An SQL engine for YottaDB and other M-like databases

mgsql

MGateway Ltd.
Chris Munt

Revision History:
13 June 2019
1 November 2019
15 January 2020
8 January 2021
22 February 2021
3 February 2022
8 June 2023

Contents

1 Introduction

mgsql is an Open Source SQL engine developed primarily for the **YottaDB** database. It will also work with the GT.M database and other M-like databases.

SQL access is provided via the following routes:

- Embedded SQL statements in M code.
- REST.
- ODBC.

Note that the **mgsql** project is very much 'work in progress'. Use cautiously!

2 Pre-requisites

The **YottaDB** database (or similar M database):

```
https://yottadb.com/
```

3 Installing mgsql

3.1 YottaDB

The instructions given here assume a standard 'out of the box' installation of **YottaDB** (version 1.30) deployed in the following location:

```
/usr/local/lib/yottadb/r130
```

The primary default location for routines:

```
/root/.yottadb/r1.30_x86_64/r
```

Copy all the routines (i.e. all files with an 'm' extension) held in the GitHub **/yottadb** directory to:

```
/root/.yottadb/r1.30_x86_64/r
```

Change directory to the following location and start a **YottaDB** command shell:

```
cd /usr/local/lib/yottadb/r130
./ydb
```

Link all the **mgsql** routines and check the installation:

```
do ylink^%mgsql

do ^%mgsql

MGSQL by MGateway Ltd.
Version: 1.4; Revision 23 (8 June 2023) %mgsql
```

Note that the version of **mgsql** is successfully displayed.

3.2 Other M systems

Log in to the Manager UCI and, using the %RI utility (or similar) load the **mgsql** routines held in **/m/mgsql.ro**.

For InterSystems Caché and IRIS, log in to the %SYS Namespace and install the **mgsql** routines held in **/isc/mgsql_isc.ro**.

```
do $system.OBJ.Load("/isc/mgsql_isc.ro","ck")
```

Change to your development Namespace and check the installation:

```
do ^%mgsql

MGSQL by MGateway Ltd.
Version: 1.4; Revision 23 (8 June 2023) %mgsql
```

4 Creating tables

Tables defined in **mgsql** map directly to M global variables for their physical storage. The standard SQL '***create table'*** constructs are used to create **mgsql** tables, but with a number of extensions to define the precise mapping to physical storage.

As an example, consider a simple table defining a hospital patient keyed by a unique patient number (**num**):

```
create table patient (
  num int not null,
  name varchar(255),
  address varchar(255) separate,
  phone_no varchar(32),
  constraint pk_patient primary key (num))
  /*! global=mgpat, delimiter=# */
```

Data is stored in the column order implied in the '***create'*** statement. Note the definition of the physical global name and a data-string delimiter as a comment line. By default, column fields are stored as a data string separated by the delimiter character defined here. For cases where fields may themselves contain the delimiter character, it is possible to configure the table so that such fields are stored separately outside the data string. Note the '***separate'*** keyword defined against the address field.

Now consider the following '***insert'*** statement:

```
insert into patient (num, name, address, phone_no)
   values(100001,
          'John Smith',
          'Apartment #3, Some Street, London',
          '1234 567890');
```

When executed, this will result in the following physical storage:

```
^mgpat(100001)="John Smith##1234 567890"
^mgpat(100001,2)="Apartment #3, Some Street, London"
```

4.1 Additional mapping techniques

In the previous section we saw how table columns could be mapped to physical values held either within a data string or separately subscripted. In both cases the column number is used to determine either the position in the data string or becomes the trailing subscript in cases where the column is separately subscripted.

The '***separate'*** keyword can be further qualified to allow alternative values to be used as trailing subscripts.

Example 1 (a single schema-defined trailing subscript):

```
create table patient (
  num int not null,
  name varchar(255),
  address varchar(255) separate 'address',
  phone_no varchar(32),
  constraint pk_patient primary key (num))
  /*! global=mgpat, delimiter=# */
```

- Note that, unlike in M code, the string 'address' is contained within single-quotation marks.

