Optimizing Large Data Imports from a Relational Database Management System

Francis Harvey, Westat

ABSTRACT

When integrating SAS® analysis capabilities with a relational database management system (RDBMS), costly time delays can be introduced by failing to optimize the transfer of data from RDBMS to SAS. This paper will illustrate several methods for performing such optimization by limiting the data that needs to be transferred. If a large data transfer would still be required even with these methods, the paper provides alternative techniques for compressing the data to be transferred or for ensuring that no transfer is required at all during analysis by ensuring an up-to-date copy of the data is already available to SAS.

INTRODUCTION

One of the enduring strengths of SAS has been its ability to provide simple and timely access to data from a variety of external data sources, allowing SAS programmers to take full advantage of the comprehensive platform of analysis tools SAS provides while ensuring acceptable performance when working with external data sources. Among the more successful examples of this data integration strategy has been the merger of SAS analysis tools with RDBMS. Using an RDBMS for data management offers many key benefits:

1) Provides a data model and method (normalization) for eliminating redundant data
2) System ensures data integrity and consistency through use of the model
3) Provides data access through unique keys
4) Expresses relationships between tables through matches on these keys
5) Physical storage of data managed by system
6) Storage and database operations optimized to provide best execution times
7) Allow multiple users access to data while ensuring data integrity

SAS data access solutions preserve these capabilities while allowing analysts to effectively access and query large-scale data sets from RDBMS storing huge amounts of information. As the size of this data has grown, inevitable delays occur during the analysis process. However, in some situations, these delays have less to do with the performance of data retrieval operations in the RDBMS or analysis processes in SAS than with the size of data being transferred between the two systems.

A first approach to avoiding these transfer delays is to find some method of limiting the amount of data that must be transferred. One technique is to limit the variables whose data must be retrieved to the smallest possible set necessary for analysis. Another approach is to similarly limit the observations being returned. With some data, it may be beneficial to split large sparsely filled variables and densely filled variables into separate data transfers that are reintegrated in SAS where only non-null values of the sparsely filled variables would need to be transferred. Alternatively, the data could be reformatted to condense variable values into shorter formatted values that still allow for analysis.

For some analysis, it is not necessary to have all observations transferred as long as summarized calculations are sent for each set of analysis variables. One potentially less well-known summarization method is the use of the FREQ statement (WEIGHT statement in PROC FREQ) with a frequency variable for indicating in some SAS procedures how sets of analysis variables should be weighted. Various other statistical measures can also be calculated and transferred by the RDBMS for a set of analysis variables making it unnecessary to transfer all of the observations for that set to perform the same calculation in SAS. If such summarization is a large part of your analysis, creating a data cube may help to standardize this process.

If these methods cannot be exploited or fail to sufficiently reduce the size of the data that must be transferred, more elaborate approaches will be needed requiring specialized customization of the SAS and/or RDBMS platforms. Data compression seeks to reduce data transfer sizes by representing the same data in an alternative format requiring less space. As with many techniques, the goal is reached by many means; compression can be performed by installing interfaces on both the SAS and RDBMS platforms that provide automatic compression of the data at one end and decompression at the other, by writing specialized code on the RDBMS platform for placing the compressed data into a file that is then transferred to the SAS platform, or by writing specialized code on the RDBMS platform for placing the compressed data within the RDBMS data that can then be transferred to the SAS platform using normal data access methods.

Finally, although it should not necessarily be the last approach to be considered, there is an option for avoiding data transfer delays by ensuring the data necessary for analysis is available locally to the SAS platform using data replication. With this approach, a separate process runs at regular intervals to transfer the necessary data from the RDBMS to a location and format that can be accessed locally by the SAS platform. This method preserves the RDBMS data management benefits while allowing the data to be accessed immediately without extended transfer delays.

