ABSTRACT

This paper discusses the process of using a SAS basic package to build an OLAP data mart on a Windows-based server using operational data extracted from the corporate mainframe. Key tasks, including hardware selection, ETL, dimensional modeling, data partitioning, indexing and parallel processing are discussed, with examples from our development effort.

INTRODUCTION

The work described here was done in support of the Kaiser-Permanente Infectious Disease Reporting System (IDRS), chaired by Dr. Roger Baxter of Kaiser-Permanente (KP). In a clinical and operational context, the IDRS addresses a critical reporting and clinical surveillance need; namely, monitoring trends in and identifying the correlates of an increase in antibiotic-resistant infections observed in the population. The analytical result is of value to both KP MCP and to the public health and infectious disease professionals in the community, since the phenomenon is widespread and difficult to evaluate with precision and certainty. However, the process of developing the data system capable of meeting these needs is an extraordinary challenge and our experience would also be of value to engineers and other computer professionals.

At a minimum, the data mart would need to address each patient's risk of infection from community and clinical exposures, previous use of antibiotics, history of infection, and other key elements of his/her short- and long-term medical history in order to support the reporting and analytical requirements. The KP laboratory, pharmacy, and clinical systems capture the required data, but merging these diverse sources into a meaningful, analyzable format proved to be a challenging problem.

The literature suggests that this problem may well be unique to KP. While insurance companies typically have vast quantities of pharmacy and other clinical utilization information from billing records and may be able to derive key diagnostic information by inference, they do not typically have access to specific laboratory results and have not had to address this issue. Similarly, organizations that provide laboratory services or perform disease surveillance have well-defined algorithms for evaluating test results, but have little or no access to the patient-level clinical data that could explain observed changes in the incidence of infection and antibiotic resistance in their population of interest. Finally, health services research studies may be based on comprehensive data, but it is typically culled from chart review and limited to very small study samples – which made them unsuitable to the sheer volume of KP data.

Objective

Initial work on the system began in January 2003. The Phase I objective was to build an OLAP Antibiotic data mart that could

- Support multi-users
- Produce outputs in HTML, Excel, PDF formats
- Capture key clinical data from 1996 to current
- Refresh data quarterly

Environment:

Data Source – Most of the data reside on the KP IBM mainframe in DB2, IMS or flat file formats. Some data are also migrated onto tapes. In addition, a small number of user-defined translation tables are on desktop PCs in Excel format.

User – Typical Microsoft workshop; PCs are Window-based equipped with communication software Attachmate connecting to mainframe SAS to get data/and reports.

Decision:

SAS was selected to be the database engine and development tool for the following reasons:

- SAS is the primary reporting tool used within KP
- SAS provides excellent tools to move data across platforms
- SAS support and programmers are widely available within the organization

Due to budgetary constraints, a number of advanced SAS tools that would have been well-suited to this project, such as OLAP server, Data Warehouse, MDDB, could not be used. The project relies primarily upon the basic SAS package.

HARDWARE SELECTION

Hardware was selected based on (1) database size and (2) performance needs.

Database Size

An enormous amount of operational data is required to support this system – the current extract is over 1 terabyte. However, the final database size will come down to about 120 gigabyte after the ETL process.

Performance

In Phase I of this project, the datamart will support only a few research analysts. Utilization of the system is expected to be evenly distributed (i.e. simultaneous access by users is unlikely), but individual jobs (e.g., a complex query) could required an intensive amount of computing resources.
Hardware List
- Intel Zeon 3.06 GHz dual processors
- 2 GB memory
- 3 SCSI hard disk with 150 GB capacity each
- Raid 0 Controller

ETL
The Extraction, Transformation, and Load (ETL) is the single most critical and labor-intensive piece of work related to the data mart. If not carefully managed, it is also the area most likely to cause delays and deadline shifts in a project timeline.

Extraction and Transformation Process
Since the source data are on the mainframe, we developed a series of SAS batch jobs to handle the data extraction and transformation process. SAS excels at handling a range of data formats with relative ease. It consolidates relational database tables with VSAM or flat files into the SAS formats seamlessly. However, even with the power of SAS, this process did not go smoothly and took more time than originally estimated.