Now consider the following '***insert'*** statement:

```
insert into patient (num, name, address, phone_no)
   values(100001,
          'John Smith',
          'Apartment #3, Some Street, London',
          '1234 567890');
```

When executed, this will result in the following physical storage:

```
^mgpat(100001)="John Smith##1234 567890"
^mgpat(100001,"address")="Apartment #3, Some Street, London"
```

Example 2 (multiple schema-defined trailing subscripts):

```
create table patient (
  num int not null,
  name varchar(255),
  address varchar(255) separate ('e','address',1),
  phone_no varchar(32),
  constraint pk_patient primary key (num))
  /*! global=mgpat, delimiter=# */
```

- Note that, unlike in M code, strings are contained within single-quotation marks.
- Note also that in cases where multiple trailing subscripts are required, the set must be enclosed in (rounded) brackets.

Now consider the following '***insert'*** statements:

```
insert into patient (num, name, address, phone_no)
   values(100001,
          'John Smith',
          'Apartment #3, Some Street, London',
          '1234 567890')
```

```
insert into patient (num, name, address, phone_no)
   values(100002,
          'Jane Jones',
          '7 Another Street, London',
          '1234 987654')
```

When executed, these statements will result in the following physical storage:

```
^mgpat(100001)="John Smith##1234 567890"
^mgpat(100001,"e","address",1)="Apartment #3, Some Street, London"

^mgpat(100002)="Jane Jones##1234 987654"
^mgpat(100002,"e","address",1)="7 Another Street, London"
```

4.2 Derived Fields

Derived Fields are defined in the schema as M extrinsic functions. The deriving function can take any number of values from the same row as input parameters. For example, the calculation of a person's age from a stored date of birth field can be implemented as a *Derived Field* - the value of 'age' being dependant on the time of data retrieval.

Example (Adding the column 'age' as a Derived Field in the 'patient' table):

```
create table patient (
  num int not null,
  name varchar(255),
  address varchar(255) separate ('e','address',1),
  phone_no varchar(32),
  dbirth date,
  age int derived age^%mgsqls(dbirth),
  constraint pk_patient primary key (num))
  /*! global=mgpat, delimiter=# */
```

- Note that in this example, we use the supplied function (`age^%mgsqls`) to calculate the age from the date of birth field (`dbirth`).

The invocation of *Derived Field* functions is completely transparent to SQL retrieval scripts. For example, consider a SQL script to list all patients older than 40 years.

```
select a.num, a.name, a.dbirth, a.age
from patient a
where a.age > 40
```

5 Executing SQL statements from the M command prompt

Before executing SQL statements do familiarise yourself with the M system resources (i.e. globals) used by **mgsql**. Refer to the ***Resources used by mgsql*** section.

The general form for executing SQL statements from within M code (or from the M command line) is as follows:

```
set status=$$exec^%mgsql(<schema>,<sql statement>,.%zi,.%zo)
```

Where:

- **%zi** is an M array representing data that needs to be input to the script.
- **%zo** is an M array representing parameters controlling output from the script.

The query output, by default, will be written to the primary device (for example, a terminal window), line by line with columns returned as a comma-separated list.

The top-level routine **%mgsql** (physical file **_mgsql.m**) contains a number of sample SQL scripts. These work to a simple database representing hospital patients and their associated admissions. View the embedded scripts in this routine.

Create the test schema:

```
do create^%mgsql
```

Insert a few test records:

```
do insert^%mgsql
```

Run the various SQL retrieval scripts:

```
do sel1^%mgsql
```

A number of SQL scripts are available at line labels: `sel1, sel2, sel3 … to` ***`sel[n]`***`.`

6 Embedding SQL statements in M code

The general form for executing SQL statements from within M code is as follows:

```
set status=$$exec^%mgsql(<schema>,<sql statement>,.%zi,.%zo)
```

Where:

- **%zi** is an M array representing data that needs to be input to the script.
- **%zo** is an M array representing parameters controlling output from the script.