DATA REDUCTION

A direct approach to reducing the delay in transferring RDBMS data is to somehow limit the data or to reformat it in a more condensed form while still allowing the necessary information to be transferred for analysis. With these methods, the decrease in data transfer delay comes from simply finding an acceptable way of decreasing the amount of data that needs to be transferred. As these techniques offer the potential for large cuts in transfer time and require little to no special setup on the RDBMS or in SAS, they should always be among the first techniques considered for solving the data transfer delay problem.

LIMITING VARIABLES

This first technique, limiting variables, is well known as a method for improving SAS program efficiency when working with SAS data sets. The same reasoning behind this improvement also applies for input from RDBMS queries. By reducing the number of variables retrieved to just those required for the analysis, it may be possible to achieve a substantial decrease in the time required to transfer the data if the time spent obtaining data for the extraneous variables was a significant portion of the time required for the original retrieval. There are several scenarios where this could occur. One or more of the dropped variables could be exceedingly requiring a large number of bytes to be transferred, the number of variables dropped could be very large resulting in a large cumulative byte count, or the number and size of variables dropped may be small but the number of observations is large enough to again result in a large drop in the number of bytes to be transferred. Creating a new query within the RDBMS without the extra variables would normally be the preferred choice for performing this limiting as it allows the RDBMS to calculate and store a new optimized execution plan for the query. However, if the query is the result of a complicated SAS pass-through query to the RDBMS or if it would be inconvenient to create new queries for each combination of variables desired from an RDBMS query, it is still possible to obtain good results by simply modifying the query passed to the RDBMS from SAS. This can be done directly using PROC SQL pass-through facility or indirectly using the SAS/ACCESS libname with an appropriate keep or drop statement.

LIMITING OBSERVATIONS

Another approach that works well for improving SAS program efficiency and also can help reduce data transfer delay is to limit the observations retrieved to just those required for analysis. In this case, the reduction in data transfer delay comes from finding observations that need not be retrieved for analysis and eliminating them with an appropriate subsetting condition prior to the retrieval. This can be a difficult task as it is essential to ensure that the subsetting is done on the RDBMS rather than after the transfer to SAS. Again, there are a number of methods for implementing this approach, with the most desirable being the creation of a new query with matching optimized execution plan within the RDBMS to limit the observations. Alternatively, the PROC SQL pass-through facility provides an acceptable substitute for ensuring the initial subsetting is done by the RDMBS. However, if methods such as the WHERE= data set option, WHERE data step statement, WHERE clause with PROC SQL, or other conditional statements are used for this task, it is possible that the subsetting will be performed in SAS rather than the RDBMS. Although SAS has been optimized to attempt to pass such conditions through to the underlying RDBMS whenever possible, there is no guarantee that the more complicated conditions will always succeed. For these other methods it is important that you spend time researching the particulars for your individual RDBMS to become familiar with what conditions can successfully be passed by SAS to the RDBMS. A useful check to verify your conclusions is to use the SASTRACE and SASTRACELOC system options to verify that the query run on the RDBMS contains the condition.

SPLITTING SPARSE AND DENSE DATA VARIABLES

If limiting the variables or observations is not possible or fails to reduce the data transfer delay enough, it may still be possible to perform a different type of limiting based on the division of the data into sparse and dense variables. Sparse variables have relatively few values filled in across all observations while dense variables are relatively tightly filled out in comparison. The sparse variables of most interest would be those that are typically assigned a relatively large data length while remaining generally unfilled, such as comment fields in a questionnaire. For these types of data, it may be beneficial to split the RDBMS query into one or more queries with one query containing the dense variables and the other query or queries containing only the observations for the sparse variables where the values are not null. These multiple queries can then be recombined in SAS after the data transfer using an appropriate linking key. Again, the most effective method for implementing this is to create a new set of RDBMS queries but alternative versions using PROC SQL pass-through or a SAS/ACCESS libname with appropriate keep and drop statements will also be effective. In most cases, this technique should not be expected to reduce the delay as it is likely that SAS will already have been optimized to try and restructure this type of data, but it is possible that certain types of retrieval requests from certain RDBMS may still experience data transfer delay that this technique can reduce.

