MySQL Shell User Guide

Abstract

This is the MySQL Shell User Guide extract from the MySQL 5.7 Reference Manual.

MySQL Shell enables you to prototype code using the X DevAPI to communicate with a MySQL Server running the X Plugin. The X Plugin is a new MySQL Server feature available with MySQL Server 5.7.12 and higher.

MySQL Shell is an advanced command-line client and code editor for the MySQL Server. In addition to SQL, MySQL Shell also offers scripting capabilities for JavaScript and Python. When MySQL Shell is connected to the MySQL Server through the X Protocol, the X DevAPI can be used to work with both relational and document data.

For help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists, where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Document generated on: 2017-09-20 (revision: 54039)

Table of Contents

Preface and Legal Notices

This is the MySQL Shell User Guide extract from the MySQL 5.7 Reference Manual.

Legal Notices

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Chapter 1 MySQL Shell Features

The following features are available in MySQL Shell.

Interactive Code Execution

MySQL Shell provides an interactive code execution mode, where you type code at the MySQL Shell prompt and each entered statement is processed, with the result of the processing printed onscreen.

Supported Languages

MySQL Shell processes code in the following languages: JavaScript, Python and SQL. Any entered code is processed as one of these languages, based on the language that is currently active. There are also specific MySQL Shell commands, prefixed with `\`, which enable you to configure MySQL Shell regardless of the currently selected language. For more information see Chapter 4, *MySQL Shell Commands*.

Batch Code Execution

In addition to the interactive execution of code, MySQL Shell can also take code from different sources and process it. This method of processing code in a non-interactive way is called *Batch Execution*.

As batch execution mode is intended for script processing of a single language, it is limited to having minimal non-formatted output and disabling the execution of commands. To avoid these limitations, use the `--interactive` command-line option, which tells MySQL Shell to execute the input as if it were an interactive session. In this mode the input is processed *line by line* just as if each line were typed in an interactive session. For more information see Section 3.5, “Batch Mode Made Interactive”.

Output Formats

MySQL Shell provides output in different formats depending on how it is used: Tabbed, Table and JSON. For more information see Section 3.3, “Output Formats”.

Multiple-line Support

Multiple-line code can be written using a command, enabling MySQL Shell to cache multiple lines and then execute them as a single statement. For more information see Section 3.5.1, “Multiple-line Support”.

Application Log

MySQL Shell can be configured to log information about the execution process. For more information see Chapter 5, *MySQL Shell Application Log*.

Supported APIs

MySQL Shell includes the following APIs implemented in JavaScript and Python which you can use to develop code that interacts with MySQL.

- The X DevAPI enables you to work with both relational and document data when MySQL Shell is connected to a MySQL server using the X Protocol. For more information, see Using MySQL as a Document Store. For general information on using X DevAPI, see X DevAPI User Guide.
- The AdminAPI enables you to work with InnoDB cluster, which provides an integrated solution for high availability and scalability using InnoDB based MySQL databases, without requiring advanced MySQL expertise. See InnoDB Cluster.

For specific documentation on the implementation of the APIs see JavaScript and Python.

X Protocol Support

MySQL Shell is designed to provide an integrated command-line client for all MySQL products which support X Protocol. The development features of MySQL Shell are designed for sessions using the X Protocol. MySQL Shell can also connect to MySQL Servers that do not support the X Protocol using the legacy MySQL Protocol. A minimal set of features from the X DevAPI are available for sessions created using the legacy MySQL protocol.

Global Session

Interaction with a MySQL Server is done through a Session object. For Python and JavaScript, a Session can be created through the `getSession` and `getNodeSession` functions of the mysqlx module. If a session is created in JavaScript mode using any of these methods, it is available only in JavaScript mode. The same happens if the session is created in Python mode. None of these sessions can be used in SQL mode.

For SQL Mode, the concept of Global Session is supported by the MySQL Shell. A Global Session is created when the connection information is passed to MySQL Shell using command-line options, or by using the `\connect` command.

The Global Session is used to execute statements in SQL mode and the same session is available in both Python or JavaScript modes. When a Global Session is created, a variable called `session` is set in the scripting languages, so you can execute code in the different languages by switching the active mode.

For more information, see Section 2.1, “MySQL Shell Sessions”.

Chapter 2 Getting Started with MySQL Shell

Table of Contents

This section describes how to get started with MySQL Shell. This section assumes you have a MySQL Server running X Plugin and that you have installed MySQL Shell, see Setting Up MySQL as a Document Store.

2.1 MySQL Shell Sessions

This section explains the different types of sessions in MySQL Shell and how to create and configure them.

2.1.1 MySQL Shell Sessions Explained

MySQL Shell is a unified interface to operate MySQL Server through scripting languages such as JavaScript or Python. To maintain compatibility with previous versions, SQL can also be executed in certain modes. A connection to a MySQL server is required. In MySQL Shell these connections are handled by a *Session* object.

The following types of Session object are available:

- *XSession*: Use this session type for new application development. It offers the best integration with MySQL Server, and therefore, it is used by default. SQL execution is not supported and therefore it is not compatible with MySQL Shell's *SQL Mode*.
- *Node Session*: Use this session type for SQL execution on a MySQL Server with the X Protocol enabled. SQL execution is available with this session type, therefore it can be used in MySQL Shell's *SQL Mode*.

