the time required for database queries to access the data. Due to the nature of the data, it made sense to partition the tables by date range, one partition for every month. Thus, each table would have 24 partitions, plus an overflow partition for any records that fell outside of the 24-month range. To deal with the challenge of maintaining this table structure each month, while at the same time maintaining the data within each table, a **monthly process was required to clean up old partitions that no longer fell within the last 24-month period, as well as add a new partition for the new month.** This process also needed to occur during non-business hours on the last day of the month and complete before business hours the following morning. **Additionally, the process had to run without affecting the data within each table.**

A third challenge we faced due to heavy data volumes involved statistics gathering on the ODS tables. Because Oracle, by default, uses a cost-based optimizer to find the least
expensive access path to retrieve data for a query, table statistics are required by the optimizer to determine the cost for each access path and choose the best option. The method we used for gathering statistics on these tables, however, was not always sufficient for the tables with the largest volume of data. We determined that these tables needed the capability of being analyzed outside of the normal method as necessary. Furthermore, since these tables should have the capability of being analyzed on an ad-hoc basis at any time of the day, ODS users should have the capability of collecting statistics on these tables without involving the DBA.

In the next chapter, the author will discuss further these ODS data processing challenges and document the solutions he developed to fulfill the data processing requirements dictated by the project.
CHAPTER 4

Performance Enhancements

During the course of the ODS project, part of the author’s role was to work through the ODS design with the data modeling team and application developers to provide suggestions and best practices for creating the ODS structures (i.e. tables, indexes, procedures), troubleshoot and tune SQL code as necessary for performance, and design custom component solutions for the ODS as the need was discovered. Some of these responsibilities were fulfilled in conjunction with other database engineers dedicated to the project. In addition, there were custom solutions created for the ODS on an individual basis. This section highlights some of the author’s individual contributions to the development of the project ODS.

4.1 Outbound Data Loads

The original architectural design for the marketing application system called on using the Oracle utility SQL*Loader to move data from the ODS outbound tables into the data mart for campaign processing and execution. Conceptually, it was a viable solution and technically possible. Three steps would be involved in the execution of this process, which would include (1) the outbound table data being extracted from the database and landing on the server file system as a flat file or files, (2) the file(s) being transferred to
the data mart server via FTP, and (3) the flat file(s) then loaded into the data mart staging schema. However, as data volumes increase, this process becomes increasingly time-consuming.

One of the limitations of using SQL*Loader in this case is that all schema data would be fully processed twice, once during extraction from the ODS outbound schema, and again during the data mart load. In between these steps, large, numerous data files would have to be transferred to the data mart via FTP. In the case of the ODS, it is possible that the outbound schema could be populated with up to half a terabyte of data. In addition, it was required that this data load process happen twice for each outbound load, once for the data mart and then again for the reporting/analysis database. It became clear that using the SQL*Loader and FTP technologies would not be sufficient to meet the service level agreements required by the business.

To solve the problem, the author looked to design a solution using Oracle’s Transportable Tablespace technology, as well as employing an extra storage volume group to move data quickly between database servers. This technology involves moving a user tablespace across databases by copying datafiles to, and importing tablespace metadata into the destination database. Unlike SQL*Loader, complete extraction and re-insertion of the entire dataset from the source database to the destination database is not required.

In this design, the server data files that belong to the outbound tablespace are first duplicated (synced) to a spare storage volume group mounted to the ODS server. The storage technology is such that this operation usually takes less than ten minutes. In
contrast, it could take hours to FTP these files over the network to another database server.

Next, the tablespace containing the ODS outbound schema must be placed into read-only mode. The technology requires that no queries or transactions can occur against the tablespace while it is being “unplugged” from the source. Since this kind of activity very rarely occurs against the outbound schema, there is no real drawback to this requirement. Once this is complete, the tablespace metadata, which consists of object information and definitions and referential integrity information, must be extracted manually. This will be used later to define the tablespace in the data mart and reporting/analysis database.

