Customizing Clinical Data Warehouse for Business Intelligence
An analyst’s view

Jianming He  Solucient LLC  Berkeley Heights  NJ 07922
Haibin Shu  Barr Research  Bala Cynwyd  PA 19004
Sonia Pulgar  Solucient LLC  Berkeley Heights  NJ 07922

ABSTRACT

Many companies choose SAS/BASE, SAS/ACCESS etc as the analytical tool, and data warehousing software other than SAS for data warehouse management. This paper discusses the analyst perspectives in such setting to deal with data loading, extraction and data cleaning. It shows basic methodology in data extraction and data management for achieving the business goals and making strategic business decisions efficiently. SAS users need to work with database administrators effectively to make the data warehouse more users friendly.

KEYWORD

Data warehousing, data cleaning, data query, data loading

INTRODUCTION

Data warehousing involves four processes: data extraction, data transformation/cleansing, data loading and arguably, most important, data mining. The primary function of business analysts is mining the data for business intelligence. To achieve the business goals and make strategic business decisions, data warehousing architects and administrators need to work closely with analysts or end-users of the data to make changes in the structure, the data loading schedules, etc. Effective communication between analysts and database administrators can make the databases more efficient and easier to use.

Often, clinical data warehouses can achieve massive size. To obtain business or clinical insight from a diversified data warehouse in a timely manner, what is required is a user-friendly interface, a good understanding of medical data and clinical practices and practical knowledge about common data warehousing techniques. A good option for storage of vast amounts of clinical information is a data warehouse that can be accessed using SAS programming environment. In fact, many companies like to use SAS/BASE plus the SAS statistical package to perform ad hoc analysis, using other OLAP (On-Line Analytical Processing) software (like Microstrategy™, Datastage™ Powercenter™, Transformation Server™, etc.) as data warehousing tools. Analysts can use SAS to access the data to glean business or clinical insights from the data. Although the set up and maintenance of the data warehouses are not the primary duties of analysts, communication between analysts and database architects and database administrators can lead to improvements in run times, less need for error correction, etc.

WORKING WITH CLINICAL DATA WAREHOUSE

A clinical data warehouse can contain heterogeneous data sources. For instance, some data may reside in Informix, other data could be housed in Oracle, or some other type of relational database. Also, some data may not be loaded to save space and time, but may be stored offline.

No matter how seemingly diverse the data can appear, they mostly come from HCFA-1500, UB-92 (uniform billing), UHDDS (uniform Hospital Discharge Data set), Health Level Seven (HL7), HSCRC (Health Services Cost Review Commission) etc. In their raw form, they share the basic elements:

- Patient demographical information: can include information such as name, address, birth date, sex, marital status, race, etc;
- Payer information: insurance company, insurance plan, type of payer (primary or secondary), etc;
- Diagnosis information: diagnosis code, type of diagnosis (principal or secondary);
- Procedure information: procedure code, type of procedure (principal or secondary);
- Medication information: service date, drug charges, dosage, administration, route, unit etc;
• Laboratory test information: test name, literal test value, numeric test value, reference range, test ordering time, etc;
• Facilities: bed size, teaching status, region, urban/rural indicator, etc;

In most clinical database systems, there are several supporting relational databases, which are the foundation of ad-hoc queries, routine reports, as well as online client analytical applications.

**SETTING UP A SAS ENVIRONMENT FOR CLINICAL RESEARCH**

It is convenient to set up a SAS environment for clinical data. Although many reference tables can reside in relational databases, it can be inefficient to use them for translation of codified variables and may result in performing unnecessary joins. One neat solution is to create a format library.

Consider the example of a business question posed to explore the daily lab testing volume by several distinct lab sites. Suppose the requisition table which has requisition_id and lab_id exists in one table. There is a separate table that contains data concerning the lab id and the lab name. In this example, the best way to answer the business question would be to count the lab test by lab id and use the lab_name to format the output so it is useable by most audiences.

```sas
proc sql;
    create table labvolume as
        select distinct lab_id, count(distinct requisition_id) as lab_ct from requisition group by lab_id;
    quit;

proc print data= labvolume; format lab_id $labname.;
run;
```

This is more efficient than joining the requisition table with the lab table and then running a frequency count. When you have a format library, it precludes translation of the numeric fields each time you use them for different queries or studies.