  This session type should only be used when connecting *directly* to an X Protocol enabled MySQL Server.
- *Classic Session* Use this session type to interact with MySQL Servers that do not have the X Protocol enabled. SQL execution is available with this session type, therefore it can be used in MySQL Shell's *SQL Mode*.

  The development API available for this type of session is very limited. For example, there are no CRUD operations, no collection handling, and binding is not supported.

2.1.2 Choosing a MySQL Shell Session Type

MySQL Shell creates an XSession object by default. To choose which type of session should be created, use one of these options:

- `--node` creates a Node Session.
- `--classic` creates a Classic Session.
- `--x` creates an XSession.

For more information, see Section 2.2.1, “Connecting using a URI String” and Section 2.2.2, “Connecting using Individual Parameters”.

2.2 MySQL Shell Connections

MySQL Shell can be configured to connect to a MySQL Server using command options when starting the application, or from within MySQL Shell itself using the `\connect` command. The address of the MySQL Server which you want to connect to can be specified using individual parameters, such as user, hostname and port, or using a Uniform Resource Identifier (URI) in the format `user@host:port/schema`, such as `mike@myserver:33060/testDB`. The following sections describe these connection methods.

> **Important**
>
> Regardless of the method you choose to connect it is important to understand how passwords are handled by MySQL Shell. By default connections are assumed to require a password. The password is requested at the login prompt. To specify a password-less account use the `--password=` option and do not specify a password, or use a `:` after the `user` in a URI and do not specify a password.

If you do not specify parameters for a connection the following defaults are used:

- user defaults to the current system user name
- host defaults to localhost
- port defaults to the X Plugin port 33060 when using an X Session, and port 3306 when using a Classic session

MySQL Shell connections using X Protocol *always* use TCP, using Unix sockets is not supported. MySQL Shell connections using MySQL Protocol default to using Unix sockets when the following conditions are met:

- `--port` is not specified
- `--host` is not specified or it is equal to `localhost`
- `--socket` is provided with a path to a socket
- `--classic` is specified

If `--host` is specified but it is not `localhost`, a TCP connection is established. In this case, if `--port` is not specified the default value of 3306 is used. If the conditions are met for a socket connection but `--socket` is not specified then the default socket is used, see Connecting to the MySQL Server.

2.2.1 Connecting using a URI String

You configure the MySQL Server which MySQL Shell connects to by passing the connection data in string format using the `--uri` command option.

Use the following format:

```
[dbuser[:[dbpassword]]@]host[:port][/schema]
```

Descriptions of these options:

- *dbuser*: specifies the MySQL user account to be used for the authentication process.
- *dbpassword*: specifies the dbuser password to be used for the authentication process.

  > **Warning**
  >
  > Storing the password in the URI is not recommended.

- *host*: specifies the host to which the session object connects. If not specified, *localhost* is used by default.
- *port*: specifies which port the target MySQL server is listening on for the connection. If not specified, 33060 is used by default for the X Protocol enabled sessions, and 3306 is the default for traditional MySQL protocol sessions.
- *schema*: specifies the database to be set as default when the session is established.

If no password is specified using the URI, which is recommended, then the password is prompted for. The following examples show how to use these command options:

- Connect with a Node Session at port 33065.

  ```
  shell> mysqlsh --uri user@localhost:33065 --node
  ```

- Connect with a Classic Session.

  ```
  shell> mysqlsh --uri user@localhost --classic
  ```

Although using a password-less account is not recommended, you can specify a user without a password using a `:` after the user name, for example:

```
shell> mysqlsh --uri user:@localhost
```

2.2.2 Connecting using Individual Parameters

In addition to specifying connection parameters using a URI, it is also possible to define the connection data using separate parameters for each value.

Use the following parameters:

- `--dbuser` (`-u`) *value*
- `--dbpassword` *value*
- `--host` (`-h`) *value*
- `--port` (`-P`) *value*
- `--schema` (`-D`) *value*
- `--password` (`-p`)
- `--socket` (`-S`)

The first 5 parameters match the tokens used in the URI format described at Section 2.2.1, “Connecting using a URI String”.

The `--password` parameter indicates the user should connect *without* a password.

For consistency, the following aliases are supported for some parameters:

- `--user` is equivalent to `--dbuser`
- `--password` is equivalent to `--dbpassword`
- `--database` is equivalent to `--schema`

When parameters are specified in multiple ways, the following rules apply:

- If an argument is specified more than once the value of the last appearance is used.
- If both individual connection arguments and `--uri` are specified, the value of `--uri` is taken as the base and the values of the individual arguments override the specific component from the base URI.

Attempt to establish an XSession with a specified user at port 33065.

```
shell> mysqlsh -u user -h localhost -P 33065
```

Attempt to establish a Classic Session with a specified user.

```
shell> mysqlsh -u user -h localhost --classic
```

Attempt to establish a Node Session with a specified user.

```
shell> mysqlsh --node -u user -h localhost
```

2.2.3 Using SSL for Encrypted Connections

Using SSL is possible when connecting to an SSL enabled MySQL server.