![Figure 4.1 Transportable Tablespace process](image)

**Figure 4.1 Transportable Tablespace process**

Once the metadata is extracted, the synced volume group is un-mounted from the ODS server and mounted onto the data mart server. Once this is complete, the ODS outbound tablespace can be placed once again in read-write mode and available for reloading on the next batch cycle. In the meantime, the outbound tablespace data files,
now synced from the ODS and mounted onto the data mart server, are ready to be
discovered by the data mart. Discovery occurs when the tablespace metadata, exported
earlier from the ODS, is imported into the data mart. When this information is exported
from the ODS, it is stored as a flat file on the server. This is a small file (less than 10
kilobytes) and can easily be transferred via FTP. To save a step in the process and bypass
the FTP transfer, when exporting the metadata, the resulting flat file is placed onto the
same file system containing the outbound tablespace data files. Thus, when these data
files are synced with the extra volume group, the metadata file is also synced and brought
over to the data mart server. The metadata can then be imported and the ODS outbound
tablespace is now contained inside the data mart as well. The volume group now mounted
to the data mart server is then un-mounted and the process starts over to move the
tablespace into the reporting/analysis database. Whereas unloading the outbound data
into a flat file, using FTP to move the data files, and loading the flat file data into another
database would take several hours, the solution using Transportable Tablespaces
completes a full load from the ODS to the data mart and the reporting/analysis database
in approximately one hour. The only extra cost is in creating an additional storage
volume group to use for duplicating the original tablespace data files. However, when
considering that unloading the outbound data to flat files would also require additional
storage on the server file system, there is not any extra cost in this solution.
4.2 Table Partition Management

The project requirements dictated that the data inside the ODS persist for at least 24 months. Once data becomes older than 24 months, it is then subject to being archived and purged from the database. However, not all customer records older than 24 months are necessarily purged. As long as a customer is active (i.e. has made a purchase in the last 24 months), the record will stay in the ODS.

In most cases, a special process handles this data offload without having to make any structural changes to the core schema. In the case of partitioned tables, there needed to be a process that would keep the table structure with as few partitions as needed in order to enhance ETL and query performance. Since the partitioned tables in question are partitioned by date (one partition for each month of the year), a new partition would need to be created each month to handle all new records. However, the single-step process of adding table partitions each month does not address the cleanup and removal of the earliest month partition in the table (which would become the 25th table partition each time a new month partition is added to the end of the partition list). Additionally, since data in the 25th table partition may not necessarily be archived and purged, it might need to remain in the ODS. This eliminates the possibility of simply dropping the earliest table partition outright after the new month partition is added.

The solution to this problem called for a design that would retain old but “active” ODS records after the archive and purge process executed every month while removing obsolete partition structures from the table. This solution is based on Kimball’s concept of the rolling month, whereby “…when the last day of the month is reached, the current rolling month becomes the most recent member of the standard months in the time series
and a new current rolling month is created [8].” Every month, at the end of the archive and purge process, a stored procedure will identify the names of the earliest two partitions in each partitioned table. Oracle’s data dictionary holds this information and will associate the partition names with a value named “partition position.” In this case, the earliest two partitions of a table hold partition position values of 1 and 2, respectively.

Once these names are determined, the procedure then leverages Oracle’s partition merge technology, which has the dual purpose of extracting the data from partition 1 and loading it into partition 2, and then dropping the now empty partition 1.

Once this is complete, a new partition must still be created for the new month. This is
not as easy as it sounds, due to the fact that each partitioned table has an overflow partition which stores data that does not fit into any of the other partitions based on the date of each record. This overflow partition is always placed in the last partition position on the table, immediately after the partition representing the current month, and it cannot simply be dropped as there may be overflow records contained within it. In addition, Oracle does not allow the creation of a partition that would hold a position in between other existing partitions. When creating a new partition, it must be added to the end of the list and have the greatest partition position value for the table.

<table>
<thead>
<tr>
<th>Step #</th>
<th>Job Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get tablespace name/partition name for each of the two oldest partitions.</td>
</tr>
<tr>
<td>2</td>
<td>Drop any existing table constraints.</td>
</tr>
<tr>
<td>3</td>
<td>Drop any existing table indexes.</td>
</tr>
<tr>
<td>4</td>
<td>Create temporary (temp) table, if not already created.</td>
</tr>
<tr>
<td>5</td>
<td>Merge two oldest partitions into a single partition, now the oldest partition.</td>
</tr>
<tr>
<td>6</td>
<td>Exchange overflow partitions between base and temp tables.</td>
</tr>
<tr>
<td>7</td>
<td>Drop overflow partition from base table.</td>
</tr>
<tr>
<td>8</td>
<td>Create new month &amp; new overflow partitions on base table.</td>
</tr>
<tr>
<td>9</td>
<td>Recreate indexes and constraints on base table (if necessary).</td>
</tr>
<tr>
<td>10</td>
<td>Load data (if any) into base table from overflow partition in temp table.</td>
</tr>
<tr>
<td>11</td>
<td>Drop the temp table.</td>
</tr>
</tbody>
</table>

Table 4.1 Steps in partition management process
To get around this issue, the procedure was designed to create a temporary partitioned table. The temporary table is identical to the partitioned table requiring the structural manipulation, but it contains no data. Once this is complete, the procedure leverages Oracle’s capability to exchange partitions with another table. The empty overflow partition of the temporary table is exchanged with the base table’s overflow partition which may contain data. Now, after the exchange, the overflow partition of the base table is guaranteed to be empty and can be dropped. The procedure then creates two new partitions at the end of the partition list, one for the new current rolling month and one for overflow data. This step completes the monthly structural modifications for the base table. The last step of the process is to load the data (if any) from the overflow partition in the temporary table that was previously exchanged with the base table. This is a simple load statement, at the end of which the temporary table no longer has any purpose and can be dropped.

