1 Contents

SQL2GMS

Note: the bitmaps are best viewed with Large Fonts display settings.

2 Overview

SQL2GMS is a tool to convert data from an *SQL database* into GAMS readable format. The source is any data source accessible through Microsoft's Data Access components including ADO, ODBC and OLEDB. The target is a *GAMS Include File* or a GAMS GDX File.

When running the executable *SQL2GMS.EXE* without command line parameters the tool will run interactively with a built-in GUI interface. Alternatively **SQL2GMS** can be run in batch mode which is useful when calling it directly from a GAMS model using the *$call* command.

If you need to read data from an *MS Access database* or *MS Excel spreadsheet* the accompanying tools MDB2GMS or XLS2GMS may be more appropriate. In some cases it may be useful to use **SQL2GMS** to read from Access and Excel data sources.

3 Requirements

SQL2GMS runs only on PC's running Windows (95/98/NT/XP) and with **MDAC** (Microsoft Data Objects) installed. In many cases this will be installed on your computer. Otherwise it can be downloaded from the Microsoft website http://www.microsoft.com/downloads.

To use this tool effectively you will need to have a working knowledge of **SQL** in order to formulate proper database queries. In addition you will need some knowledge on how to connect to your database using ODBC or ADO.

From version 3.0 the help file is in .chm format.

4 Converting database tables to GAMS data

Database tables can be considered as a generalization of a GAMS parameter. GAMS parameters are **single valued** indicating written as a table GAMS parameters have multiple index columns but just one value column. If the table is organized as multi-valued table, a UNION can be used to generate the correct GAMS file.

Besides parameters it is also possible to generate **set** data. The following tables summarizes some of the possibilities:

Single valued table	`parameter d(i,j) /`
	`$include data.inc`
	`/;`
		`SELECT city1,city2,distance`
		`FROM distances`

Multi valued table
 	
Use two separate parameters and queries or a parameter with an extra index position and a UNION select.
	`parameter sales(year,loc,prd)`
	`/`
	`$include sales.inc`
	`/;`
	`parameter profit(year,loc,`
	`prd) /`
	`$include profit.inc`
	`/;`
	
	or
	
	`set typ /sales,profit/`
	`parameter data(year,loc,prd,`
	`typ) /`
	`$include data.inc`
	`/;`
		`SELECT year,loc,prod,sales`
		`FROM data`
		
		`SELECT year,loc,prod,profit`
		`FROM data`
		
		`SELECT year,loc,prod,'sales',`
		`sales`
		`FROM data`
		`UNION`
		`SELECT year,loc,prod,'profit',`
		`profit`
		`FROM data`

Single dimension set
 	
Make sure elements are unique
	`set i /`
	`$include set.inc`
	`/;`
		`SELECT distinct(indexcolumn)`
		`FROM datatable`

Multi dimensional set
 	
Add dummy value field to make sure the elements are separated by a dot or use string concatenation (|| or & depending on DBMS).
	`set ij(i,j) /`
	`$include ij.inc`
	`/;`
		`SELECT indx1,indx2," "`
		`FROM datatable`
		
		`SELECT indx1&'.'&indx2`
		`FROM datatable`
		
		`SELECT indx1||'.'||indx2`
		`FROM datatable`

There are no special requirements on the data types used in the database. The data are converted to strings, which is almost always possible. Data types like LONG BINARY may not be convertible to a string, in which case an exception will be raised. In general NULL's should not be allowed to get into a GAMS data structure. The handling of NULL's can be specified in an option.

4.1 Example: single-valued tables

Example: single-valued table

Consider the simple example table:

table: **distances**

City1	City2	Distance
SEATTLE	NEW-YORK	2.5
SAN-DIEGO	NEW-YORK	2.5
SEATTLE	CHICAGO	1.7
SAN-DIEGO	CHICAGO	1.8
SEATTLE	TOPEKA	1.8
SAN-DIEGO	TOPEKA	1.4

with the accompanying query:

```
SELECT City1,City2,Distance
FROM Distances
```

This can be represented in GAMS as:

```
set i /seattle, san-diego/;
set j /new-york, chicago, topeka/;
parameter dist(i,j) 'distances' /
$include distances.inc
/;
```

where the include file **distances.inc** has been generated using the above query. This file can look like:

```
* ----------------------------------------------------
* SQL2GMS Version 3.0, November 2006
* Erwin Kalvelagen, GAMS Development Corp
* ----------------------------------------------------
* ADO version: 2.8
* Connection string: Driver={Microsoft Access Driver (*.mdb)}; Dbq=d:
\sql2gms\sample.mdb
* Provider: MSDASQL
* Query: SELECT city1, city2, distance
*        FROM distances
* ----------------------------------------------------
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* ----------------------------------------------------
```

The standard export format is to consider the last column the value column and the previous columns as the indices. The indices are separated by a dot, allowing the generated include file to be used as part of a **parameter** declaration statement.

4.2   Example: Multi-valued tables

Consider the table with two value columns:

TABLE:**DATA**

year	loc	prod	sales	profit
1997	la	hardware	80	8
1997	la	software	60	16

1997	nyc	hardware	110	5
1997	nyc	software	100	10
1997	sfo	hardware	80	9
1997	sfo	software	50	10
1997	was	hardware	120	7
1997	was	software	70	20
1998	la	hardware	70	6
1998	la	software	70	10
1998	nyc	hardware	120	7
1998	nyc	software	120	14
1998	sfo	hardware	90	12
1998	sfo	software	70	15
1998	was	hardware	130	12
1998	was	software	80	15

A simple way to import this into GAMS is to use two parameters and two SQL queries. The SQL queries can look like:

```
SELECT year,loc,prod,sales
FROM data

SELECT year,loc,prod,profit
FROM data
```

If the results are stored in include files **sales.inc** and **profit.inc** then this can be read into GAMS as follows:

```
parameter sales(year,loc,prd) /
$include sales.inc
/;
parameter profit(year,loc,prd) /
$include profit.inc
/;
```

The operation can also be performed in one big swoop by using a different GAMS datastructure:

```
set typ /sales,profit/
parameter data(year,loc,prd,typ) /
$include data.inc
/;
```

This parameter has an extra index **typ** which indicates the data type. To generate the correct include file we can use the following query:

```
SELECT year,loc,prod,'sales',sales
FROM data
UNION
SELECT year,loc,prod,'profit',profit
FROM data
```

The generated include file will look like:

```
* ------------------------------------------------
* SQL2GMS Version 3.0, November 2006
```

```
* Erwin Kalvelagen, GAMS Development Corp
* ------------------------------------------------------
* ADO version: 2.8
* Connection string: Driver={Microsoft Access Driver (*.mdb)}; Dbq=d:
\sql2gms\sample.mdb
* Provider: MSDASQL
* Query: SELECT year,loc,prod,'sales',sales
*        FROM data
*        UNION
*        SELECT year,loc,prod,'profit',profit
*        FROM data
* ------------------------------------------------------
1997.la.hardware.profit 8
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit 9
1997.sfo.hardware.sales 80
1997.sfo.software.profit 10
1997.sfo.software.sales 50
1997.was.hardware.profit 7
1997.was.hardware.sales 120
1997.was.software.profit 20
1997.was.software.sales 70
1998.la.hardware.profit 6
1998.la.hardware.sales 70
1998.la.software.profit 10
1998.la.software.sales 70
1998.nyc.hardware.profit 7
1998.nyc.hardware.sales 120
1998.nyc.software.profit 14
1998.nyc.software.sales 120
1998.sfo.hardware.profit 12
1998.sfo.hardware.sales 90
1998.sfo.software.profit 15
1998.sfo.software.sales 70
1998.was.hardware.profit 12
1998.was.hardware.sales 130
1998.was.software.profit 15
1998.was.software.sales 80
* ------------------------------------------------------
```

4.3   Index mapping

In some cases the index elements used in the database are not the same as in the GAMS model. E.g. consider the case where the GAMS model has defined a set as:

```
set i /NY,DC,LA,SF/;
```

Now assume a data table looks like:

TABLE:**EXAMPLE TABLE**

city	value

new york	100
los angeles	120
san francisco	105
washington dc	102

This means we have to map 'new york' to 'NY' etc. This mapping can be done in two places: either in GAMS or in the database.