In the simplest case, the query output will be written to the primary device (for example, a terminal window), line by line with columns returned as a comma-separated list.

Example:

```
new %zi,%zo
set status=$$exec^%mgsql("","select * from patient",.%zi,.%zo)
```

Output:

```
100001,John Smith,'Apartment #3, Some Street, London',1234 567890
100002,Jane Jones,'7 Another Street, London',1234 987654
```

Information about the columns will be returned in the output **(%zo)** array, the structure of which is as follows:

```
%zo(0,<column no>)=<column name>
%zo(0,<column no>,0)=<type>
```

For example, for the above query this will be:

```
%zo(0,1)="patient.num"
%zo(0,1,0)="int"
%zo(0,2)="patient.name"
%zo(0,2,0)="varchar(255)"
%zo(0,3)="patient.address"
%zo(0,3,0)="varchar(255)"
%zo(0,4)="patient.phone_no"
%zo(0,4,0)="varchar(255)"
%zo("routine")="x0011"
```

Note that the name of the M routine generated for this query is returned under the '***routine'*** node of this array.

6.1 Supplying variable inputs to a query

You can supply variable inputs to the query via the inputs array - **%zi** in the examples. The rationale for doing this, as opposed to embedding values directly in the scripts, is that it reduces the amount of compilation time needed. Once the query is compiled the same generated code can be used for all sets of input variables.

Example:

```
new %zi,%zo
set %zi("number")=100000
set status=$$exec^%mgsql("","select * from patient
                         where num = :number",.%zi,.%zo)
```

6.2 Using the SQL output spool file

Rather than dumping query output to the current device, it is possible to mandate that output is directed to the spool file, which is a global called ***mgsqls***. To do this, specify a query ***statement ID*** in the input array.

Example:

```
new %zi,%zo
set %zi(0,"stmt")="all patients"
set status=$$exec^%mgsql("","select * from patient",.%zi,.%zo)
```

The query will execute silently and write the output to the spool file, the structure of which is as follows:

```
^mgsqls($Job, <statement_ID>, 0, <row_no>, <column_no>)=<value>
```

For the example given, the contents of the spool file will be as follows:

```
^mgsqls([$Job],"all patients",0,1,1)=100001
^mgsqls([$Job],"all patients",0,1,2)="John Smith"
^mgsqls([$Job],"all patients",0,1,3)="Apartment #3, Some Street, London"
^mgsqls([$Job],"all patients",0,1,4)="1234 567890"
^mgsqls([$Job],"all patients",0,2,1)=100002
^mgsqls([$Job],"all patients",0,2,2)="Jane Jones"
^mgsqls([$Job],"all patients",0,2,3)="7 Another Street, London"
^mgsqls([$Job],"all patients",0,2,4)=1234 987654
```

Where **$Job** is the current process ID.

6.3 Using an SQL output callback

Finally, it is possible to mandate that query output is directed to a ***callback*** – an M function defined in the application. To do this, specify the name of the ***callback*** function (including the routine name) in the input array.

The ***callback*** function must be defined as follows:

```
callback(%zi, %zo, rn)
```

Where:

- **%zi** is the query input array.
- **%zo** is the query output array.
- **rn** is the current row number.

