

An Automated Tool for Data Change Implementation

Alexander Pakalniskis, M.S., Cedars-Sinai Medical Center, Los Angeles, CA

Alein T. Chun, Ph.D., Cedars-Sinai Medical Center, Los Angeles, CA

Nilesh Bajania, B.S., Cedars-Sinai Medical Center, Los Angeles, CA

Peixin Wang, M.S., Cedars-Sinai Medical Center, Los Angeles, CA

ABSTRACT

Created by the Data Quality Management Unit (DQMU) of the Cedars-Sinai Medical Center (CSMC) Resource and Outcomes Management (ROM) Department, the Data Change Management System (DCMS) is an automated tool supporting the controlled introduction of data change in Structured Query Language (SQL) environments. By flagging all programs and owners affected by any given data change, the tool helps ensure ongoing report quality, reduces the time and effort required for change implementation, and provides transparency for project management. The tool may be used in any programming environment utilizing embedded SQL queries, including SAS, and is especially valuable in complex database migrations.

INTRODUCTION

CSMC is located in Los Angeles, California and is one of the largest private academic hospitals in the Western United States. With nearly 1,000 beds, this medical facility admits more than 50,000 inpatients and 150,000 outpatients annually. Its work force is comprised of about 10,000 employees, 2,000 physicians on the voluntary staff and 200 on the clinical faculty.

Two departments share the principal responsibility for data processing and clinical reporting at CSMC. EIS (Enterprise Information Systems) supports the CSMC-wide Data Warehouse (CSDW), an Oracle database, while ROM provides *ad hoc*, routine and scheduled production clinical reporting, as well as support for data quality management issues. Most ROM reports consist of SAS-embedded SQL queries to the CSDW.

EIS was mandated by hospital leadership to migrate all financial, business, and clinical applications to an integrated information system housed in the CSDW. This resulted in a rebuilding of all principal data feeds into the enterprise CSDW, and required a variety of changes to the data themselves as well as the table structure in the CSDW. Dubbed "CS-Link," this multi-year, multi-phase project is expected to include addition/restructuring/removal of tables, change of variable definitions, change/addition/removal of variable names, and change of variable formats.

Because of the unprecedented level of data change, we recognized that an automated tool would provide the most efficient means to identify all programs affected by impending or previously implemented changes in CSDW table structure or definition. This was the primary impetus for the creation of DCMS.

Multiple additional benefits have been realized as a result of the implementation of DCMS. An automated structured inventory of production code and associated programmer has been put in place. DCMS is now used to assess the scope of any impending or recent change in CSDW at program, associated programmer, or summary level. An email notification is automatically sent to programmers identifying all their programs affected by data change and providing useful information regarding the change. Management now has the ability to evaluate data change impact and status at individual programmer, departmental, and project levels. Created to support a database migration project, DCMS now provides ongoing support for data quality control and change management.