When we export the table directly (with the option '*Quote Blanks*' turned on), we get:

```
* ----------------------------------------------------
* SQL2GMS Version 3.0, November 2006
* Erwin Kalvelagen, GAMS Development Corp
* ----------------------------------------------------
* ADO version: 2.8
* Connection string: Driver={Microsoft Access Driver (*.mdb)}; Dbq=d:
\sql2gms\sample.mdb
* Provider: MSDASQL
* Query: SELECT city, value
*         FROM [example table]
* ----------------------------------------------------
'new york' 100
'los angeles' 120
'san francisco' 105
'washington dc' 102
* ----------------------------------------------------
```

As the index elements contain blanks, the option **Quote Blanks** was used. To import this file and convert it to a different index space we can use the following GAMS code:

```
set i /NY,DC,LA,SF/;

set idb 'from database' /
   'new york',
   'washington dc',
   'los angeles',
   'san francisco'
/;

parameter dbdata(idb) /
$include data.inc
/;

set mapindx(i,idb) /
    NY.'new york'
    DC.'washington dc'
    LA.'los angeles'
    SF.'san francisco'
/;

parameter data(i);
data(i) = sum(mapindx(i,idb), dbdata(idb));
display data;
```

The second approach is to handle the mapping inside the database. We can introduce a mapping table that looks like:

TABLE:**MAPCITY**

city	gcity
new york	la
los angeles	ny
san francisco	sf
washington dc	dc

This table can be used in a join to export the data in a format we can use by executing the query:

```
SELECT gcity, value
FROM [example table],mapcity
WHERE [example table].city=mapcity.city
```

The GAMS import step could look like:

```
set i /NY,DC,LA,SF/;

parameter data(i) /
$include data.inc
/;
display data;
```

where the data file looks like:

```
* ----------------------------------------------------
* SQL2GMS Version 3.0, November 2006
* Erwin Kalvelagen, GAMS Development Corp
* ----------------------------------------------------
* ADO version: 2.8
* Connection string: Driver={Microsoft Access Driver (*.mdb)}; Dbq=d:
\sql2gms\sample.mdb
* Provider: MSDASQL
* Query: SELECT gcity, value
*        FROM [example table], mapcity
*        WHERE [example table].city=mapcity.city
* ----------------------------------------------------
la 120
ny 100
sf 105
dc 102
* ----------------------------------------------------
```

Note: MS Access allows table names with embedded blanks. In that case the table name can be surrounded by square brackets. Other databases may not allow this.

5   Connection Strings

The connection string determines to which database the tool will try to connect. You can give simply the name of an ODBC Data Source or provide much more elaborate connection strings. Here is an example list:

ODBC Examples

ODBC Data Source	`MyDSN`
ODBC Data Source	`DSN=MyDSN`
ODBC DSN with userid and password	`DSN=`*xxx*`;UID=`*yyy*`;PWD=`*zzz*`;`
ODBC File DSN	`FILEDSN=`*d:\ppp\fff.dsn*`;UID=`*yyy*`;PWD=`*zzz*`;`

ODBC DSN-less Text Driver	`Driver={Microsoft Text Driver (*.txt; *.csv)};Dbq=`*d:\ppp*`;`
	`Extensions=asc,csv,tab,txt;Persist Security Info=False`
	
	(Note: the filename is used in the `FROM` clause in the query string.)

ODBC DSN-less MS Access Driver	`Driver={Microsoft Access Driver (*.mdb)};Dbq=`*d:\ppp\fff.* *mdb*`;` `Uid=`*yyy*`;Pwd=`*zzz*`;`
ODBC DSN-less driver for MS SQL Server	`Driver={SQL Server};Server=`*sss*`;Database=`*ddd*`;Uid=`*yyy*`;Pwd=` *zzz*`;`
ODBC Driver for Oracle	`Driver={Microsoft ODBC for Oracle};Server=`*sss*`;Uid=`*yyy*`;Pwd=` *zzz*

ODBC Driver for Oracle (old)	`Driver={Microsoft ODBC Driver for Oracle};ConnectString=`
	sss`;`
	`Uid=`*yyy*`; Pwd=`*zzz*`;`

ODBC driver for MySQL	`DRIVER={MySQL ODBC 3.51 Driver};SERVER=localhost;`
	`DATABASE=test; UID=xxx;PWD=yyy;OPTION=3`
	
	See http://www.mysql.com/products/myodbc/manual_toc.html

ODBC driver for Interbase 6/ Firebird	`DRIVER={XTG Systems InterBase6 ODBC driver};DB=localhost:`
	`d:\gams projects\sql2gms\ver2.0\ib.gdb;UID=xxx;PWD=yyy`
	
	See http://www.xtgsystems.com

OLE DB Examples

OLE DB Data link file	`File name=`*d:\ppp\fff.udl*`;`
OLE DB Provider for ODBC Access (Jet)	`Provider =MSDASQL; Driver={Microsoft Access Driver (*.` `mdb)};Dbq= `*d:\ppp\fff.mdb*`; Uid=`*yyy*`; Pwd=`*zzz*`;`
OLE DB Provider for ODBC SQL Server	`Provider=MSDASQL; Driver={SQL Server}; Server=`*sss*`;` `Database= `*ddd*`; Uid=`*yyy*`; Pwd=`*zzz*`;`
OLE DB Provider for Microsoft Jet (Access)	`Provider=Microsoft.Jet.OLEDB.4.0; Data Source=`*d:\ppp\fff.* *mdb*`; User Id=`*yyy*`; Password=`*zzz*`;`
OLE DB Provider for Microsoft Jet (Access) with System Database	`Provider=Microsoft.Jet.OLEDB.4.0; Data Source=`*d:\ppp\fff.* *mdb*`; Jet OLEDB:System Database=`*fff.mdw*`;` Specify user id and password in the U=*xxx* and P=*zzz* SQL2GMS options. If MDB file has a password, add: Jet OLEDB:Database Password= *xxx*;.

OLE DB Provider for SQL Server	`Provider=sqloledb; Network Library=DBMSSOCN; Data `
	`Source=`*ddd* `; Initial Catalog=`*ccc*`; User Id=`*yyy*`; Password=`
	zzz`;`

OLE DB Provider for SQL Server with trusted connection security	`Provider=sqloledb; Network Library=DBMSSOCN; Data ` `Source=`*ddd*`; Initial Catalog=`*ccc*`; Trusted_Connection=yes;`

MS Remote Examples

MS Remote - Access (Jet) through ODBC DSN	`Remote Server=`*http://xxx*`; Remote Provider=MSDASQL; DSN=` *nnn*`; Uid=`*yyy*`; Pwd=`*zzz*`;`
MS Remote - Access (Jet) through OLE DB Provider	`Provider=MS Remote; Remote Server=`*http://xxx*`; Remote ` `Provider=Microsoft.Jet.OLEDB.4.0; Data Source=`*d:\ppp\fff.*

	`mdb; Uid=`*yyy*`; Pwd=`*zzz*`;`

MS Remote - Access (Jet) through OLE DB Provider with an RDS Datafactory Custom Handler	`Provider=MS Remote; Remote Server=`*http://xxx*`;`
	
	`Handler=MSDFMAP.Handler; Data Source=MyConnectTag`
	
	The entry in `\winnt\Msdfmap.ini` **is:**
	`[connect MyConnectTag]`
	`Access=ReadWrite`
	`Connect="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=xxx.`
	`mdb; User Id=yyy; Password=zzz;"`

MS Remote - SQL Server using ODBC DSN	`Remote Server=`*http://xxx*`; Remote Provider=MSDASQL; Network` `Library=DBMSSOCN; DSN=`*nnn*`; Uid=`*yyy*`; Pwd=`*zzz*`;`

MS Remote - SQL Server using OLE DB Provider	`Provider=MS Remote; Remote Server=`*http://xxx*`; Remote`
	`Provider=SQLOLEDB; Network Library=DBMSSOCN; Data Source=`
	nnn`; Initial Catalog=`*ddd*`; User Id=`*yyy*`; Password=`*zzz*`;`

MS Remote - SQL Server through OLE DB Provider with an RDS Datafactory Custom Handler	`Provider=MS Remote; Remote Server=`*http://xxx*`;`
	
	`Handler=MSDFMAP.Handler; Data Source=MyConnectTag`
	
	The entry in `\winnt\Msdfmap.ini` **is:**
	`[connect MyConnectTag]`
	`Access=ReadWrite`
	`Connect="Provider=SQLOLEDB; Network Library=DBMSSOCN; Data`
	`Source=nnn; Initial Catalog=ddd; User Id=yyy;`
	`Password=zzz;"`

For more information consult the documentation with your database driver. ODBC drivers can be had from several sources: Micrsoft delivers ODBC with a number of drivers; database providers may have likely an ODBC driver for their RDBMS available and finally there are a number of third party ODBC drivers available (e.g. from http://ww.easysoft.com).