To configure an SSL connection, use the following command options:

- `--ssl` : This enables or disables connections through SSL. If set to 0, the other SSL command options are ignored.
- `--ssl-ca=filename`: The path to a file in PEM format that contains a list of trusted SSL certificate authorities.
- `--ssl-cert=filename`: The name of the SSL certificate file in PEM format to use for establishing an encrypted connection.
- `--ssl-key=filename`: The name of the SSL key file in PEM format to use for establishing an encrypted connection.

The `--ssl` option is assumed to be 1 (enabled) if the other SSL options are set.

2.2.4 Connecting using both URI and Individual Parameters

When the `--uri` option is specified in combination with some of the individual parameters, the address specified by the `--uri` option is used as the base connection data and the values provided using individual parameters override the corresponding value from the URI. If the `--user` option is used, it would replace any user specified as part of a URI.

For example to establish an XSession and override *`user`* from the URI:

```
shell> mysqlsh --uri user@localhost:33065 --user otheruser
```

2.2.5 Creating a Session Using Shell Commands

If you open MySQL Shell without specifying connection parameters, MySQL Shell opens without an established global session. It is possible to establish a global session once MySQL Shell has been started using the following *Shell Commands*:

- `\connect URI`: Creates an XSession.
- `\connect -n URI`: Creates a Node Session.
- `\connect -c URI`: Creates a Classic Session.

Configure the connection using the URI parameter, which follows the same syntax as for the `--uri` command option. For additional information, see Section 2.2.1, “Connecting using a URI String”.

For example:

```
mysql-js> \connect root@localhost
Creating XSession to root@localhost...
Enter password: ****
No default schema selected.
mysql-js>
```

2.2.5.1 Creating an Encrypted Session Using SSL

To establish an SSL connection, the URI parameter passed to the connect shell commands must include the SSL information as URL parameters. For example:

```
mysql-js> \connect root@localhost?ssl_ca=/path/to/ca/file&\
ssl_cert=/path/to/cert/file&ssl_key=/path/to/key/file
Creating XSession to root@localhost...
Enter password: ****
No default schema selected.
mysql-js>
```

2.2.6 Connections in JavaScript and Python

When a connection is made using the command options or by using any of the shell commands, a global session object is created. This session is global because once created, it can be used in any of the MySQL Shell execution modes.

Any global session object is available in JavaScript or Python modes because a variable called **session** holds a reference to it.

In SQL mode, both Node and Classic sessions can be used because they both expose SQL execution. In SQL mode an XSession cannot be used.

In addition to the global session object, sessions can be established and assigned to a different variable by using the functions available in the mysql and mysqlx JavaScript and Python modules.

For example, the following functions are provided by these modules:

- **mysql.getSession(connectionData[, password])**
- **mysqlx.getNodeSession(connectionData[, password])**
- **mysql.getClassicSession(connectionData[, password])**

The first of these functions is used to create an XSession which features the most comprehensive development API and supports X Protocol.

The second creates a Node Session which connects to a X Protocol enabled MySQL Server and allows SQL Execution.

The latter returns a Classic Session object which uses the traditional MySQL protocol and has a very limited development API.

connectionData can be either a URI as specified above or a dictionary containing the connection parameters. See Section 2.2.1, “Connecting using a URI String”.

The following example shows how to create a Node Session using the X Protocol:

```
mysql-js> var mysqlx=require('mysqlx').mysqlx;
mysql-js> var session=mysqlx.getNodeSession('root@localhost');
mysql-js> print(session)
<NodeSession:root@localhost>
mysql-js>
```

The following example shows how to create a Node Session using the X Protocol so that you can execute SQL:

```
mysql-js> var mysqlx=require('mysqlx').mysqlx;
mysql-js> var session=mysqlx.getNodeSession({host: 'localhost', dbUser: 'root'});
mysql-js> print(session)
<NodeSession:root@localhost>
mysql-js>
```

The following example shows how to create a Classic Session:

```
mysql-js> var mysql=require('mysql').mysql;
mysql-js> var session = mysql.getClassicSession('root@localhost:3307');
mysql-js> print(session)
<ClassicSession:root@localhost:3307>
mysql-js>
```

2.2.6.1 Using SSL for Encrypted Connections

To establish an SSL connection, set the SSL information in the connectionData dictionary. For example:

```
mysql-js> var mysqlx=require('mysqlx').mysqlx;
mysql-js> var session=mysqlx.getNodeSession({host: 'localhost',
                                             dbUser: 'root',
                                             dbPassword: 'mypasswd',
                                             ssl_ca: "path_to_ca_file",
                                             ssl_cert: "path_to_cert_file",
                                             ssl_key: "path_to_key_file"});
mysql-js> print(session)
<NodeSession:root@localhost>
```

2.3 MySQL Shell Global Variables

MySQL Shell reserves certain variables as global variables, which are assigned to commonly used objects in scripting. This section describes the available global variables and provides examples of working with them. The global variables are:

- `session` represents the global session if one has been established.
- `db` represents a schema if one has been defined, for example by a URI.

