#### Paper 3340-2015

# Performing Efficient Transposes on Large Teradata Tables Using SQL Explicit Pass-Through

Tao Cheng, Accenture, Almere, the Netherlands

## **ABSTRACT**

It is a common task to reshape your data from long to wide for the purpose of reporting or analytical modeling and PROC TRANSPOSE provides a convenient way to accomplish this. However, when performing the transpose action on large tables stored in a database management system (DBMS) such as Teradata, the performance of PROC TRANSPOSE can be significantly compromised. In this case, it may be more efficient for the DBMS to perform the transpose task. SAS® provides in-database processing technology in PROC SQL, which allows the SQL explicit pass-through method to push some or all of the work to the DBMS. This technique has helped facilitate integration between SAS and a wide range of data warehouses and databases, including Teradata, EMC Greenplum, IBM DB2, IBM Netezza, Oracle, and Aster Data.

This paper uses the Teradata database as an example DBMS and explains how to transpose a large table that resides in it using the SQL explicit pass-through method. The paper begins with comparing the execution time using PROC TRANSPOSE with the execution time using SQL explicit pass-through. From this comparison, results have demonstrated that SQL explicit pass-through is more efficient than the traditional PROC TRANSPOSE when transposing Teradata tables, especially large tables. The paper explains how to use the SQL explicit pass-through method and discusses the types of data columns that you might need to transpose, such as numeric and character. The paper presents a transpose solution for these types of columns. Finally, the paper provides recommendations on packaging the SQL explicit pass-through method by embedding it in a macro. SAS programmers who are working with data stored in an external DBMS and who would like to efficiently transpose their data will benefit from this paper.

## INTRODUCTION

The TRANSPOSE procedure is a simple and powerful procedure in the Base SAS. You can use PROC TRANSPOSE to turn variables into observations or observations into variables. The output from the procedure is a data set that contains the transposed data. This output data set can be used in subsequent DATA or PROC steps for analysis, reporting, or further data manipulation.

When using PROC TRANSPOSE to transpose a Teradata table, the data is fetched from the DBMS to SAS. Large tables mean that a significant amount of data must be transferred. PROC TRANSPOSE can still do the job, but based on this comparison appears to be a less efficient way of transposing data when compared to the method of using SQL explicit pass-through, which has several advantages over processing within SAS. These advantages may consist of:

- Increased security,
- Reduced network traffic
- Faster processing

Increased security is possible because sensitive data does not have to be extracted from the DBMS. Reduced network traffic is possible because large amount of data are not transported across network connections. Faster processing is possible because data is processed locally using high-speed secondary storage devices on the DBMS and because the DBMS may be capable of optimizing a query for execution in a highly parallel fashion.

Figure 1 shows the execution time using PROC TRANSPOSE and the execution time using SQL explicit pass-through method. The SQL explicit pass-through method results in significant performance improvements, especially when transposing large Teradata tables.

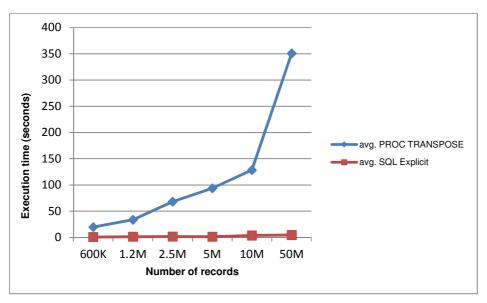


Figure 1. Execution Time Comparison: PROC TRANSPOSE vs. SQL Explicit Pass-Through

This paper discusses Teradata table transposition using the SQL explicit pass-through method to turn observations into variables. The focus is on transposing tables using BY groups and an ID variable. The solution presented here is derived from the Author's recent analytics project. There is definitely more to SQL explicit pass-through than what is explained in this paper. This paper demonstrates a way of using SQL explicit pass-through to transpose DBMS tables in the following format:

- One or more BY-group variable(s), e.g. the NAME column in Table 1.
- One ID variable values of this variable will be used as column names in the transposed table, e.g. the SUBJECT column in Table 1 is an ID variable.
- One VAR variable values of this variable will be transposed, e.g. the VALUE column in Table 1.