During this task, the business and technical team met repeatedly. The business team, who will be the future users of the system, helped the technical team to understand the data. The technical team then began work on the routine sub-tasks like data validation, data scrubbing, data integration, and data derivation. We successfully completed the process -- after three attempts; the work had to be repeated because of iterative solutions to the following problems:
- Data quality and consistency
- Lack of operational data documentation
- Missing data (Tapes missing and/or stored off-site)
- Changes to the analytical/reporting requirements

Dimensional Modeling
Like all other OLAP databases, IDRS database is derived from operational data, which are transactional based. The existing operational systems are able to handle simple, specific questions, e.g., “What was this patient prescribed after a positive test for e.coli?” but cannot answer more complex questions. The main purpose of the IDRS is analytical, e.g., “Trend the last 5 years of cultures positive for pseudomonas that were resistant to Ampicillin.”

Most IDRS requests will require a large volume of data manipulation to answer. We needed to design a database structure that responds better to these types of complex request. Our data mart is based on the Dimension Model, or “Star Schema,” a common design in the OLAP world.

The basic architecture of the dimension model, as its name suggested, looks like a star (see diagram below). It has one large central table (Fact table) and a set of smaller attendant tables connected in a radial pattern around the central table.

Fact Table
The Fact table is the central table. It is highly normalized. It contains the most important data, from a business standpoint, and is the most frequently used table in the system. If designed well, a fact table alone can take care of many kinds of queries without joining to other dimension tables.

Each record of the fact table carries a unit of data that can be classified into two groups:
1. Key variables – These are keys (sometimes variables are combined to form the composite key) for the fact table and other dimension tables. In general they’re in short form, for example, TEST_NUM instead of TEST_NAME.
2. Grain variables – Non-key data

![Diagram of Dimensional Model (Star Schema)](image)

The Fact table of our data mart carries laboratory order test data. Each record captures the clinical and patient information at the date/time of the laboratory test was ordered. Here is part of the variable list:

1. MRN (key to Patient Dimension)
2. Provider_id (key)
3. Order_fac_id (key)
4. Order_dept_id
5. Order_date (key to Date Dimension)
6. Order_time (key)
7. Ordered_test_cd
8. Patient_age_in_mon
9. Patient_sex
10. Patient_status_cd
11. Memberonth
12. ADT_case_no (key)
13. First_abx
Note that some variables are both key and grain variables. For example, the Ordered_test_cd is the key variable to the Lab Directory Dimension table but it is also the grain variable of the fact table.

**Dimension Tables**

Dimension tables store detailed information about each dimension that can be accessed by the fact table. These tables are not typically normalized. They can be linked with the fact table by key(s). Here is an example of the variable list of the Date Dimension table:

1. DateKey (key variable)
2. DayOfWeek
3. DayOfMonth
4. DayOfYear
5. YearNum
6. QuarterNum
7. MonthNum
8. HolidayFlag
9. LsstDayInMonthFlag

**Multiple Dimension Database**

From the Star Schema diagram above, with multiple dimension tables connecting to the fact table, it is clear that we are indeed creating a Multiple Dimension Database (MDDB).

**Data Partitioning**

Data Partitioning is one the database design tools used to yield better query performance. In general, data partitioning means (1) breaking up a table or (2) spreading out a table.

**Breaking Up A Table**

When data tables grow so large that affect disk storage and query performance, they must be broken up into multiple smaller tables. For example, we can split a large “ANTIBIOTIC” table into multiple small “ANTIBIOTIC,yyyy” tables by year.

The benefit of splitting a large table into multiple small tables is obvious. With the exception of entire table scan, it provides better query performance, in general. However, like everything in computing, this benefit has a price. The fast access comes at the cost of database maintenance and programming efforts. After breaking up a large table, we then need to maintain multiple tables and, therefore, we need to write more code to get the same job done. Consider the above example: We now need to read two or more small tables versus one large table in order to get cross-year data. This means more programming effort is required to answer the same question. One good way to reduce the programming effort is to write a SAS macro to generate SAS code dynamically.