Setting up a format library to be shared by all users can eliminate the need for an additional translation step. It is easy to convert reference tables from a relational database into a SAS format. Take the lab table as an example.

```sas
libname oracle oracle user=*** password=*** path='***' ;

data lab;
    set oracle.lab;
        start=lab_id;
        label=lab_name;
        fmtname='$lab';
    keep fmtname start label;

proc sort data =lab out=fmt nodupkey; by start;

proc format cntlin=fmt;
run;
```

The code referenced above will convert a reference table into a useable SAS format. When you try to save your format in your format library, you may code as follows.

```sas
libname lib '/yourformatdirectory';

proc format cntlout=outdata lib=lib;
run;
```

The format library is ready for use. To ensure that the reference formats are always loaded, make this libname statement part of the autoexec file that runs each time a new SAS session is initiated.
Another way to improve the efficiency and standardization of analysis by data analysts is to create a macro library for use by all.

The macro library can contain code to standardize the output of certain units or formats for particular fields. One advantage of housing such programs as macro programs is that it will be easy to update and maintain in the event that the codes change in the future. It can also be used to standardize the raw data that may come in slightly different forms. For instance, you may want to convert all different forms of the word “street” such as “street”, “st.” or “St.”, into one standard word.

Macros may contain standard algorithms for data transformation and data validation. Most clinical data warehouses store diagnosis codes and procedure codes without decimal points that signify boundaries of the code. However, most end-users of clinical data are accustomed to seeing the decimal points within the code. To remedy this situation, a macro program can be created to place the decimal points back into a diagnosis codes and procedure codes when the report is finalized. Therefore, we can create a macro called as %DIAGNOSISCODE to delete all diagnosis code not start with letter E with a length greater than 5, and to add a decimal point if it is missing.

Again, the way to ensure that all users have access to the macro library is to make it part of the autoexec file. You may put following code in your autoexec.sas file.

```sas
filename macdir '/yourmacdirectory';
options mautosource sasautos=macdir;
```

We may also use %include to retrieve macro saved in other places. This option needs to explicitly specify the location of that macro in the code.

**Dynamically Adjust Reference Table**

To be responsive to the analysts and end-users of the data, colleagues charged with loading, maintenance and enhancements to the database may need to modify the database to make it more conducive for data mining. Frequently one search argument is determined by several different codes, which can keep changing. In the clinical lab setting, local labs might change the local test code frequently, or use different abbreviations for the test name depending on the individuals keying in the data. Or, we may need to search for specific drugs using different fields such as generic name, brand name, etc. Such wild card searches basically need to be researched by someone with strong clinical background. In some environments, this is referred to as analytical code determination.

In the healthcare data warehouse, analytical code determination basically involves in ICD-9-CM (International Classification of Diseases) Diagnosis and Procedure codes, CPT (Current Procedural Terminology) code, HCPCS (HCFA Common Procedure Coding System), Drug code\(^1\), test code\(^2\), etc. Most ad hoc questions are related to the analytical code determination. Generally, it is difficult to query directly against the data warehouse using wild card search. An efficient strategy is to create a new cross-reference table to help analysts to perform data quickly. Such table is also an importance base for the OLAP tool. Once the business model is set up, system developers can efficiently create and distribute reports by linking it with the cross-reference tables.

For instance, we have a reference drug table which contains generic_xref, NDC code, CTC code, generic_name and brand_name. We can map those codes into drug name.

<table>
<thead>
<tr>
<th>Generic_xref</th>
<th>NDC</th>
<th>Brand_name</th>
<th>Generic_name</th>
<th>Therapeutic class</th>
<th>CTC, etc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard drug name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For the laboratory test table we can map as follows.

\(^1\) including generic_xref, NDC (National Drug Code), generic name, product name, CTC code, therapeutic class etc

\(^2\) test name and LOINC (Logical Observation Identifiers Names and Codes) code, CPT 4 codes etc
It is better to keep those mapped cross-reference tables in the relational databases. Since those tables are derived from the raw data, they need to be updated regularly.

With these created cross-reference tables, clinical analysts now do not need to bother medical expert each time to QA their wild card search. We may create an index on the standard code, which will improve the performance of query significantly.

**INDEXED DATA EXTRACTION FOR DATA MINING**

1) **Extract data from relational databases**

For Clinical data mining setting, except some data cuts, most data extraction is indexed. That is, you are searching your data with specification of diagnosis, medication, lab, time etc. To perform the data query correctly, it is extremely important to use the appropriate methodology.

Through SAS/ACCESS provides tremendously tools to query database, you need to find your query methodology from relational database like DBA handbook. Many books provide tips of how to tune up your SQL query from a relational database.

Every data query needs to be analyzed before sent to the database. It is a frequent complaint that a data query is running very slowly. In most cases, data should be extractable from the database within a reasonable time frame. Most queries of well-designed databases probably have some error or inefficiencies if there is no result after a day or more. To assess the speed and status of your query, you may go the temporary working space you created to check the query speed. Generally, you will see some indication of the data extraction performance in the working space.