Table 1 shows an example table illustrating the required table format for using the transpose solution introduced in the paper.

NAME	SUBJECT	VALUE
Jack	id_num	101
Jack	new_patient	1
Jack	expense	65.20
Lisa	id_num	102
Lisa	new_patient	0
Lisa	expense	27.35

Table 1. Example Table Format

The following sections explain the solution and introduces you to the applications of transposing numeric data columns, character data columns and performing possible data definition (data type, data format, or both) conversion. This paper also packages the solution into a SAS Macro so that it would be convenient to re-use this solution to transpose your tables whenever you need it. After reading this paper you will be able to use the solution or customize it as needed to efficiently transpose your DBMS tables.

## **BASIC UNDERSTANDINGS**

The solution introduced in this paper requires some basic understanding about SAS explicit pass-through method, SAS Macros and Teradata databases. This paper does not explain how SAS Macros work. Listed below are some relevant and basic understandings of Teradata database and SAS system.

#### **TERADATA BASICS**

- Teradata will generate an error when reserved words are used as column names. In this case use double quotes around the word.
  - For example, 'value' is a Teradata reserved word and an error will be generated when you use it as a column name. If you have to have the column 'value', then you could prevent the error by enclosing the word in double quotes, i.e. "value".
- If you create a table that already exist in Teradata database, you will get a database error.
- Null value in Teradata means an absence of information and is analogous to missing value in SAS system.
- When obtaining row summary information of DBMS tables using GROUP BY clause and Aggregate functions, the missing values are excluded from the operation before the functions take place.
- This paper uses the aggregate function MAX(). The MAX() for a set of known numeric values is the largest one. The MAX() for a set of known DATE and DATETIME data is the one farthest in the future or most recent. The MAX for a set of non-blank character strings is the last one in the ascending sort order.

# Some common Teradata data types:

- CHARACTER Data Type. Represents a fixed length character string. For example, CHARACTER(7) is the CHARACTER data type with a length of seven.
- VARCHAR Data Type. Represents a variable length character string of length 0 to *n*. For example, VARCHAR(10) is the character data type with a varying length string of length 0 to 10.
- INTEGER Data Type. Represents a signed, binary integer value from -2,147,483,648 to 2,147,483,647.
- DECIMAL Data Types. Represents a decimal number of *n* digits, with *m* of those *n* digits to the right of the decimal point. For example, number 256.78 has the type DECIMAL(5,2).
- DATE Data Type. Identifies a field as a DATE value and simplifies handling and formatting of date variables. The default format is ANSI date YYYY-MM-DD.
- TIMESTAMP Data Type. Identifies a field as a TIMESTAMP value. The default format is YYYY-MM-DDBHH:MI:SS.S(F)Z. 'S(F)Z' represents the fractional portion of the SECOND field. Values for fractional seconds precision range from zero through six inclusive. The default precision is six. For example, when fractional seconds precision is 6, '1999-01-01 23:59:59.999999'; when fractional seconds precision is 0, '1999-01-01 23:59:59'.

#### **SAS SYSTEM OPTIONS**

Below is a set of SAS system options supporting in-database operations. Once set these options write to the SAS log the SQL that the DBMS such as Teradata engine passes to the DBMS.

MSGLEVEL=N

Specifies the level of detail in messages that are written to the SAS log. N is the default.

NOSTSUFFIX

The log output is much easier to read if you specify NOSTSUFFIX.

```
SASTRACE=',,,ds'
```

Specifies that all SQL statements that are sent to the DBMS are sent to the log and that a summary of timing information for calls made to the DBMS is sent to the log.

```
SASTRACELOC=SASLOG
```

Prints SASTRACE information to a SAS log.

```
SQL_IP_TRACE=SOURCE
```

Specifies the level of information to be included in the trace output. When set to SOURCE, the major SQL commands that are submitted for in-database computation are included in trace output, but not minor SQL commands such as those used to obtain metadata.

```
DBIDIRECTEXEC
```

Indicates that the SQL pass-through facility optimizes handling of SQL statements by passing them directly to the database for execution, which optimizes performance.