MySQL Shell provides interactive error resolution for common situations related to using the global variables. For example:

- Attempting to use an undefined `session` global variable.
- Attempting to retrieve an nonexistent schema using `session`.
- Attempting to use an undefined `db` global variable.

Undefined Global Session

The global `session` variable is set when a global session is established. When a global session is established, issuing a `session` statement in MySQL Shell displays the session type and its URI as follows:

```
mysql-js> session
<XSession:root@localhost:33060>
mysql-js>
```

If no global session has been established, MySQL Shell displays the following:

```
mysql-js> session
<Undefined>
mysql-js>
```

If you attempt to use the `session` variable when no global session is established, interactive error resolution starts and you are prompted to provide the required information to establish a global session. If the session is successfully established, it is assigned to the `session` variable. The prompts are:

- An initial prompt explains that no global session is established and asks if one should be established.
- If the you choose to set a global session, the session type is requested.
- Either the URI or the alias of a stored session is requested.
- If required, a password is requested.

For example:

```
mysql-js> session.uri
The global session is not set, do you want to establish a session? [y/N]: y
Please specify the session type:

1) X
2) Node
3) Classic

Type: 2
Please specify the MySQL server URI (or $slias): root@localhost
Enter password:*******
root@localhost:33060
mysql-js>
```

Undefined db Variable

The global `db` variable is set when a global session is established and a default schema is configured. For example, using a URI such as `root@localhost/sakila` to establish a global session connected to the MySQL Server at `localhost`, on port 33060, as the user `root`, assigns the schema `sakila` to the global variable `db`. Once a schema is defined, issuing `db` at the MySQL Shell prompt prints the schema name as follows:

```
mysql-js> db
<Schema:world_x>
mysql-js>
```

If there is no global session established, the following is displayed:

```
mysql-js> db
<Undefined>
mysql-js>
```

If you attempt to use the `db` variable when no global session has been established, the following error is displayed:

```
mysql-js> db.getCollections()
LogicError: The db variable is not set, establish a global session first.
at (shell):1:2
in db.getCollections()
^
```

If a global session has been established but you attempt to use an undefined `db`, interactive error resolution begins and you are prompted to define an active schema by providing the schema name. If this succeeds the `db` variable is set to the defined schema. For example:

```
mysql-js> db.getCollections()
The db variable is not set, do you want to set the active schema? [y/N]:y
Please specify the schema:world_x
[
<Collection:countryinfo>
]
mysql-js> db
<Schema:world_x>
mysql-js>
```

Retrieving an Nonexistent Schema

If you attempt to use `session` to retrieve an nonexistent schema, interactive error resolution provides the option to create the schema.

```
mysql-js> var mySchema = session.getSchema('my_test')
The schema my_test does not exist, do you want to create it? [y/N]: y

mysql-js> mySchema
<Schema:my_test>
mysql-js>
```

In all cases, if you do not provide the information required to resolve each situation, a proper result of executing the requested statement on an undefined variable is displayed.

Chapter 3 MySQL Shell Code Execution

Table of Contents

This section explains how code execution works in MySQL Shell.

3.1 Interactive Code Execution

The default mode of MySQL Shell provides interactive execution of database operations that you type at the command prompt. These operations can be written in JavaScript, Python or SQL depending on the type of session being used. When executed, the results of the operation are displayed on-screen.

As with any other language interpreter, MySQL Shell is very strict regarding syntax. For example, the following JavaScript snippet reads and prints the documents in a collection:

```
var mysqlx = require('mysqlx').mysqlx;
var mySession = mysqlx.getSession('user:pwd@localhost');
var result = mySession.world_x.countryinfo.find().execute();
var record = result.fetchOne();
while(record){
  print(record);
  record = result.fetchOne();
}
```

As seen above, the call to `find()` is followed by the `execute()` function. CRUD database commands are only actually executed on the MySQL Server when `execute()` is called. However, when working with MySQL Shell interactively, `execute()` is implicitly called whenever you press `Return` on a statement. Then the results of the operation are fetched and displayed on-screen. The rules for when you need to call `execute()` or not are as follows:

- When using MySQL Shell in this way, calling `execute()` becomes optional on:
  - `Collection.add()`
  - `Collection.find()`
  - `Collection.remove()`
  - `Collection.modify()`
  - `Table.insert()`
  - `Table.select()`
  - `Table.delete()`
  - `Table.update()`
  - `NodeSession.sql()`

- Automatic execution is disabled if the object is assigned to a variable. In such a case calling `execute()` is mandatory to perform the operation.
- When a line is processed and the function returns any of the available `Result` objects, the information contained in the Result object is automatically displayed on screen. The functions that return a Result object include:
  - The SQL execution and CRUD operations (listed above)
  - Transaction handling and drop functions of the session objects in both mysql and mysqlx modules: -
    - `startTransaction()`
    - `commit()`
    - `rollback()`
    - `dropSchema()`
    - `dropTable()`
    - `dropCollection()`
    - `dropView()`
    - `ClassicSession.runSql()`

Based on the above rules, the statements needed in the MySQL Shell in interactive mode to establish a session, query, and print the documents in a collection are:

```
mysql-js> var mysqlx = require('mysqlx').mysqlx;
mysql-js> var mySession = mysqlx.getSession('user:pwd@localhost');
```

No call to `execute()` is needed and the Result object is automatically printed.

```
mysql-js> mySession.world_x.countryinfo.find();
```

Formatting Results Vertically

When executing SQL using MySQL Shell you can display results in a column-per-row format with the `\G` command, in a similar way to `mysql`. If a statement is terminated with `\G` instead of the active delimiter (which defaults to `;`), it is executed by the server and the results are displayed in vertical format, regardless of the current default output format. For example issuing a statement such as

```
SELECT * FROM mysql.user \G
```

displays the results vertically.