4.3 Ad-Hoc Table Analysis

In Oracle 9i, the DBMS used for the project ODS, the default behavior for query optimization is to use a cost-based optimizer, which chooses the least expensive execution plan, in terms of system resources, to access data. Oracle will use statistics gathered on database objects to determine the least expensive access path to retrieve the data requested in a query. For example, for a given query on a table, if the optimizer knows about the data in the table (e.g. number of records, data distribution, and partitioned data), it can decide if a full table scan to retrieve the result set would be faster that using a table index. Thus, to make sure that query performance remains at an
acceptable level, statistics on database tables and indexes need to be gathered on a regular basis.

The project standard for analyzing database objects was to execute the analysis daily at a schema level basis. For the ODS, since the freshness of data was determined by each batch cycle, it was decided that this analysis should be executed once each batch cycle was complete. Because gathering object statistics is a resource-intensive operation, we determined the following three guidelines for execution: (1) analysis was to occur after business hours when the system was not as active; (2) only a database administrator could have access to run the analysis, thus preventing application analysts from triggering the job at inopportune times; and (3) statistics would only be gathered on those tables and associated indexes whose data volume had changed by at least 10% from the last analysis. This last point was decided so that tables containing data volumes that were relatively stable would be skipped. As volume changes less than this amount would not likely skew the current table/index statistics, thus not affecting query performance to an unsatisfactory level, database resources could be conserved.

However, for some of the larger tables in the ODS, this was not an acceptable solution. For example, more than one table in the ODS was estimated to hold over one billion records. This would mean that for each table at this size, data volumes could increase to any level up to, but just below, 100 million records and the table would still be skipped under the statistics analysis guidelines. This change in data volume, without gathering fresh statistics, could adversely affect query performance as the cost-based optimizer would be basing its data retrieval path on outdated statistics. We decided that there needed to be a utility that could provide statistics analysis on these large tables.
independent of the 10% data volume change rule. We also decided that since there were only a few tables that would fall into this classification, the application analysts should have access to update statistics on only these tables as needed.

<table>
<thead>
<tr>
<th>ANALYSIS_CONTROL_TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER</td>
</tr>
<tr>
<td>TABLE_NAME</td>
</tr>
<tr>
<td>ESTIMATE_PERCENT</td>
</tr>
<tr>
<td>DEGREE</td>
</tr>
</tbody>
</table>

Table 4.2 Control table for ad-hoc table analysis

To do this, the author created a table that would contain control data on these large tables. This table would contain four pieces of information for each table to be analyzed on an ad-hoc basis, namely (1) the table owner, (2) the table name, (3) the data sample size used to estimate statistics for the entire table (taken as a percentage of each data block held by the table), and (4) the degree of parallelism to use in analysis. To avoid unexpected results by making sure that only one data sample size and a specific degree of parallelism is used for each table, a primary key is created on the table owner and table name columns.

Next, the author created a procedure used to call this table and extract the information when an application analyst needed to gather table statistics (e.g. on noticeably poor query performance). The procedure takes the table name as one argument and gathers the
associated information from the control table if the table name exists there. If the table name does not exist in the control table, the attempt is logged in an audit table along with the name of the database account making the attempt, and the procedure exits gracefully.

If the table name exists in the control table, all associated table information is gathered. The individual values for each column in the record are then used as arguments in a separate procedure that is called to execute the actual statistics gathering process. All indexes associated with the table being analyzed are also analyzed. In addition, the execution is logged in an audit table and includes the name of the database account making the attempt, the table analyzed, the elapsed time of the gathering process, and the return code. Because application analysts have been given privileges not usually available to them, the audit table is used to help the database administrator control the process by keeping records of who is using the procedure, what is being analyzed, and how often the procedure is executed. This process is designed to increase database performance by pinpointing potential problem tables with heavy data volumes and keeping their statistics current through an ad-hoc, non-standard means of object analysis.
CHAPTER 5

Implementation

5.1 Outbound Data Loads

The process to move the outbound schema data from the ODS into the data mart and the reporting/analysis database was implemented and tested in all testing environments before making it available in production (see Appendix A for details regarding the project test environments). In all phases of testing previous to the performance test phase, the process was implemented mostly as a proof-of-concept. The true functionality and performance of this process could not be tested in the early testing phases since the system databases were all being hosted on the same server, rather than existing in a distributed environment on different servers. Once implemented in the performance test environment (which would eventually become the distributed production environment), we could then test the process and get a good sense of its behavior.