For detailed information about each system option, please refer to SAS online resources.

# SYNTAX OF SQL EXPLICIT PASS-THROUGH

Below is the syntax of SQL explicit pass-through. This form may be used to pass any SQL - including Data Definition Language (DDL) commands - from SAS to the database for execution. By default SAS/ACCESS to Teradata uses ANSI Mode and ANSI Mode requires the COMMIT Statement with explicit pass-through.

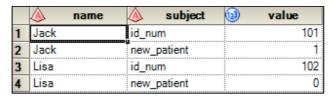
```
proc sql;
```

## **UNDERSTANDING THE SOLUTION**

In this section the SQL explicit pass-through transposition solution is discussed with an example. The transposition solution is applied to the input table as shown in Display 1. The result is the output table as shown in Display 2. Please note that in the input table for each BY-group i.e. column 'NAME' there are no double or duplicated entries in column 'SUBJECT'. Double or duplicated entries are often not desirable and are considered to be data quality problems, which should be resolved before performing the transposition.

## **INPUT TABLE**

This is the table that we want to transpose. 'new\_patient' is a binary field identifying if a person is a new patient or not.



**Display 1. Table Before Transposition** 

## **OUTPUT TABLE**

This is the table that we want to have after transposition.



**Display 2. Table After Transposition** 

## **CODE THAT DOES THE JOB**

Here is the SQL explicit pass-through code that performs the transposition. The key part of the solution is the highlighted MAX() statement. The word <code>value</code> is enclosed in double quotes because it is one of the Teradata reserved words and will throw an error otherwise.

## **Two-step Execution**

Here is an example to explain how the solution works.

The solution code first establishes a table with a structure of 4 columns: name, id\_num, new\_patient, and expense. The data is populated into this table through two steps:

- First Step: the CASE statement is executed
- Second step: the aggregate function MAX() is carried out.

# First step

In the first step the CASE statement is executed. This gives an interim table as follows.

name	id_num	new_patient
Jack	101	
Jack		1
Lisa	102	
Lisa		0

Table 2. Interim Result from Execution of the CASE Statement

# Second step

In the second step function MAX() aggregates the data for each value in the BY-group variable in the interim table. The result of the second step is Table 3. Please note that the missing values are excluded from the operation before the function MAX() takes place.

name	id_num	new_patient
Jack	101	1
Lisa	102	0

Table 3. Table after transposition

We now use a simple example to discuss the SQL explicit pass-through.

It is often necessary to have a change in data type after the transposition. In the section 'USING THE SOLUTION' examples are given to show the application of the SQL explicit pass-through transposition solution and how to convert data type while carrying out the transposition.

## **USING THE SOLUTION**

#### TRANSPOSE NUMERIC COLUMN

Display 3 shows an input table that has an BY-group variable 'name', an ID variable 'subject' and a numeric variable 'value'.

	name	subject	100 value
1	Jack	id_num	101
2	Jack	new_patient	1
3	Jack	expense	65.2
4	Lisa	id_num	102
5	Lisa	new_patient	0
6	Lisa	expense	27.35

Display 3. Input Table with Numeric Column

Display 4 shows that in the table after the transposition. There are three numeric columns: 'id\_num', 'new\_patient', 'expense'. By default these three columns are numeric columns because the source column of them i.e. 'value' is a numeric column. Please note that a data format conversion is applied to the column 'expense' and the data in this column are standardized to have 3 digits after the decimal point.

	🔌 name	id_num	new_patient	expense
1	Jack	101	1	65.200
2	Lisa	102	0	27.350

Display 4. Output Table with 3 Numeric Columns

The following code does the transposition and data format conversion. The data format conversion is implemented in the highlighted CAST() function. Decimal(7,3) specifies the data format of the column 'expense' in the output table. CAST() is a Teradata function that performs the explicit data type conversion. A data type conversion modifies the data definition (data type, data attributes, or both) of an expression.

