

MySQL Reader/Writer

FORMAT NOTES: This format is not supported by FME Base Edition.

Overview

MySQL^{®1} is a well known DataBase Management System (DBMS) that provides various types of tables for different database applications. The MySQL (Attributes Only) reader and writer plug-in provides the Feature Manipulation Engine (FME) with the ability to read and write attribute data stored in a MySQL database. There are two versions of the reader and writer, MYSQL which includes the spatial extension and supports geometry features, and MYSQL_DB which is an attribute only version which ignores geometry. Currently the table types that can be read are MyISAM, InnoDB and MEMORY.

The MySQL (Attributes Only) reader and writer communicate directly with the MySQL C API interface for maximum throughput.

This chapter assumes familiarity with MySQL, the table types, column types, available server daemons, indexing mechanisms and connection parameters.

Please note that MySQL functionality that only exists in the MySQL road map and not in practice such as server side cursors and views were considered but are not currently integrated into the MySQL reader for FME.

FME has been "MySQL[®] Network[™] Certified" against the MySQL Pro certified binary version 4.1.10a. MySQL Network is the only MySQL version certified for FME; users are encouraged to migrate to MySQL Network. For more information, please see the MySQL home page at: <http://www.mysql.com/>

1. MySQL is a registered trademark of MySQL AB in the United States, the European Union and other countries.

MySQL Quick Facts

Format Type Identifier	MYSQL_DB
Reader/Writer	Both
Licensing Level	
Dependencies	
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	None
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Always
Schema Required	Yes
Transaction Support	Yes
Geometry Type	mysql_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no
circular arc	no	raster	no
donut polygon	no	solid	no
elliptical arc	no	surface	no
ellipses	no	text	no
line	no	z values	no
none	yes		

Reader Overview

FME considers a MySQL dataset to be a database containing a collection of relational tables, and a table to be an FME feature type with each row corresponding to at least one FME feature.

Tables schemas must be defined in the FME mapping file before they can be read.

Arbitrary `WHERE` clauses and joins are fully supported as well as an entire arbitrary SQL `SELECT` statement. Support for `@SQL` and `@Relate` functions has also been added and further information is available for these functions in the *FME Functions, Factories and Transformers* document available from the Safe Software website.

The basic reading process involves opening a connection to the database, querying metadata, and querying data. The data is read by submitting SQL queries and parsing the returned result sets. Currently, result sets are either entire fetched to the client machine or fetched one at a time from.

Reader Directives

The suffixes listed are prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the MySQL reader is `MYSQL_DB_IN`.

DATASET

Required/Optional: *Required*

This specifies the name of the MySQL database. The database must exist in the DBMS. This can be verified by executing the query `SHOW DATABASES` in the MySQL query interpreter.

Example:

```
MYSQL_DB_DATASET testdb
```

HOST

Required/Optional: *Required*

This specifies the machine running the MySQL DBMS as either an IP address or host name. The database must have proper permissions and be set up to accept TCP/IP connections if connecting from a remote machine.

```
MYSQL_DB_IN_HOST myserver
```

PORT

Required/Optional: *Required*

When connecting remotely, this specifies the TCP/IP port on which to connect to the DBMS service. The default port is 3306.

```
MYSQL_DB_IN_PORT 3306
```

USER_NAME

Required/Optional: *Required*

The name of user who will access the database. The named user must exist with appropriate MySQL permissions. The default user name is `mysql`.

```
MYSQL_DB_IN_USER_NAME mysql
```

PASSWORD

Required/Optional: *Optional*

The password of the user accessing the database. This parameter is optional when connecting with a username that has a blank password associated with it.

```
MYSQL_DB_IN_PASSWORD secret
```

DEF**Required/Optional:** *Required*

The syntax of the definition is:

```
MYSQL_DB_DEF <tableName> \
  [mysql_where_clause <whereClause>] \
  [<fieldName> <fieldType>] +
```

OR

```
MYSQL_DB_DEF <queryName> \
  [mysql_sql_statement <sqlQuery>]
```

The <tableName> must match the name of an existing MySQL table in the database. This will be used as the feature type of all the features read from the table. The exception to this rule is when using the `sql_statement` keyword. In this case, the `DEF` name may be any valid alphabetic identifier; it does not have to be an existing table name – rather, it is an identifier for the custom SQL query. The feature type of all the features returned from the SQL query are given the query name.