FORMATTING DATA

Although it may not possible to find sparse variables for remerging, it may still be possible to find variables with an unnecessarily large data length in the RDBMS query. For analysis, it may only be necessary to have a condensed code for each variable value or it may even be possible to consolidate many values in a single code grouping, allowing for a variable with a shorter formatted data value to be substituted for the original variable. How this reformatting would be implemented depends a great deal on the RDBMS and the nature of the reformatting that must be done. For more extensive reformatting, one effective approach for some RDBMS is to create a formats table and use subqueries on each variable to lookup the appropriate formatted value for the variable's original value. A simple example of this technique for SQL Server is illustrated in Appendix A. Although this may slow the RDBMS query, the resulting reduction in data transfer delay may more than make up for the increased complexity. As with the other techniques, the most effective method for implementing this is to create a new RDBMS query, possibly with other suitable database objects for implementing the formatting, but PROC SQL pass-through is an acceptable alternative.

SUMMARIZATION

Often analysis begins with the need to find a single aggregated value for a variable from all observations that have a matching group of values in other analysis variables. Since only the values for the group of variables and aggregate value are actually needed for analysis, it would be possible to reduce data transfer delay by replacing all observations for each group with a single observation containing the values for the group along with a new variable to hold the aggregated value. Most RDBMS offer a variety of functions for calculating aggregated values from a set of values as well as the capability to create new user-defined functions for calculating such aggregated values. In some cases, the need to work with summarized data throughout the analysis may be so pervasive that building an appropriate data cube may be necessary to help standardize the process.

WEIGHTED SUMMARIZATION

Although all observations may need to be taken account of for analysis, it may be possible to condense several observations together into a single observation for each set of analysis variables by adding an additional frequency variable to indicate how many observations were found for each combination of analysis variables. SAS makes working with such data easier through the inclusion of the FREQ statement (WEIGHT statement for PROC FREQ) to specify how observations should be weighted for most procedures. If the analysis is done using code that does not include this statement, it would still be an easy matter to rebuild the original uncondensed RDBMS data in SAS using a data step. This technique is most effective when combined with some level of format groupings as a large number of essentially duplicate observations could be grouped together into a single observation. As with the other techniques, the most effective method of implementation is to create a new version of your RDBMS query that groups sets of analysis variables while adding the new frequency variable, but PROC SQL pass-through is an acceptable alternative.

COMPLEX SUMMARIZATION

Typically RDBMS also support aggregate functions for calculating the maximum value, minimum value, sum, standard deviation, and statistical variance from a set of values in addition to a simple count of the number of values. Using these functions, it is simple to create a new variable for each statistic with the appropriate aggregated value for each group of analysis variables. However, even if an RDMBS does not offer the particular aggregate function desired, most RDBMS offer some version of programming language for the creation of user-defined functions that can also be used to create an aggregated value from a set of values. Both in-build and user-defined functions can then be used to condense the observations from the RDBMS query into a single observation per group of analysis variables with additional variables added as needed to hold the aggregate value from each aggregate function. Again, the most effective means of implementation is to create a new RDBMS query for performing the grouping and adding the aggregated value, but PROC SQL pass-through is an acceptable alternative.

DATA CUBE

Rather than create a summarized version of each RDBMS query as the need arises during analysis, you may find it more beneficial to construct a data cube containing the presummarized aggregate values for each combination of analysis variables likely to arise during analysis. In addition to avoiding data transfer delays by retrieving a condensed version of the data from the RDBMS, it also offers a performance benefit in that the summarized data is already available prior to the retrieval request rather than needing to be calculated at the same time as the retrieval. However, implementing a data cube for analysis can require a great deal of customization of your RDBMS possibly even necessitating the use of a separate multi-dimensional database that will serve as the repository for the data of your data cube. As this is a complicated issue, you will need to perform further research reviewing the benefits and tradeoffs from such an approach to determine if this option is appropriate for your analysis needs.