#### TRANSPOSE CHARACTER COLUMN

Display 5 shows an input table that has an BY-group variable 'name', an ID variable 'subject' and a character variable 'value'. Note that the column 'value' contains different types of data: character, integer, decimal, date, and datetime. Because these data are stored in one column, they have to be stored as character data.

	A name	subject	
1	Jack	id_num	101
2	Jack	date of birth	21-09-1980
3	Jack	country	NL
4	Jack	new_patient	1
5	Jack	expense	65.20
6	Jack	receive_dt	2015-09-16 09:18:27
7	Jack	process_dt	09-17-2015 09:00:00
8	Lisa	id_num	102
9	Lisa	date of birth	22-07-2001
10	Lisa	country	UK
11	Lisa	new_patient	0
12	Lisa	expense	
13	Lisa	receive_dt	2015-12-21 16:20:14
14	Lisa	process_dt	12-22-2015 09:00:00

# **Display 5. Input Table with Character Column**

Display 6 shows the table after transposition. In the input the data in the column 'value' are all in character data type and in the output table some columns are converted to more sensible data types, for instance, 'date of birth' is converted to a DATE column, 'expense' is converted to DECIMAL column and 'receive\_dt' is converted to a TIMESTAMP column.



## Display 6. Output Table with Numeric and Character Columns

The following code performs the transposition. As before note how data definition conversions are implemented in the CAST() function. The highlighted format statement code specifies the input data format and ensures that Teradata correctly interprets the input data in the conversion. The source value to the column 'receive\_dt' is in the default Teradata TIMESTAMP format YYYY-MM-DDBHH:MI:SS and therefore Teradata can interpret it correctly during conversion. In contrast with column 'receive\_dt', data in column 'process\_dt' are in a format other than the Teradata default one, therefore a FORMAT statement is used to instruct Teradata how to interpret the input data. Similarly, a FORMAT statement is also used for conversion when creating the column 'date of birth'.

The transposition solution is able to cope with missing data. In the input table, Lisa does not have an expense. In the transposed output table Lisa's expense is missing and is represented by a dot.

```
proc sql;
  connect to teradata (server=testserver user=testuser password=testpass);
  execute (create table testdatabase.patients_out as(
             select name,
                    cast(max((case subject WHEN 'id_num' then "value" else
                         null end)) as integer) id_num,
                    cast(max((case subject WHEN 'date of birth' then "value"
                              else null end)) as date
                         format 'dd-mm-yyyy') "date of birth",
                    cast(max((case subject WHEN 'country' then "value" else
                         null end)) as varchar(5)) country,
                    cast(max((case subject WHEN 'new_patient' then "value"
                         else null end)) as integer) new_patient,
                    cast(max((case subject WHEN 'expense' then "value" else
                         null end)) as decimal(7,3)) expense,
                    cast(max((case subject WHEN 'receive_dt' then "value"
                         else null end)) as timestamp(0)) receive_dt,
                    cast(max((case subject WHEN 'process_dt' then "value"
                         else null end)) as timestamp(0)
                         format 'mm-dd-yyyybhh:mi:ss') process_dt
              from testdatabase.patients
              group by name) with data primary index(name)
           ) by teradata;
   execute (commit) by teradata;
quit;
```

# **PUT IT IN SAS MACRO**

When frequent table transposition are required, and the tables share the same table structure as illustrated in the above examples, it is a good idea to create a SAS macro for the SQL explicit pass-through solution and use it whenever you need.

In this section the SQL explicit pass-through transposition solution is implemented in a SAS macro named %sqltranspose. It is discussed how to use this macro to realize the same transposition as shown in the section TRANSPOSE CHARACTER COLUMN.

```
 \$ \textit{sqltranspose} \texttt{(DSIN=patients,DSOUT=patients\_out,BY=name,ID=subject,VAR=value);}
```

%sqltranspose has parameters for the input table, the output table, the BY-group variable, ID variable, and the actual column for transposition.