The <fieldType> of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored. The exception to this is the `geometry` field type which is not placed in the `DEF`.

The definition allows specification of separate search parameters for each table. If any of the per table configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader keywords listed above. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
<code>where_clause</code>	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, then all the tuples are returned. This keyword will be ignored if the <code>sql_statement</code> is present.
<code>sql_statement</code>	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the MySQL reader will execute the query, and use the resulting rows as the features instead of reading from the table <queryName>. All returned features will have a feature type of <queryName>, and attributes for all columns selected by the query.

If no <whereClause> is specified, all rows in the table will be read and returned as individual features. If a <whereClause> is specified, only those rows that are selected by the clause will be read. Note that the <whereClause> does not include the word `WHERE`.

The MySQL (Attributes Only) reader allows one to use the `sql_statement` parameter to specify an arbitrary SQL `SELECT` query on the `DEF` line. If this is specified, FME will execute the query, and use each row of data returned from the query to define at least one feature. Each of these features will be given the feature type named in the `DEF`

line, and will contain attributes for every column returned by the `SELECT`. In this case, all `DEF` line parameters regarding a `WHERE` clause or spatial querying are ignored, as it is possible to embed this information directly in the text of the `<sqlQuery>`.

The following example selects rows from the table `ROADS`, placing the resulting data into FME features with a feature type of `MYROADS`. Imagine that `ROADS` defines a numeric field named `ID` and a text field named `NAME`.

```
MYSQL_DB_DEF MYROADS \
  sql_statement 'SELECT id, name FROM ROADS'
```

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables files that will be read. If no `IDs` are specified, then all defined and available tables are read. The syntax of the `IDs` keyword is:

```
MYSQL_DB_IDS <featureType1> \
  <featureType2> \
  <featureTypeN>
```

The feature types must match those used in `DEF` lines.

The example below selects only the `ROADS` table for input during a translation:

```
MYSQL_DB_IDS ROADS
```

FETCH_ALL_FEATURES

Required/Optional: *Optional*

This keyword is used to specify that the entire result set of the main feature query should be retrieved into a large buffer in client memory. Otherwise, the default is to be less memory intensive and to retrieve one row at a time from the server, caching them in a smaller buffer. This allows for large datasets to be processed by default without the possibility of running out of client memory. This keyword can be set to `YES` to improve the performance of smaller queries.

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

When set to 'Yes', indicates to the reader to return all the schemas of the tables in the database.

If this specification is missing then it is assumed to be 'No'.

Range: YES | NO

Default: NO

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to `RETRIEVE_ALL_SCHEMAS`: this optional specification is used to tell the reader to only retrieve the table names of all the tables in the source database. If `RETRIEVE_ALL_SCHEMAS` is also set to Yes, then `RETRIEVE_ALL_SCHEMAS` takes precedence. If this value is not specified, then it is assumed to be No.

Range: YES | NO

Default: NO

Writer Overview

The MySQL (Attributes Only) writer module stores both geometry and attributes into an MySQL database. The MySQL (Attributes Only) writer provides the following capabilities:

- **Transaction Support:** The MySQL (Attributes Only) writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication. Performance can also be improved by reducing transactional overhead for multiple small queries such as inserts.
- **Index Creation:** The MySQL (Attributes Only) writer can set up and populate indexes as part of the loading process. By default, no indexes are created. Columns can be individually indexed. Composite column indexes are not supported at this time.
- **Insert Binding:** By default, the MySQL (Attributes Only) writer uses prepared statements and query parameter binding ensure speedy data loading.

Writer Directives

The suffixes shown are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the MySQL (Attributes Only) writer is `MYSQL_DB_OUT`.