5.1   ODBC Examples

In this section we show a few examples using SQL2GMS with ODBC data sources.

> ODBC Driver Manager
> Reading from an MS Access database
> Reading from an MS Excel spreadsheet
> Reading from a Text File

6   ODBC Driver Manager

To configure ODBC data sources use the *ODBC Data Source Administrator*. This tool can be invoked from the Start button menu: *Settings|Control Panel*, and click on the *ODBC Data Sources* icon:

Under Windows XP the sequence is: *Control Panel|Performance and Maintenance|Administrative Tools* and click on the *Data Sources (ODBC)* icon
The ODBC Data Source Administrator tool looks like:
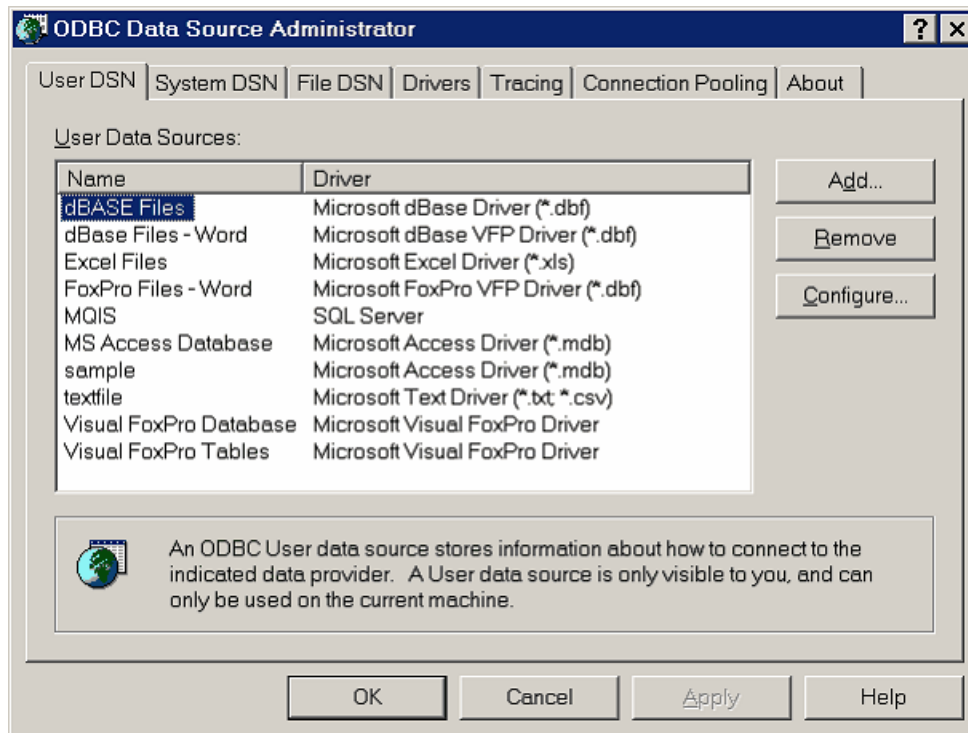
To create a new data source, click the Add button, select a driver, give it a name (this is the DSN name) and configure the data source.

7 Reading from an MS Access Database

There are several ways to export data from an SQL database into a GAMS include file:

1. Export a CSV (Comma Separated Values) file using Access Export. See http://www.gams.com/~erwin/interface/interface.html for an example.
2. Use the MDB2GMS tool.
3. Use SQL2GMS with a configured ODBC data source. The connection string will look like: " `DSN=mydsn`".
4. Use SQL2GMS with a DSN-less ODBC connection. In this case we need to specify both the driver and the location of the database file explicitly in the connection string. The connection string will look like:
   "`Driver={Microsoft Access Driver (*.mdb)};Dbq=D:\data\mydata.mdb`".
5. Use SQL2GMS with an OLE DB driver. The connection string can look like:
   "`Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\data\mydata.mdb`"

8 Reading from an MS Excel spreadsheet

There are numerous ways to export data from an Excel spreadsheet into a GAMS include file:

1. Export a CSV (Comma Separated Values) file using Excel Export. See http://www.gams.com/~erwin/interface/interface.html for an example.

2. Use the XLS2GMS tool.
3. Use the GDXXRW tool. See http://www.gams.com/docs/excel/
4. Use SQL2GMS with an Excel ODBC connection. An example is shown below.

Consider the spreadsheet:

After configuring a data source *ExcelDist* that uses the Excel ODBC driver and points to the .XLS file containing the above sheet, we can use the connection string: "DSN=ExcelDist". With the database browser we see:
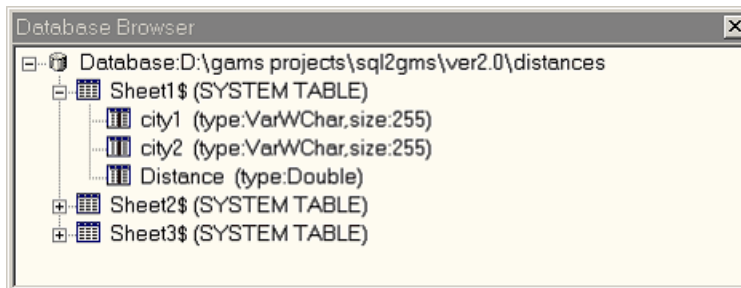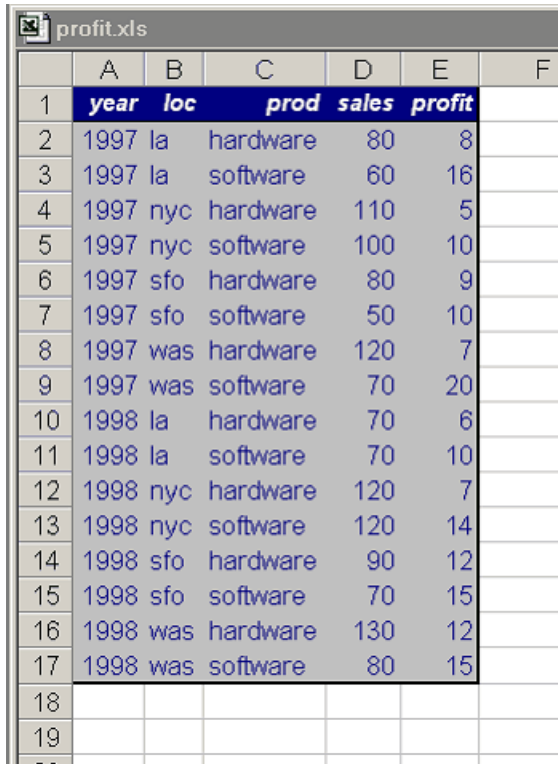

I.e. the table name is *Sheet1$*. We now can formulate the query: SELECT city1,city2,distance FROM [Sheet1$]. We need the brackets to protect the funny table name. The result is:

```
* -------------------------------------------------
* SQL2GMS Version 2.0, January 2004
* Erwin Kalvelagen, GAMS Development Corp
* -------------------------------------------------
* ADO version:       2.7
* Connection string: DSN=ExcelDist
* Query:             SELECT city1,city2,distance
*                    FROM [sheet1$]
* Provider:          MSDASQL
* -------------------------------------------------
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -------------------------------------------------
```

Although other tools are often more convenient to use, this approach is useful if you need to select a subsection of the spreadsheet table. It is easy to select just a few columns or rows from a table using a properly formulated SQL query. The skeleton would be: SELECT *columns_to_extract* FROM [sheet1$] WHERE *rows_to_extract*.

