Lab Manual on Scheduled Events (Temporal Triggers) (June 2020)

Allan O. Omondi,
Doctoral Fellow, Faculty of Information Technology,
Strathmore University, Nairobi, Kenya
aomondi [at] strathmore.edu

Objectives

By the end of this lab, you will be able to:

1. **List** the background processes that are running on a database
2. **Turn ON/OFF** the event scheduler
3. **Create** a scheduled event that is executed every minute for 1 hour
4. **List** the scheduled events in a database
5. **Disable/Enable** a scheduled event
6. **Alter** the details of a scheduled event
7. **Delete** a scheduled event

Tools

1. MySQL DBMS (>=MySQL DBMS 8.0)
2. MySQL Workbench (>=MySQL Workbench 8.0)

Source: https://mauvesea.wordpress.com/2011/06/16/the-calendar/

Approximate Time Required

1 Hour

Prerequisites

1. You should have installed MySQL DBMS (>= MySQL 8.0) and MySQL Workbench (>=MySQL Workbench 8.0). They are available via this link.
2. You should have imported the sample database called "classicmodels" into your database. It is available via this link.

Narrative:

The "classicmodels" sample database contains data of a retail business. The retail business, in this case, is engaged in the sale of models of classic cars. It contains typical business data such as customers, products, product lines (categories of products), orders, order line items (details of an order), payments received, employees, and branch details.

Source: https://www.shannons.com.au/auctions/2013-shannons-melbourne-autumn-classic-auction/cd6dd3cdt3ap0g4k/

Recommended citation:
Omondi, A. O. (2020). *BBT 3104 – Advanced Database Systems – Lab Manual on Scheduled Events (Temporal Triggers) [PDF Lab Manual]*. Retrieved from Strathmore University eLearning website https://elearning.strathmore.edu/pluginfile.php/225801/mod_resource/content/1/4.a.Concept4of6-LabManual-ScheduledEvents.pdf

A temporal trigger is a named object which contains one or more SQL statements. However, unlike normal triggers which are triggered by DML actions (i.e. actions that manipulate data using the Data Manipulation Language section of SQL – **C**reate, **U**pdate, **D**elete), **temporal triggers are triggered by time**. The word "temporal" is an adjective that means "related to time".

Source: https://pixabay.com/photos/stopwatch-timer-clock-symbol-icon-2624277/

They are, therefore, stored in the database and executed based on time at one or more intervals. This execution can be done indefinitely or within a pre-specified period (interval).

Possible intervals include:

(i) YEAR (e.g. `INTERVAL 1 YEAR` means it is executed after every 1 year)
(ii) QUARTER
(iii) MONTH
(iv) DAY
(v) HOUR (e.g. `INTERVAL 1 HOUR` means it is executed after every 1 hour)
(vi) MINUTE
(vii) WEEK
(viii) SECOND
(ix) YEAR_MONTH (e.g. `INTERVAL '2:5' YEAR_MONTH` means it is executed after every 2 years and 5 months)
(x) DAY_HOUR
(xi) DAY_MINUTE
(xii) DAY_SECOND
(xiii) HOUR_MINUTE (e.g. `INTERVAL '1:25' YEAR_MINUTE` means it is executed after every 1 hour 25 minutes)
(xiv) HOUR_SECOND
(xv) MINUTE_SECOND

A temporal trigger is also known as a "**scheduled event**" or an "**event**" and it is similar to an automated task created using task scheduler (in Windows) or using cron (in Linux). The **event scheduler** module (in the MySQL DBMS) is responsible for scheduling and executing temporal triggers.

