Connect to an external database server in FileMaker

Your FileMaker solution may have a need to connect to another database. While you can do ESS, we offer an alternative way. You can connect via script to a database, run SQL statements and close the connection. That is a great way to move a few records in one direction or even implement a synchronization routine.

Alternatively you can have a script to open the connection when the solution opens. Then just have various calculations and scripts refer to the open connection and run queries as needed. Like have a formula field on a layout to query a number for a value from the foreign table and show it to the user. Like a solution to manage shows and query from the web shop how many tickets got sold.

So what do you need?

1. Database Type

First question is what database you like to connect to.
We support these database types: CubeSQL, Centura SQLBase, DB2, Firebird, Informix, InterBase, MariaDB, Microsoft Access, Microsoft SQL Server, MySQL, ODBC, Oracle Database Server, PostgreSQL, SQL Anywhere, SQLite, SQLCipher and Sybase.

For most of them we can do without ODBC. For example PostgreSQL can be connected directly with the native driver and without ODBC. Still we also support ODBC as connectivity and thus we can also connect via ODBC to PostgreSQL. But we clearly prefer to skip ODBC as another layer between you and the database server.

Once you know what type of database you have, you need to know where to find it:

2. IP and Port

To connect to a sever, you need to get a connection string. That string usually encapsulates various values needed to connect. The server is identified by an IP address or a domain name. If the server is on the same computer, you can use "localhost". The network connections to localhost usually skips the firewall and are more likely to work. If you have a connection via TCP/IP, you need a port number. Usually there are default port numbers used, so you don't need to specify them. For PostgreSQL it is 5432, for MySQL it is 3306 or for MS SQL Server 1433.

Once you have IP and port, you can even try to connect via our socket functions and see if it works. If you get a timeout, the IP and port combination may be wrong. If you get connection refused, the firewall is probably blocking you. But if you get a connection, your IP and port combination works. We can try it in terminal with curl command line tool:
curl -v http://localhost:5432

This attempts a connection on localhost to port 5432 for PostgreSQL server.
You may see output like this if no server is running there or Firewall blocks you:
* Trying ::1:5432...
* connect to ::1 port 5432 failed: Connection refused

Or with a timeout:
* Trying 192.168.2.222:5432...
* connect to 192.168.2.222 port 5432 failed: Operation timed out
* Failed to connect to 192.168.2.222 port 5432: Operation timed out
* Closing connection 0
curl: (28) Failed to connect to 192.168.2.222 port 5432: Operation timed out

Or with success:
* Trying ::1:5432...
* Connected to localhost (::1) port 5432 (#0)

If it fails, check your firewalls and allow the access. For MS SQL Server, you may need to enable TCP/IP first if you like to use it for connections. Once you know where your server is, you need to know credentials:

3. Credentials

Next you need to know the login data. Usually you setup a new user role on the database server for your automation with a new password. This user should be limited to see only the tables (or views) that you need to automate. And most tables should be read only. If you need to write records, consider only allowing inserts and still forbid delete, truncate and drop commands. Think about what damage someone could do to your company if they get those credentials. Sadly we see too many apps just connect with an admin login and could do anything they want!

So we may get details like this:
Username: FMAutomation
Password: FileMakerIsGreat!!!
Database: Staff

Tables to query: Person
Table to insert: Timesheet

So this FileMaker solution would measure times for staff and then insert them into one table. And it may query another table to learn who is in the staff group. All other tables are blocked. Person table is of course set to read only for this login.

4. Libraries

To connect to various databases, you need drivers.

For MySQL you need the libmysqlclient file for your platform. On Windows the libmySQL.dll and on MacOS a libmysqlclient.16.dylib. For example you can go on our Libraries download folder and get a recent MySQL 8 library, we provide e.g. libssl.1.1.dylib, libmysqlclient.21.dylib and libcrypto.1.1.dylib for MacOS. Since that version of the library supports encrypted connection, it needs the SSL and Crypto libraries in the same folder as the MySQL library. You can copy the files in whatever folder you like, e.g. the one where the plugin is located. Please note that you may find older and newer libraries and sometimes you need an older library, because the server is not up to date.

For PostgreSQL you need the libraries: libssl.1.0.0.dylib, libpq.5.11.dylib and libcrypto.1.0.0.dylib. The version may be different and Windows has of course DLL as extension. Like MySQL you can copy them where you like and tell our plugin to load them.

For SQLite, you can use built-in library within MBS FileMaker Plugin. See SQL.InternalSQLiteLibrary.Activate function. Alternatively you can tell our plugin to use whatever version you need.

For Microsoft SQL Server on Windows you don't need a library file since it comes pre-installed on Windows.
If you try it on MacOS, you can load FreeTDS library and tell the plugin in the connection string where it is.

In all cases, the library files you download may not be notarized for MacOS. You may need to use Terminal commands like "xattr -cr /Users/cs/Documents/PostgreSQL" to remove quarantine flags for a folder. Then you should be able to load the libraries.