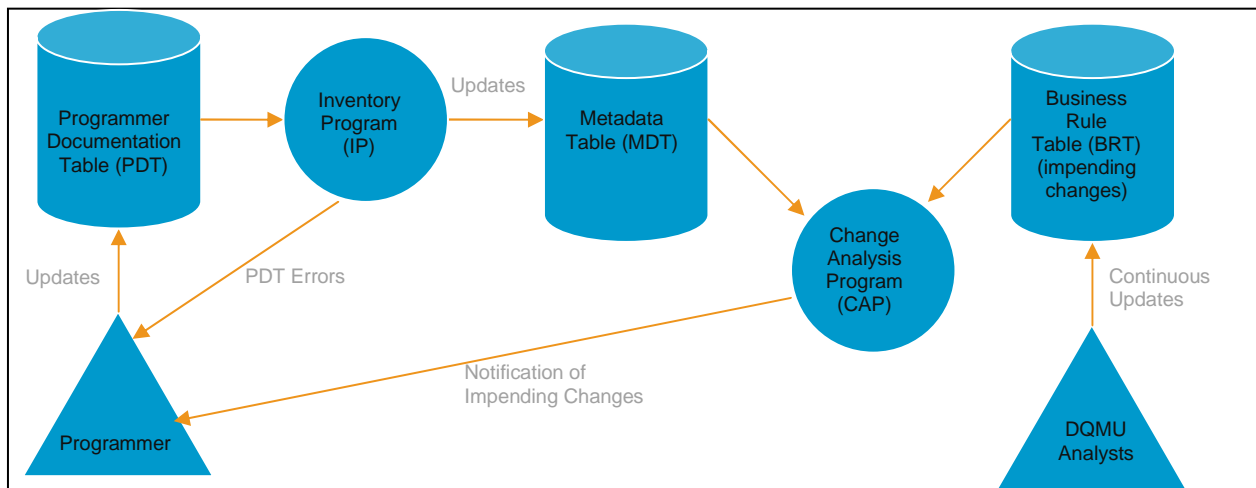


Figure 1. DCMS Iterative Process

DCMS

Working seamlessly with the most recent Oracle enterprise integrated grid computing technology, DCMS is currently housed on 32 bit Linux servers. It consists of two SAS programs, the Inventory Program (IP) and the Change Analysis Program (CAP), and utilizes three tables: the Programmer Documentation Table (PDT), the Metadata Table (MDT), and the Business Rule Table (BRT). The iterative DCMS process is depicted in Figure 1.

DCMS INVENTORY PROGRAM

The first DCMS program is the Inventory Program (IP). The function of the IP is to maintain a complete and accurate inventory of all SAS-embedded SQL programs accessing the CSDW. This inventory is stored in the Metadata Table (MDT). Figure 2 shows the MDT variables, their origin, and an example.

<u>Variable</u>	<u>Example</u>	<u>Origin</u>
Programmer_Name	Alex Pakalniskis	PDT
Library_Name	ROMR2237	PDT
Program_Folder	\\cshsmedaff\RO_Mgmt\shared\sas_code\other_from_Kdrive\ROMR2237	PDT
Program_Name	ROMR2237.sas	PDT
Program_Description	Grouping Table for Classic DRG & APR groupings	PDT
Database_Name	CSDW	IP, ROMR2237.sas
Table_Name	VISIT	IP, ROMR2237.sas
Field_Name	ACCT_NO	IP, ROMR2237.sas
Run_Date	29may2009	IP, ROMR2237.sas
Program_Errors		IP, ROMR2237.sas

Figure 2. DCMS MDT Variables

The IP utilizes a Programmer Documentation Table (PDT) to update the Metadata Table (MDT). The PDT is maintained by programmers and the table contents are shown in Figure 3.

<u>Variable</u>	<u>Example</u>
Programmer_Name	Alex Pakalniskis
Program_ID	ROMR2237
Program_Folder	\\cshsmedaff\RO_Mgmt\shared\sas_code\other_from_Kdrive\ROMR2237
Program_Name	ROMR2237.sas
Program_Description	Grouping Table for Classic DRG & APR groupings

Figure 3. DCMS PDT Variables

The first execution of the IP populates the MDT with relevant information for all SAS-embedded SQL programs for all programmers. Subsequent executions of the IP, which are performed at programmer level, reinitialize the MDT for that programmer.

The first step of the inventory program is the validation of PDT entries. For each Program_ID in the PDT, the IP constructs a network address using the Program_Folder/Program_Name variables. It then attempts to open and read the contents of the file found at that derived network drive address.

If the derived network drive address is found to be invalid or the file does not exist, the programmer is alerted of this issue for corrective action. Also, if the file at the derived network drive address contains no SQL code (i.e., it is purely SAS code), the programmer is alerted of this anomaly. Both notifications are automatically sent using Microsoft Outlook email. This automated feature helps ensure the currency of program and associated programmer documentation and with it the ongoing usefulness of DCMS.

The second step of the IP is to gather all relevant information for each program being processed. First it locates and extracts any SQL code from the SAS program found at the derived network address (Program_Folder\Program_Name). Then it parses the SQL code to identify all queried databases, database table names, and database table field names.