When we tested the process in the performance test environment, the ODS outbound data was successfully transported to the application data mart and the reporting/analysis database within the allotted time window of 5 p.m. to 8 a.m. (non-business hours). Once the entire end-to-end process was thoroughly tested and prepared for production, it consistently completed successfully for both of the destination databases in one hour or less. During testing, two issues were discovered that required modification of the process.
Test # | Outbound Data Volume | Time to Import to ADM | Time to Import to RAR | Total Time of Execution |
--- | --- | --- | --- | --- |
1 | ~225 GB | 22 minutes | 17 minutes | 39 minutes |
2 | ~270 GB | 25 minutes | 18 minutes | 43 minutes |
3 | ~350 GB | 34 minutes | 22 minutes | 56 minutes |
4 | ~500 GB | 41 minutes | 27 minutes | 68 minutes |

Table 5.1 Sample test results for ODS outbound data loads

5.1.1 Placing Outbound Schema Tables in Read-Only Mode

In Section 4.1, the author explained that the ODS outbound schema tables must be in read-only mode before tablespace metadata can be extracted. Oracle’s Transportable Tablespace technology requires that no queries or transactions can occur against the tablespace while it is being “unplugged” from the source (i.e. when metadata is being extracted). During testing, however, it was also discovered that if ODS analysts were logged into the database and performing queries against these outbound tables (a rare but possible occurrence), they could not be placed into read-only mode until the queries were completed. As such, if a query was running against an outbound table or tables during an execution of the data load process, the process would hang and wait until all queries were complete before resuming. Since queries against ODS data can take a significant amount of time, the data load process could be frozen beyond an acceptable time window. The priority in this case was to stop the user processes running queries against the outbound tables and continue with the data load process.
One possible solution to this problem would be to write a script that queries the database catalog and finds the processes running against these tables and then kill these processes. What made this solution inadequate was the fact that while the script was querying the database catalog to find the user processes and kill them, it was possible that a second set of user queries could execute during the time it took to kill the first set. Then, once the first set of queries was killed, the second set of queries could prevent the tables from being placed in read-only mode. This scenario could be repeated continuously, thus further delaying the outbound data load beyond an acceptable window.

In this case, it was decided that a brief outage to the ODS would be acceptable in order to process the data load in a timely manner. The solution then was to simply shutdown and restart the ODS immediately before placing the outbound tables into read-only mode. This allowed us to make sure all user processes were eliminated at one time. Upon restart of the ODS, the tables were immediately placed into read-only mode before user processes were able to begin a query against an outbound table or tables.

5.1.2 Metadata Export

The export of tablespace metadata from the ODS includes data describing the structure of the outbound tables, indexes, constraint and referential integrity, stored procedures and packages, user privileges, and other objects. When testing the import of this metadata into the data mart and reporting/analysis database, the job would fail if privileges had been granted on an object to an ODS user (i.e. an analyst account) that did not exist in either of the other databases.
One way to solve this problem would be to extract object privilege metadata from the ODS for only those users that exist in the other databases. This solution, however, was impractical. If user privileges changed over time in the ODS, or if users were added to or removed from the ODS, the process of only extracting certain privilege metadata would have to be corrected manually for each alteration. In addition, the metadata should only be extracted from the ODS once per data load for both destinations. But, since the data mart and the reporting/analysis database may have different user accounts, the metadata import would fail on at least one of these databases, thus stopping the process.

The solution to these problems was to leverage Oracle’s export/import technology by allowing the import of metadata to ignore errors raised when trying to apply object privileges to users that don’t exist. As such, when importing metadata into each destination database, the import process will attempt to apply any user privilege associated with an outbound table or other object. If the privilege belongs to a user that does not exist in the destination database, the import process simply ignores the error and continues the import. This allows us to extract all object privileges at one time from the ODS and transfer them to their destination without having to worry about if the user account exists in the destination database. Also, since all privileges on the outbound objects are extracted at once, there is no manual alteration necessary if users are added, deleted, or modified over time in the ODS.
5.2 Table Partition Management

The process of managing the data and structure for large, partitioned tables was implemented and tested in all testing environments before making it available in production. In all phases of testing previous to the performance test phase, the process was implemented mostly as a proof-of-concept. The true functionality and performance of this process could not be tested in the early testing phases since these environments were not built to hold production data volumes. Once implemented in the performance test environment with production-like data volumes, we could then test the process and get a good sense of its behavior.