DATA COMPRESSION

After failing to find a data reduction or summarization technique that applies to your RDBMS query or failing to find one that is effective in eliminating the data transfer delay, you may then need to turn to methods that require some degree of customization of your SAS and/or RDBMS platforms. A general-purpose approach that does not require the special conditions necessary for data reduction is the usage of a lossless compression algorithm to compact the data on the RDBMS prior to its transmission to the SAS platform for eventual decompression. With these methods, the decrease in transfer delay comes from sending the same data in a greatly compacted format while incurring little time cost from the compression and decompression of the data.

COMPRESSED COMMUNICATION

A first technique would be to replace the communication drivers currently used by your RDBMS and SAS platforms with those that offer more advanced compression capabilities. As this will likely involve an extensive modification to your RDBMS setup and would require a great deal of time and effort to create a driver superior to those provided with your RDBMS, you are unlikely to want to create these replacement communication drivers yourself. However, if no suitable existing replacement driver can be found, this should not necessarily dissuade you from this approach. Fortunately, when creating a communication driver tailored to your needs, it is not necessary to create a fully formed ODBC or OLEDB driver solution. Instead, you can choose to avoid implementing a lot of the unnecessary functionality normally found in those solutions and create your own type of communication driver suited to your needs. You may wish to use this driver only for data transfers from the RDBMS and use your normal communication driver for all other tasks. After the arduous work of implementing the communication protocol is finished, adding a component to first compress your data using your particular compression algorithm should be relatively easy.

FILE TRANSFER
Although you are unlikely to find a compression algorithm available as a RDBMS function that can be run against the data from the RDBMS query, most RDBMS allow data to be exported from the database to local or networked file location. Appendix B contains an example of this process for SQL Server. From there, it is usually a simple matter to access an appropriate operating system command or program on your RDBMS server to compress the data extract. This may be all that is required if your SAS platform has access to the location where the compressed file was created. If not, you can use FTP or other methods to get the compressed data file to your SAS platform for eventual decompression and loading into SAS.

BINARY TRANSFER
This technique is perhaps the least realistic of all that have been offered, but may be useful in certain rare situations. As with the file transfer method, you must first create the compressed file from the exported RDBMS data. However, if your RDBMS provides this capability, this compressed data can then be imported back into the RDBMS as a binary data object. Since SAS cannot work directly with a table containing such an object, it would then be necessary to convert this to a table with a text field by breaking up the data into chunks with a maximum length of 32767 bytes for each value. After the compressed data has been transferred to SAS, it can then be decompressed by outputting each text chunk to a file using binary format. After an identical version of the compressed file has been created on the SAS platform, you can use an appropriate program to decompress the file and then reload the data into SAS. Appendix C extends the compressed file example from Appendix B to illustrate an example of binary transfer for SQL Server.

DATA REPLICATION
Up till this point, all of these techniques have focused on the costly time delay associated with sending data from the RDBMS to SAS at the time of the analysis. However, this is not the only option. If enough information is known about what type of data from the RDBMS will be needed for the eventual analysis and the data is relatively static for an acceptable period of time, it may be feasible to schedule a recurring process to replicate the RDBMS data in an area local to the SAS platform. With this method, the avoidance of transfer delay comes from ensuring the necessary analysis data is already available locally to the SAS platform without needing to query the RDBMS. However, this benefit comes with the overhead of the replication infrastructure including both the mechanism for implementing the replication process and the additional space and security requirements for storing the replicated data. There is also a potential for unsynchronized data to be used during your analysis. Finally, since you might not be sure of the exact needs of the analysis until it is actually run, the replication process may need to transfer more data than is actually necessary.

CONCLUSION
As storage space has become ever cheaper, methods for reducing the size of stored data have come to play an ever smaller role in analysis work. However, the techniques that were developed for this problem have not become outdated. Rather, they have now shifted to a new arena where the challenge has become how to reduce bandwidth usage when transferring data. Despite having a highly efficient RDBMS for data retrieval and a SAS platform optimized for integrating with your RDBMS and performing the analysis, a potential bottleneck still looms when it comes to the transfer of data between the two systems. It is here that you may encounter your greatest time cost, but before more bandwidth is assumed to be the only possible solution, you should run through these variations of some classic techniques to see if they now can be applied to this new problem.

