Events (MySQL Event Scheduler)

In Unix and Linux systems there is a program called cron, which allows users to execute scripts or commands at a specified time. For example, jobs, such as removing system cache, running a backup script, or connecting to the Internet to retrieve email at a specific time.

Similarly, event support was added in **MySQL 5.1.6**. MySQL Events are tasks that run according to a schedule (i.e., scheduled events). When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time. Conceptually, this is similar to the idea of the Unix crontab (also known as a "cron job") or the Windows Task Scheduler.

As mentioned, the MySQL Event Scheduler is similar to the Linux cron jobs and Windows Task Scheduler. It allows tasks to be executed at a predefined scheduled time. These scheduled tasks called "Events" can be set to run at certain intervals of time, rather than just as a onetime task, as well. For example, a task can be created to run every day, or on a monthly-basis, at a specific time—freeing the DBA (or programmer) from the repetitive task(s).

Creating an Event is similar to creating a named database object containing SQL statements to be executed at one or more intervals, each beginning and ending at a specific date and time. This feature of events in MySQL is mainly intended at database administrators, who can use it for scheduling jobs that would execute in the background as a one-time or a recurrent process at certain intervals.

MySQL events are executed by a special event-scheduler thread. The event-scheduler thread and its current state can be visualized in the output of the `SHOW PROCESSLIST;` command by users who have their privilege set to 'super' or by the database administrators. The default state of the event-scheduler thread is OFF. To switch off the thread we can pass the value OFF for event_scheduler for the global variable. **Remember:** before creating an event, the event- scheduler thread must be enabled (i.e., turned on). The thread can be enabled by the following command.

```
SET GLOBAL event_scheduler = ON;
```

Creating Events:

The SQL standard does not provide any set of official standard for events. The MySQL concept for creating events is based on the syntax of Sybase SQL Anywhere's CREATE EVENT syntax. MySQL uses the following syntax for creating events:

```
CREATE EVENT [IF NOT EXISTS ] event_name
 ON SCHEDULE schedule
 [ON COMPLETION [ NOT ] PRESERVE ]
 [ENABLED | DISABLED ]
 [COMMENT 'comment' ]
 DO sql_statement;
```

The 'event_name' must be a valid identifier of up to 64 characters. And since events are database objects which are stored within a database; the event names must be unique. The IF NOT EXISTS clause works the same as it does in the CREATE TABLE statement, if an 'event_name' exists, no action will be taken. The ON SCHEDULE clause determines at what time and how often the 'sql _statement' defined for the event has to be executed. The 'schedule' can be a timestamp in the future, recurring interval or a combination of both timestamp and the interval. The two clauses it can accept are **AT** and **EVERY**.

One-time event: AT timestamp is used for one-time events, which are executed once on a given date and time as specified. Suppose we want to drop a table named 'test' from the database 'mydb', after two hours from now, we can create an event as follows:

```
CREATE EVENT deleteTable
 ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 2 HOUR
 DO DROP TABLE mydb.test;
```

The event will be created and stored in the database 'mysql' under the table 'event'. The CURRENT_TIMESTAP will take the current time from the system and INTERVAL adds two hours of delay to that time. Similarly, a specific time can also be set for the event to be executed by the use of **AT** TIMESTAMP 'time' with the ON SCHEDULE clause. And as the event-scheduler thread is enabled, it will trigger up the event after two hours from the current time. The execution of the event would result in dropping of the table from the database. **NOTE:** it will ***NOT*** execute more than once. When testing, change name.

Recurring events: On the other hand, **EVERY** clause is used for tasks that are to be repeated at scheduled intervals. For instance if a job has to be executed after every 2 hours, the ON SCHEDULE clause for the event will be like: ON SCHEDULE EVERY 2 HOUR The interval that the AT and EVERY clause can accept is: YEAR, MONTH, WEEK, DAY, HOUR, MINUTE, or SECOND. The EVERY clause also accepts two optional clauses: STARTS and ENDS. The STARTS clause defines at what date and time the repeating of the event process should begin. Similarly the ENDS clause defines at what date and time the event repetition should cease. For example, if you want to insert the current time into a table at an interval of 20 seconds for the next five hours, then you can do that using the following event.

```
CREATE EVENT addTimer
 ON SCHEDULE EVERY 20 SECOND
 STARTS CURRENT_TIMESTAMP
 ENDS CURRENT_TIMESTAMP + INTERVAL 5 HOUR
 DO INSERT INTO mydb.timer (timer_id) VALUES (NOW());
```

The addTimer event, once created will start inserting current time into the timer_id column of the table at an interval of every 20 seconds and this repetition will go on until the next 5 hours, and then it will stop.