**Spreading Out Tables**

Spanning table records across multiple disk packs is a common way to improve query speed in the OLTP database design. With multiple read-write heads working at the same time, data can be extracted faster. The same idea can be applied to OLAP design with a few slight changes. In our project we partitioned tables into two different hard disks. This configuration enables our queries to take the advantage of SAS MP Connect parallel process feature. Worth noting here is that the decision to partition tables requires careful evaluation. We needed to essentially forecast the patterns by which tables would be read once the users started making queries. In general, if two tables are likely to be merged often, then they should be reside on two separate hard disks.

For the IDRS, our database server was partitioned as follows:

- **1st Hard Disk** – This is storage for files that are not part of data table structure. All system files, applications including SAS and others were installed on this drive.
- **2nd Hard Disk** – This is the first drive (hard disk) for the database. It is storage for two groups of data tables, the highest- and lowest-utilization tables. We placed the Fact table and some detail-record tables here.
- **3rd Hard Disk** – This is the second drive (hard disk) for the database. It is storage for other frequently used tables, other than the Fact.

Our database configuration is intended to fully utilize the hardware capacity (dual-processor and multiple disks) by evenly distributing the data tables based on their anticipated utilization. We wanted to speed up the read process via parallel process (two disk heads read simultaneously) and avoid sequential process where possible.
Indexing

Indexing is a common way to improve query performance. However, it achieves the speed at the expense of disk space. There are different types of indexes, but the best performance comes from building an index based on a thorough understanding of the access patterns used to query the underlying table. Here are some highlights for building an index for an OLAP database:

- Take time to talk to users. Learn about how tables will be queried by the users and their expectations. In general, OLAP users don’t expect query results to come back in a couple of seconds, but they may need to routinely query the system along non-standard lines, so a dialog will give you additional information about the kinds of table indexes needed to meet the users’ needs.
- Indexing cannot help a small file. In fact, small files with an index yield even slower performance than those without. Files with less than 10,000 records, in general, are not good candidates for indexing.
- Indexing takes disk space. Depending on the number and type of indexes, it is possible that it will increase the original table size by 50% or more.
- Do NOT lift existing indexes from data source tables. Data source tables are operational data and are transactional based. Most of their table indexes are composite (multiple fields) and carry unique value and they enable queries to access detail record(s) in second. In an OLAP database, data is accessed according to a different type of query, and one that rarely requires use of space-consuming composite indexes.
- Take advantage of the Sort feature before building an index. Storing data in the right order is critical to efficient access. This will be discussed in detail below.
- Most OLAP database tables are designed for read-only access. Avoid real-time updates, if possible. If there are special situations that require real-time update, test carefully as an index could be heavy overhead on the update process.
- Performed tests are essential after building indexes; fine tune the indexes, as needed.
- Index maintenance is an on-going process. As business requirements change, so will the table access patterns and the indexes have to be changed accordingly.

Fact Table Index

Before building fact table index, we first interviewed users (clinicians and analysts) to learn their expectations from the IDRS data mart. We also researched the types of applications will be accessing IDRS. Typically we collected two types of information:

1. Business questions – These are the outputs (reports) that users need. For example: “Report the last 5 years utilization trending of a specific test at a specific location.” We then wrote the queries based on the business logic.
2. Technical questions – Typical types of queries that programmers will be using.

With this information in-hand, we then scored variables by the frequencies that they were used to access data. In SQL statements, we checked the variables inside the Where clause. We came up with a variable list that accounts for more than 90% queries:

1. Order_date
2. Order_date + Ordered_test_cd
3. Order_date + ESI_indicator
4. Order_date + Order_time + MRN + Ordered_test_cd

This confirms that most of the time the fact table will be accessed by the time variable, i.e. the Order_date. We also noticed that the Order_date variable is the first variable in virtually all access patterns. We decided not to build a simple index on OrderPdate, since one of the other three composite indexes will provide partial index power (as Order_date). It worked, but was suboptimal -- we missed a critical step: We forgot to sort the data file by the Order_date variable.