5. Connect

In a script we may need to connect to the server and for that we start a new connection:

Set Variable [$Connection; Value:MBS("SQL.NewConnection")]

If you store it in a global variable, you may keep it open and use it in calculations. Please be aware that the database server may have a timeout and close the connection when it is inactive too long. In that case you may run your connection script again, if you need the database again. The SQL.isAlive function may help to detect the connection state.

Next we tell the plugin where to find the library files:

Set Variable [$r; MBS("SQL.SetConnectionOption"; $Connection; "MYSQL.LIBS"; "c:\MySQL\libmySQL.dll")]

This is different option names for various database types: SQLANY.LIBS, CUBESQL.LIBS, DB2CLI.LIBS, DUCKDB.LIBS, IBASE.LIBS, INFCLI.LIBS, MYSQL.LIBS, ODBC.LIBS, OCI8.LIBS, LIBPQ.LIBS, SQLBASE.LIBS, SQLITE.LIBS, SQLNCLI.LIBS, SYBCT.LIBS, SYBINTL.LIBS, SYBCOMN.LIBS, SYBTCL.LIBS, SYBCS.LIBS.

All those have default names and if the library is findable in the path, you may only need the file name. For example installing Oracle client library may add the path for the DLL folder to the global path list and it just works.

Next you connect to the server:

Set Variable [$result; Value: MBS("SQL.Connect"; $Connection; $SQLConnectionString; $Username; $Password; $Type)]

You pass the type of the database for $Type on the end, e.g. "PostgreSQL". The connection string is usually a combination of database name, server name and various options.

For MySQL or PostgreSQL it may be just "MyServer@MyDatabase" where first part is IP or domain (optional with port separated with comma) for the server and after the @ follows the name of the database.

For Microsoft SQL Server on Windows you may just pass "MyServer\SQLEXPRESS@pubs" where the first part is the name of the instance for the server. Windows then resolves that to the actual IP. From macOS you may try with freeTDS and a connection thing like this: "DRIVER={" & $path &

"};Server=192.168.2.32;UId=SA;PWD=test;Database=test;TDS_VERSION=7.2;Port=1433". The path to the freetds.dylib is passed on macOS right away in the connection string followed with details to the server IP or domain, port and the requested TDS Version. If the user name and password is part of the connection string, you may not need to pass it separately. But usually we don't include passwords in connection string to avoid those leaking in a log file.

If you use ODBC, you can also configure a data source on the machine in ODBC configuration application. Then you can just pass "ODBC" as type and then pass the name of the data source as connection string. Otherwise a connection string for ODBC can point to the driver and pass parameters like for Microsoft Access: "Driver={Microsoft Access Driver (*.mdb, *.accdb)};Dbq=C:\mydatabase.accdb;". As you see we include name of the file path for the local database.

If $result now contains OK, we have a connection, but otherwise you can show an error.

6. Use connection

Next you may run your first query, for example a SELECT command to query the number of active staff for the company. As you see we use AS to name the column in the result. And for the parameter CompanyID we use a named parameter. Here is the line:

```
Set Variable [ $Command ; Value: MBS ( "SQL.NewCommand" ; $Connection ; "SELECT COUNT(*) AS \"Count\" FROM \"Person\" WHERE \"Active\" = 1 AND \"CompanyID\" = :CompanyID" ) ]
```

Next we set the parameter to the text for the company identifier (probably an UUID):

```
Set Variable [$SetParamResult; MBS( "SQL.SetParamAsText"; $Command; "CompanyID"; $CompanyID ) ]
```

Once parameters are set, we can execute the command:

```
Set Variable [$ExecuteResult; MBS( "SQL.Execute"; $Command ) ]
```

If that succeeds, we can fetch first record of result. For our example we get just one record and then can query the result:

```
Set Variable [$FetchResult; MBS( "SQL.FetchNext"; $Command ) ]
```

And if that succeeds we can query the field either by name:

```
Set Variable [$Count; MBS( "SQL.GetFieldAsNumber"; $Command; "Count" ) ]
```

Or by index:

```
Set Variable [$Count; MBS( "SQL.GetFieldAsNumber"; $Command; 1 ) ]
```

Then you can free the command and eventually free the connection.
Don't forget to check after each line for errors, e.g. with MBS("IsError") function.
You find more examples in our documentation, in older blog posts and the example databases.

See also:

- Using TLS when connecting to MySQL server
- ODBC driver and MBS SQL Functions in FileMaker
- Can FileMaker connect to a Microsoft Access database?
- CubeSQL Library for Mac
- Prefetching records from databases
- Tip of the day: Connect to MySQL and run a query
- SQLite in memory databases
- Connecting to Microsoft Visual FoxPro
- SQL Connectivity for FileMaker

Please do not hesitate to contact us if you have questions.