DISCLAIMER
The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

REFERENCES
Bromberg, Peter A., Ph.D. “In-Memory Data Compression in .NET” *EggHeadCafe.com, Articles*. December 2001. <http://www.eggheadcafe.com/articles/20011226.asp> (7/28/2004).

Cohen, John J. “The Dirty Secret Behind the Box: Why Programming Efficiency Should Matter to All of Us” *Proceedings of the Sixteenth Annual NorthEast SAS Users Group Conference*, 16, CD-ROM. Paper AT015. Available <http://www.nesug.org/html/Proceedings/nesug03/at/at015.pdf>.

DataDirect Technologies. “ODBC Basics” *DataDirect Technologies, Articles*. 2004. <http://www.datadirect.com/techzone/odbc/basics/basics/index.ssp> (7/28/2004)

Jorgensen, Greg. “Introduction to Relational Database Systems” *Computer Bits*. April 2003. <http://www.computerbits.com/archive/2003/0400/introdatabases.html> (7/28/2004).

Willcocks, Roger. “Why Use Data Compression for Your Web Service?” *L-Space Design, Software Development Articles*. 2004. <http://www.l-space-design.com/Articles/Why_use_data_compression_in_web_services.aspx> (7/28/2004).

ACKNOWLEDGMENTS
Special thanks to Sigurd Hermansen for his multiple suggestions for this paper, especially data reduction techniques. Just when I was about to create my own method for collapsing repeating rows of analysis variables in the RDBMS and recreating them in SAS, he pointed out the FREQ statement was already available for doing precisely that. I also never would have considered the spare/dense variable split approach until he mentioned it.

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

Francis Harvey
Westat
harveyf1@westat.com

APPENDIX A

DATA REDUCTION – FORMATTING DATA

VARIABLE FORMATS TABLE

VariableName	DataType	FormatName	BeginValue	EndValue	Label
ContactTitle	A	$CNTCTTF	Accounting Manager	Accounting Manager	AMG
ContactTitle	A	$CNTCTTF	Assistant Sales Agent	Assistant Sales Agent	ASA
ContactTitle	A	$CNTCTTF	Assistant Sales Representative	Assistant Sales Representative	ASR
ContactTitle	A	$CNTCTTF	Marketing Assistant	Marketing Assistant	MAS
ContactTitle	A	$CNTCTTF	Marketing Manager	Marketing Manager	MMG
ContactTitle	A	$CNTCTTF	Order Administrator	Order Administrator	OAD
ContactTitle	A	$CNTCTTF	Owner	Owner	OWN
ContactTitle	A	$CNTCTTF	Owner/Marketing Assistant	Owner/Marketing Assistant	OMA
ContactTitle	A	$CNTCTTF	Sales Agent	Sales Agent	SAG
ContactTitle	A	$CNTCTTF	Sales Associate	Sales Associate	SAS
ContactTitle	A	$CNTCTTF	Sales Manager	Sales Manager	SMG
ContactTitle	A	$CNTCTTF	Sales Representative	Sales Representative	SRP

```
-- RETRIEVE LIST OF CUSTOMER NAMES WITH TITLE CODE

SELECT [ContactName],
    ISNULL((
    SELECT [VariableFormats].[Label]
    FROM [Northwind].[dbo].[VariableFormats]
    WHERE ([VariableFormats].[VariableName] = 'ContactTitle')
        AND (([VariableFormats].[BeginValue] = '' AND [Customers].[ContactTitle] IS NULL)
            OR ([VariableFormats].[BeginValue] <= [Customers].[ContactTitle] And
[Customers].[ContactTitle] <= [VariableFormats].[EndValue]))
    ),[ContactTitle]) AS [ContactTitle]
FROM [Northwind].[dbo].[Customers]
```