An example of a more complex spreadsheet is:

A DSN-less connection string would be: "`DRIVER=Microsoft Excel Driver (*.xls); DBQ=d:\gams projects\sql2gms\ver2.0\profit.xls`". The browser will show:
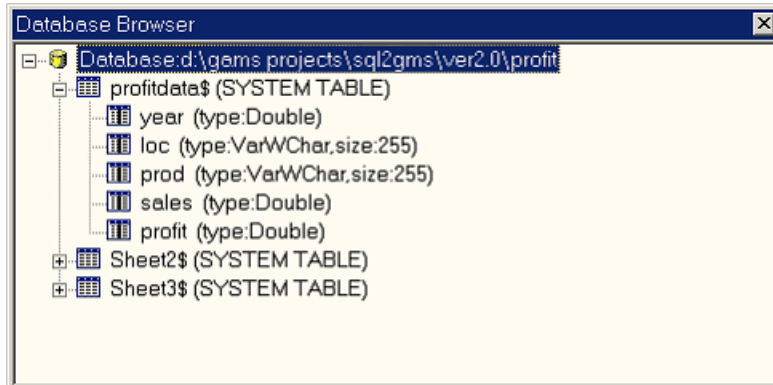
A possible query that maps the two value columns into a GAMS parameter is:

```
SELECT year,loc,prod,'sales',sales
FROM [profitdata$]
UNION
SELECT year,loc,prod,'profit',profit
FROM  [profitdata$]
```

The result is:

```
* ------------------------------------------------
```

```
* SQL2GMS Version 2.0, January 2004
* Erwin Kalvelagen, GAMS Development Corp
* ---------------------------------------------------
* ADO version:        2.7
* Connection string: DRIVER=Microsoft Excel Driver (*.xls);dbq=d:\gams
projects\sql2gms\ver2.0\profit.xls;
* Query:              SELECT year,loc,prod,'sales',sales
*                     FROM [profitdata$]
*                     UNION
*                     SELECT year,loc,prod,'profit',profit
*                     FROM  [profitdata$]
* Provider:           MSDASQL
* ---------------------------------------------------
1997.la.hardware.profit 8
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit 9
1997.sfo.hardware.sales 80
1997.sfo.software.profit 10
1997.sfo.software.sales 50
1997.was.hardware.profit 7
1997.was.hardware.sales 120
1997.was.software.profit 20
1997.was.software.sales 70
1998.la.hardware.profit 6
1998.la.hardware.sales 70
1998.la.software.profit 10
1998.la.software.sales 70
1998.nyc.hardware.profit 7
1998.nyc.hardware.sales 120
1998.nyc.software.profit 14
1998.nyc.software.sales 120
1998.sfo.hardware.profit 12
1998.sfo.hardware.sales 90
1998.sfo.software.profit 15
1998.sfo.software.sales 70
1998.was.hardware.profit 12
1998.was.hardware.sales 130
1998.was.software.profit 15
1998.was.software.sales 80
* ---------------------------------------------------
```

9   Reading text files

Microsoft delivers ODBC with a text file driver which allows you to read a text file as if it is database table.
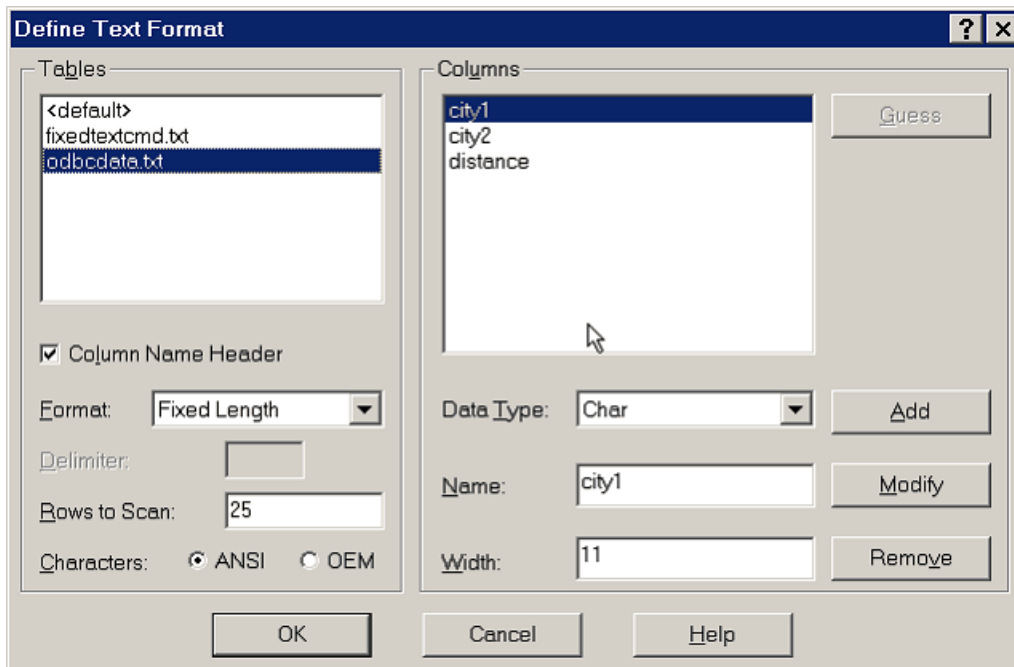
A fixed format file such as:

```
City1      City2      Distance
SEATTLE    NEW-YORK   2.5
SAN-DIEGO  NEW-YORK   2.5
SEATTLE    CHICAGO    1.7
```

```
SAN-DIEGO   CHICAGO    1.8
SEATTLE     TOPEKA     1.8
SAN-DIEGO   TOPEKA     1.4
```

can be read using Fixed Length setting of the text driver:

The resulting include file will look like:

```
* -----------------------------------------------------
* SQL2GMS Version 2.0, January 2004
* Erwin Kalvelagen, GAMS Development Corp
* -----------------------------------------------------
* ADO version:       2.7
* Connection string: DSN=text
* Query:             select city1,city2,distance from odbcdata.txt
* Provider:          MSDASQL
* -----------------------------------------------------
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -----------------------------------------------------
```
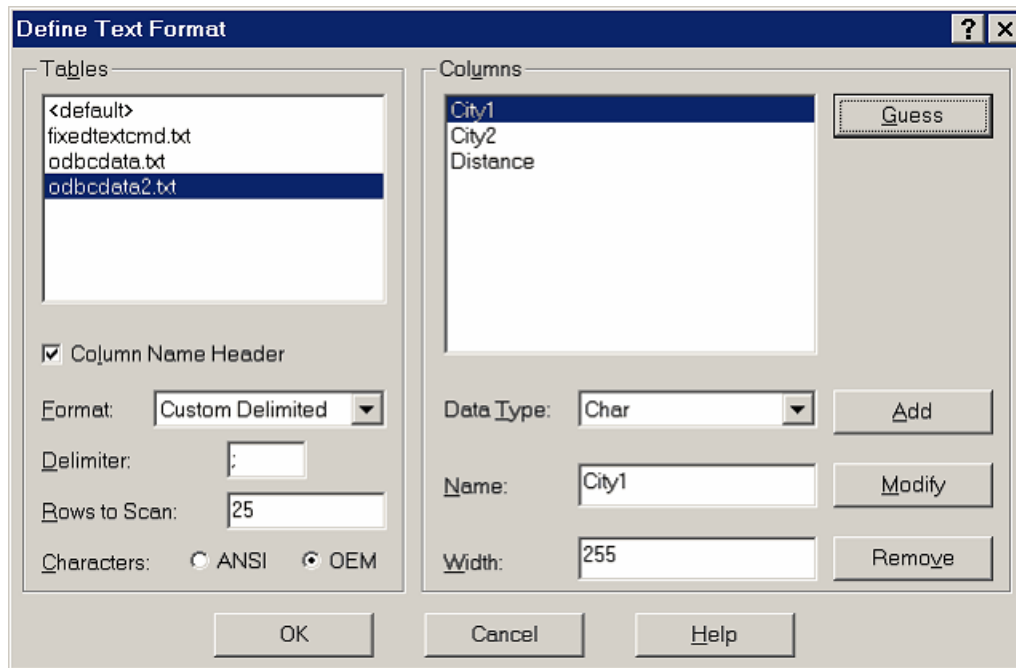
A CSV file can be interpreted as a table as well, or any other separated format. We will try to read:

```
City1;City2;Distance
SEATTLE;NEW-YORK;2.5
SAN-DIEGO;NEW-YORK;2.5
SEATTLE;CHICAGO;1.7
SAN-DIEGO;CHICAGO;1.8
SEATTLE;TOPEKA;1.8
SAN-DIEGO;TOPEKA;1.4
```

This can be read using:

The actual formats used are stored by ODBC in an INI file **schema.ini** (located in the directory of the data files) which can be inspected directly:

```
[odbcdata.txt]
ColNameHeader=True
Format=FixedLength
MaxScanRows=25
CharacterSet=ANSI
Col1=city1 Char Width 11
Col2=city2 Char Width 11
Col3=distance Float Width 10

[odbcdata2.txt]
ColNameHeader=True
Format=Delimited(;)
MaxScanRows=25
CharacterSet=OEM
Col1=City1 Char Width 255
Col2=City2 Char Width 255
Col3=Distance Float
```

10   Interactive use

When the tool is called without command line parameters, it will startup interactively. Using it this way, one can specify options such as the connection string, the query and the final destination file (a GAMS include file or a GDX file) using the built-in interactive environment. The main screen contains a number of buttons and edit boxes, which are explained below.