The embedded SQL is also analyzed to identify any violation of ISO/ANSI SQL coding standards, governmental (Health Insurance Portability & Accountability Act of 1996) regulations, and/or corporate coding standards. An example of an SQL coding standard violation is the referring to an unprefix variable when that variable is housed in more than one multiply joined table of an SQL query, as shown in Figure 4. An example of a governmental (and CSMC) coding regulation/standard violation is the hard-coding of user and password parameters in the "CONNECT TO ORACLE" clause of an SQL query. If violations of this type are found, an alert concerning the situation is generated. These alerts, triggered by the inventory program, are automatically delivered in the form of Microsoft Outlook emails, one email for each program with which the programmer is associated.

```

...
Select  v.VISIT_ID,
        v.ADMIT_DATE,
        DISCHARGE_NURSING_STATION /* although not an error, it is unclear from which table this variable is mined */
From    VISIT                    v,
        VISIT_SUMMARY            s
...

```

Figure 4. Example of ISO/ANSI SQL and CSMC Coding Standard Violation

The third step of the inventory program is to store the extracted information in the MDT. Five of the ten MDT variables (Programmer_Name, Program_Folder, Program_Name, Program_ID, and Program_Description) come directly from the PDT. The remaining five variables (Database_Name, Table_Name, Field_Name, Run_Date, and Program_Errors) are created by the parsing procedure of the IP.

DCMS CHANGE ANALYSIS PROGRAM

The second DCMS program is the Change Analysis Program (CAP). The purpose of CAP is to notify all programmers when and if their programs will be affected by impending data changes. The CAP is executed as needed and utilizes the MDT as well as the BRT to generate notifications of impending change.

The BRT is maintained by DQMU analysts and contains a record for each CSDW field undergoing change. BRT records, one for each field undergoing change, contain the information shown in Figure 5.

<u>Variable</u>	<u>Example</u>
Field_Name	Payor_Priority
Table_Name	Visit_Payor
Data_Format_Changed_Flag	Yes
Field_Name_Changed_Flag	Yes
CS_Link_Field_Name	Payor_Priority
CS_Link_Table_Name	Visit_Coverage
Notes	Link Visit_Coverage table to Coverage table using Visit_Id. Payor_Priority is in Coverage table.
Analyst_Initials	NB

Figure 5. DCMS BRT Variables

The Notes field of BRT records contains free form text describing the impending change.

The DCMS CAP compares entries in the BRT with entries in the MDT and identifies all programs listed on the MDT which would be impacted by impending data changes listed on the BRT. Associated programmers of affected programs are notified of the impending change and relevant information regarding the change is provided. Notifications are automatically generated utilizing Microsoft Outlook email.

The DCMS CAP also performs multi-level change impact analyses on an as-needed basis. One such analysis determines, for each programmer, the number of programs requiring her/his attention due to impending CSDW table or field change. Because it identifies the impact of CSDW changes on each individual programmer, the CAP has proven to be a useful resource and project management tool. Other analyses are performed by CAP as needed, assisting in the evaluation of priorities, scheduling of migrations and assignment of resources to ensure ongoing data quality.

PARSING METHODOLOGY OF THE DCMS INVENTORY PROGRAM

Traditionally, SAS is viewed as a good tool for data analysis and reporting. However, it is not limited to just that. There are novel uses of SAS, such as the ability to parse other SAS programs with string manipulation.

Perhaps the most complex processing in the DCMS IP is this unconventional SAS code which locates the embedded SQL code and extracts from it the information required by the MDT. This code warrants a closer scrutiny.

The SAS program found at the network address indicated on the PDT is read into a SAS data set as a text file. Each line of the SAS program to be analyzed is a separate record in this SAS data set.

Once the SAS data set is completed, the IP removes all SAS comments of the form “/* COMMENTS */” from this data set. These types of comments can span multiple lines of code in the original program, and would appear as multiple records in the SAS data set. Any record containing non-SQL code preceding the first occurrence of the token “SQL” is removed from the SAS data set. Any blank lines are removed. Two new variables are created for each line of the SAS data set. One variable tags each line of code to a separate “SQL” token (this creates an “SQL block”). The other variable indicates the line number within that SQL block. In effect, the first variable counts the number of SQL blocks, while the second variable indicates how many lines are associated with each of these SQL blocks.

All blanks are removed from the beginning of each line of code (the line is left justified).