<table>
<thead>
<tr>
<th>Partitioned Table Name</th>
<th>Average Table Data Volume</th>
<th>Average Time of Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales Discount</td>
<td>~453 million rows</td>
<td>70 minutes</td>
</tr>
<tr>
<td>Sales Tender</td>
<td>~320 million rows</td>
<td>29 minutes</td>
</tr>
<tr>
<td>Sales Transactions</td>
<td>~366 million rows</td>
<td>64 minutes</td>
</tr>
<tr>
<td>Sales Transaction Line Item</td>
<td>~1.4 billion rows</td>
<td>188 minutes</td>
</tr>
</tbody>
</table>

Table 5.2 Sample test results for “rolling partition” process

When we tested the process in the performance test environment, the partition management process, executing against partitioned tables holding production data volumes, completed successfully within the allotted time window of 5 p.m. to 8 a.m. (non-business hours). Once the entire end-to-end process was thoroughly tested and prepared for production, it consistently completed successfully in between five to six
hours. On average, the process, which consists of partition management for four of the largest ODS tables, took approximately one minute of clock time for every seven million records of data held within the tables collectively.

The main issue faced in implementing this process involved indexes created on the partitioned tables. A side effect of merging partitions and creating new partitions for each table was that indexes associated with the original partition structure became invalid and were no longer usable. It was required, then, to drop all indexes and any associated constraints on each partitioned table before moving forward with partition manipulation. Once the new partition structure for each table was in place, all previously existing indexes had to be rebuilt based on the new structure. Due to the volume of these tables (each of which contained at least 250 million records, with the largest containing approximately 1.4 billion records), much of the execution time spent on the process was used to rebuild table indexes. However, even with the extra processing time required to rebuild indexes, the process still executed successfully within the allotted time window.

5.3 Ad-Hoc Table Analysis

The process to allow select ODS users to analyze large tables on an ad-hoc basis was not implemented and tested until the ODS was built in the performance test environment. In all phases of testing previous to the performance test phase, the data volumes used represented a fraction of the estimated production data volume. As such, the standard process of statistics analysis at the schema level was sufficient. It was not until the ODS was holding production-like data volumes that we realized an alternative was required.
After some tuning of the information held within the control table, the process for user-enabled, ad-hoc table analysis was executed successfully by the author and selected ODS users. Execution times varied based on the size of the table, the data sample size used to estimate statistics for the entire table, and the degree of parallelism used for each table. All process executions completed within three to ten minutes, depending on the data volume of each table. These results were acceptable for the ODS users.

The main issue faced in implementing this process was determining the best data sample size and degree of parallelism to be used for each table. Several tests were executed against the ODS tables registered in the control table to determine an acceptable level of accuracy based on the data sample size, and an acceptable execution time based on the degree of parallelism used in each run. ODS tables that contained over 10 million records were placed in the control table and analysis could be executed by a select group of ODS analysts. Any table with less than 10 million records would be left out of the control table and would be analyzed based on the standard process. The values specified in the control table, determined by these tests to be the optimal parameters for analysis based on table size, are illustrated in Table 5.3.

<table>
<thead>
<tr>
<th># of Records Per Table</th>
<th>Data Sample Size</th>
<th># of Parallel Processes Used</th>
<th>Time of Execution</th>
<th>Accuracy Variance (Maximum)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-50 million</td>
<td>10%</td>
<td>16</td>
<td>3-5 minutes</td>
<td>5%</td>
</tr>
<tr>
<td>50-100 million</td>
<td>5%</td>
<td>32</td>
<td>3-5 minutes</td>
<td>2%</td>
</tr>
<tr>
<td>&gt; 100 million</td>
<td>1%</td>
<td>32</td>
<td>3-10 minutes</td>
<td>2%</td>
</tr>
</tbody>
</table>

Table 5.3 Optimum values used in ad-hoc analysis control table
In this thesis, the author provided an overview of an ODS and explored some of its fundamental characteristics. As described earlier, an ODS is a collection of detailed, subject-oriented data, derived from various external sources. The data is operational in nature (as opposed to the summarized data typically found in a data warehouse) and used as a decision-support system for a company based on that company’s specific subject areas. The ODS can typically fall into one of three classifications, based on the frequency of data updates. Class I refers to the Online Synchronous ODS, in which updates made to the source operational data are reflected in the ODS in a matter of seconds. Class II refers to the “Store-and-Forward” ODS, in which operational data updates are gathered in a separate location, independent of both the primary data sources and the ODS, and propagated to the ODS at one time at regular intervals. Class III refers to the Batch Load ODS, in which updates are executed asynchronously using batch processes. Because the Class III ODS is loaded less frequently than a Class I or II ODS (usually not more than once daily), updates to source data are often subjected to heavy integration and transformation.