Appendix B

DATA COMPRESSION – FILE & BINARY TRANSFER

```
-- CREATE COMMA SEPARATE VALUES FILE (CSV) FOR CUSTOMERS TABLE

declare @sql_headers varchar(8000)
-- Retrieve column names for Customers table from SQL Server metadata,
-- surrounding with double quotes and adding a column alias by column position
select @sql_headers=isnull(@sql_headers+',','')+
        'char(34)+'''+column_name+'''+char(34) c'+cast(ordinal_position as varchar)
from information_schema.columns
where table_catalog = 'Northwind'
    and table_schema = 'dbo'
    and table_name='Customers'
-- Create query returning this column headers list
select @sql_headers='select top 1 '+@sql_headers+' from Northwind.dbo.Customers'

-- Retrieve column names for Customers table from SQL Server metadata,
-- placing it in a suitable expression to ensure doubling of single quotes where necessary
-- for the CSV and adding a column alias by column position
declare @sql_data varchar(8000)
select @sql_data=isnull(@sql_data+',','')+
    case data_type
        when 'smallint'
            then 'cast('+column_name+' as varchar)'
        when 'int'
            then 'cast('+column_name+' as varchar)'
        when 'money'
            then 'cast('+column_name+' as varchar)'
        when 'real'
            then 'cast('+column_name+' as varchar)'
        when 'nchar'
            then 'quotename('+column_name+',char(34))'
        when 'nvarchar'
            then 'quotename('+column_name+',char(34))'
        else column_name
    end+' c'+cast(ordinal_position as varchar)
from information_schema.columns
where table_catalog = 'Northwind'
    and table_schema = 'dbo'
    and table_name='Customers'
select @sql_data='select '+@sql_data+' from Northwind.dbo.Customers'

-- Create view for pulling CSV data from Customers table with leading column headers row
declare @sql nvarchar(4000)
if exists (select name from dbo.sysobjects where id =
  object_id(N'[dbo].[vwOutputCustomers]') and OBJECTPROPERTY(id, N'IsView') = 1)
drop view [dbo].[vwOutputCustomers]
select @sql='create view vwOutputCustomers as '+@sql_headers+' UNION ALL '+@sql_data
EXEC sp_executesql @sql

-- Export CSV data from SQL Server to file on local drive
exec master..xp_cmdshell
  'bcp "northwind.dbo.vwOutputCustomers" out "c:\~cust.txt" -c -t "," -T'

-- COMPRESS CSV FILE

-- Use pkzip for compression
EXEC master..xp_cmdshell 'c:\pkzip.exe -ex c:\~cust.zip c:\~cust.txt',no_output
```