Multiple consecutive spaces are replaced with a single space on each line. Each occurrence of a left parenthesis immediately followed by one or more spaces is replaced with a left parenthesis. Any line containing an embedded semicolon is split into two lines, with the semicolon ending the first new line and all text following the semicolon placed into the second line.

All lines are concatenated so that each resultant line ends in a semicolon.

All lines of an SQL block containing the token "DROP VIEW" are dropped since this Oracle construct yields no table and/or variable information required by the MDT.

Only lines beginning with the tokens "CREATE TABLE" or "CREATE VIEW" are now considered. Furthermore, if the line begins with the token "CREATE TABLE", it must also contain the tokens "AS SELECT" and "(SELECT" in order to be considered.

The token "LEFT JOIN" is replaced with a comma. The token "DISTINCT" found in any SQL "SELECT" statement is removed. The Oracle-specific token "(+)" is removed.

All multiple spaces are replaced with a single space. Spaces before and/or after parentheses, commas, equal signs, asterisks, and plus signs are removed. SQL tokens "SELECT*FROM" are replaced with the phrase "SELECT all_variables_from_this_table FROM".

The SQL "SELECT" statement is now split into separate variables which represent the Oracle clauses SELECT, FROM, ON, WHERE, GROUPBY, HAVING, and ORDERBY.

Calls to "SQL ROUND" and "SUM" functions are removed. Trailing ");" are removed from each clause. Tokens "FROM", "WHERE", "GROUP BY", and "ORDER BY" are removed.

References to tables listed in the SQL "FROM" clause are arranged in length of table alias order.

Variables in the SQL "SELECT" statement are counted as well as the tables in the SQL "FROM" clause. The number of variables in the "GROUP BY" clause is determined.

Column aliases in the "SELECT" part are replaced with true table names as indicated on the "FROM" clause. SQL tokens such as "OR", ">", "<", "IS NOT NULL", "BETWEEN", etc. are replaced with a comma in the "WHERE" clauses.

Table aliases with table names in the SQL "WHERE" clause are replaced as proscribed in the SQL "FROM" clause.

At this point in the parsing procedure, the current SAS data set contains up to five character strings of the form "Table1.Variable1,Table1.Variable2, ..., TableN.Variable1,TableN.Variable2, ...", one character string for each of the Oracle clauses "GROUP BY", "WHERE", "SELECT", "HAVING", and "ORDER BY". In addition, the SAS data set contains one character string containing the "CONNECT TO ORACLE" information. The penultimate step in this parsing process is to create a SAS data set containing simply two character variables. The SAS data set, which is derived exclusively from SQL statements within SAS programs identified on the PDT, has the format and content shown in Figure 6.

Table1.Variable1	CONNECT TO ORACLE information for this table
Table1.Variable2	CONNECT TO ORACLE information for this table
...	CONNECT TO ORACLE information for this table
TableN.Variable1	CONNECT TO ORACLE information for this table
TableN.Variable2	CONNECT TO ORACLE information for this table
...	...

Figure 6. SAS Data Set from Which MDT Is Created

Finally, this SAS data set is used to load the MDT table with the variables Database_Name (from the CONNECT TO ORACLE information), Table_Name and Field_Name (from the TableX.VariableY constructs).

DCMS USAGE

1. Automated Notification of Data Change Impact

The value of the DCMS notification of Data Change Impact is in the analysis performed. It does not simply alert the programmer that data is changing. It also notifies her/him which of his programs are affected and how. This is best appreciated by comparing a DCMS notification of Data Change and a notification of Data Change with no supporting analysis. Suppose the programmer receives the data change notification illustrated in Figure 7, not supported by any detailed analysis.

To Data Warehouse Users:

The Data Warehouse is updated with data transmitted through 17jan2009. The database connect string is CSDW. Projected changes in the 07mar2009 release of CS-Link:
The range for the variable DISCH_DISP_CODE will be expanded;
The range for the variable ACCT_NO will be expanded; and
The Payor_Code table will be retained for historical data only. A new table (PAYOR_CS LINK) is being introduced. Two separate tables (pre- and post- CS-Link) will need to be mined for multi-year analysis.