Multiple SQL statements on the same line which are separated by `\G` are executed separately as if they appeared one per line., for example

```
select 1\Gselect 2\Gselect 3\G
```

In other words `\G` functions as a normal delimiter.

3.2 Batch Code Execution

As well as interactive code execution, MySQL Shell provides batch code execution from:

- A file loaded for processing.

- A file containing code that is redirected to the standard input for execution.
- Code from a different source that is redirected to the standard input for execution.

For example:

Loading SQL code from a file for batch processing.

```
shell> mysqlsh --file code.js
```

Redirecting a file to standard input for execution.

```
shell> mysqlsh < code.js
```

Redirecting code to standard input for execution.

```
shell> echo "show databases;" | mysqlsh --sql --uri root@192.168.1.141:33060
```

Executable Scripts

Starting with version 1.0.4, on Linux you can create executable scripts that run with MySQL Shell by including a `#!` line as the first line of the script. This line should provide the full path to MySQL Shell and include the `--file` option. For example:

```
#!/usr/local/mysql-shell/bin/mysqlsh --file
print("Hello World\n");
```

The script file must be marked as executable in the filesystem. Running the script invokes MySQL Shell and it executes the contents of the script.

3.3 Output Formats

The output of the commands processed on the server can be formatted in different ways. This section details the different available output formats.

3.3.1 Table Format

The table format is used by default when MySQL Shell is in interactive mode. The output is presented as a formatted table for a better view and to aid analysis.

```
mysql-sql> select * from sakila.actor limit 3;
+----------+-------------+----------------+---------------------+
| actor_id | first_name  | last_name      | last_update         |
+----------+-------------+----------------+---------------------+
|        1 | PENELOPE    | GUINESS        | 2006-02-15 4:34:33  |
|        2 | NICK        | WAHLBERG       | 2006-02-15 4:34:33  |
|        3 | ED          | CHASE          | 2006-02-15 4:34:33  |
+----------+-------------+----------------+---------------------+
3 rows in set (0.00 sec)
mysql-sql>
```

To get this output format when running in batch mode, use the `--table` command-line option.

3.3.2 Tab Separated Format

This format is used by default when running MySQL Shell in batch mode, to have better output for automated analysis.

```
>echo "select * from sakila.actor limit 3;" | mysqlsh --classic --uri root@192.168.1.141:33460
actor_id        first_name      last_name       last_update
```

```
1       PENELOPE        GUINESS 2006-02-15 4:34:33
2       NICK    WAHLBERG        2006-02-15 4:34:33
3       ED      CHASE   2006-02-15 4:34:33
```

3.3.3 JSON Format Output

MySQL Shell supports the JSON format for output and it is available both in interactive and batch mode. This output format can be enabled using the `--json` command-line option:

JSON Format in Batch Mode

```
shell>echo "select * from sakila.actor limit 3;" | mysqlsh --json --sqlc --uri root@192.168.1.141:3306
{"duration":"0.00 sec","info":"","row_count":3,"rows":[[1,"PENELOPE","GUINESS",{"year":2006,"month":1,"day"
shell>echo "select * from sakila.actor limit 3;" | mysqlsh --json=raw --sqlc --uri root@192.168.1.141:3306
{"duration":"0.00 sec","info":"","row_count":3,"rows":[[1,"PENELOPE","GUINESS",{"year":2006,"month":1,"day"
shell>echo "select * from sakila.actor limit 3;" | mysqlsh --json=pretty --sqlc --uri root@192.168.1.141:33
{
    "duration": "0.00 sec",
    "info": "",
    "row_count": 3,
    "rows": [
        [
            1,
            "PENELOPE",
            "GUINESS",
            {
                "year": 2006,
                "month": 1,
                "day": 15,
                "hour": 4,
                "minute": 34,
                "second": 33.0
            }
        ],
        [
            2,
            "NICK",
            "WAHLBERG",
            {
                "year": 2006,
                "month": 1,
                "day": 15,
                "hour": 4,
                "minute": 34,
                "second": 33.0
            }
        ],
        [
            3,
            "ED",
            "CHASE",
            {
                "year": 2006,
                "month": 1,
                "day": 15,
                "hour": 4,
                "minute": 34,
                "second": 33.0
            }
        ]
    ],
    "warning_count": 0
}
shell>
```