DATASET/DATABASE, HOST, PORT, USER_NAME, PASSWORD

These directives operate in the same manner as they do for the MySQL (Attributes Only) reader. The remaining writer-specific directives are discussed in the following sections.

DEF

Required/Optional: *Optional*

Each MySQL table must be defined before it can be written. The general form of a MySQL (Attributes Only) definition statement is:

```

MYSQL_DB_DEF <tableName> \
    [mysql_type <mysql_type>] \
    [mysql_mode (inherit_from_writer|insert|update|delete)] \
    [mysql_table_type <mysql_table_type>] \
    [mysql_drop_table (yes|no)] \
    [mysql_truncate_table (yes|no)] \
    [<fieldName> <fieldType>][,<indexType>]*
    
```

The table definition allows control of the table that will be created. If the table already exists, the majority of the `mysql_` parameters will be ignored and need not be given. If the fields and types are listed, they must match those in the database.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a `<fieldType>` is given, it may be any field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
<code>tableName</code>	The name of the table to be written. If a table with the specified name exists, it will be overwritten if either the <code>mysql_overwrite_table DEF</code> line parameter set to <code>YES</code> or if the global writer keyword type <code>mysql_out_overwrite</code> is set to <code>YES</code> . Otherwise the table will be appended. Valid values for table names include any character string devoid of SQL-offensive characters and less than 32 characters in length.
<code>mysql_type</code>	The type of entity stored within the feature. This should always be set to the following: <code>mysql_none</code>
<code>mysql_mode</code>	The the default operation mode of the feature type in terms of the types of SQL statements sent to the database. Valid values are <code>INSERT</code> , <code>UPDATE</code> , <code>DELETE</code> and <code>INHERIT_FROM_WRITER</code> . Note that <code>INSERT</code> mode allows for only <code>INSERT</code> operations where as <code>UPDATE</code> and <code>DELETE</code> can be overwritten at the feature levels. <code>INHERIT_FROM_WRITER</code> simply indicates to take this value from the writer level and not to override it at the feature type level. Default: <code>INHERIT_FROM_WRITER</code>
<code>mysql_table_type</code>	The type of table to be created. The valid values for the type are listed below: <code>MEMORY</code> <code>InnoDB</code> <code>MYISAM</code> Please refer to the MySQL manual for further information on these table types. Default: <code>MyISAM</code>
<code>mysql_drop_table</code>	This specifies that if the table exists by this name, it should be dropped and recreated before any features are written to it. This parameter, along with <code>mysql_truncate_table</code> , deprecates the older <code>mysql_overwrite_table</code> parameter. Default: <code>NO</code>

Parameter	Contents
<code>mysql_truncate_table</code>	This specifies that if the table exists by this name, it should be truncated before any features are written to it. This parameter, along with <code>mysql_drop_table</code> , deprecates the older <code>mysql_overwrite_table</code> parameter. Default: NO
<code>fieldName</code>	The name of the field to be written. Valid values for field name include any character string devoid of SQL-offensive characters and less than 63 characters in length.
<code>fieldType</code>	The type of a column in a table. The valid values for the field type are listed below: int smallint tinyint mediumint bigint decimal(width, precision) float(width, precision) double(width, precision) char(width) varchar(width) date time datetime timestamp year tinyblob blob mediumblob longblob tinytext text mediumtext longtext
<code>indexType</code>	The type of index to create on the given field. The valid values for the index type are listed below: BTREE (default attribute index) PRIKEY (primary key)

STRIP_WHITESPACE

Required/Optional: *Optional*

This statement tells the MySQL (Attributes Only) writer module whether or not whitespace should be stripped from field values.

If the `MYSQL_DB_OUT_STRIP_WHITESPACE` statement is not specified or is set to `YES`, both leading and trailing whitespace will be stripped from field values.

