

# **An SQL engine for YottaDB and other M-like databases**

## **mysql**

M/Gateway Developments Ltd.  
Chris Munt

### Revision History:

13 June 2019  
1 November 2019  
15 January 2020

## **Contents**

1	Introduction.....	2
2	Pre-requisites.....	2
3	Installing mysql.....	2
3.1	YottaDB .....	2
3.2	Other M systems.....	3
4	Creating tables .....	3
4.1	Additional mapping techniques.....	4
4.2	Derived Fields .....	6
5	Executing SQL statements from the YottaDB/M command line .....	7
6	Setting up the network service.....	7
6.1	YottaDB .....	8
6.2	Other M systems.....	9
7	Access to mysql using REST.....	9
8	Access to mysql using ODBC.....	10
8.1	Building the ODBC Driver from source .....	12
9	Resources used by mysql.....	12
10	License .....	13

# 1 Introduction

**mgsq**l 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 **mgsq**l project is very much 'work in progress'. Use cautiously!

## 2 Pre-requisites

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

`https://yottadb.com/`

## 3 Installing mgsq

### 3.1 YottaDB

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

`/usr/local/lib/yottadb/r122`

The primary default location for routines:

`/root/.yottadb/r1.22_x86_64/r`

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

`/root/.yottadb/r1.22_x86_64/r`

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

```
cd /usr/local/lib/yottadb/r122
./ydb
```

Link all the **mgsq1** routines and check the installation:

```
do ylink^%mgsq1

do ^%mgsq1

MGSQ1 by M/Gateway Developments Ltd.
Version: 1.1; Revision 9 (15 January 2020) %mgsq1
```

Note that the version of **mgsq1** is successfully displayed.

### 3.2 Other M systems

Log in to the Manager UCI and, using the %RI utility (or similar) load the **mgsq1** routines held in **/m/mgsq1.ro**.

For InterSystems Cache and IRIS, log in to the Manager UCI and install the **mgsq1** routines held in either **/m/mgsq1\_cache.xml** or **/m/mgsq1\_iris.xml** as appropriate. For example:

```
do $system.OBJ.Load("/m/zmgsi_mgsq1.xml","ck")
```

Change to your development UCI and check the installation:

```
do ^%mgsq1

MGSQ1 by M/Gateway Developments Ltd.
Version: 1.1; Revision 9 (15 January 2020) %mgsq1
```

## 4 Creating tables

Tables defined in **mgsq1** map directly to M global variables for their physical storage. The standard SQL '*create table*' constructs are used to create **mgsq1** 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’*** 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,"e","address",1)="Apartment #3, Some 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^%mgsqsls(dbirth),  
    constraint pk_patient primary key (num))  
/!* global=mgpat, delimiter=# */
```

- Note that in this example, we use the supplied function (age^%mgsqsls) 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 YottaDB/M command line

Before executing SQL statements do familiarise yourself with the M system resources (i.e. globals) used by **mgsq1**. Refer to the *Resources used by mgsq1* section.

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

```
set status=$$exec^%mgsq1(<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 top level routine **%mgsq1** (physical file **\_mgsq1.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^%mgsq1
```

Insert a few test records:

```
do insert^%mgsq1
```

Run the various SQL retrieval scripts:

```
do sel1^%mgsq1
```

A number of SQL scripts are available at line labels sel1 to sel8.

## 6 Setting up the network service

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 **mgsq1** installation must be accessible over the network. The service described here will concurrently support access to **mgsq1** via REST and ODBC. The default TCP server port for **mgsq1** is **7041**. If you wish to use an alternative port then modify the following instructions accordingly.

## 6.1 YottaDB

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

```
/usr/local/lib/yottadb/r122/mgsqldb
```

Content:

```
#!/bin/bash
cd /usr/local/lib/yottadb/r122
export ydb_dir=/root/.yottadb
export ydb_dist=/usr/local/lib/yottadb/r122
export
ydb_routines="/root/.yottadb/r1.22_x86_64/o* (/root/.yottadb/r1.22_x86_64/r /root/.yottadb/r) /usr/local/lib/yottadb/r122/libyottadbutil.so"
export ydb_gblldir="/root/.yottadb/r1.22_x86_64/g/yottadb.gld"
$ydb_dist/ydb -r xinetd^%mgsqldb
```



Create the **xinetd** script (called **mgsqldb\_xinetd** here):

```
/etc/xinetd.d/mgsqldb_xinetd
```

Content:

```
service mgsqldb_xinetd
{
    disable           = no
    type              = UNLISTED
    port              = 7041
    socket_type       = stream
    wait              = no
    user              = root
    server             = /usr/local/lib/yottadb/r122/mgsqldb_ydb
}
```

- Note: sample copies of **mgsqldb\_xinetd** and **mgsqldb\_ydb** are included in the `/unix` directory.

Edit the services file:

```
/etc/services
```

Add the following line to this file:

```
mgsqldb_xinetd          7041/tcp          # MGSQDB
```

Finally restart the **xinetd** service:

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

## 6.2 Other M systems

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

```
do start^%mgsqldb(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).

## 7 Access to mgsqldb 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/mgsq1/mg.sql?sql=select * from patient
```

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

```
http://127.0.0.1:7041/mgsq1/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 **mgsq1** service.

## 8 Access to mgsq1 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)\mgsq1\
```

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

```
C:\Program Files\mgsq1\
```

You will have to create the **/mgsq1** 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)\mgsq1\mgodbc32.reg
C:\Program Files\mgsq1\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 **mysql** drivers as appropriate:

Mysql ODBC x86  
Mysql ODBC x64

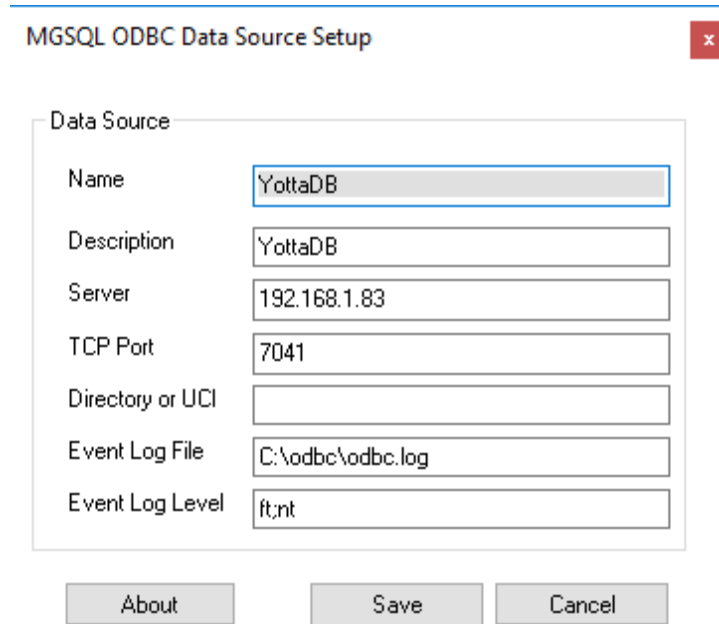
Complete the **mysql** configuration dialogue box and save:

- **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.

## 8.1 Building the ODBC Driver from source

The **mysql** 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
```

## 9 Resources used by mysql

**mysql** 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.
- **mgsq** will generate M Routines prefixed by 'x'.

## 10 License

Copyright (c) 2018-2020 M/Gateway Developments Ltd,  
Surrey UK.  
All rights reserved.

<http://www.mgateway.com>  
Email: [cmunt@mgateway.com](mailto:cmunt@mgateway.com)

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