Temporal triggers can be used to periodically:

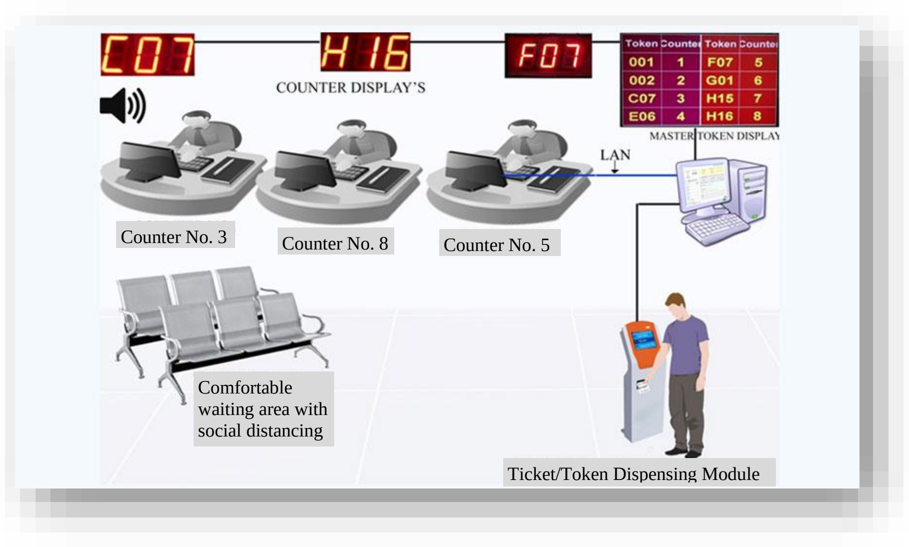
(i) Optimize relations to improve the performance of the database system
(ii) Archive data
(iii) Generate complex reports during off-peak hours
(iv) Clean up logs

amongst many other innovative tasks.

The following steps present a scenario where the manager of the customer service department has requested for a dashboard that displays how long a customer has been waiting on the queue to be served. It essentially avoids a situation where clients are waiting in an uncomfortable setting and can easily insight each other to be impatient.

Source: https://pngimage.net/queue-png-7/

The use of tickets or tokens can be implemented to manage the queue and keep track of how long a customer has been waiting.