Example:

```
   new %zi,%zo
   set %zi(0,"callback")="allpatients^thisroutine"
   set status=$$exec^%mgsql("","select * from patient",.%zi,.%zo)
   quit
   ;
allpatients(%zi,%zo,rn) ; query callback
   new stop
   set stop=0
   ; process the row of output here
   Kill %zo(rn) ; we don't want to keep the current row
   quit stop
```

Where ***rn*** is the current row number and stop is a ***stop*** flag. Set the ***stop*** flag to 1 in the ***callback*** function to force the early termination of the query. The data for the row is held in the output array **(%zo),** the structure of which is as follows:

```
%zo(<row_no>, <column_no>)=<value>
```


Full example:

```
select    ; embedded query using a callback function
          new %zi,%zo
          set %zi(0,"callback")="selectcb^thisroutine"
          set status=$$exec^%mgsql("","select * from patient",.%zi,.%zo)
          quit
          ;
selectcb(%zi,%zo,rn) ; callback function
          new n,stop
          set stop=0
          write !,"row number: ",rn
          for n=1:1 quit:'$data(%zo(0,n))  do
          . write !,"  column name: ",$get(%zo(0,n))
          . write !,"     type: ",$get(%zo(0,n,0))
          . write !,"     value: ",$get(%zo(rn,n))
          kill %zo(rn)
          quit stop
```

7 Transaction Processing

mgsql supports the standard SQL Transaction Processing commands:

```
START TRANSACTION
COMMIT
ROLLBACK
```

The implementation of these commands is based on the underlying M commands: **tstart**, **tcommit** and **trollback**. SQL Transactions can be implemented in M code or via an external program connecting to **mgsql** via the ODBC driver.

The M Transaction Processing commands can be used directly in M code. For example:

```
new %zi,%zo
tstart
set status=$$exec^%mgsql("",<sql update statement>,.%zi,.%zo)
; further update statements …
tcommit
```

Alternatively, the corresponding SQL Transaction Processing commands can be used in SQL statements. For example:

```
new %zi,%zo
set status=$$exec^%mgsql("","transaction start",.%zi,.%zo)
set status=$$exec^%mgsql("",<sql update statement>,.%zi,.%zo)
; further update statements …
set status=$$exec^%mgsql("","commit",.%zi,.%zo)
```

It is also possible to implement transactions in M ***callback*** functions. This method is mandatory for YottaDB.

The transaction ***callback*** function must be defined as follows:

```
callback(%zi, %zo)
```

Where:

- **%zi** is the query input array.
- **%zo** is the query output array.

For example:

```
tp   ; transaction implemented in a callback function
     new %zi,%zo
     set %zi(0,"callback")="tpcb"
     set status=$$exec^%mgsql(“”,”transaction start”,.%zi,.%zo)
     q
     ;
tpcb(%zi,%zo) ; callback function
     set status=$$exec^%mgsql(“”,<sql update statement>,.%zi,.%zo)
     ; further update statements …
     set status=$$exec^%mgsql(“”,”commit”,.%zi,.%zo)
     quit status
```

8 SQL Query Optimisation

mgsql will attempt to find the most optimal route through the set of tables queried in accordance with the information that the optimiser can extract from the WHERE predicate together with any JOIN constraints specified.

In cases where **mgsql** does not come up with an optimal route through the tables it is possible to provide ***hints*** in the form of explicitly defining the indices to use. If an index is defined in the FROM statement, **mgsql** will attempt to process the tables in the order specified in the FROM statement.

The general form for explicitly optimising queries is as follows:

```
select [columns] from table1:index1, table2 where [predicate]
```

Or if aliases are used:

```
select [columns] from table1 a:index1, table2 b where [predicate]
```

In the above example, **mgsql** will attempt to parse **table1** first using **index1**, followed by **table2.**

As a convenience, the primary key may be defined as index ‘0’:

```
select [columns] from table1 a:0, table2 b:idx2 where [predicate]
```

In the above example, **mgsql** will attempt to parse **table1** first using the **primary key**, followed by **table2** using index **idx2.**

9 Starting the mgsql network Superserver

So far, we have covered the basics of executing SQL statements from M code. In order to execute SQL queries over REST or ODBC the **mgsql** installation must be accessible over the network. The Superserver service described here will concurrently support access to **mgsql** via REST and ODBC. The default TCP server port for **mgsql** is **7041**. If you wish to use an alternative port then modify the following instructions accordingly.