Regards, EIS

Figure 7. Notification of Impending CSDW Change without Analysis

The programmer is interested in knowing if the upcoming CS-Link changes (on 07mar2009) to CSDW will affect the viability of the program, ROMR2237. To determine this, the programmer would be required to manually peruse the thousands of lines of SAS program code from within the SAS editor and locate all code accessing the changing variables. This is time-consuming and error-prone. In comparison, DCMS analyzes the impact of the impending changes and triggers the Microsoft Outlook email shown in Figure 8.

Dear Alex Pakalniskis:

ROMR2237.sas, located at \\cshsmedaff\RO_Mgmt\shared\sas_code\other_from_Kdrive\ROMR2237, has been analyzed to determine the impact of upcoming CS-Link changes. Please review these automatically generated notes:

Table DISCHARGE_DISPOSITION.

Field DISCH_DISP_CODE. (Contact Nilesh Bajania at bajanian@cshs.org).

Field DISCH_DISP_UB92_CODE will be reformatted in CS-Link DW. Field DISCH_DISP_UB92_CODE will be renamed in CS-Link DW. Column is blank for values from Clarity, but current DISCH_DISP_CODE does not seem like UB codes either. (Contact Ann Chu at chua@cshs.org).

Table PAYOR will be renamed in CS-Link DW to PAYOR_BENEFIT_PLAN.

Field PAYOR_CATEGORY_CODE will be reformatted in CS-Link DW. In DW-Link DW you need: Link VISIT_COVERAGE table to COVERAGE table using VISIT_ID. Link COVERAGE table to PAYOR_BENEFIT_PLAN table using BENEFIT_PLAN_ID. PAYOR_CATEGORY_CODE is in BENEFIT_PLAN table. (Contact Nilesh Bajania at bajanian@cshs.org).

Field PAYOR_CODE. Effective 3-1-9 PAYOR table will no longer be updated in CS-Link DW. This table is retained for historical data. (Contact Nilesh Bajania at bajanian@cshs.org).

Table VISIT.

Field ACCT_NO will be reformatted in CS-Link DW. Field ACCT_NO will be renamed in CS-Link DW: In Epic Clarity Database Account # is called HAR (Hospital Account Record) Pre-registration/admission to assign each patient stay/visit a HAR. All CSNs for that stay/visit/diagnosis will be attached to that HAR. Same day multiple CSNs will be attached to the same HAR. Therefore, if encounters need to be counted the same way as before, use the HOSPITAL_CONTACT table instead. The HAR number is even (7) digits, starting with "1000000" – Current ACCT_NO is eleven (11) digits. Note: Recurring visits for the same diagnosis to be on the same HAR. Certain Rules apply for IP and OP based on PAYOR (e.g., Medicare's 72 hr rule). Contact Serial Number (CSN): Each episode (encounter) will be assigned a CSN. Invoice: The invoice number is the HAR number plus three (3) assigned digits. The claim to generate a summary of all CSN provided. (Contact Nilesh Bajania or Ann Chu at bajanian@cshs.org or chua@cshs.org).

Field DISCH_DISP_CODE will be reformatted in CS-Link DW. Field DISCH_DISP_CODE will be renamed in CS-Link DW.

Check field DISCH_DISP_SOURCE_CODE in CS-Link DW DISCHARGE_DISPOSITION table. "CLR" are DISCHARGE_DISPOSITION codes coming from Clarity. Some Clarity DISCHARGE_DISPOSITION codes are UB92 codes. Note: CS-Link DW DISCHARGE_DISPOSITION codes are NUMERIC but data type is still VARCHAR2. That is, if you are expecting code "06", but code is stored as "6". (Contact Nilesh Bajania or Ann Chu at bajanian@cshs.org or chua@cshs.org).

Table VISIT_PAYOR will be renamed in CS-Link DW to VISIT_COVERAGE, COVERAGE, PAYOR_EPIC.