JSON Format in Interactive Mode (started with --json=raw)

```
mysql-sql> select * from sakila.actor limit 3;
{"duration":"0.00 sec","info":"","row_count":3,"rows":[[1,"PENELOPE","GUINESS",{"year":2006,"month":1,"day"
```

```
mysql-sql>
```

JSON Format in Interactive Mode (started with --json=pretty)

```
mysql-sql> select * from sakila.actor limit 3;
{
    "duration": "0.00 sec",
    "info": "",
    "row_count": 3,
    "rows": [
        [
            1,
            "PENELOPE",
            "GUINESS",
            {
                "year": 2006,
                "month": 1,
                "day": 15,
                "hour": 4,
                "minute": 34,
                "second": 33.0
            }
        ],
        [
            2,
            "NICK",
            "WAHLBERG",
            {
                "year": 2006,
                "month": 1,
                "day": 15,
                "hour": 4,
                "minute": 34,
                "second": 33.0
            }
        ],
        [
            3,
            "ED",
            "CHASE",
            {
                "year": 2006,
                "month": 1,
                "day": 15,
                "hour": 4,
                "minute": 34,
                "second": 33.0
            }
        ]
    ],
    "warning_count": 0
}
mysql-sql>
```

3.3.4 Result Metadata

When an operation is executed, in addition to any results returned, some additional information is available. This includes information such as the number of affected rows, warnings, duration, and so on, when any of these conditions is true:

- JSON format is being used for the output
- MySQL Shell is running in interactive mode.

3.4 Active Language

MySQL Shell can execute SQL, JavaScript or Python code, but only one language can be active at a time. The active mode determines how the executed statements are processed:

- If using SQL mode, statements are processed as SQL which means they are sent to the MySQL server for execution.
- If using JavaScript mode, statements are processed as JavaScript code.
- If using Python mode, statements are processed as Python code.

When running MySQL Shell in interactive mode, activate a specific language by entering the commands: `\sql`, `\js`, `\py`.

When running MySQL Shell in batch mode, activate a specific language by passing any of these command-line options: `--js`, `--py` or `--sql`. The default mode if none is specified is JavaScript.

Use MySQL Shell to execute the content of the file `code.sql` as SQL.

```
shell> mysqlsh --sql < code.sql
```

Use MySQL Shell to execute the content of the file `code.js` as JavaScript code.

```
shell> mysqlsh < code.js
```

Use MySQL Shell to execute the content of the file `code.py` as Python code.

```
shell> mysqlsh --py < code.py
```

3.5 Batch Mode Made Interactive

This section describes code execution in batch mode.

- In batch mode, all the command logic described above is not available, only valid code for the active language can be executed.
- When processing SQL code, it is executed statement by statement using the following logic: read/process/print result.
- When processing non-SQL code, it is loaded entirely from the input source and executed as a unit.

Use the `--interactive` (or `-i`) command-line option to configure MySQL Shell to process the input source as if it were being issued in interactive mode; this enables all the features provided by the Interactive mode to be used in batch processing.

> **Note**
>
> In this case, whatever the source is, it is read line by line and processed using the interactive pipeline.

3.5.1 Multiple-line Support

It is possible to specify statements over multiple lines. When in Python or JavaScript mode, multiple-line mode is automatically enabled when a block of statements starts like in function definitions, if/then statements, for loops, and so on. In SQL mode multiple line mode starts when the command `\` is issued.

Once multiple-line mode is started, the subsequently entered statements are cached.

For example:

```
mysql-sql> \
... create procedure get_actors()
```

```
... begin
...   select first_name from sakila.actor;
... end
...
mysql-sql>
```

Chapter 4 MySQL Shell Commands

MySQL Shell provides commands which enable you to modify the execution environment of the code editor, for example to configure the active programming language or a MySQL Server connection. The following table lists the commands that are available regardless of the currently selected language. As commands need to be available independent of the *execution mode*, they start with an escape sequence, the `\` character.

Command	Alias/Shortcut	Description
`\help`	`\h` or `\?`	Prints help about MySQL Shell commands.
`\quit`	`\q` or `\exit`	Exit MySQL Shell.
`\`		In SQL mode, begin multiple-line mode. Code is cached and executed when an empty line is entered.
`\status`		Show the current MySQL Shell status.
`\js`		Switch execution mode to JavaScript.
`\py`		Switch execution mode to Python.
`\sql`		Switch execution mode to SQL.
`\connect`	`\c`	Connect to a MySQL Server with a URI using an XSession (X Protocol).
`\connect_node`	`\cn`	(Removed in version 1.0.4, use `\connect -n`) Connect to a MySQL Server with a URI using a Node session.
`\connect_classic`	`\cc`	(Removed in version 1.0.4, use `\connect -c`) Connect to a MySQL Server with a URI using a Classic session (MySQL Protocol).
`\use`		Specify the schema to use.
`\source`	`\.`	Execute a script file using the active language.
`\warnings`	`\W`	Show any warnings generated by a statement.
`\nowarnings`	`\w`	Do not show any warnings generated by a statement.
`\lsconn`	`\lsc`	Print the connection data for the stored sessions.
`\saveconn`	`\savec`	Save connection data of a session, optionaly use `-f` to force overwriting an existing connection.
`\addconn`	`\addc`	(Removed in version 1.0.4, see `\saveconn`) Store the connection data of a session.
`\rmconn`		Removes a stored session.
`\chconn`		(Removed in version 1.0.4, see `\saveconn`) Updates a stored session.