Output GAMS Include file (*.INC). If you want to create a GAMS include file, then specify here the destination file.

The include file will be an ASCII file that can be read by GAMS using the *$include* command. If the include file already exists, it will be overwritten.

Output: GDX file (.gdx)   💾 Browse...

e:\gamsmodels\data.gdx

Output GDX file (*.GDX) As an alternative to a GAMS include file, the tool can also generate a GDX file. If the GDX file already exists it will be overwritten – it is not possible to append to a GDX file. One or both of the output files need to be specified.

SQL query

SELECT city1,city2,distance
FROM transport

SQL Query The SQL Query box is the place to provide the query. Note that the actual area for text can be larger than is displayed: use the cursor-keys to scroll. The query is passed on directly to the RDBMS so the complete power and expressiveness of SQL is available including stored procedures etc. For an exact description of allowed expressions consult a text on your database system.

ADO version: 2.7
Number of rows: 32
Elapsed time: 0.16 seconds
Done

Progress Memo. This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.

▼

The edit boxes above all have a drop down list, which can be used to access quickly file names and queries that have been used earlier (even from a previous session).

🗄 Tables

The **tables button** will pop up a new window with the contents of the database file selected in the input file edit line. This allows you to see all table names and field names needed to specify a correct SQL query. An exception will be generated if no database file name is specified in the input edit line.

🔍 Options

The **options button** will pop up a window where you can specify a number of options. The connection string is an important option, which needs to be set correctly before a query can be submitted successfully.

? Help

The **help button** will show this help.

✓ OK
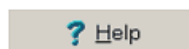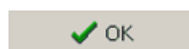
If the **OK button** is pressed the query will be executed and an include file or GDX file will be generated.

Batch

Pressing the **batch button** will give information on how the current query can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied to the clipboard. In the IDE press Ctrl-C or choose Edit|Paste to copy the contents of the clipboard to a GAMS text file.

Close

The **close button** will exit the application. The current settings will be saved in an INI file so when you run SQL2GMS again all current settings will be restored.

11   Options

The **Options** window can be created by pressing the options button:
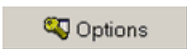
Options

The following options are available in the options window:

User name: Here you can specify the user name for logging in to the RDBMS. For databases without user authentication, this can be left empty.

Password: this edit box allows you to specify the password for the database system. The characters are echoed as a '*'.

Connection String: The connection string determines how SQL2GMS talks to the database. For more information see Connection Strings.

ODBC Data Sources/Drivers: this drop down list can be used to compose a connection string when an ODBC data source or driver is involved. The list will show all configured data sources and all available ODBC drivers.

Quote blanks: Quote strings if they contain blanks or embedded quotes. If a string does not contain a blank or embedded quotes, it will remain unquoted. If the string contains a single quote the quotes applied will be double quotes and if the string contains already a double quote, the surrounding quotes will be single quotes. (In the special case the string contains both, double quotes are replaced by single quotes). For more information see this example. This option only applies to an output include file.

Mute: Don't include the extra informational text (such as used query etc.) in the include file.

No listing: Surround the include file by *$offlisting* and *$onlisting* so that the data will not be echoed to the listing file. This can be useful for very large data files, where the listing file would

become too large to be handled comfortably.

Format SQL: If an SQL text is reloaded in the SQL Edit Box, it will be formatted: keywords will be printed in CAPS and the FROM and WHERE clause will be printed on their own line. If this check box is unchecked this formatting will not take place and SQL queries will be shown as is.

The following options are only needed in special cases:

NULL: This radio box determines how NULL's are handled. A NULL in an index position or a value column will usually make the result of the query useless: the GAMS record will be invalid. To alert you on NULL's the default to throw an exception is a safe choice. In special cases you may want to map NULL's to an empty string or a 'NULL' string.

Time-out values for connection time and command execution time expressed in seconds. If -1 is specified then it will the use the default value which is 15 seconds for the connection and 30 for the commands. If you set the value to zero, ADO will wait indefinitely.

The buttons have an obvious functionality:
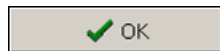
The **OK button** will accept the changes made.

The **Cancel button** will ignore the changes made, and all option settings will revert to their previous values.
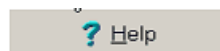
The **Help button** will show this help text.

Test Connection will try to make a connection to the database using the given connection string. If the settings are correct you will see something like:

The following options can only be specified in an ini file; there is no interactive equivalent:

Key	Type	Meaning
D		Generate debug output
E		Allow an empty result set; without this option an empty result set will generate an error
R	*integer*	Row batch size; the default is 100 records

12   Batch use

When calling **SQL2GMS** directly from GAMS we want to specify all command and options directly from the command line or from a command file. An example is:

```
C:\tmp> sql2gms C="DSN=sample" O=C:\tmp\data.inc Q="select City1,City2,Distance
from Distances"
```

This call will perform its task without user intervention. The batch facility can be used from inside a GAMS model, e.g.:

```
parameter c(i,j) 'data from database' /
$call =sql2gms C="DSN=sample" O=C:\tmp\data.inc Q="select City1,City2,Distance from
Distances"
$include C:\tmp\data.inc
/;
```

The $call statement is rather error prone and you will need to spend a little it of time to get the call correct and reliable.

All the possible command line options are listing in command line arguments section. A proper **batch** call will at least contain the following command line parameters:
1. C=*connectionstring*
2. O=*outputincludefile* or X=*outputgdxfile*
3. Q=*querystring*

13   Example

Consider the database table:

DATA

year	loc	prod	sales	profit
1997	la	hardware	80	8
1997	la	software	60	16
1997	nyc	hardware	110	5
1997	nyc	software	100	10
1997	sfo	hardware	80	9
1997	sfo	software	50	10
1997	was	hardware	120	7
1997	was	software	70	20
1998	la	hardware	70	6

1998	la	software	70	10
1998	nyc	hardware	120	7
1998	nyc	software	120	14
1998	sfo	hardware	90	12
1998	sfo	software	70	15
1998	was	hardware	130	12
1998	was	software	80	15

We want to extract the following information:

- The set **year**
- The set **loc**
- The parameter **sales**
- The parameter **profit**

This can be accomplished using the following GAMS code:

```
set y 'years' /
$call =sql2gms C="DSN=sample" Q="select distinct(year) from data" O="D:\data\years.inc"
$include d:\data\years.inc
/;

set loc 'locations' /
$call =sql2gms C="DSN=sample" Q="select distinct(loc) from data" O="D:\data\locations.inc"
$include d:\data\locations.inc
/;

set prd 'products' /hardware, software/;

parameter sales(prd,loc,y) /
$call =sql2gms C="DSN=sample" Q="select prod,loc,year,sales from data" O="d:\data\sales.
inc"
$include d:\data\sales.inc
/;
display sales;

parameter profit(prd,loc,y) /
$call =sql2gms C="DSN=sample" Q="select prod,loc,year,profit from data" O="d:\data\profit.
inc"
$include d:\data\profit.inc
/;
display profit;
```

The $CALL statements assume that **sql2gms.exe** is in the path. This can be achieved by placing **sql2gms.exe** in GAMS system directory (the directory where also **gams.exe** is located; the directory can be easily located in the IDE by looking at GAMS executable path in File|Options|Execute). If **sql2gms.exe** is not in the search path, you can call it explicitly as:

```
$call ="D:\util\sql2gms.exe" C="DSN=sample" Q="select prod,loc,year,profit from data" O="d:
\data\profit.inc"
```