Additionally, the author went on to discuss the Class III ODS that he helped to design and implement, including descriptions of the performance improvements he provided for
the ODS project. These performance improvements centered on the areas of handling large data transfers between systems in a timely and efficient manner, automating data management and size control in large, partitioned database tables, and providing a flexible, user-friendly utility to gather database statistics necessary for the ODS cost-based optimizer. The specific results of these performance enhancements were discussed in Chapter 5. Overall, the design and implementation of these enhancements allowed for the execution and successful completion of critical processing requirements, internal data management, and efficient query performance in a timely manner.

Since the new system has been in production, the author has identified some opportunities to enhance the ODS to address further possible performance issues. Specifically, the subject of ODS user resource consumption has been an issue to address in production. One aspect of this issue involves limiting the amount of computing resources a given user can consume when logged on to ODS and issuing queries. With the exception of the time period in which the daily batch cycle is running, ODS users are typically free to logon to the ODS and issue queries as necessary. As such, there may be periods of time where several users are logged on and issuing large, resource-intensive queries at the same time. Since an ODS user may not know that other users are logged on and vying for resources, a resource sharing policy should be designed and implemented at the database level. To date, the author has designed a resource plan that allows the database to limit CPU and parallel process utilization for a given user. It has been tested successfully for a common ODS user ID (this ID is accessible by several users for general reporting purposes), but it has not yet been implemented for individual account IDs. More testing is required to make sure a given user is allowed to retain the computing
resources required for that user’s work without limiting resources for other users, or limiting resources as little as possible.

Additionally, a second aspect of resource contention has been identified. In this case, individual ODS users may unknowingly attempt to issue queries during the daily batch cycle. There have been instances where batch processing was substantially slowed due to resource contention caused by the additional workload on the ODS from user queries. This has also required a review of the user policy, in terms of hours of operation. A possible solution here would be to implement a script that would identify individual account IDs and do the following: (1) before the daily batch cycle processing begins, kill all individual user account sessions that are open; (2) lock down the accounts during the batch cycle; (3) unlock the accounts once the batch cycle is complete. The author has already implemented this user policy in other systems successfully. This ODS system is an excellent candidate for implementation of this policy.

During the period of time working on the corporate ODS project, the author learned to think of the ODS as a single, yet critical component of a larger computing system. Although ODS design and implementation were the main scope of the project, these cannot not accomplished without understanding the various computing components that surround the ODS. Thinking about the various data sources, data formats, and data flow transfers both into and out of the ODS play major roles in the eventual design and implementation. Additionally, ODS design is impacted by data volumes that grow continuously over time. Challenges concerning performance of execution arise as the volume of data within the ODS (or any database) increases, requiring creative and robust solutions to maintain a high level of data processing performance.


an Architectural Approach to Meet the Challenges of Customer Orientation.”
Proceedings of the 36th Annual Hawaii International Conference on System

http://www.dbmsmag.com/9712d05.html.

Support Business Initiatives and Needs.” Proceedings of the 37th Hawaii
International Conference on System Sciences, January 5-8, 2004. Piscataway, New


Kaufmann, 1999.

Approach to Immediate View Maintenance for an Operational Data Store.” Annual
Conference of the North American Fuzzy Information Processing Society, July 25-
Appendix A: ODS Implementation Phases
Build Environments

The implementation of the project ODS was executed in four build iterations during
the development and testing phases. The fourth build iteration in the performance test
environment also served as the pre-production environment. Once performance testing of
the application was completed, the performance test environment was converted into the
actual production database environment.

Development/Unit Testing

The development database environment was designed and built to hold 10% of the
estimated volume of 2.5 terabytes of production data. As such, the ODS, data mart, and
metadata repository were all small enough to be hosted on one server. This did not
provide a conflict in building the ODS to handle large data transfers across physical
servers, as was the original design of the system. Since this environment was only
designed to support and test each basic component of the application system (i.e.
designing and testing each part individually), there was no need to test the system end-to-
end.