For most M systems, the **mgsql** Superserver can be started from the M command prompt. For YottaDB there is the option of starting Superserver child processes via the ***xinetd*** service.

- Note that if you are using the generic MGateway Superserver **(%zmgsi**) then no action is required here as the generic Superserver is able to serve **mgsql**.

9.1 Starting the mgsql Superserver

Start the M-hosted concurrent TCP service in the Manager UCI:

```
do start^%mgsql(0)
```

To use a server TCP port other than 7041, specify it in the start-up command (as opposed to using zero to indicate the default port of 7041).

9.2 Starting YottaDB Superserver child processes via xinetd

Network connectivity to **YottaDB** is managed via the **xinetd** service. First create the following launch script (called **mgsql_ydb** here):

```
/usr/local/lib/yottadb/r130/mgsql_ydb
```

Content:

```
#!/bin/bash
cd /usr/local/lib/yottadb/r130
export ydb_dir=/root/.yottadb
export ydb_dist=/usr/local/lib/yottadb/r130
export
ydb_routines="/root/.yottadb/r1.30_x86_64/o*(/root/.yottadb/r1.30_x86_6
4/r /root/.yottadb/r) /usr/local/lib/yottadb/r130/libyottadbutil.so"
export ydb_gbldir="/root/.yottadb/r1.30_x86_64/g/yottadb.gld"
$ydb_dist/ydb -r xinetd^%mgsql
```

Note that ‘execute’ permissions must be assigned to this script. For example:

```
chmod a=rx mgsql_ydb
```

Create the **xinetd** script (called **mgsql_xinetd** here):

```
/etc/xinetd.d/mgsql_xinetd
```

Content:

```
service mgsql_xinetd
{
     disable         = no
     type            = UNLISTED
     port            = 7041
     socket_type     = stream
     wait            = no
     user            = root
     server          = /usr/local/lib/yottadb/r130/mgsql_ydb
}
```

- Note: sample copies of **mgsql_xinetd** and **mgsql_ydb** are included in the /unix directory.

Edit the services file:

```
/etc/services
```

Add the following line to this file:

```
mgsql_xinetd          7041/tcp                        # MGSQL
```

Finally restart the **xinetd** service:

```
/etc/init.d/xinetd restart
```

10 Access to mgsql using REST

Now that the network service has been configured and deployed it is possible to execute SQL scripts via REST calls. Results are returned formatted as JSON.

For example, using the **curl** utility from the UNIX command line:

```
        curl -d "select * from patient" -H "Content-Type: text/sql"
http://localhost:7041/mg.sql/execute
```

Assuming that the simple test database described previously has been created the above request will generate the following output:

```
        {"sqlcode": 0, "sqlstate": "00000", "error": "", "result":
[{"num": "1","name": "Chris Munt","address": "Banstead"},{"num":
"2","name": "Rob Tweed","address": "Redhill"},{"num": "3","name": "John
Smith","address": "London"},{"num": "4","name": "Jane Doe","address":
"Oxford"}]}
```

Simple invocation from a browser (Hint: Firefox does a good job of rendering JSON):

```
        http://127.0.0.1:7041/mgsql/mg.sql?sql=select * from patient
```

Alternatively, enter an SQL statement in the form generated by:

```
        http://127.0.0.1:7041/mgsql/mg.sql
```

In a live environment a production-grade web server should be used. For example, using the Apache server the **mod_proxy** module can be used to ***front*** the **mgsql** service.