Example **21**

The resulting listing file will show:

```
   1  set y 'years' /
INCLUDE    d:\data\years.inc
   4  * ------------------------------------------------------
   5  * SQL2GMS Version 2.0, January 2004
   6  * Erwin Kalvelagen, GAMS Development Corp
   7  * ------------------------------------------------------
   8  * ADO version:       2.7
   9  * Connection string: DSN=sample
  10  * Query:             select distinct(year) from data
  11  * Provider:          MSDASQL
  12  * ------------------------------------------------------
  13   1997
  14   1998
  15  * ------------------------------------------------------
  16  /;
  17
  18  set loc 'locations' /
INCLUDE    d:\data\locations.inc
  21  * ------------------------------------------------------
  22  * SQL2GMS Version 2.0, January 2004
  23  * Erwin Kalvelagen, GAMS Development Corp
  24  * ------------------------------------------------------
  25  * ADO version:       2.7
  26  * Connection string: DSN=sample
  27  * Query:             select distinct(loc) from data
  28  * Provider:          MSDASQL
  29  * ------------------------------------------------------
  30   la
  31   nyc
  32   sfo
  33   was
  34  * ------------------------------------------------------
  35  /;
  36
  37  set prd 'products' /hardware, software/;
  38
  39  parameter sales(prd,loc,y) /
INCLUDE    d:\data\sales.inc
  42  * ------------------------------------------------------
  43  * SQL2GMS Version 2.0, January 2004
  44  * Erwin Kalvelagen, GAMS Development Corp
  45  * ------------------------------------------------------
  46  * ADO version:       2.7
  47  * Connection string: DSN=sample
  48  * Query:             select prod,loc,year,sales from data
  49  * Provider:          MSDASQL
  50  * ------------------------------------------------------
  51  hardware.la.1997 80
  52  software.la.1997 60
  53  hardware.nyc.1997 110
  54  software.nyc.1997 100
  55  hardware.sfo.1997 80
  56  software.sfo.1997 50
  57  hardware.was.1997 120
  58  software.was.1997 70
  59  hardware.la.1998 70
  60  software.la.1998 70
  61  hardware.nyc.1998 120
  62  software.nyc.1998 120
  63  hardware.sfo.1998 90
  64  software.sfo.1998 70
  65  hardware.was.1998 130
  66  software.was.1998 80
```

```
 67  * ------------------------------------------------------
 68  /;
 69  display sales;
 70
 71  parameter profit(prd,loc,y) /
INCLUDE    d:\data\profit.inc
 74  * ------------------------------------------------------
 75  * SQL2GMS Version 2.0, January 2004
 76  * Erwin Kalvelagen, GAMS Development Corp
 77  * ------------------------------------------------------
 78  * ADO version:      2.7
 79  * Connection string: DSN=sample
 80  * Query:            select prod,loc,year,profit from data
 81  * Provider:         MSDASQL
 82  * ------------------------------------------------------
 83  hardware.la.1997 8
 84  software.la.1997 16
 85  hardware.nyc.1997 5
 86  software.nyc.1997 10
 87  hardware.sfo.1997 9
 88  software.sfo.1997 10
 89  hardware.was.1997 7
 90  software.was.1997 20
 91  hardware.la.1998 6
 92  software.la.1998 10
 93  hardware.nyc.1998 7
 94  software.nyc.1998 14
 95  hardware.sfo.1998 12
 96  software.sfo.1998 15
 97  hardware.was.1998 12
 98  software.was.1998 15
 99  * ------------------------------------------------------
100  /;
101  display profit;
```

Indeed the includes contain the correct data sets. The include file summary shows that the $CALL and $INCLUDE statements were executed without errors:

```
Include File Summary


   SEQ    GLOBAL TYPE       PARENT    LOCAL  FILENAME

     1         1 INPUT          0        0  D:\gams projects\sql2gms\ver2.0\Untitled_1.gms
     2         2 CALL           1        2  =sql2gms C="DSN=sample" Q="select distinct(year)
from data" O="D:\data\years.i
                                            nc"
     3         3 INCLUDE        1        3  .d:\data\years.inc
     4        19 CALL           1        7  =sql2gms C="DSN=sample" Q="select distinct(loc) from
data" O="D:\data\location
                                            s.inc"
     5        20 INCLUDE        1        8  .d:\data\locations.inc
     6        40 CALL           1       14  =sql2gms C="DSN=sample" Q="select prod,loc,year,
sales from data" O="d:\data\sa
                                            les.inc"
     7        41 INCLUDE        1       15  .d:\data\sales.inc
     8        72 CALL           1       20  =sql2gms C="DSN=sample" Q="select prod,loc,year,
profit from data" O="d:\data\p
                                            rofit.inc"
     9        73 INCLUDE        1       21  .d:\data\profit.inc
```

Finally the DISPLAY statements show that the data arrived correctly in the required format:

```
----      69 PARAMETER sales
```

Example | **23**

```
                        1997          1998

      hardware.la        80.000        70.000
      hardware.nyc      110.000       120.000
      hardware.sfo       80.000        90.000
      hardware.was      120.000       130.000
      software.la        60.000        70.000
      software.nyc      100.000       120.000
      software.sfo       50.000        70.000
      software.was       70.000        80.000


----      101 PARAMETER profit

                        1997          1998

      hardware.la         8.000         6.000
      hardware.nyc        5.000         7.000
      hardware.sfo        9.000        12.000
      hardware.was        7.000        12.000
      software.la        16.000        10.000
      software.nyc       10.000        14.000
      software.sfo       10.000        15.000
      software.was       20.000        15.000
```

14 Multi-Query Batch use

In some cases a number of small queries need to be performed on the same database. To do individual calls to **SQL2GMS** can become expensive: there is significant overhead in starting Access and opening the database. For these cases we have added the option to do multiple queries in one call. To write several GAMS include files we can use a command file that looks like:

```
C=DRIVER={Microsoft Access Driver (*.mdb)};dbq=sample.mdb

Q1=select distinct(year) from data
O1=year.inc

Q2=select distinct(loc) from data
O2=loc.inc

Q3=select distinct(prod) from data
O3=prod.inc

Q4=select prod,loc,year,sales from data
O4=sales.inc

Q5=select prod,loc,year,profit from data
O5=profit.inc
```

We see that the option **Q**_n_ is matched by an option **O**_n_. That means that the results of the _n_-th query are written to the _n_-th output file.

In case we want to store the results of a multi-query call to a single GDX file, we can use:

```
C=DRIVER={Microsoft Access Driver (*.mdb)};dbq=sample.mdb
X=sample.gdx

Q1=select distinct(year) from data
S1=year
```

```
Q2=select distinct(loc) from data
S2=loc

Q3=select distinct(prod) from data
S3=prd

Q4=select prod,loc,year,sales from data
A4=sales

Q5=select prod,loc,year,profit from data
A5=profit
```

Here we see that a query **Q***n* is matched by either a set name **S***n* or a parameter name **A***n* (the letter **P** was taken already: it is used to specify a password). The name of the GDX file is specified with the **X** option.

For a complete example see section Multi-query Batch example.

15   Multi-Query Batch example

As an example database we use the Access table:

DATA

year	loc	prod	sales	profit
1997	la	hardware	80	8
1997	la	software	60	16
1997	nyc	hardware	110	5
1997	nyc	software	100	10
1997	sfo	hardware	80	9
1997	sfo	software	50	10
1997	was	hardware	120	7
1997	was	software	70	20
1998	la	hardware	70	6
1998	la	software	70	10
1998	nyc	hardware	120	7
1998	nyc	software	120	14
1998	sfo	hardware	90	12

1998	sfo	software	70	15
1998	was	hardware	130	12
1998	was	software	80	15

We want to extract the following information:

- The set **year**
- The set **loc**
- The parameter **sales**
- The parameter **profit**

This can be accomplished using the following GAMS code:

```
$ontext

  Example database access with SQL2GMS (ODBC)

  Multiple queries in one call

$offtext

$onecho > cmd.txt
C=DRIVER={Microsoft Access Driver (*.mdb)};dbq=%system.fp%sample.mdb

Q1=select distinct(year) from data
O1=year.inc

Q2=select distinct(loc) from data
O2=loc.inc

Q3=select distinct(prod) from data
O3=prod.inc

Q4=select prod,loc,year,sales from data
O4=sales.inc

Q5=select prod,loc,year,profit from data
O5=profit.inc
$offecho

$call =sql2gms @cmd.txt

set y years /
$include year.inc
/;
set loc locations /
$include loc.inc
/;
set prd products /
$include prod.inc
/;

parameter sales(prd,loc,y) /
$include sales.inc
/;
display sales;
```

```
parameter profit(prd,loc,y) /
$include profit.inc
/;
display profit;
```

The same example imported through a GDX file can look like:

```
$ontext

  Example database access with SQL2GMS (OLEDB)

  Multiple queries in one call, store in GDX file

$offtext

$onecho > cmd.txt
C=Provider=Microsoft.Jet.OLEDB.4.0;Data Source=%system.fp%sample.mdb
X=sample.gdx

Q1=select distinct(year) from data
s1=year

Q2=select distinct(loc) from data
s2=loc

Q3=select distinct(prod) from data
s3=prd

Q4=select prod,loc,year,sales from data
a4=sales

Q5=select prod,loc,year,profit from data
a5=profit
$offecho

$call =sql2gms @cmd.txt

$call =gdxviewer sample.gdx

set y 'years';
set loc 'locations';
set prd 'products';
parameter sales(prd,loc,y);
parameter profit(prd,loc,y);

$gdxin 'sample.gdx'
$load y=year loc prd sales profit


display sales;
display profit;
```

The call to **gdxviewer** will display the gdx file in the stand-alone GDX viewer.