Field PAYOR_CODE will be reformatted in CS-Link DW. Field PAYOR_CODE will be renamed in CS-Link DW: PAYOR_ID. CSDW PAYOR table will be retained as is for historical data. In CS-Link DW new table PAYOR_EPIC is introduced. DQMU needs to analyze your need and advise how can you get the same data from EPIC. You will use PAYOR table for per-EPIC PAYOR data and PAYOR_EPIC table for EPIC PAYOR data for multi-year analysis. Link VISIT_COVERAGE table to COVERAGE table using COVERAGE_ID. Link COVERAGE table to PAYOR_EPIC table using PAYOR_ID. PAYOR_ID is equivalent to PAYOR_CODE of VISIT_PAYOR table. (Contact Nilesh Bajania or Ann Chu at bajanian@cshs.org or chua@cshs.org).

Field PAYOR_PRIORITY will be reformatted in CS-Link DW. Field PAYOR_PRIORITY will be renamed in CS-Link DW: PAYOR_PRIORITY. Link VISIT_COVERAGE table to COVERAGE table using VISIT_ID. PAYOR_PRIORITY code is in COVERAGE table. (Contact Nilesh Bajania at bajanian@cshs.org).

Analysis of these tables/fields by DQMU indicates no change requirements:

Table=DISCHARGE_DISPOSITION

Field=DISCH_DISP_DESC (Contact Nilesh Bajania at bajanian@cshs.org).

Field=DISCH_DISP_UB92_DESC (Contact Nilesh Bajania at bajanian@cshs.org).

Table=PAYOR_CATEGORY

Field=PAYOR_CATEGORY_CODE (Contact Nilesh Bajania at bajanian@cshs.org).

Field=PAYOR_CATEGORY_DESC (Contact Nilesh Bajania at bajanian@cshs.org).

Field=PAYOR_CATEGORY_GROUP_CODE (Contact Nilesh Bajania at bajanian@cshs.org).

Table=PAYOR_CATEGORY_GROUP

Field=PAYOR_CATEGORY_GROUP_CODE (Contact Nilesh Bajania at bajanian@cshs.org).

Field=PAYOR_CATEGORY_GROUP_DESC (Contact Nilesh Bajania at bajanian@cshs.org).

Table=VISIT_PAYOR

Field=VISIT_ID (Contact Nilesh Bajania at bajanian@cshs.org).

Figure 8. Automated Output of Change Analysis Program

The MDT table contains 67 records for this program. Nine of these records refer to a database other than CSDW, so they are of no consequence. And only sixteen of the remaining 58 records reference discharge disposition, account, or payer information. The automated email is indicating that only eight of those sixteen table/variable combinations will need to be altered because of the impending CSDW changes. Furthermore, as much information as available on the BRT about the impending changes is included in this email to assist the programmer. Some information isn't yet available (in the DISCHARGE_DISPOSITION table no information is given for the field DISCH_DISP_CODE), although the programmer is urged to contact the business analyst and other contact information is provided. Using the DCMS analysis, the programmer can save time and can be confident that he has not overlooked a piece of code that should be changed. With less programmer effort, ongoing data quality is assured in the face of data change.

2. Data Change Impact Report for Management

The MDT contains a wealth of information for both project managers and resource managers. They can request reports to help level load programming work, to understand the status of a database migration project, to prioritize programming work, and, in general, to help manage their project and department teams. Continuing with the CSDW change example above, ROM leadership might desire to know the impact on their team of the payer structure change. Using the MDT as the source, an impact report is readily available. When the insurance payer tables were flagged in the BRT as undergoing imminent changes in the new CSDW, the CAP identified that three programmers would be affected by these changes. One programmer was associated with 36 programs, a second programmer was associated with 5 programs, and a third programmer was found to be associated with 15 programs. This was valuable information for ROM leadership.

CONCLUSION

By performing automated change impact analyses, DCMS saves programmer time and avoids errors natural to manual processes. In addition, the information it gathers provides transparency to project and resource managers. Initially created to help identify the impact and scope of a database migration project, DCMS has proven its value as a tool for ongoing program quality management and control. A centralized repository of data change information and suggested programming solutions, made available to all programmers, insures uniform report quality

ACKNOWLEDGEMENTS

Thanks to Mirga Girnius, Ph.D. for editing and to Bruce Davidson, Ph.D. for reviewing this paper and providing their helpful remarks.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.
Contact the authors at email address pakalniskisa@cshs.org.