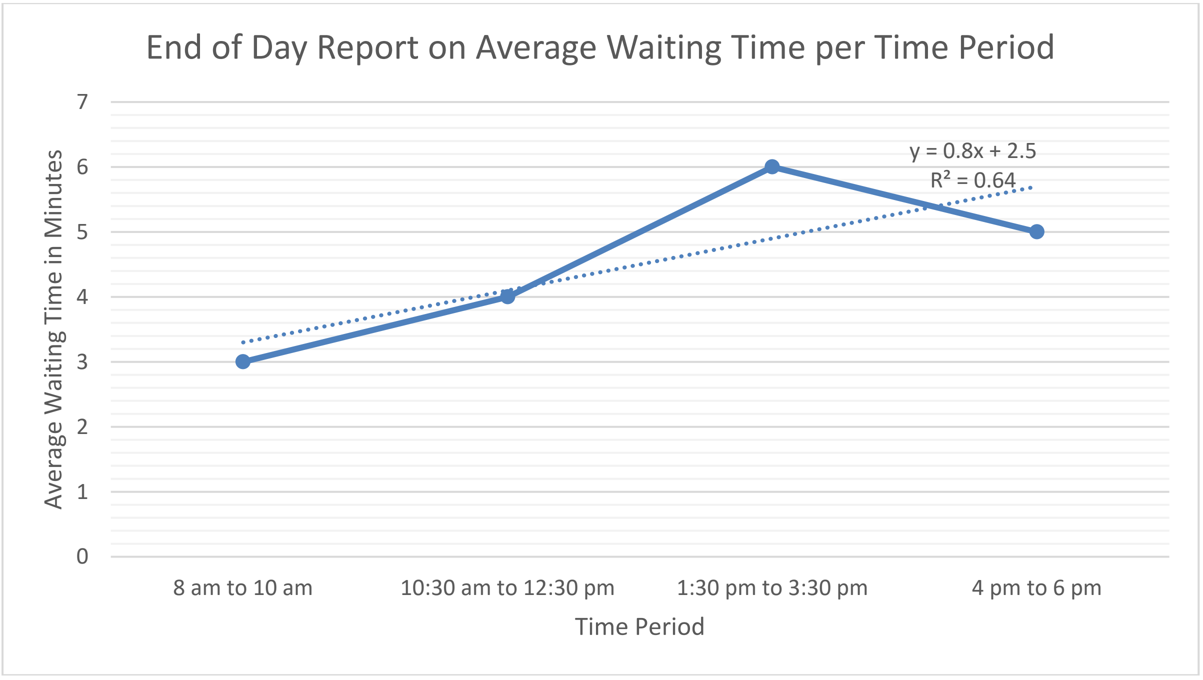


Source: http://www.breakts.com/queue-management-token-display-system/

Providing the right employees with the right information at the right time, enables them to make the right decisions. For example, if the database is able to store data and provide information that indicates that the average waiting time is too long, the customer service manager can decide to **redesign the customer service business process** and perhaps make the waiting area more comfortable to promote patience in customers.

Source: https://www.hotelcontractbeds.co.uk/blog/2015/06/hotel-lobby-design/

The scenario requires a customer to get a ticket number and wait for their turn to speak to an employee who can address their issues. The timer starts immediately a customer receives a ticket number from the ticketing module of the queue management system and stops immediately the employee (customer service officer (CSO)) has resolved the customer's issue. As a DBA, you can ensure the database records the total waiting time data so that all the programmers need to do is to provide a nice user interface to present the data read from the database. For example, a user interface that can display a graph like this on a dashboard:

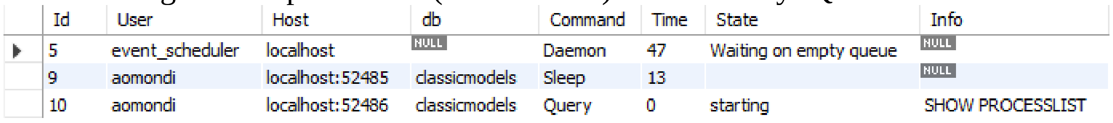


STEP 1. Confirm that the event scheduler is ON by default

Execute the following command to confirm that the event scheduler is on by default. Remember, it is the event scheduler module of the MySQL DBMS that is responsible for scheduling and executing temporal triggers.

```
SHOW PROCESSLIST;
```

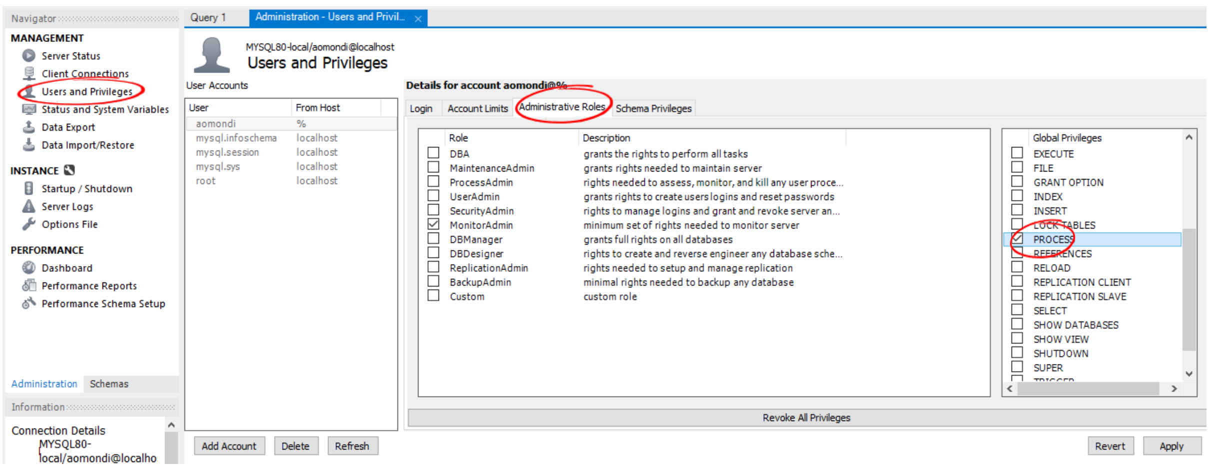
You should get an output similar (not identical) to this on MySQL Workbench:



Id	User	Host	db	Command	Time	State	Info
5	event_scheduler	localhost	NULL	Daemon	47	Waiting on empty queue	NULL
9	aomondi	localhost:52485	classicmodels	Sleep	13		NULL
10	aomondi	localhost:52486	classicmodels	Query	0	starting	SHOW PROCESSLIST

The first line indicates that the event scheduler is ON by default.