Performance Testing

Background:

- Two identical SAS data sets, one sorted by Order_date and one unsorted.
- Both data sets have identical composite indexes (Order_date + Ordered_test_cd and Order_date + ESI_indicator)

We ran the following SAS program on each data set.

```sas
Libname lab 'D:\idrs\data\lab';
       *libname lab 'E:\idrs\data\lab';
options source msglevel=i;
data _null_; 
  call symput('D20020101', 15341) ;
  call symput('D20020131', 15371) ;
run ;
proc sql _method ;
create table jan2002 as 
  select *
  from lab.labOrder 
  where order_date between &D20020101 and &D20020131;
quit ;
data jan2002 ;
set lab.labOrder 
  (where=(&D20020101<=order_date<=&D20020131));
run;
data jan2002 ;
set lab.labOrder ;
  if &D20020101<=order_date<=&D20020131 ;
run;
```

Finding:
The data set with sort provided 3x faster access (see Attachment-A for details). Depending on how data will be accessed, sorting data set in advance could yield even better performance. In SAS programming, sorting data set
is the first step for most procedures. Sorting our data set in advance by the most frequently accessed variable, Order_date, will definitely save processing time. Based on this testing, we changed our plan for building the index for the fact table:

1. Sort the data set by the Order_date variable
2. Build 3 composite indexes
   - Order_date + Ordered_test_cd
   - Order_date + ESI_indicator
   - Order_data + Order_time + MRN + Ordered_test_cd

MP Connect

IDRS data mart was built in the architecture for parallel processing. It enables SAS programs to use the MP Connect parallel processing feature, if possible. The following example demonstrates how a SAS program fires up two other SAS sessions at the same time to execute two Sort procedures concurrently and then brings control back at the end.

```sas
options autosignon=yes comamid=tcp sascmd="SAS";
rsubmit process=task1 wait=no;
  libname lab 'e:\idrs\data\lab';
  proc sort data=lab.labOrder out=labOrder;
    by order_date order_time;
  run;
  endrsubmit;

rsubmit process=task2 wait=no;
  libname rx 'd:\idrs\data\pims';
  proc sort data=rx.outpat out=rxOuptpat;
    by pos_date ;
  run;
  endrsubmit;
waitfor task1 task2;
data local;
  a=5;
run;
signoff task1;
signoff task2;
```

CONCLUSION

There were a number of key findings from our work, but the most important learning is that we can indeed build an OLAP database and MDDB with the basic SAS package. Other recommendations based on this work include advising developers to:

- Spend more time with users to understand how they are going to use the OLAP database and their expectations. This is the foundation of good project specifications.
- Allocate as much time as possible for ETL, especially the Transformation process – there’s always a surprise or two.
- Design MDDB based on business requirements. Build multiple MDDBs if needed.
- Remember that fine-tuning query performance as an ongoing task – even if you get it right at the first time, your structure may become obsolete as the users’ needs change.
- Ensure that there are adequate resources for maintenance, which could be considerable.

ACKNOWLEDGMENTS

1. Roger Baxter, MD, The Permanente Medical Group: Oakland, CA
2. Dave Schweppe, director of Information Consulting Services, Operations Support Services, The Permanente Medical Group, Oakland, CA

REFERENCES

1. Strategic Data Warehousing Principles Using SAS Software, by Peter R. Welbrock
2. The Data Warehouse Toolkit, by Ralph Kimball
3. SAS papers:
   - A practical Approach to Solving Performance Problems with the SAS System, by Tony Brown
   - Multiprocessing With Version 8 of the SAS System, by Cheryl Doninger

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Berwick Chan
Berwick Consulting, Inc
760 Crestview Drive
Pinole, CA 94564
510.724.2928
berwickchan@comcast.net

