MySQL variables and System data

mysql.data and system.data

These two databases have a completely identical structure:

```
CREATE TABLE IF NOT EXISTS [metric_master] (
   [metric_id] INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
   [metric_desc] TEXT ASC UNIQUE )
```

```
CREATE TABLE IF NOT EXISTS [snapshot_master] (
   [timestamp_id] INTEGER NOT NULL,
   [metric_id] INTEGER NOT NULL,
   [metric_now] TEXT,
   [metric_diff] TEXT,
   PRIMARY KEY (metric_id, timestamp_id))
```

```
CREATE INDEX IF NOT EXISTS [timestamp_id_index] ON [snapshot_master] ([timestamp_id])
```

```
CREATE TABLE IF NOT EXISTS [timestamp_master] (
   [timestamp_id] INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
   [server_timestamp] INTEGER,
   [server_start_time] INTEGER,
   [server_uptime] INTEGER,
   [server_uptime_diff] INTEGER,
   [server_is_connected] INTEGER)
```

```
  CREATE INDEX IF NOT EXISTS [server_timestamp_idx] ON [timestamp_master] ([server_timestamp])
```

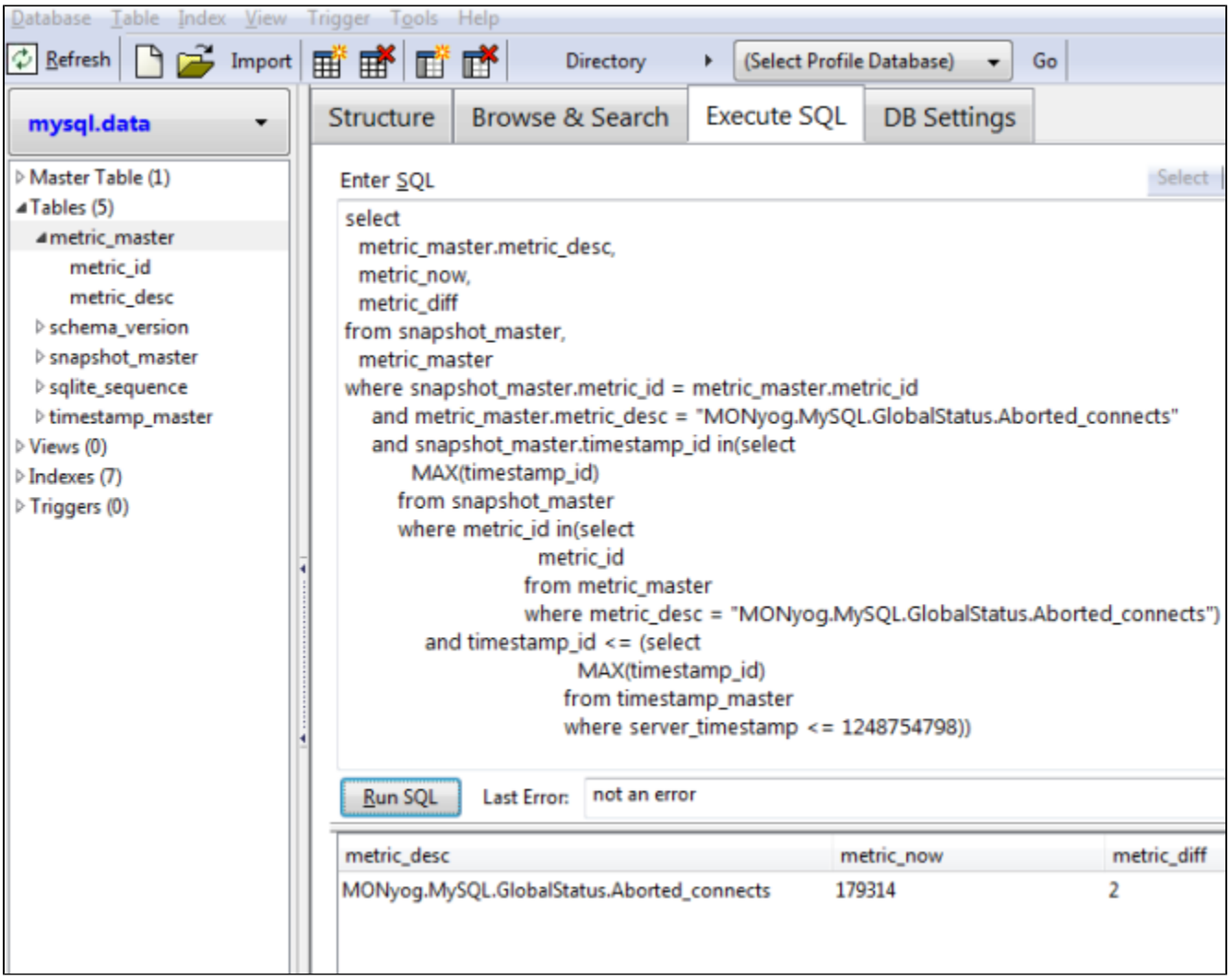
What is most important to understand here is the [timestamp_id] column occurring in both [snapshot_master] and [timestamp_master] tables. Actually, with MySQL and InnoDB you would probably create a Foreign Key from [snapshot_master] to [timestamp_master] for constraining and clarity. To get meaningful results you need to `JOIN` the two in the query or use a `SUBQUERY`.