- DSIN input table before transposition, e.g. table 'patients'
- DSOUT output table after transposition, e.g. table 'patients' out'
- BY variable enabling aggregation, e.g. column 'name' in example
- ID variable contributing column names, e.g. column 'subject' in example
- VAR column providing source data for transposition, e.g. column 'value' in example

To ensure the right column definition in the output table and to make it possible for the macro to perform the data type conversion, information about column data type and source data format should be provided to the macro %sqltranspose. Therefore some preparatory work needs to be done before running the macro for DBMS transposition.

#### PREPARATORY WORK

The preparatory work passes the required information about data types and formats to the macro %sqltranspose through the use of macro variables. There are different ways to achieve this. This paper uses the method of creating macro variables from a data definition dataset. Display 7 shows an example of a data definition dataset. The data definition dataset has three columns and contains information about the data format of the columns in the input table and the data type of the columns in the output table. If you understand your data well you can simply create this dataset manually. In Appendix A Complete Solution a method of creating the data definition dataset is given.

- Subject distinct values of the ID variable in the input table Display 6.
- Type the preferred data type in the output table. It is suggested to specify a type value to avoid data truncation and to store data efficiently. This is an optional column and therefore if you do not specify a type value %sqltranspose still works.
- Format source data format to ensure correct data conversion. This is an optional column. However
  a format value is required when a column has data that are not in one of the default Teradata data
  formats.

	subject	▲ type	format
1	country	varchar(5)	
2	date of birth	date	dd-mm-yyyy
3	expense	decimal(7,3)	
4	id_num	integer	
5	new_patient	integer	
6	process_dt	timestamp(0)	mm-dd-yyyybhh:mi:ss
7	receive_dt	timestamp(0)	

**Display 7. Data Definition Dataset** 

The following code is used to pass the information in the data definition dataset into macro variables.

```
%let id_var=subject;
data _null_;
  set definition end=last;
  /*create macro variables - column name*/
  call symputx("col_name" || strip(put(_N_, 8.)), &id_var.);
  /*create macro variables - values of ID column*/
  col_val= "'"|| strip(&id_var.) ||"'";
  call symputx("col_val" || strip(put(_N_, 8.)), col_val);
  /*create macro variables - number of variables*/
  if last then do;
    call symputx("num", strip(put(_N_, 8.)));
  /*create macro variables - column data type*/
  call symputx("col_type" || strip(put(_N_, 8.)), strip(type));
  /*create macro variables - source data format*/
 col_format= "'"|| strip(format) ||"'";
  call symputx("col_fmt" || strip(put(_N_, 8.)), col_format);
```

The preparatory work makes sure that necessary information such as column data type and source data format are available to the macro %sqltranspose.

#### MACRO %SQLTRANSPOSE

The macro %sqltranspose code is shown below. Note that the macro uses the macro variables that are created in the preparatory work.

Similar to the SQL explicit transposition code in previous examples, the key part of the transposition macro is the CAST() statement containing the MAX() function. The %IF-%THEN/%ELSE Statement gives the flexibility of applying a data type and a format for the correct data conversion.

```
/*declaring macro variables*/
%let td_string=server=testserver user=testuser password=testpass;
%let db=testdatabase;
/*creation of macro %sqltranspose*/
%macro sqltranspose(DSIN=, DSOUT=, BY=, ID=, VAR=);
proc sql;
connect to teradata as td (&td_string.);
execute (create table &db..&DSOUT. as (
        select &BY.
%do i=1 %to &num.;
/*condition when column has no type and no format information*/
%if "&&col_type&i." = "" and &&col_fmt&i.='' %then %do;
        , max((case &ID. when &&col_val&i. then "&VAR." else null end)) "&&col_name&i."
%end:
/*condition when column has type information but no format information*/
%else %if "&&col_type&i." ^= "" and &&col_fmt&i.='' %then %do;
,cast(max((case &ID. when &&col_val&i. then "&VAR." else null end)) as &&col_type&i.)
"&&col_name&i."
%end;
/*condition when column has type and format information*/
%else %if "&&col_type&i." ^= "" and &&col_fmt&i.^='' %then %do;
,cast(max((case &ID. when &&col_val&i. then "&VAR." else null end)) as &&col_type&i.
format &&col_fmt&i.) "&&col_name&i."
%end;
%end:
from &db..&DSIN.
group by &BY.) with data primary index (name)
) by td;
execute (commit) by td;
auit:
%mend sqltranspose;
```