Note that the account you have used to login to the database must have the "PROCESS" global privilege to view the list of processes. This can be confirmed in MySQL Workbench by going to "Users and Privileges" followed by the "Administrative Roles" tab as shown in the following figure:



Connection Details
MYSQL80-local/aomondi@localho

MYSQL80-local/aomondi@localhost
Users and Privileges

User Accounts

User	From Host
aomondi	%
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

Details for account aomondi@%

	Role	Description
☐	DBA	grants the rights to perform all tasks
☐	MaintenanceAdmin	grants rights needed to maintain server
☐	ProcessAdmin	rights needed to assess, monitor, and kill any user proce...
☐	UserAdmin	grants rights to create users logins and reset passwords
☐	SecurityAdmin	rights to manage logins and grant and revoke server an...
✓	MonitorAdmin	minimum set of rights needed to monitor server
☐	DBManager	grants full rights on all databases
☐	DBDesigner	rights to create and reverse engineer any database sche...
☐	ReplicationAdmin	rights needed to setup and manage replication
☐	BackupAdmin	minimal rights needed to backup any database
☐	Custom	custom role

	Global Privileges
☐	EXECUTE
☐	FILE
☐	GRANT OPTION
☐	INDEX
☐	INSERT
☐	LOCK TABLES
✓	PROCESS
☐	REFERENCES
☐	RELOAD
☐	REPLICATION CLIENT
☐	REPLICATION SLAVE
☐	SELECT
☐	SHOW DATABASES
☐	SHOW VIEW
☐	SHUTDOWN
☐	SUPER

STEP 2. Turn OFF the event scheduler

There are situations where you can decide to turn off the event scheduler as a DBA, e.g. when a scheduled event has been created incorrectly or maliciously (by a hacker) and is overwhelming the database system with requests for services to read or manipulate data. The performance of the database can slow down if it is overwhelmed with too many requests, e.g. hundreds of thousands of requests per second. This essentially denies genuine users service because the database system is "too busy" attending to many other fake requests. It is common to use the term "Denial of Service (DoS)" attack to describe such a situation.

Turning off the event scheduler will prevent all the events that have been created in the database system from being executed automatically until you turn on the event scheduler again. Execute the following command to turn OFF the event scheduler:

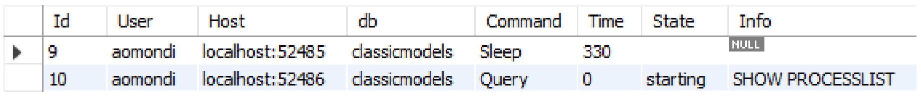
```
SET GLOBAL event_scheduler = OFF;
```

STEP 3. Confirm that the event scheduler is OFF and then turn it back ON again

Execute the following command to confirm that the event scheduler is OFF:

```
SHOW PROCESSLIST;
```

You should get an output similar (not identical) to this:



Id	User	Host	db	Command	Time	State	Info
9	aomondi	localhost:52485	classicmodels	Sleep	330		NULL
10	aomondi	localhost:52486	classicmodels	Query	0	starting	SHOW PROCESSLIST

Notice that the event scheduler is no longer listed as one of the processes currently running on the database system.