Example:

```
MYSQL_DB_OUT_STRIP_WHITESPACE YES
```

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the MySQL (Attributes Only) writer module when to start actually writing features into the database. The MySQL (Attributes Only) writer does not write any features until the feature number of features are skipped, and then it begins writing the following features. Normally, the value specified is zero – a non-zero value is usually only specified when a data load operation is being resumed after failing partway through.

Example:

```
MYSQL_DB_OUT_START_TRANSACTION 0
```

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database. Setting the transaction interval 0 results in the enabling of auto commit transaction mode.

If the `MYSQL_DB_OUT_TRANSACTION_INTERVAL` statement is not specified, then a value of 1000 is used as the transaction interval.

Example:

```
MYSQL_DB_OUT_TRANSACTION_INTERVAL 2000
```

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter *Database Writer Mode* on page 19.

This directive informs the MySQL (Attributes Only) writer which SQL operations will be performed by default by this writer. This operation can be set to `INSERT`, `UPDATE` or `DELETE`. The default writer level value for this operation can be overwritten at the feature type or table level. The corresponding feature type def parameter name is called `MYSQL_MODE`. It has the same valid options as the writer level mode and additionally the value `INHERIT_FROM_WRITER` which causes the writer level mode to be inherited by the feature type as the default for features contained in that table.

The operation can be set specifically for individual feature as well. Note that when the writer mode is set to `INSERT` this prevents the mode from being interpreted from individual features and all features are inserted unless otherwise marked as `UPDATE` or `DELETE` features. These are skipped.

If the `MYSQL_DB_WRITER_MODE` statement is not specified, then a value of `INSERT` is given.

Example:

```
MYSQL_DB_OUT_WRITER_MODE INSERT
```

Feature Representation

Features read from MySQL consist of a series of attribute values but no geometry. The feature type of each feature is as defined on its `DEF` line.

Features written to the database have the destination table as their feature type, and attributes as defined on the `DEF` line.

Features read from, or written to, MySQL have an attribute for each column in the database table. The feature attribute name will be the same as the source or destination column name. The attribute and column names are case-insensitive as opposed to database and table names whose case sensitivity depends on the underlying operating system.

All Features

mysql_type: `mysql_none`

All features are tagged with this value when reading or writing to or from MySQL (Attributes Only).

Troubleshooting

Problems sometimes arise when attempting to connect to a MySQL database. This is almost always due to a misconfiguration in the user's environment. The following suggestions can often help detect and overcome such problems.

- Ensure you can connect to the database with the host, port, database, user name, and password using the `mysql` client. See MySQL documentation for proper security and connection information, and for the usage of the `mysql` client.
- If you try to list the tables and nothing happens, check the log file. There may have been an underlying error that didn't generate a dialog. Usually this means a parameter does not exist or permissions are not sufficient to access the requested resource.
- In most cases, the `MYSQL_DB_DATABASE` keyword should be left with blank values, with the `MYSQL_DB_DATASET` keyword containing the name of the MySQL database.
- When using a UNIX operating system, the environment variables `MYSQL_HOST`, `MYSQL_TCP_PORT`, `USER` and `MYSQL_PWD` can be used to specify the MySQL connection parameters.
- If you receive an error like *Lost connection to MySQL server during query*, you may have to increase your maximum allowed packet size server variable. This can be done by editing the `my .ini` file in the MySQL server installation directory. The line to edit, or add if not present, is:

```
max_allowed_packets=8M
```

The default is 1M, meaning maximum packet size is 1 MB. We recommend 8MB (8M) or more if you are moving large geometric features.

- Various other issues may arise depending on the values of the various option files permitted for a MySQL database and for specific servers.
- Lastly, try reading up on common MySQL problems at <http://dev.mysql.com/doc/mysql/en/problems.html>

Translation Errors in Workbench

There is a known issue with the MySQL (Attributes Only) Writer and Workbench Feature Type Properties.

The Feature Type Properties for the MySQL Writer are shown at right.

The **Database User** field should be left blank. Entering a username in the field may cause the FME translation to fail. However, even if the translation is successful, MySQL will not be able to read the resulting table.