An aspect of this environment that is not commonly discussed in the literature is that
of multiple schema management. For example, in Section 3.4, the author discussed the
fact that the project ODS was designed to hold several schemas that would each serve a
purpose in the flow of data through the system. This design held true for all testing
environments. Additionally, in the development environment, several different
incarnations of the core ODS schema needed to be built and managed. The multiple incarnations became necessary because multiple application developers working on the project needed their own environment in the database for component testing. While this dictated that a significant amount of time and energy would be needed from the database engineering team in order to build and manage the different schemas (during which the author created weekly DDL code releases for distribution among all application developers), it ultimately enabled developers to work independently and in parallel during this phase of implementation. It did, however, become incumbent upon developers to keep their own schemas updated with the weekly DDL releases, as all incremental DDL updates represented changes in ODS design (as decided on by the database engineers, in concert with data modelers and application developers) that would ultimately become the production release.

Database environments for each testing phase

System Test Environment

The system test database environment was designed and built to hold 20% of the estimated volume of 2.5 terabytes of production data. As such, the ODS, data mart, and metadata repository were all small enough to be hosted on one server. This did not
provide a conflict in building the ODS to handle large data transfers across physical servers, as was the original design of the system. Since the system test environment was only designed to make sure that the components built in the development environment would function properly on an integrated hardware and software system (whereby the system meets the specified requirements of the developed application components), there was no need to test the system end-to-end. To support the testing of an integrated system, this environment also included a dedicated application server as well as an ETL server to test data loads into, through, and out of the ODS database.

**Integration Test Environment**

The integration test database environment was designed and built to hold 35% of the estimated volume of 2.5 terabytes of production data. As such, the ODS, data mart, and metadata repository were all small enough to be hosted on one server. The purpose of this environment was to begin end-to-end testing of the application system in order to validate how a change to the system would operate in its surrounding environment. The focus here was more on functionality then on performance or scalability, hence the decision (along with budgetary decisions and available computing resources) to have the ODS hold only a fraction of the estimated production data. Although the performance test environment would be the true test in judging how the system handled large data transfers across physical servers, the concept was also tested in this environment. For example, the data load from the ODS into the data mart using Oracle’s Transportable Tablespace technology was tested in this environment. However, this was done here more as a proof-of-concept validation. Since the ODS and data mart databases were hosted by the same
server, it was not necessary to test the synchronization of the extra storage volume group and subsequent mount to a different server. For the purposes of this environment, it was sufficient to test the export of metadata from the ODS, the transfer of data files from one file system on the server to a different file system on a different storage volume group for the data mart, and the metadata load into the data mart to discover the outbound tablespace data files.

In addition, to support end-to-end testing throughout the application system, this environment included a dedicated application server as well as an ETL server to test data loads into, through, and out of the ODS database.

**Performance Test/Pre-Production Environment**

The performance test database environment was designed and built to hold 100% of the estimated volume of 2.5 terabytes of production data. The purpose of the performance test environment was to measure how the system behaved under production-sized data volumes and computing resources, which could then be compared against known or expected performance levels. This environment would also serve as pre-production, whereby once testing was complete and the ODS was ready for production, the performance environment would transition into the production build. All naming conventions, external data source connections and loads, ETL configuration, and application server configuration in this test environment were built to production standards in anticipation of the transition to production once testing was complete. Because of the large estimated size of the ODS, there was a need for dedicated computing resources on its host server. As such, in this phase of testing, the ODS was separated
from the data mart and metadata repository and hosted on its own high-powered server (8 CPUs at 1300 MHz each with 24 GB of dedicated memory).

The decision to bypass a performance test environment separate from the eventual production environment was based on project budgets. In terms of building and testing a production-ready ODS in its eventual destination environment, the decision made for a seamless transition to the production build. This eliminated the need for migrating the ODS components (e.g. database DDL, stored code, security configuration, and backup configuration), once completely built, into a separate environment. The trade-off in this approach, however, is that once the ODS was ready for production, this environment completely became the working ODS production environment, and the performance test environment was lost. As enhancements have been made to the ODS over time, new stored code and database object configuration have been limited to testing in the development, system test, and integration test environments, all with well-below production data volumes and computing resources. While much can be learned from testing enhancements in these environments, there are limitations encountered by not being able to test against a production-like system. As such, performance results learned here may not necessarily apply to what will happen once the code is released for production. This was a known risk during the project.