This is basically how they work:

- Everytime SQL DM for MySQL sends a statement like `SHOW VARIABLES/STATUS` of some kind (or fetching a OS metric from Linux/proc folder) one row is INSERTED into [timestamp_master] table with information about current time and the metrics retrieved for each such statement is INSERTED into [snapshot_master]. The [snapshot_master] table contains the metrics details. The [timestamp_id] column identifies when metric details were like that. Also, that timestamps in SQL DM for MySQL databases are unix_timestamps.
- And actually, we do not always `INSERT` into [snapshot_master]. Only if the particular metric was changed since last time something was INSERTED for that metric we will INSERT again. So if you want to find the value of a metric at some particular time you need to find the most recent value stored before that particular time for that particular metric.
- Finally, the [snapshot_master] does not have the names of the metrics. It is not possible to know in advance what metrics the server returns as it depends on server details (version and configuration). And actually a server may be upgraded.
  Saving each textual description only once which would save disk space. So [snapshot_master] only contains a number in the [metric_id] column referring to the textual description the [metric_master] table. So, if the query returns the name of the metric or the metric name should be used in a WHERE-clause also [metric_master] table must be referenced in the query. If you are familiar with the SHOW statements and what information they return you identify the meaning of each row in [metric_master] table.

The term Metric here refers to the discrete values returned for `SHOW` statements themselves (SHOW GLOBAL VARIABLES; SHOW GLOBAL STATUS; SHOW SLAVE STATUS etc.). Whatever calculations SQL DM for MySQL does in its web interface are done after and not before storage. But we do one calculation before storing however: whenever a metric is INSERTED we also retrieve that latest stored value for the same metric and calculate the difference. Both the current value and this difference is stored (in [metric_now] and [metric_diff] columns respectively).

An example of an easily understandable query doing all this could look like:



An example of a query that we actually execute (optimized for large SQLite databases) to populate a graph is as follows:

```
SELECT metric_now
FROM snapshot_master
WHERE snapshot_master.metric_id = my_metric_id
   AND snapshot_master.timestamp_id IN(
      SELECT MAX(timestamp_id)
      FROM snapshot_master
      WHERE metric_id = my_metric_id
      AND timestamp_id <= (
         SELECT MAX(timestamp_id)
         FROM timestamp_master
         WHERE server_timestamp <= my_metric_timestamp)
   )
```

Actually SQLite support has recommended using SUBQUERIES and not JOINS in most cases with SQLite for best performance with big databases. That is also the experience we have ourselves when profiling different queries returning same results.

udo.data

This is the database where we store data from the Custom SQL Objects(CSOs). This database has 3 different tables- Snapshot_master, Column_master and timestamp_master

```
CREATE TABLE [column_master]
   (id INTEGER NOT NULL PRIMARY KEY,
    timestamp_id INTEGER NOT NULL,
    udo_id INTEGER NOT NULL,
    key_column_value VARCHAR(255),
    column VARCHAR(255),
    UNIQUE([udo_id], [key_column_value], [column], [timestamp_id]));
```

```
CREATE TABLE [snapshot_master] ([timestamp_id] INTEGER NOT NULL,
   [metric_id] INTEGER NOT NULL,
   [metric_now] TEXT,
   [metric_diff] TEXT,
   PRIMARY KEY (metric_id, timestamp_id));
```

```
CREATE TABLE [timestamp_master] (
   [timestamp_id] INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
   [server_timestamp] INTEGER,
   [server_start_time] INTEGER,
   [server_uptime] INTEGER,
   [server_uptime_diff] INTEGER,
   [server_is_connected] INTEGER, [udo_id] INTEGER);
```

events.data

This database holds information related to events in SQL DM for MySQL.

```
CREATE TABLE [events]  (id INTEGER PRIMARY KEY AUTOINCREMENT,
   first_seen INTEGER(5) NOT NULL,
   last_seen INTEGER(5) NOT NULL,
   down_count INTEGER(5) NOT NULL DEFAULT 0,
   server_id VARCHAR(255) NOT NULL,
   server_name VARCHAR(255) NOT NULL,
   group_id INTEGER(5) NOT NULL DEFAULT 0,
   counter_id INTEGER(5) NOT NULL DEFAULT 0,
   grp VARCHAR(255),
   name VARCHAR(255),
   sampling_time_frame VARCHAR(255),
   type INTEGER NOT NULL DEFAULT 0,
   threshold VARCHAR(255),
   value VARCHAR(255),
   advice TEXT,
   mail_alert VARCHAR(10) NOT NULL DEFAULT '',
   down_count_override INTEGER(5) NOT NULL DEFAULT 0,
   notify_stable_override VARCHAR(10) NOT NULL DEFAULT '',
   smtp_alert_cnt INTEGER(5) NOT NULL DEFAULT 0,
   snmp_alert_cnt INTEGER(5) NOT NULL DEFAULT 0,
   ignored_timestamp INTEGER NOT NULL DEFAULT 0);
```

```
CREATE TABLE [timestamp_master](
   [timestamp_id] INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
   [server_timestamp] INTEGER UNIQUE);
```