Help Command

The `\help` command can be used with or without parameters. When used without parameters a general help is printed including information about:

- Available commands.
- Available commands for the active mode.

When used with a parameter, the parameter must be a valid command. If that is the case, help for that specific command is printed including:

- Description

- Supported aliases if any
- Additional help if any

For example:

```
\help connect
```

If the parameter is not a valid command, the general help is printed.

Connect Command

The `\connect` command is used to connect to a MySQL Server using an URI. This command creates an X Protocol connection by default.

For example:

```
\connect root@localhost:3306
```

If a password is required you are prompted for it.

Use the `-n` option to create a Node session, using the X Protocol to connect to a single server. For example:

```
\connect -n root@localhost:3306
```

Use the `-c` option to create a Classic session, enabling you to use the MySQL Protocol to issue SQL commands directly on a server. For example:

```
\connect -c root@localhost:3306
```

Source Command

The `\source` command is used to execute code from a script at a given path. For example:

```
\source /tmp/mydata.sql
```

It can be used to execute either SQL, JavaScript or Python code. The code in the file is executed using the active language, so to process SQL code the MySQL Shell must be in SQL mode.

> **Warning**
>
> As the code is executed using the active language, executing a script in a different language than the currently selected execution mode language could lead to unexpected results.

Use Command

The `\use` command enables you to choose which schema is active, for example:

```
\use schema_name
```

The `\use` command requires a global development session to be active. If the global development session is an XSession then the `\use` command only sets `db` to the object representing the *`schema_name`* but does not set a current schema on the database. If the global development session is one of NodeSession or ClassicSession the `\use` command sets the current schema to the specified *`schema_name`* and updates the `db` variable to the object that represents the selected schema.

Chapter 5 MySQL Shell Application Log

This section explains the logging provided by MySQL Shell, where to find logs and how to configure the level of logging.

MySQL Shell can be configured to generate an application log file with information about issues of varying severity. You can use this information to verify the state of MySQL Shell while it is running. The log format is plain text and entries contain a timestamp and description of the problem. For example:

```
2016-04-05 22:23:01: Error: Default Domain: (shell):1:8: MySQLError: You have an error
in your SQL syntax; check the manual that corresponds to your MySQL server version for
the right syntax to use near '' at line 1 (1064) in session.sql("select * from t
limit").execute().all();
```

The amount of information to add to the log can be configured using `--log-level`. See Configuring Logging.

MySQL Shell Log File Location

The location of the log file is the user configuration path and the file is named `mysqlsh.log`.

Log File on Windows

On Windows, the default path to the log file is `%APPDATA%\MySQL\mysqlsh\mysqlsh.log`

To find the location of `%APPDATA%` on your system, echo it from the comand-line. For example:

```
C:>echo %APPDATA%
C:\Users\exampleuser\AppData\Roaming
```

On Windows, the path is determined by the result of gathering the `%APPDATA%` folder specific to that user, and then appending `MySQL\mysqlsh`. Using the above example results in `C:\Users\exampleuser\AppData\Roaming\MySQL\mysqlsh\mysqlsh.log`.

Log File on Unix-based Systems

For a machine running Unix, the default path is `~/.mysqlsh/mysqlsh.log` where "~" represents your home directory. The environment variable `HOME` also represents the home directory. Appending `.mysqlsh` to the this home directory determines the default path to the logs. For example:

```
shell>echo $HOME
/home/exampleuser
```

Therefore the location of the MySQL Shell file on this system is `/home/exampleuser/.mysqlsh/mysqlsh.log`.

These paths can be overridden on all platforms by defining the environment variable `MYSQL_USER_CONFIG_PATH`. The value of this variable replaces `%APPDATA%` in Windows or `$HOME` in Unix.

Configuring Logging

By default, logging is disabled in MySQL Shell. To enable logging use the `--log-level` command option when starting MySQL Shell. The value assigned to `--log-level` controls the level of detail in the log. The level of logging can be defined using either numeric levels from 1 to 8, or equivalent named levels as shown in the following table.

Log Level Number	Log Level Name	Meaning
1	none	No logging, the default
2	internal	Internal Error
3	error	Errors are logged
4	warning	Warnings are logged
5	info	Information is logged
6	debug	Debug information is logged
7	debug2	Debug with more information is logged
8	debug3	Debug with full information is logged

The numeric and named levels are equivalent. For example there is no difference in logging when starting MySQL Shell in either of these ways:

```
shell>mysqlsh --log-level=4
shell>mysqlsh --log-level=warning
```

Chapter 6 Customizing MySQL Shell

Table of Contents

MySQL Shell offers the ability to customize the behavior and code execution environment through startup scripts, which are executed when the application is first run. Using such scripts enables you to:

- Add additional search paths for Python or JavaScript modules.
- Override the default prompt used by the Python and JavaScript modes.
- Define global functions or variables.
- Any other possible initialization through JavaScript or Python.