Due to the distributed nature of this environment and the volume of data it was designed to hold, the author was able to test his ODS contributions here and observe practical results.
LIST OF ACRONYMS

ADM – Application Data Mart
AMR – Application Metadata Repository
API – Application Program Interface
B2B – Business-to-Business
CC – Credit Card
COA – Change of Address
DBMS – Database Management System
DDL – Data Definition Language
DDU – De-Duplication Utility
EDW – Enterprise Data Warehouse
ETL – Extract/Transform/Load
FTP – File Transfer Protocol
ODS – Operational Data Store
OLTP – Online Transaction Processing
POS – Point-of-Sale
RAR – Reporting/Analysis DB Repository
RMAN – (Backup and) Recovery Manager
SQL – Structured Query Language
XML – Extensible Markup Language
Appendix C: Procedures for ODS Performance Enhancements
1. Outbound Data Loads

Script #1: Drop the outbound tablespace from the application data mart (SQL script w/ shell wrapper)

- Set environment variables.
- Login.
- Drop the outbound data tablespace.
- Log out.
- Write timestamp, success/failure status to log.

Script #2: Unmount volume group from data mart server (shell script)

- Set environment variables.
- Check if volume group is mounted. If not then exit, unmount not required.
- Execute proprietary unmount script.
- Write timestamp, success/failure status to log.

Script #3: Re-sync volume group with ODS outbound data (shell script)

- Set environment variables.
- Check for lock file, is the script already running? If so then exit, script failed due to existing lock file.
- Execute proprietary re-sync script.
- Write timestamp, success/failure status to log.

Script #4: Shutdown ODS to clear out any active processes running against outbound data (SQL script w/ shell wrapper)

- Set environment variables.
- Login.
- Force a database checkpoint.
- Shutdown the database.
- Log out.
- Write timestamp, success/failure status to log.
Script #5: Restart the ODS (SQL script w/ shell wrapper)

- Set environment variables.
- Login.
- Start up the database.
- Query status to confirm that database is open and in read/write mode.
- Log out.
- Write timestamp, success/failure status to log.

Script #6: Put ODS outbound tablespace in read-only mode (SQL script w/ shell wrapper)

- Set environment variables.
- Login.
- Alter tablespace to read-only mode.
- Log out.
- Write timestamp, success/failure status to log.

Script #7: Export ODS outbound tablespace metadata to a flat file (shell script)

- Set environment variables.
- Use Oracle export utility to export metadata, define tablespace here.
- Write timestamp, success/failure status to log.

Script #8: Unmount synced volume group from the ODS server

- Set environment variables.
- Check if volume group is mounted. If not then exit, unmount not required.
- Execute proprietary unmount script.
- Write timestamp, success/failure status to log.

Script #9: Put ODS outbound tablespace back in read/write mode (SQL script w/ shell wrapper)

- Set environment variables.
- Login.
- Alter tablespace to read/write mode.
- Log out.
- Write timestamp, success/failure status to log.
Script #10: Mount the synced volume group to the data mart server (shell script)

- Set environment variables.
- Execute proprietary mount script.
- Write timestamp, success/failure status to log.

Script #11: Import tablespace metadata into application data mart (shell script)

- Set environment variables.
- Use Oracle import utility to import metadata. Define the metadata flat file from export and tablespace datafiles here.
- Write timestamp, success/failure status to log.
2. Table Partition Management

A shell script with embedded SQL is called on a monthly basis to manage the size of tables partitioned according to a date range, in this case by month. The embedded SQL calls an Oracle stored procedure for each of the four range partitioned tables in the ODS. For each table, the procedure follows the same process.

- For the oldest and second-oldest table partitions,
  - Select tablespace name for each partition into a variable.
  - Select partition name for each partition into a variable.

- Select the number of constraints on the table. If greater than 0, drop the constraints.

- Select the number of indexes on the table. If greater than 0, drop the indexes.

- Check if the temporary table exists. If not, create with same structure as base table.

- Merge the two oldest partitions together into one partition.

- Exchange the overflow partition of base table with overflow partition of temporary table.

- Drop the overflow partition from base table.

- Create partition for new rolling month. Create new overflow partition.

- If dropped, recreate indexes and constraints.

- Load data from overflow partition of temporary table into base table.

- Drop temporary table.
3. Ad-Hoc Table Analysis

The ad-hoc table analysis procedure is called using an SQL execute command, which queries the control table for analysis information and then executes the analysis. The control table is created to hold analysis parameters for each table that can be analyzed using this procedure (see Table 4.2). A log table is also created to hold information for each execution of the procedure.

- Create a cursor to select the analysis parameters for the table specified in the execution call.
- If the table does not exist in the control table, then write to the log table. Send error output to the user, and exit.
- Fetch cursor elements into declared variables.
- Run table analysis using variable values collected from the cursor.
- Record the execution in the log table, which includes table name, elapsed time, success/failure, owner of the executing process, and date of execution.