You can then turn the event scheduler back ON so that you can use it for the subsequent steps in this lab. Execute the following command to turn it back ON:

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

STEP 4. Create a relation to record the time taken to serve a client

DDL stands for Data Definition Language. It is the section of SQL that can be used to define storage structures that can store data. Execute the following DDL statement to create the customer service ticket relation:

```
CREATE TABLE `classicmodels`.`customer_service_ticket` (
  `customer_service_ticket_ID` int unsigned NOT NULL AUTO_INCREMENT COMMENT
'Identifies the ticket number',
  `customer_service_ticket_resolved` tinyint NOT NULL DEFAULT '0' COMMENT
'Indicates whether the issue raised via the ticket has been resolved (1 if
resolved and 0 if not resolved)',
  `customer_service_ticket_raise_time` timestamp NOT NULL COMMENT 'Records
the time when the ticket was raised by the client. Required to know when to
start the timer.',
  `customer_service_total_wait_time_minutes` int DEFAULT NULL COMMENT
'Records the total amount of time elapsed since the customer raised the
ticket.',
  `customer_service_ticket_last_update` text DEFAULT NULL COMMENT 'Records
a message that specifies when the last update was made by firing the
event',
  `customerNumber` int DEFAULT NULL COMMENT 'Identifies the customer who
has raised the ticket',
  PRIMARY KEY (`customer_service_ticket_ID`),
  CONSTRAINT `FK_1_customers_TO_M_customer_service_ticket` FOREIGN KEY
(`customerNumber`) REFERENCES `customers` (`customerNumber`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
COMMENT='Used to keep track of how long an issue raised via a IT ticketing
system has remained unresolved.';
```

STEP 5. Insert sample data into the created relation

Execute the following statements to insert sample data into the "customer_service_ticket" relation:

This is meant to represent a ticket raised by "Danish Wholesale Imports" (customer number 145 in the relation called customers):

```
INSERT INTO `classicmodels`.`customer_service_ticket`
(`customer_service_ticket_raise_time`, `customerNumber`)
VALUES (CURRENT_TIMESTAMP, 145);
```

Wait for at least 1 minute before executing the next insertion. This is meant to represent a ticket raised by "Kelly's Gift Shop" (customer number 496 in the relation called customers):

```
INSERT INTO `classicmodels`.`customer_service_ticket`
(`customer_service_ticket_raise_time`, `customerNumber`)
VALUES (CURRENT_TIMESTAMP, 496);
```

Wait for at least 2 minutes before executing the next insertion. This is meant to represent a ticket raised by "American Souvenirs Inc" (customer number 168 in the relation called customers):

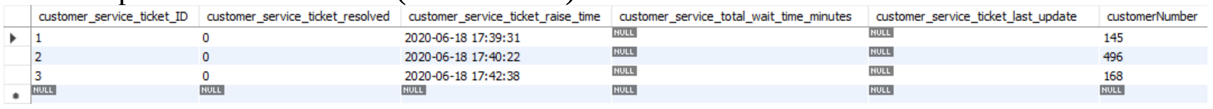
```
INSERT INTO `classicmodels`.`customer_service_ticket`
(`customer_service_ticket_raise_time`, `customerNumber`)
VALUES (CURRENT_TIMESTAMP, 168);
```

STEP 6. View the data in the "customer_service_ticket" relation before the event is fired

Execute the following query to view the data after the insertions:

```
SELECT * FROM classicmodels.customer_service_ticket;
```

Your output should be similar (not identical) to:



customer_service_ticket_ID	customer_service_ticket_resolved	customer_service_ticket_raise_time	customer_service_total_wait_time_minutes	customer_service_ticket_last_update	customerNumber
1	0	2020-06-18 17:39:31	NULL	NULL	145
2	0	2020-06-18 17:40:22	NULL	NULL	496
3	0	2020-06-18 17:42:38	NULL	NULL	168
NULL	NULL	NULL	NULL	NULL	NULL

Take note of the time when the ticket was raised as recorded in the attribute "customer_service_ticket_raise_time".