To use the macro %sqltranspose to transpose the table in Display 5, you should first complete the preparatory work. Then the macro code %sqltranspose needs to be complied so that it is available for use in your SAS session. Next transpose the table simply by invoking the macro:

```
% sqltranspose(DSIN=patients,DSOUT=patients_out,BY=name,ID=subject,VAR=value);
```

## CONCLUSION

When transposing large DBMS tables, it can be more efficient to push the work to the DBMS side using SQL explicit pass-through. This paper presents a possible method for performing the transpose work in the Teradata database. When you need to transpose a DBMS table which has a similar structure to the one that is presented in this paper, you could directly apply the solution. Once you understand the solution you should be able to customize it to address your specific transposition problems.

## **REFERENCES**

Teradata Corporation. Teradata Database, Tools and Utilities Release 14.10. <a href="http://www.info.teradata.com/htmlpubs/DB">http://www.info.teradata.com/htmlpubs/DB</a> TTU 14 10/index.html#page/SQL Reference/B035 1141 11 2A/appb.111.01.html

SAS Institute Inc., SAS/ACCESS(R) 9.2 for Relational Databases: Reference, Fourth Edition <a href="http://support.sas.com/documentation/cdl/en/acreldb/63647/HTML/default/viewer.htm#a001384390.htm">http://support.sas.com/documentation/cdl/en/acreldb/63647/HTML/default/viewer.htm#a001384390.htm</a>

SAS Institute Inc., SAS® 9.4 In-Database Products

http://support.sas.com/documentation/cdl/en/indbug/67366/HTML/default/viewer.htm#titlepage.htm

SAS Institute Inc., SAS/ACCESS Interface to Teradata <a href="https://support.sas.com/resources/papers/teradata.pdf">https://support.sas.com/resources/papers/teradata.pdf</a>

Frank Capobianco. 2011. Explicit SQL Pass-Through: Is It Still Useful?. Teradata Corporation, Lexington, SC. <a href="http://support.sas.com/resources/papers/proceedings11/105-2011.pdf">http://support.sas.com/resources/papers/proceedings11/105-2011.pdf</a>

Cynthia L. Zender. 2013. Macro Basics for New SAS® Users. SAS Institute Inc., Cary, NC https://support.sas.com/resources/papers/proceedings13/120-2013.pdf

# **CONTACT INFORMATION**

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

Author name : Tao Cheng

Company : Accenture Technology Solutions Netherlands

 Work phone
 : +31 (0) 36 5461470

 E-mail
 : t.cheng@accenture.com

 Website
 : www.accenture.nl

Copyright © 2015 Accenture

All rights reserved.

Accenture, its logo and High Performance Delivered are trademarks of Accenture.

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.