To alter a predefined event, the syntax for altering the event is as follows:

```
ALTER EVENT event_name
 [ ON SCHEDULE schedule ]
 [ RENAME TO event name2 ]
 [ ON COMPLETION [ NOT ] PRESERVE ]
 [ COMMENT 'comment' ]
 [ ENABLED | DISABLED ]
 [ DO sql_statement ]
```

The ALTER EVENT statement clauses are same as that for the CREATE EVENT clauses. With RENAME TO clause, an existing event can be renamed. The clauses that you mention for altering will be the only ones that will be changed and the rest will remain intact as in the originally created event.

If you want to see all the events that are in the database server, you can run a select query on the event table of the mysql database of MySQL server. Moreover, the events that have occurred and are now not required can be dropped from the database by using the command mentioned below:

DROP EVENT 'event_name'

Event Scheduling is the key for tasks that includes removing sessions, cache etc. from database internal tables, or generating monthly database reports during the night when the data transaction stress on the database server is less.

Events should not be confused with triggers. Whereas a trigger is a database object whose statements are executed in response to a specific type of event that occurs on a given table, a (scheduled) event is an object whose statements are executed in response to the passage of a specified time interval.

What is the event scheduler?

- Temporal triggers
- NOT related to a specific table
- Execute SQL code: at a given time, or at given intervals

Why use the event scheduler?

- Cross platform scheduler
- No external applications needed
- Very little overhead

When is the event scheduler useful?

- Data cleanup
- Consistency checks
- Refreshing stale data:
    - dropping old partitions
    - creating new ones
- Extract-Transform-Load (ETL) operations (i.e., data warehousing):
    - Creating summary tables
    - Prepare detail data for warehousing

MySQL Events have the following major features and properties:

•In MySQL 5.1.12 and later, an event is uniquely identified by its name and the schema to which it is assigned.

•An event performs a specific action according to a schedule. This action consists of an SQL statement, which can be a compound statement in a BEGIN ... END block if desired.

•An event's timing can be either one-time or recurrent. A one-time event executes one time only. A recurrent event repeats its action at a regular interval, and the schedule for a recurring event can be assigned a specific start day and time, end day and time, both, or neither. (By default, a recurring event's schedule begins as soon as it is created, and continues indefinitely, until it is disabled or dropped.)

•If a repeating event does not terminate within its scheduling interval, the result may be multiple instances of the event executing simultaneously. If this is undesirable, you should institute a mechanism to prevent simultaneous instances. For example, you could use the GET_LOCK() function, or row or table locking.

•Users can create, modify, and drop scheduled events using SQL statements intended for these purposes. Syntactically invalid event creation and modification statements fail with an appropriate error message. A user may include statements in an event's action which require privileges that the user does not actually have. The event creation or modification statement succeeds but the event's action fails. See Section 19.4.6, “The Event Scheduler and MySQL Privileges” for details.

•Many of the properties of an event can be set or modified using SQL statements. These properties include the event's name, timing, persistence (that is, whether it is preserved following the expiration of its schedule), status (enabled or disabled), action to be performed, and the schema to which it is assigned.

The default definer of an event is the user who created the event, unless the event has been altered, in which case the definer is the user who issued the last ALTER EVENT statement affecting that event. An event can be modified by any user having the EVENT

privilege on the database for which the event is defined. (Prior to MySQL 5.1.12, only an event's definer, or a user having privileges on the mysql.event table, could modify a given event.) See Section 19.4.6, "The Event Scheduler and MySQL Privileges".

•An event's action statement may include most SQL statements permitted within stored routines.

Creating Events:

1. Enable event scheduler:
    a. option file: event_scheduler=ON
    b. online: SET GLOBAL event_scheduler=ON;
2. Create an event
3. Check effects

NOTE:
You can ensure the scheduler starts when MySQL is launched by setting event_scheduler=ON in your MySQL configuration file (my.cnf or my.ini on Windows).

If the Event Scheduler status has **not** been set to DISABLED in the option file, event_scheduler can be toggled between ON and OFF (using SET). It is also possible to use 0 for OFF, and 1 for ON when setting this variable. Thus, any of the following 4 statements can be used in the mysql client to turn on the Event Scheduler:

```
SET GLOBAL event_scheduler = ON;
SET @@global.event_scheduler = ON;
SET GLOBAL event_scheduler = 1;
SET @@global.event_scheduler = 1;
```

Similarly, any of these 4 statements can be used to turn off the Event Scheduler:

```
SET GLOBAL event_scheduler = OFF;
SET @@global.event_scheduler = OFF;
SET GLOBAL event_scheduler = 0;
SET @@global.event_scheduler = 0;
```

Although ON and OFF have numeric equivalents, the value displayed for event_scheduler by SELECT or SHOW VARIABLES is always one of OFF, ON, or DISABLED. DISABLED has no numeric equivalent. For this reason, ON and OFF are usually preferred over 1 and 0 when setting this variable.