16 Strategies

Strategies

Including SQL statements to extract data from a database inside your model can lead to a number of difficulties:

- The database can change between runs, leading to results that are not reproducible. A

possible scenario is a user calling you with a complaint: "the model is giving strange results". You run the model to verify and now the results are ok. The reason may be because the data in the database has changed.

- There is significant overhead in extracting data from a database. If there is no need to get new data from the database it is better to use a snapshot stored locally in a format directly accessible by GAMS.
- It is often beneficial to look at the extracted data. A first reason, is just to make sure the data arrived correctly. Another argument is that viewing data in a different way may lead to a better understanding of the data. A complete "under-the-hood" approach may cause difficulties in understanding certain model behavior.

Often it is a good strategy to separate the data extraction step from the rest of the model logic.

If the sub-models form a chain or a tree, like in:

Data extraction --> Data manipulation --> Model definition --> Model solution --> Report writing

we can conveniently use the save/restart facility. The individual submodel are coded as:

Step 0: sr0.gms

```
$ontext

  step 0: data extraction from database
  execute as:  > gams sr0 save=s0

$offtext

set i 'suppliers';
set j 'demand centers';

parameter demand(j);
parameter supply(i);
parameter dist(i,j) 'distances';

$onecho > cmd.txt
C=Provider=Microsoft.Jet.OLEDB.4.0;Data Source=%system.fp%transportation.mdb

Q1=select name from suppliers
O1=i.inc

Q2=select name from demandcenters
O2=j.inc

Q3=select name,demand from demandcenters
O3=demand.inc

Q4=select name,supply from suppliers
O4=supply.inc

Q5=select supplier,demandcenter,distance from distances
O5=dist.inc
$offecho

$call =sql2gms.exe @cmd.txt

set i /
$include i.inc
/;
```

```
set j /
$include j.inc
/;

parameter demand /
$include demand.inc
/;

parameter supply /
$include supply.inc
/;

parameter dist /
$include dist.inc
/;

display i,j,demand,supply,dist;
```

Step 1: sr1.gms

```
$ontext

  step 1: data manipulation step
  execute as:  > gams sr1 restart=s0 save=s1

$offtext

Scalar f 'freight in dollars per case per thousand miles' /90/ ;

Parameter c(i,j) 'transport cost in thousands of dollars per case';

c(i,j) = f * dist(i,j) / 1000 ;
```

Step 2: sr2.gms

```
$ontext

  step 2: model definition
  execute as:  > gams sr2 restart=s1 save=s2

$offtext

Variables
    x(i,j)  'shipment quantities in cases'
    z       'total transportation costs in thousands of dollars' ;

Positive Variable x ;

Equations
    ecost       'define objective function'
    esupply(i)  'observe supply limit at plant i'
    edemand(j)  'satisfy demand at market j' ;

ecost ..       z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

esupply(i) ..  sum(j, x(i,j))  =l=  supply(i) ;

edemand(j) ..  sum(i, x(i,j))  =g=  demand(j) ;
```

Step 3: sr3.gms

```
$ontext
```

```
  step 3: model solution
  execute as:  > gams sr3 restart=s2 save=s3
```

$offtext

option lp=cplex;

Model transport /all/ ;
Solve transport using lp minimizing z ;

Step 4: sr4.gms
$ontext

```
  step 4: report writing
  execute as:  > gams sr4 restart=s3
```

$offtext

abort$(transport.modelstat <> 1) "model not solved to optimality";

display x.l,z.l;

A model that executes all steps can be written as:
execute '=gams.exe sr0 lo=3 save=s0';
abort$errorlevel "step 0 failed";

execute '=gams.exe sr1 lo=3 restart=s0 save=s1';
abort$errorlevel "step 1 failed";

execute '=gams.exe sr2 lo=3 restart=s1 save=s2';
abort$errorlevel "step 2 failed";

execute '=gams.exe sr3 lo=3 restart=s2 save=s3';
abort$errorlevel "step 3 failed";

execute '=gams.exe sr4 lo=3 restart=s3';
abort$errorlevel "step 4 failed";

If you only change the reporting step, i.e. generating some output using PUT statements, then you only need to change and re-execute step 4. If you change solver or solver options, then only steps 3 and 4 need to be redone. For a small model like this, this exercise may not be very useful, but when the model is large and every step is complex and expensive, this is a convenient way to achieve quicker turn-around times in many cases.

In some cases the save/restart facility is not appropriate. A more general approach is to save the data from the database in a GDX file, which can then be used by other models. We can use the model from step 0 to store the data in a GDX file:

MDB2GDX1.GMS:
$ontext

```
  Store data from Access database into a GDX file.
```

$offtext

execute '=gams.exe sr0 lo=3 gdx=trnsport.gdx';
abort$errorlevel "step 0 failed";

execute '=gdxviewer.exe trnsport.gdx';

We can also let SQL2GMS create the GDX file:

MDB2GDX2.GMS:

```
$ontext

   Store data from Access database into a GDX file.

$offtext

$onecho > cmd.txt
C=Provider=Microsoft.Jet.OLEDB.4.0;Data Source=%system.fp%transportation.mdb
X=%system.fp%transportation.gdx

Q1=select name from suppliers
S1=i

Q2=select name from demandcenters
S2=j

Q3=select name,demand from demandcenters
A3=demand

Q4=select name,supply from suppliers
A4=supply

Q5=select supplier,demandcenter,distance from distances
A5=dist
$offecho

$call =sql2gms.exe @cmd.txt
```

The first approach has the advantage that a complete audit record is available from the data moved from the database to the GDX file in the **sr0.lst** listing file. If someone ever wonders what came out of the database and how this was stored in the GDX file, that file gives the answer.

To load the GDX data the following fragment can be used:

GDXTRNSPORT.GMS:

```
$ontext

   Load transportation data from GDX file

   Compile time loading

$offtext

set i 'suppliers';
set j 'demand centers';

parameter demand(j);
parameter supply(i);
parameter dist(i,j) 'distances';

$gdxin transportation.gdx
$load i j demand supply dist
```

```
display i,j,demand,supply,dist
```

In one application I had to retrieve data from the database each morning, at the first run of the model. The rest of the day, the data extracted that morning could be used. The following logic can implement this:

```
$ontext

    Retrieve data from data base first run each morning.

$offtext

$onecho > getdate.txt
C=Provider=Microsoft.Jet.OLEDB.4.0;Data Source=%system.fp%transportation.mdb
Q=select day(now())
O=dbtimestamp.inc
$offecho

$if not exist dbtimestamp.inc $call "echo 0 > dbtimestamp.inc"

scalar dbtimestamp 'day of month when data was retrieved' /
$include dbtimestamp.inc
/;

scalar currentday 'day of this run';
currentday = gday(jnow);

display "compare", dbtimestamp,currentday;

if (dbtimestamp<>currentday,

   execute '=gams.exe sr0 lo=3 gdx=transportation.gdx';
   abort$errorlevel "step 0 (database access) failed";

   execute '=sql2gms.exe @getdate.txt'
);
```

The include file **dbtimestamp.inc** contains the day of the month (1,..,31) on which the data was extracted from the database. If this file does not exist, we initialize it with 0. We then compare this number with the current day of the month. If the numbers do not agree, we execute the database extraction step and rewrite the **dbtimestamp.inc** file. This last operation could be done using a PUT statement, but in this case we used an SQL statement.

17   Command-line Arguments

Command line arguments

C=*ConnectionString*	This option is required and specifies which database is to be used and how. Examples of valid connection strings are in Connection Strings. Often the connection string will need to be surrounded by quotes, as in: `C="DSN=sample".`
O=*OutputIncludeFile*	This option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter statement. Make sure the directory is writable. UNC names can be used. An output file must be specified for batch operation: i.e. either O= or X= needs to be specified (or both).