Appendix C

COMPRESSION – BINARY TRANSFER – SQL SERVER

```
-- IMPORT COMPRESSED FILE

-- Create procedure to create format specification for loading binary data
if exists (select name from dbo.sysobjects where name =
  N'spFileLoad_CreateFormatFile' AND type = 'P')
drop procedure spFileLoad_CreateFormatFile
GO
CREATE PROCEDURE spFileLoad_CreateFormatFile
 @Size int
AS
BEGIN
 PRINT '8.0'
 PRINT '1'
 PRINT '1 SQLIMAGE 0 ' + CAST( @Size AS varchar(10) ) + ' "" 1 FileContent ""'
END
GO

-- Find size of compressed file
create table #info (
   alt_name            varchar(255) null,
   size_in_bytes       int null,
   creation_date       int null,
   creation_time       int null,
   last_written_date       int null,
   last_written_time       int null,
   last_accessed_date  int null,
   last_accessed_time  int null,
   attributes          int null
)
INSERT INTO #info
EXEC master..xp_getfiledetails 'c:\~cust.zip'
DECLARE @size_in_bytes int
SELECT @size_in_bytes = size_in_bytes
FROM #info
drop table #info

-- Create format specification file
declare @cmd nvarchar(4000)
set @cmd = 'osql -E -Q"Northwind.dbo.spFileLoad_CreateFormatFile ' +
    CAST(@size_in_bytes AS varchar(10)) +
    '" -o c:\~cust.fmt'
exec master..xp_cmdshell @cmd,no_output

-- Create temporary table to hold binary data
if exists (select name from dbo.sysobjects where id =
  object_id(N'[dbo].[temp]') and OBJECTPROPERTY(id, N'IsTable') = 1)
drop table northwind.dbo.temp
create table northwind.dbo.temp (
    FileContent image
)

-- Load binary data into temporary table
exec master..xp_cmdshell
  'bcp northwind.dbo.temp in c:\~cust.zip -T -f c:\~cust.fmt',no_output
```

```
-- CREATE PROCEDURE TO RETRIEVE BINARY DATA IN CHUNKS OF 32767 BYTES

if exists (select name from dbo.sysobjects where name =
  N'spRetrieveCustomers' AND type = 'P')
drop procedure spRetrieveCustomers
GO
CREATE PROCEDURE spRetrieveCustomers
AS
-- Ensure only data sent back to SAS
SET NOCOUNT ON
BEGIN
-- Find size of compressed file
create table #info (
   alt_name            varchar(255) null,
   size_in_bytes       int null,
   creation_date       int null,
   creation_time       int null,
   last_written_date       int null,
   last_written_time       int null,
   last_accessed_date  int null,
   last_accessed_time  int null,
   attributes          int null
)
INSERT INTO #info
EXEC master..xp_getfiledetails 'c:\~cust.zip'
DECLARE @size_in_bytes int
SELECT @size_in_bytes = size_in_bytes
FROM #info
drop table #info

DECLARE @ptrval varbinary(16)
SELECT @ptrval = TEXTPTR(FileContent)
   FROM northwind.dbo.temp
DECLARE
    @offsetval INT,
    @bufferval INT
SELECT @offsetval = 0
SELECT @bufferval = 32767

-- Last chunk, reduce buffer size to the nChars remaining
IF (@offsetval + @bufferval) > @size_in_bytes
    SELECT @bufferval = @size_in_bytes
WHILE @offsetval < @size_in_bytes
BEGIN
    READTEXT northwind.dbo.temp.FileContent @ptrval @offsetval @bufferval
    SELECT @offsetval = @offsetval + @bufferval
    -- Last chunk, reduce buffer size to the get the last nChars remaining
    IF (@offsetval + @bufferval) > @size_in_bytes
    SELECT @bufferval = @size_in_bytes - @offsetval
END
END
GO
```

COMPRESSION – BINARY TRANSFER – SAS

```
/* LOAD BINARY DATA FROM SQL SERVER */

proc sql;
connect to oledb as ncidiet (provider=sqloledb
   properties=("Data Source"=. "Initial Catalog"=Northwind
     'integrated security' = SSPI) dbMax_text=32767);

create table filecontent as
select filecontent
from connection to ncidiet
(exec spRetrieveCustomers);

disconnect from ncidiet;
quit;


/* EXPORT BINARY DATA FROM SAS USING BINARY FORMAT */

data _null_;
   file "c:\windows\temp\~cust.zip" RECFM=N;
   do while (not last);
      set filecontent end=last;
      do i = 1 to length(filecontent) by 2;
         char = byte(input(substr(filecontent,i, 2),hex2.));
         put char 1. @;
      end;
   end;

   stop;
run;


/* DECOMPRESS COMPRESSED FILE */

option xsync noxwait;
x "c:\windows\temp\pkunzip.exe c:\windows\temp\~cust.zip c:\windows\temp";


/* IMPORT CSV FILE */

PROC IMPORT OUT= WORK.temp
            DATAFILE= "C:\windows\temp\~cust.txt"
            DBMS=CSV REPLACE;
     GETNAMES=YES;
     DATAROW=2;
RUN;
```