6.1 Working With Start-Up Scripts

When MySQL Shell enters either into JavaScript or Python mode, it searches for startup scripts to be executed. The startup scripts are JavaScript or Python specific scripts containing the instructions to be executed when the corresponding mode is initialized.

Startup scripts must be named as follows:

- For JavaScript mode: `mysqlshrc.js`
- For Python mode: `mysqlshrc.py`

MySQL Shell searches the following paths for these files (in order of execution).

On Windows:

1. `%PROGRAMDATA%MySQLmysqlshmysqlshrc.[js|py]`
2. `%MYSQLSH_HOME%sharedmysqlshmysqlshrc.[js|py]`
3. `<mysqlsh binary path>mysqlshrc.[js|py]`
4. `%APPDATA%MySQLmysqlshmysqlshrc.[js|py]`

On Linux and OSX:

1. `/etc/mysql/mysqlsh/mysqlshrc.[js|py]`
2. `$MYSQLSH_HOME/shared/mysqlsh/mysqlshrc.[js|py]`
3. `<mysqlsh binary path>/mysqlshrc.[js|py]`
4. `$HOME/.mysqlsh/mysqlshrc.[js|py]`

The environment variable `MYSQLSH_HOME` defines the root folder of a standard setup of MySQL Shell. If `MYSQLSH_HOME` is not defined it is automatically calculated based on the location of the MySQL Shell binary, therefore on many standard setups it is not required to define `MYSQLSH_HOME`.

If `MYSQLSH_HOME` is not defined and the MySQL Shell binary is not in a standard install folder structure, then the path defined in option 3 in the above lists is used. If using a standard install or if `MYSQLSH_HOME` points to a standard install folder structure, then the path defined in option 3 is not used.

> **Warning**
>
> The lists above also define the order of searching the paths, so if something is defined in two different scripts, the script executed later takes precedence.

6.2 Adding Module Search Paths

There are two ways to add additional module search paths:

- Through environment variables
- Through startup scripts

6.2.1 Environment Variables

Python uses the `PYTHONPATH` environment variable to allow extending the search paths for python modules. The value of this variable is a list of paths separated by:

- A colon character in Linux and OSX
- A semicolon character in Windows

To achieve this in JavaScript, MySQL Shell supports defining additional JavaScript module paths using the `MYSQLSH_JS_MODULE_PATH` environment variable. The value of this variable is a list of semicolon separated paths.

6.2.2 Startup Scripts

The addition of module search paths can be achieved for both languages through the corresponding startup script.

For Python modify the `mysqlshrc.py` file and append the required paths into the `sys.path` array.

```
# Import the sys module
import sys
# Append the additional module paths
sys.path.append('~/custom/python')
sys.path.append('~/other/custom/modules')
```

For JavaScript the same task is achieved by adding code into the `mysqlshrc.js` file to append the required paths into the predefined `shell.js_module_paths` array.

```
// Append the additional module paths
shell.js.module_paths[shell.js.module_paths.length] = '~/custom/js';
shell.js.module_paths[shell.js.module_paths.length] = '~/other/custom/modules';
```

6.3 Overriding the Default Prompt

MySQL Shell uses a default prompt for both Python (`mysql-py>`) and JavaScript (`mysql-js>`).

You can customize the language specific prompt using the `shell.custom_prompt()` function. This function must return a string that is used as the prompt. To have a custom prompt when MySQL Shell starts, define this function in a startup script. The following example shows how this functionality can be used.

In Python `shell.custom_prompt()` could be defined as:

```
# Import the sys module
from time import gmtime, strftime
def my_prompt():
  ret_val = strftime("%H:%M:%S", gmtime())
  if session and session.isOpen():
    data = shell.parseUri(session.getUri())
    ret_val = "%s-%s-%s-py> " % (ret_val, data.dbUser, data.host)
  else:
    ret_val = "%s-disconnected-py> " % ret_val
  return ret_val
shell.custom_prompt = my_prompt
```

In JavaScript `shell.custom_prompt()` could be defined as:

```
shell.custom_prompt = function(){
  var now = new Date();
  var ret_val = now.getHours().toString()+ ":" + now.getMinutes().toString() + ":" + now.getSeconds().t
  if (session && session.isOpen()){
    var data = shell.parseUri(session.getUri());
    ret_val += "-" + data.dbUser + "-" + data.host + "-js> ";
  }
  else
    ret_val += "-disconnected-js> ";
  return ret_val;
}
```

The following example demonstrates using the custom prompt functions defined above in startup script. The prompts show the current system time, and if a session is open the current user and host:

```
Welcome to MySQL Shell 1.0.4 Development Preview
Copyright (c) 2016, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type '\help', '\h' or '\?' for help.
Currently in JavaScript mode. Use \sql to switch to SQL mode and execute queries.
14:34:32-disconnected-js> \py
Switching to Python mode...
19:34:39-disconnected-py> \connect root:@localhost
Creating an X Session to root@localhost:33060
No default schema selected.
19:34:50-root-localhost-py> \js
Switching to JavaScript mode...
14:34:57-root-localhost-js>
```