O*n*=*OutputIncludeFile*	When multiple queries are used, you can append a number to match a query with an output file: **Q2=**"sel ect a, b f r om t abl e" **O2=**ab. i nc See section Multi-query Batch use
X=*OutputGDXfile*	This option specifies the name of the output file. The format of the output file will be a GAMS GDX file. Make sure the directory is writable. UNC names can be used. An output file must be specified for batch operation: i.e. either O= or X= needs to be specified (or both).
Q=*Query*	This option can be used to specify an SQL query. Queries contain spaces and thus have to be surrounded by double quotes. For the exact syntax of the queries that is accepted by the database we refer to the documentation that comes with your RDBMS.
Q*n*=*Query*	When multiple queries are used, you can append a number to match a query with an output file: **Q2=**"sel ect a, b f r om t abl e" **O2=**ab. i nc See section Multi-query Batch use
S=*setname*	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file.
S*n*=*setname*	If multiple queries are used, you can append a number to match the query: **Q2=**"sel ect i f r om t abl e" **S2=**I See section Multi-query Batch use
Y=*setname*	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. If the set contains SetTexts (strings) these will be exported.
Y*n*=*setname*	If multiple queries are used, you can append a number to match the query. If the set contains SetTexts (strings) these will be exported: **Q2=**"sel ect i f r om t abl e" **S2=**I See section Multi-query Batch use
A=*parametername*	If we write to a GDX file, use this option to specify the name of a parameter to be used inside the GDX file. (Note: MDB2GMS uses P, but P was already taken in SQL2GMS for specifying the password).
A*n*=*parametername*	If multiple queries are used, you can append a number to match the query: **Q2=**"sel ect i , v f r om t abl e" **A2=**v See section Multi-query Batch use
L	Embed the data in $*offlisting*, $*onlisting*. A quick way to reduce the size of the listing file.
@*filename* **@"*file name*"**	Causes the program to read options from a file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if it were specified on the command line.
N=*inifilename*	Use a different Inifile than the standard **sql2gms.ini** located in the same directory as the executable **sql2gms.exe**.
T1=*ConnectionTimeOut*	Indicates how long to wait while establishing a connection before terminating the attempt and generating an error. The value sets, in

	seconds, how long to wait for the connection to open. Default is 15. If you set the property to zero, ADO will wait indefinitely until the connection is opened. A value of -1 indicates that the default value is to be used. Note: the provider needs to support this functionality.
T2=*CommandTimeOut*	Indicates how long to wait while executing a command before terminating the attempt and generating an error. The value sets, in seconds, how long to wait for a command to execute. Default is 30. If you set the property to zero, ADO will wait indefinitely until the execution is complete. A value of -1 indicates that the default value is to be used. Note: the provider needs to support this functionality.
T=*TimeOut*	Sets both **T1** and **T2**.

18 $CALL command

The **$CALL** command in GAMS will execute an external program at compile time. There are two forms:

```
$call externalprogram
$call =externalprogram
```

The version without the leading '=' calls the external through the command processor (*command.com* or *cmd.exe*). The second version with the '=', bypasses the command processor and directly executes the external program. We mention some of the differences:

1. Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE. If you want to execute these commands you will need to use the form `$call externalprogram` which uses the command processor.
2. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form `$call externalprogram`.
3. If it is important to stop with an appropriate error message if the external program does not exist, only use the form `$call =externalprogram`. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: `$call =externalprogram`.
4. When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the START command, as in: `$call start /w externalwindowsprogram`. Otherwise, it is preferred to use: `$call =externalwindowsprogram`.

In general it is recommended to use the `$call =externalprogram` version for its better error-handling.

When command line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

```
$call externalprogram parameter1 parameter2
$call =externalprogram parameter1 parameter2
```

The total length of the command line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

```
$call '"external program" "parameter 1" "parameter 2"'
```

```
$call ="external program" "parameter 1" "parameter 2"
```

It is noted that the first form needs additional quotes around the whole command line due to bugs in the parsing of the $call in GAMS. The second form work without additional quotes *only if the* = *appears outside the double quotes*.

19   Command files

Parameters can be specified in a command file. This is important if the length of the command line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command lines. Instead of specifying a long command line as in:

```
$call =sql2gms C="DSN=sample" O="c:\My Documents\data.inc" Q="Select * from mytable"
```

we can use a command line like:

```
$call =sql2gms @"c:\My Documents\options.txt"
```

The command file **c:\My Documents\options.txt** can look like:

```
C=DSN=sample
O=c:\My Documents\data.inc
Q=Select * from mytable
```

It is possible to write the command file from inside a GAMS model using the $echo command. The following example will illustrate this:

```
$set cmdfile "c:\windows\temp\commands.txt"
$echo "C=DSN=sample"            >  "%cmdfile%"
$echo "O=E:\models\labor.INC"   >> "%cmdfile%"
$echo "Q=select * from labor"   >> "%cmdfile%"
$call =sql2gms @"%cmdfile%"
parameter p /
$include "E:\models\labor.INC"
/;
display p;
```

Newer versions of GAMS allow:

```
$set cmdfile "c:\windows\temp\commands.txt"
$onecho  >  "%cmdfile%"
C=DSN=sample
O=E:\models\labor.INC
Q=select * from labor
$offecho
$call =sql2gms @"%cmdfile%"
parameter p /
$include "E:\models\labor.INC"
/;
display p;
```

If a query becomes very long, it is possible to spread it out over several lines. To signal a setting will continue on the next line insert the character \ as the last character. E.g.:

```
Q=select prod,loc,year,'sales',sales from data \
```

```
union \
select prod,loc,year,'profit',sales from data
```

20 Notes

ADO

ActiveX Data Objects. Microsoft's data-access object model. An object-oriented architecture for accessing data stored in SQL databases and related data sources. Accessible from a large number of host languages such as VB, C++, Delphi. Supersedes ODBC. Many SQL databases provide ADO access through either OLEDB or ODBC.

ODBC

An API and driver manager system for accessing data stored in an RDBMS. The API provides applications a way to talk to databases while the driver manager application allows users to install, configure and manage ODBC drivers for different databases.

OLEDB

Driver architecture for SQL databases. A driver is called a *OLE DB provider*. This is used from ADO.

UNC Names

UNC means Unified Naming Convention. UNC names are a microsoft convention to name files across a network. Th general format is:

\\<server>\<share>\<path>\<file>

Examples:

\\athlon\c\My Documents\sql2gms.rtf

GDX Files

A GDX file contains GAMS data in binary format. The following GAMS commands will operate on GDX files: $GDXIN, $LOAD, EXECUTE_LOAD, EXECUTE_UNLOAD. The GDX=filename command line option will save all data to a GDX file. A GDX file can be viewed in the IDE using File|Open.

MDB2GMS

MDB2GMS is a tool to import tables from MS Access databases. This utility directly uses MS Access and DAO (Data Access Objects) resulting in a somewhat simpler interface. It is not needed to specify a connection string, but just a .MDB file. The query mechanism is similar: a query is sent as is to the database server and the result set is translated into a GAMS representation. For more information see http://www.gams.com/~erwin/interface/interface.html.

XLS2GMS

XLS2GMS is a tool to import information from an Excel spreadsheet. It considers the content of a selected range as GAMS source code. For more information see http://www.gams.com/~erwin/interface/interface.html.

Quotes

Examples of handling of indices when the option "quote blanks" is used:

Input	output	remarks
Hello	hello	blanks or embedded quotes
"hello"	"hello"	touched, is quoted already
'hello'	'hello'	id.
"hello'	"hello'	id., but will generate an error in GAMS
o'brien	"o'brien"	
'o'brien'	'o'brien'	touched, will generate an error in GAMS
art"ificial	'art"ificial'	
art"ifi'cial	"art"ifi'cial"	

Compile time commands

All **$** commands in GAMS are performed at compile time. All other statements are executed at execution time. This means that a compile time command will be executed **before** an execution time command, even if it is below. As an example consider:

```
file batchfile /x.bat/;
put patchfile;
putclose "dir"/;
$call x.bat
```

This fragment does not work correctly as already during compilation, the $call is executed, while the put statements are only executed after the compilation phase has ended and GAMS has started the execution phase. The above code can be fixed by moving the writing of the batch file to compilation time as in

```
$echo "dir" > x.bat
$call x.bat
```

or by moving the external program invocation to execution time:

```
file batchfile /x.bat/;
put patchfile;
putclose "dir"/;
execute x.bat;
```

Notice that all **$** commands do not include a semi-colon but are terminated by the end-of-line.