STEP 7. Create a scheduled event to compute the time elapsed

Execute the following command to create a scheduled event called "EVN_record_customer_waiting_time_every_1_minute_for_1_hour":

```
CREATE EVENT EVN_record_customer_waiting_time_every_1_minute_for_1_hour
ON SCHEDULE EVERY 1 MINUTE
STARTS CURRENT_TIMESTAMP + INTERVAL 3 MINUTE
ENDS CURRENT_TIMESTAMP + INTERVAL 1 HOUR
ON COMPLETION PRESERVE
COMMENT 'This event computes the total time a customer has waited since
they raised a ticket'
DO
UPDATE `classicmodels`.`customer_service_ticket`
SET
    `customer_service_total_wait_time_minutes` = TIMESTAMPDIFF(MINUTE,
        `customer_service_ticket_raise_time`,
        CURRENT_TIMESTAMP),
    `customer_service_ticket_last_update` = CONCAT('The last 1-minute
recurring update was made at ', CURRENT_TIMESTAMP)
WHERE
    `customer_service_ticket_resolved` = 0;
```

Notice the use of the prefix to identify the named database object. Although optional, other prefixes and suffixes that can be used include:

i.) "PK_" represents a primary key
ii.) "FK_" represents a foreign key
iii.) "IDX_" represents an index
iv.) "_UNIQUE" represents a unique constraint (included as a suffix because the prefix would be "IDX" for a "unique index")
v.) "FUNC_" represents a function
vi.) "PROC_" represents a procedure
vii.) "TRG_" represents a trigger, and so on.

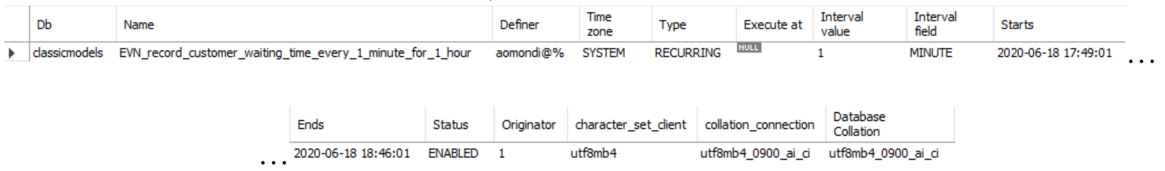
- `ON SCHEDULE EVERY 1 MINUTE` – creates a scheduled event that is fired (executed) every minute
- `STARTS CURRENT_TIMESTAMP + INTERVAL 3 MINUTE` – Specifies that the event should start being fired 3 minutes from the time it was created. Omitting this statement means that the scheduled event will start being fired immediately it has been created.
- `ENDS CURRENT_TIMESTAMP + INTERVAL 1 HOUR` – Specifies that the event should stop working 1 hour from the time when it was created. Omitting this statement means that the scheduled event will continue being executed indefinitely until it is deleted or disabled.

- `ON COMPLETION PRESERVE` – Specifies that event should remain in the database even after the last execution (in this case, even after 1 hour has passed and the event is no longer active).
- `TIMESTAMPDIFF(MINUTE, `customer_service_ticket_raise_time`, CURRENT_TIMESTAMP)` – Used to compute the difference in minutes between two timestamps. In this case, between the time when the customer raised the ticket and the current time when the event is being fired.

STEP 8. Confirm that the scheduled event has been created

Execute the following to view the list of scheduled events in the database:

```
SHOW EVENTS FROM classicmodels;
```



Db	Name	Definer	Time zone	Type	Execute at	Interval value	Interval field	Starts
classicmodels	EVN_record_customer_waiting_time_every_1_minute_for_1_hour	aomondi@%	SYSTEM	RECURRING	NULL	1	MINUTE	2020-06-18 17:49:01

Ends	Status	Originator	character_set_client	collation_connection	Database Collation
2020-06-18 18:46:01	ENABLED	1	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci

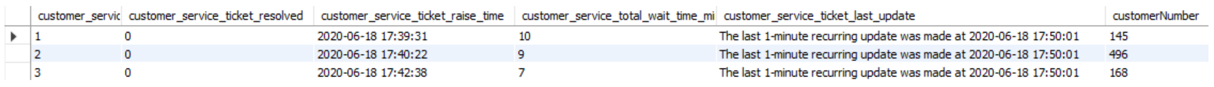
You can also see the code that was used to create a scheduled event by executing the following command:

```
SHOW CREATE EVENT
`EVN_record_customer_waiting_time_every_1_minute_for_1_hour`;
```