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.
<table>
<thead>
<tr>
<th>Attachment-A</th>
<th>SAS log (same data set with no sort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9    proc sql _method ;</td>
<td>9    proc sql _method ;</td>
</tr>
<tr>
<td>10   create table jan2002 as</td>
<td>10   create table jan2002 as</td>
</tr>
<tr>
<td>11     select *</td>
<td>11     select *</td>
</tr>
<tr>
<td>12     from lab.labOrder</td>
<td>12     from lab.labOrder</td>
</tr>
<tr>
<td>13     where order_date between &amp;D20020101 and &amp;D20020131;</td>
<td>13     where order_date between &amp;D20020101 and &amp;D20020131;</td>
</tr>
<tr>
<td>NOTE: SQL execution methods chosen are:</td>
<td>NOTE: SQL execution methods chosen are:</td>
</tr>
<tr>
<td>SqxcrtA</td>
<td>SqxcrtA</td>
</tr>
<tr>
<td>INFO: Index labOrder2 selected for WHERE clause optimization.</td>
<td>sqxsrc( LAB.LABORDER )</td>
</tr>
<tr>
<td>NOTE: Table WORK.JAN2002 created, with 107537 rows and 26 columns.</td>
<td>INFO: Index labOrder2 selected for WHERE clause optimization.</td>
</tr>
<tr>
<td>14     quit ;</td>
<td>NOTE: Table WORK.JAN2002 created, with 107537 rows and 26 columns.</td>
</tr>
<tr>
<td>NOTE: PROCEDURE SQL used:</td>
<td>14     quit ;</td>
</tr>
<tr>
<td>real time 0.59 seconds</td>
<td>NOTE: PROCEDURE SQL used:</td>
</tr>
<tr>
<td>cpu time 0.17 seconds</td>
<td>real time 1.17 seconds</td>
</tr>
<tr>
<td>15</td>
<td>cpu time 1.12 seconds</td>
</tr>
<tr>
<td>16     data jan2002 ;</td>
<td>16     data jan2002 ;</td>
</tr>
<tr>
<td>17     set lab.labOrder</td>
<td>17     set lab.labOrder</td>
</tr>
<tr>
<td>(where=((&amp;D20020101&lt;=order_date&lt;=&amp;D20020131)) );</td>
<td>(where=((&amp;D20020101&lt;=order_date&lt;=&amp;D20020131)) );</td>
</tr>
<tr>
<td>INFO: Index labOrder2 selected for WHERE clause optimization.</td>
<td>INFO: Index labOrder2 selected for WHERE clause optimization.</td>
</tr>
<tr>
<td>18     run;</td>
<td>18     run;</td>
</tr>
<tr>
<td>NOTE: There were 107537 observations read from the dataset LAB.LABORDER.</td>
<td>NOTE: There were 107537 observations read from the dataset LAB.LABORDER.</td>
</tr>
<tr>
<td>WHERE ((order_date)&gt;=15341 and order_date&lt;=15371));</td>
<td>WHERE ((order_date)&gt;=15341 and order_date&lt;=15371));</td>
</tr>
<tr>
<td>NOTE: The data set WORK.JAN2002 has 107537 observations and 26 variables.</td>
<td>NOTE: The data set WORK.JAN2002 has 107537 observations and 26 variables.</td>
</tr>
<tr>
<td>NOTE: DATA statement used:</td>
<td>NOTE: DATA statement used:</td>
</tr>
<tr>
<td>real time 0.68 seconds</td>
<td>real time 0.89 seconds</td>
</tr>
<tr>
<td>cpu time 0.28 seconds</td>
<td>cpu time 0.87 seconds</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>20     data jan2002 ;</td>
<td>20     data jan2002 ;</td>
</tr>
<tr>
<td>21     set lab.labOrder</td>
<td>21     set lab.labOrder</td>
</tr>
<tr>
<td>;</td>
<td>;</td>
</tr>
<tr>
<td>22     if &amp;D20020101&lt;=order_date&lt;=&amp;D20020131 ;</td>
<td>22     if &amp;D20020101&lt;=order_date&lt;=&amp;D20020131 ;</td>
</tr>
<tr>
<td>23     run;</td>
<td>23     run;</td>
</tr>
<tr>
<td>NOTE: There were 6317616 observations read from the dataset LAB.LABORDER.</td>
<td>NOTE: There were 6317616 observations read from the dataset LAB.LABORDER.</td>
</tr>
<tr>
<td>NOTE: The data set WORK.JAN2002 has 107537 observations and 26 variables.</td>
<td>NOTE: The data set WORK.JAN2002 has 107537 observations and 26 variables.</td>
</tr>
<tr>
<td>NOTE: DATA statement used:</td>
<td>NOTE: DATA statement used:</td>
</tr>
<tr>
<td>real time 7.56 seconds</td>
<td>real time 2.50 seconds</td>
</tr>
<tr>
<td>cpu time 1.84 seconds</td>
<td>cpu time 2.01 seconds</td>
</tr>
</tbody>
</table>