## **APPENDIX A: COMPLETE SOLUTION**

```
%** SAS Paper 3340-2015
%* Date : April 2015
%* Version : 1.0
%* Description : This program illustrates how to effectively transpose
%*
일 *
           a DBMS table using the macro %sqltranspose
%** Version control
8************************
2*****************************
                                                  *;
%* Create sample Teradata table patients
%* Table patients serves as the input table in the example
/*common options to show in log the execution of codes in DBMS*/
option msglevel=n nostsuffix sastrace=',,,ds' sastraceloc=saslog
    sql_ip_trace=source dbidirectexec;
/*common options for debugging macro*/
option macrogen symbolgen mlogic mprint;
/*create macro variables*/
%let td_string=server=testserver user=testuser password=testpass;
%let db=testdatabase;
proc sql;
  connect to teradata (&td_string.);
  execute (create table &db..patients (name varchar(10),
                           subject varchar(20),
                           "value" varchar(30)
       ) by teradata;
  execute (commit) by teradata;
  /*Jack*/
  execute (insert into &db..patients (name, subject, "value")
                    values ('Jack', 'id_num', '101')
       ) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                        values ('Jack', 'date of birth', '21-09-
1980')
       ) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                   values ('Jack', 'country', 'NL')
       ) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                         values ('Jack', 'new_patient', '1')
```

```
) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                                values ('Jack', 'expense', '65.20')
         ) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                                values ('Jack', 'receive_dt', '2015-09-
16 09:18:27')
        ) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                                values ('Jack', 'process_dt', '09-17-
2015 09:00:00')
      ) by teradata;
  /*Lisa*/
  execute (insert into &db..patients (name, subject, "value")
                           values ('Lisa', 'id_num', '102')
         ) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                               values ('Lisa', 'date of birth', '22-07-
2001')
         ) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                               values ('Lisa', 'country', 'UK')
         ) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                                values ('Lisa', 'new_patient', '0')
         ) by teradata;
  execute (insert into &db..patients (name, subject)
                                values ('Lisa', 'expense')
         ) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                                values ('Lisa', 'receive_dt', '2015-12-
21 16:20:14')
         ) by teradata;
  execute (insert into &db..patients (name, subject, "value")
                                values ('Lisa', 'process_dt', '12-22-
2015 09:00:00')
         ) by teradata;
  execute (commit) by teradata;
quit;
%* Create definition dataset with DATALINES statement
8*****************************
data definition;
  length subject $15 type $15 format $25;
  infile datalines delimiter='|';
  input subject $ type $ format $;
  datalines;
country | varchar(5) |
date of birth | date | dd-mm-yyyy
expense | decimal(5,2)|
           | integer |
id num
new_patient | integer
```

```
%* Create macro variables to store data type and data format
%let id_var=subject;
data _null_;
 set definition end=last;
 /*create macro variables - column name*/
 call symputx("col_name" || strip(put(_N_, 8.)), &id_var.);
 /*create macro variables - values of ID column*/
 col_val= "'"|| strip(&id_var.) ||"'";
 call symputx("col_val" || strip(put(_N_, 8.)), col_val);
 /*create macro variables - number of variables*/
 if last then do;
    call symputx("num", strip(put(_N_, 8.)));
 end:
 /*create macro variables - column data type*/
 call symputx("col_type" || strip(put(_N_, 8.)), strip(type));
 /*create macro variables - source data format*/
 col_format= "'"|| strip(format) ||"'";
 call symputx("col_fmt" || strip(put(_N_, 8.)), col_format);
run;
8*****************************
%* Create macro %sqltranspose that does the SQL explicit transpose
%macro sqltranspose(DSIN=, DSOUT=, BY=, ID=, VAR=);
  proc sgl;
        connect to teradata as td (&td_string.);
        execute (create table &db..&DSOUT. as (
                                select &BY.
                                %do i=1 %to &num.;
                        /*condition when column has no type and no format
information*/
                                  %if "&&col_type&i." = "" and
&&col_fmt&i.='' %then %do;
                                  , max((case &ID. when &&col_val&i.
then "&VAR." else null end)) "&&col_name&i."
                                  %end:
                        /*condition when column has type information but
no format information*/
                                  %else %if "&&col_type&i." ^= "" and
&&col_fmt&i.='' %then %do;
                                  , cast(max((case &ID. when
&&col_val&i. then "&VAR." else null end)) AS &&col_type&i.) "&&col_name&i."
                                  %end;
```

```
/*condition when column has type and format
information*/
                          %else %if "&&col_type&i." ^= "" and
&&col_fmt&i.^='' %then %do;
                          , cast(max((case &ID. when
&&col_val&i. then "&VAR." else null end)) AS &&col_type&i. format
&&col_fmt&i.) "&&col_name&i."
                          %end;
                        %end;
                        from &db..&DSIN.
                        group by &BY.
                  ) with data primary index (&BY.)
            ) by td;
        execute (commit) by td;
    quit;
%mend sqltranspose;
%sqltranspose(DSIN=patients,DSOUT=patients_out,BY=name,ID=subject,VAR=value);
```