Event caveats:

- Not preserved: by default, events are removed after their last execution.
- Errors:
    - Errors go to error log only (not visible to client that created events)
    - Factual errors detected at run time, not at creation time (e.g. table not found, procedure called with wrong parameters)
- WHAT TO DO: create procedure, test it, then attach it to an event
- The events scheduler cannot access the operating system.
- Thus, the event scheduler **CAN NOT**
    - send email
    - list directories
    - write to arbitrary files
    - run applications

Examples:

-- check if event scheduler on

```
SHOW VARIABLES LIKE 'event_scheduler';
```

-- turn on event sceduler (using "set" applies only until server restarted)
-- must change configuration file to make change permanent

```
SET GLOBAL event_scheduler = ON;
```

-- drop events

```
DROP EVENT IF EXISTS one_time_delete_audit_rows;
DROP EVENT IF EXISTS monthly_delete_audit_rows;
```

--create event

-- temporarily redefine delimiter

```
DELIMITER //
```

-- create event statement that executes only once
-- "AT" here indicates execute one month from current date/time (from creation date)
-- delete statement deletes all records from invoices_audit table more than one month old (limit 100 records)
-- "INTERVAL" keyword also works with SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, and YEAR

```
CREATE EVENT one_time_delete_audit_rows
ON SCHEDULE AT NOW() + INTERVAL 1 MONTH
DO
BEGIN
  DELETE FROM invoices_audit WHERE action_date < (NOW() - INTERVAL 1 MONTH)
LIMIT 100;
END//
```

-- change delimiter back

```
DELIMITER ;
```

-- same as above, but begins at Midnight on 6/30/12, and executes every month

```
CREATE EVENT monthly_delete_audit_rows
ON SCHEDULE EVERY 1 MONTH
STARTS '2012-06-30 00:00:00'
DO BEGIN
  DELETE FROM invoices_audit WHERE action_date < (NOW() - INTERVAL 1 MONTH)
LIMIT 100;
```

More Examples:

-- creating an event at a given time

```
CREATE EVENT event_name
ON SCHEDULE AT '2012-04-20 15:55:00'
DO
INSERT INTO some_table
VALUES
  ('testing', now());
```

Or…by calling a stored procedure

```
CREATE EVENT event_name
ON SCHEDULE AT now() + interval 20 minute
DO
CALL myProcedure();
```

-- Creating a recurring event

```
CREATE EVENT event_name
ON SCHEDULE EVERY 20 MINUTE
DO
INSERT INTO some_table
VALUES
  (' testing ', now());
```

Or…by calling a stored procedure

```
CREATE EVENT event_name
ON SCHEDULE every 7 DAY
DO
CALL myProcedure();
```

-- Creating a recurring event

-- creates an event that runs every 10 minutes, starts in 2 hours, and ends two hours later

```
CREATE EVENT event_name
ON SCHEDULE EVERY 10 MINUTE
STARTS NOW() + INTERVAL 2 HOUR
ENDS NOW() + INTERVAL 4 HOUR
DO
CALL myProcedure();
```

Additional Event Commands

-- list all events on server (list options vertically \G)

```
SELECT * FROM INFORMATION_SCHEMA.EVENTS\G
```

-- list all events in specific database

```
SHOW EVENTS IN test;
```

-- list all events in mydb beginning with "test"

```
SHOW EVENTS IN mydb LIKE 'test%';
```

-- disable event

```
ALTER EVENT monthly_delete_audit_rows DISABLE;
```

-- enable event

```
ALTER EVENT monthly_delete_audit_rows ENABLE;
```

-- rename event

```
ALTER EVENT one_time_delete_audit_rows RENAME TO one_time_delete_audits;
```

-- drop event

```
DROP EVENT monthly_delete_audit_rows;
```

-- drop event only if exists

```
DROP EVENT IF EXISTS monthly_delete_audit_rows;
```

-- show create event statement

```
SHOW CREATE EVENT one_time_delete_log_rows;
```

Notes: testing events

1. Move all code within event into stored procedure
2. Make event only call stored procedure
3. Test stored procedure with CALL

References

http://dev.mysql.com/doc/refman/5.1/en/events.html

http://dev.mysql.com/doc/refman/5.1/en/events-configuration.html

http://www.ciol.com/Databases/Tutorial/Schedule-Tasks-using-Events-on-MySQL/4808108623/0/