Mostly, we are interested in certain diagnosis groups, certain medication or certain laboratory information within specific time frame, etc.

First, you search against one specific table using searching argument like ‘OR’, ‘IN’, or wild card. Save the searching result as a new table. Secondly, you created index on the new table in the relational database, remember to keep it as a SAS format dataset. Make sure all date field types match with other tables you will be joining. Next, update the statistics on this table. Finally, join this new table with other tables. If there are more than two arguments you are searching against or if your query involves the use of an ‘OR’ operator, you may need create a new table with these arguments.

For instance, suppose we have a clinical database with 3 tables,

1) patients (service_id, sex, age, race, etc, indexed on service_id only)
2) service (service_id, date_of_service, account_name, etc., indexed on service_id only)
3) diagnosis (service_id, diagnosis_code, diagnosis_sequence, etc. indexed on service_id only)

Consider an example in which we are tasked with pulling data from all three tables for cancer patients. We may use ICD-9-CM codes from 140 to 239 to identify those patients.

1) **Base approach**

```
select a.sex, a.age, a.race, 
      b.date_of_service, b.account_name, 
      c.diagnosis_code, c.diagnosis_sequence 
from    patients a,    service b , diagnosis c
```
where  a.service_id=b.service_id  
    and a.service_id=c.service_id  
    and diagnosis_code between  '140' and '239';

Diagnosis code is a field of character, mostly it does not own an index. The above query can help to get the data. However, it might be very slow. Since diagnosis code is not indexed, such query involves in a full table scan.

2)  Better approach.

Firstly  create a temporary table in database call  tempdx which contains all distinct service_id with cancer diagnosis.

    create  table tempdx as 
    select  distinct service_id 
    from    diagnosis 
    where   diagnosis_code between  '140' and '239';

Secondly  create unique index on table tempdx and analyze the table.

    create index tempdx1 on tempdx(service_id); 
    Analyze table tempdx compute statistics ;

The step of analyzing the table is quite important. Otherwise, your created index won’t be activated. Thirdly join it with other tables.

    select a.sex, a.age, a.race, 
        b.date_of_service, b.account_name, 
        c.diagnosis_code, c.diagnosis_sequence
    from    patients a,    service b , diagnosis c, tempdx d 
    where       d.service_id=a.service_id 
    and d.service_id=b.service_id  
    and d.service_id=c.service_id;

The second approach makes every query indexed. Though it looks involving unnecessary steps, actually it takes much less time than the first approach.

It is a common mistake to do full table scans when extracting data. A full table scan could happen because:  
1). Multiple tables are joined as part of a search for an argument that has not been indexed.  
2). Created table without matching the database field type.  
3). No index created for the new table.  
4). The new table has not been updated.

If you have submitted your query to the database and nothing has happened for 30 minutes, further investigation may be required because either the query may not be structured well or the database itself may be experiencing some problems.

2) Retrieve data from plain text file.

If the data is offline, stored into tape, the main challenge of retrieving data from plain text files is space. Most frequently, the text files should be broken into smaller pieces.

If we are in UNIX environment, users need to be familiar with commands like ‘split’, ‘gzip’, ‘gunzip’, etc to clear space to accommodate reading in new data. Experienced analysts may use UNIX shell scripts to deal the data breaking, reading and consolidating problems.

DATA CLEANING

As we have mentioned, data quality is always a big concern of the clinical data warehouse. Clerks, primary physicians, consultants, nurses, therapists, machines, computer systems: any one of them might make a mistake. It may be a simple data entry mistake or an error in calculation,. In order to have results that are readily interpretable for business purposes, analysts need do data cleaning in most cases.
Data cleaning can be costly and risky. Analysts need to be equipped with a good understanding of the data and sound medical knowledge and patience. It might take a long time to investigate certain quality assurance problems. It can be risky because any alteration of the original data might result in a different conclusion. Records with “abnormal” values could be exactly what is interesting. That does not suggest that we should not do some data cleaning.

Mostly, database administrators will report data check broken out by a several categories as a reference. From an analyst’s view, these snapshots of the data are valuable in capturing high-level detail about the warehouse.

A costly way is to create an error code dictionary. This entails setting up a system of comparisons and is a data integrity check. For instance, this process may check foreign key and primary key constraints, check data bias and completeness. If the data value is out of the range, the system may create an abnormal flag to alert users. In this case, objective criteria used to judge values needs to be established and can be done in a number of ways. Benchmarks need to be used and may involve research on trending, inter-lab or inter hospital comparisons, comparisons among insurance companies, states etc.