11 Access to mgsql using ODBC

The ODBC driver is in the **/odbc** directory. Pre-built drivers for 32 and 64-bit Windows are in the **/odbc/x86** and **/odbc/x64** directories respectively. To install both drivers copy the contents of **/odbc/x86** to:

```
C:\Program Files (x86)\mgsql\
```

And copy the contents of **/odbc/x64** to:

```
C:\Program Files\mgsql\
```

You will have to create the **/mgsql** sub-directory if it doesn't already exist. To register both drivers, using Windows Explorer, double click on each of the following Registry files:

```
C:\Program Files (x86)\mgsql\mgodbc32.reg
C:\Program Files\mgsql\mgodbc64.reg
```

You can now configure an ODBC Data Source using the Windows Administrative tools for ODBC Data sources (accessed via the Windows Control Panel:

```
Control Panel\System and Security\Administrative Tools\ODBC Data Sources
```

Under the **System DSN** tab. select **Add...** and choose one of the **mgsql** drivers as appropriate:

```
MGSQL ODBC x86
MGSQL ODBC x64
```

Complete the **mgodbc** configuration dialogue box and save:

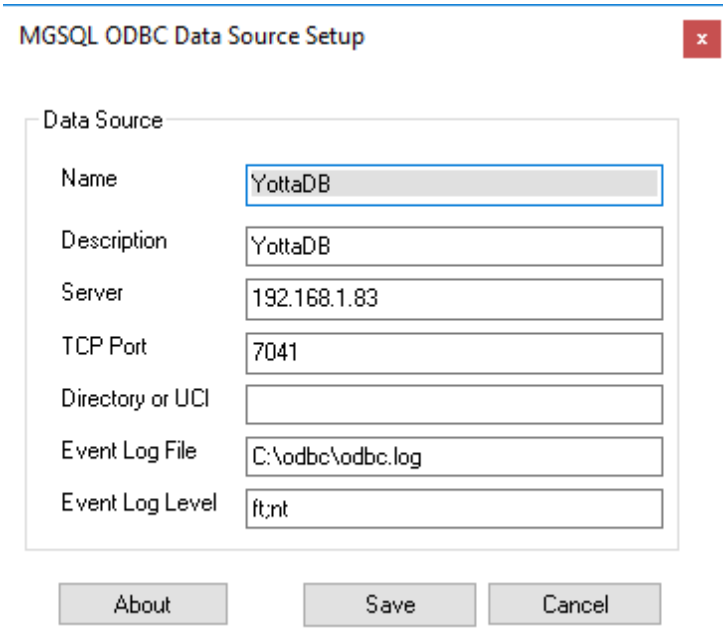
- **Name**: Your Data Source Name (DSN).
- **Description**: An optional description.
- **Server**: IP Address of your M server.
- **TCP Port**: TCP Port (the default is 7041).
- **Directory or UCI**: M UCI (leave blank for YottaDB).
- **Event Log File**: Log file (including full path).
- **Event Log Level**: Log level (a comma separated list of log directives).

Log Level Directives:

- **e**: Log Errors.

- **ft**: Log ODBC function call trace.
- **nt**: Log all network buffers sent and received.

Example:



The data source created can now be used in Windows applications.

11.1 Building the ODBC Driver from source

The **mgsql** ODBC driver (mgodbc.dll) is written in C and you will need ***Microsoft Visual Studio*** to build it from source. The ***Visual Studio Community Edition*** is available for download from Microsoft free of charge.

The Visual Studio project for building the ODBC driver is in the **/odbc** directory (**mgodbc.vcxproj**). This project can be loaded in to, and built from, the Visual Studio GUI toolset. Alternatively, you can build the project from the Developer Command Shell:

Building the 32-bit driver:

```
msbuild mgodbc.vcxproj /p:configuration=release /p:platform=Win32
```

Building the 64-bit driver:

```
msbuild mgodbc.vcxproj /p:configuration=release /p:platform=x64
```

12 Resources used by mgsql

mgsql will write to the following globals

- ^mgsqld: The catalogue or schema.
- ^mgsqls: The spool file for SQL output.
- ^mgsqlx: The cache of compiled queries.
- ^mglog: The event Log.
- ^mgtmp: A temporary file used by the SQL compiler.
- ^mgtemp: A temporary sort file used when executing SQL queries.

- **mgsql** will generate M Routines prefixed by 'x'.

13 License

http://www.mgateway.com
Email: cmunt@mgateway.com