If the data field is numeric, we may also use the following approach to identify outliers.

1) Define 
   \[ IQR = Q_3 - Q_1 \ \text{(75 percentile minus 25 percentile of the sample)}; \]
2) Define upper bound \[ = Q_3 + 1.5 \times IQR \]
3) Define lower bound \[ = Q_1 - 1.5 \times IQR \]
4) Any data that falls out of the range is deemed an outlier.

When the population is Guassian, the percentile method is equivalent to mean+(-) 2.7std. The percentiles can be calculated quite simply using the PROC UNIVARIATE procedure in SAS.

For categorical variables, we may check the percentage of each category against total, using the same strategy of the above.

Based on the results of these series of comparisons, the next step is to compile a code dictionary. This can be saved in the data warehouse as a reference guide for use in data mining.

CONCLUSION

Although the optimal analytical and data warehousing environment might be in all SAS, most companies choose SAS plus other data warehousing tool for their business intelligence. Data analyst’s role in such setting is never passive. Since many data warehouse supporting personnel may not know much about SAS, SAS programmer /analysts need work with them very closely to help them understanding business need. Analysts can learn the IT solution in the data warehouse as well. Both data warehouse administrator and business analysts need to cooperate to customize the data warehouse to be simple, effective and efficient.

REFERENCE


Judy Loren, SAS Connection to DB2: Tools and Techniques, Proceeding of the Twenty_first Annual SUGI conference, 21,498-507

Thiru Satchi, An SQL List Macro to Retrive Data from Large SAS/DB2 Databases, www2.sas.com/proceedings/sugi24/Posters/p220-24.pdf

SAS institute Inc.(199), SAS Online Document ©, Cary, NC: SAS institute
ACKNOWLEDGMENTS

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.

CONTACT INFORMATION

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

Jianming He
Connell Corporate Center, One Connell Drive, Suite 1000
Berkeley Heights, NJ 07922
(734) 669-7835 (O)
(908) 665-1764 (F)
jhe@solucient.com

Haibin Shu
Barr Research, One Bala Plaza
Suite 324, Bala Cynwyd, PA 19004
(610) 617-4449 (O)
glandpe@aol.com

Sonia Pulgar
Connell Corporate Center, One Connell Drive, Suite 1000
Berkeley Heights, NJ 07922
(734) 669-7836 (O)
spulgar@solucient.com

APPENDIX: MACRO FOR INDEXED DATA QUERY

Sybase version

libname mydblib sybase database=tempdb server=&server user=&user password=&password;
%macro pull (indata, key, master, outdata, pullvar);
proc sort data=&indata (keep=&key where=(&key~='.'))
nodupkey out=mydblib.temp (bulkcopy=yes dbtype=(&key='data type')) by &key ;
proc sql;
connect to sybase (server= &server  database=&server  user=&user password=&pwd);
   execute (create index ind on tempdb..temp (&key)) by sybase;
   execute( update statistics table temp) by sybase;
   create table &outdata  as select *  from connection to sybase
      (select &pullvar   from &master  b, tempdb..temp a (index ind)
       where a.&key=b.&key);
   execute (drop table tempdb..temp ) ; by sybase;
quit;
%mend;


Informix Version

Libname mydblib informix database=&database server=&server;

%macro pull (indata, key, master, outdata, pullvar);
proc sort data=&indata(keep=&key where=(&key=<.))
   nodupkey out=mydblib.temp (dbtype=(&key='data type'));by &key ;
proc sql;
   connect to informix (database=&temp);
   execute (create index ind on temp (&key)) by informix;
   execute (update statistics for table temp) by informix;
   create table &outdata as select * from connection to informix
      (select &pullvar from masterdatabase:mastertable b,
       temp a
      where a.&key=b.&key);
   quit;
proc delete data=mydblib.temp;
%mend;

Oracle Version

Libname mydblib oracle user=&user password=&password path="&path"
  schema=&schema dbindex=yes;

%macro pull (indata, key, master, outdata, pullvar);
proc sort data=&indata(keep=&key where=(&key=<.))
   nodupkey out=mydblib.temp (bulkload=yes dbtype=(&key='data type'));by &key ;
proc sql;
   connect to oracle (database=&temp user=&user password=&pwd path=&path);
   execute (create index ind on temp (&key)) by oracle;
   execute (analyze table temp compute statistics) by oracle;
   create table &outdata as select * from connection to oracle
      (select &pullvar from &master b, mydblib.temp a
      where a.&key=b.&key);
   quit;
proc delete data=mydblib.temp;
%mend;