STEP 9. View the data in the ticket relation after the event has been fired (at least 5 minutes after you did STEP 7)

Execute the following command to view the data:

```
SELECT * FROM classicmodels.customer_service_ticket;
```



customer_servic	customer_service_ticket_resolved	customer_service_ticket_raise_time	customer_service_total_wait_time_mi	customer_service_ticket_last_update	customerNumber
1	0	2020-06-18 17:39:31	10	The last 1-minute recurring update was made at 2020-06-18 17:50:01	145
2	0	2020-06-18 17:40:22	9	The last 1-minute recurring update was made at 2020-06-18 17:50:01	496
3	0	2020-06-18 17:42:38	7	The last 1-minute recurring update was made at 2020-06-18 17:50:01	168

Notice the automatic update of data that records the number of minutes that a customer has been waiting. This update is being done every minute by the scheduled event.

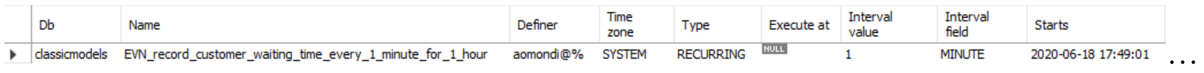
STEP 10. Disable the scheduled event

A DBA can decide to disable a scheduled event when necessary, e.g. to conduct an inspection of what a scheduled event is doing when it is being fired or to correct/update the scheduled event. Execute the following command to disable an event:

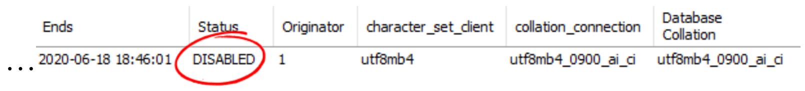
```
ALTER EVENT EVN_record_customer_waiting_time_every_1_minute_for_1_hour
DISABLE;
```

Confirm that the event has been disabled by executing the following command:

```
SHOW EVENTS FROM classicmodels;
```



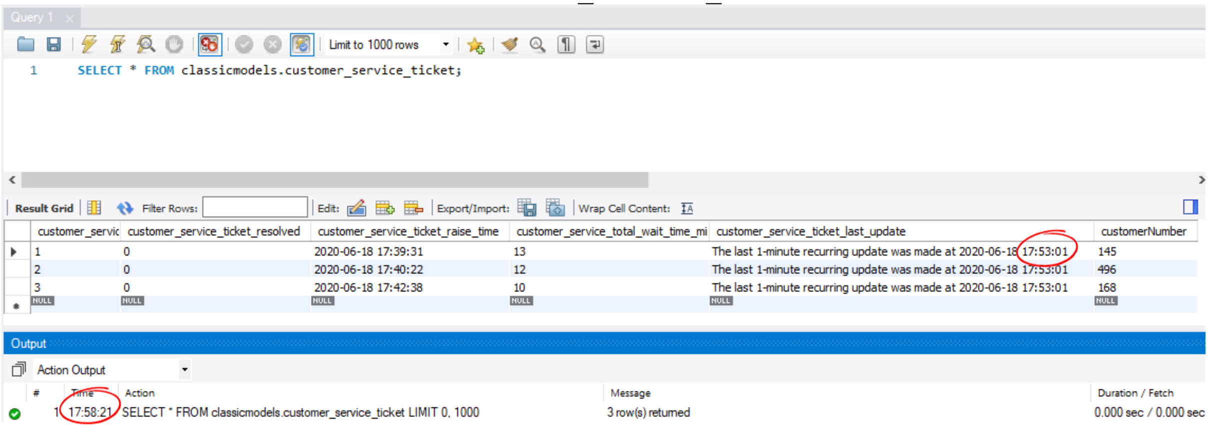
Db	Name	Definer	Time zone	Type	Execute at	Interval value	Interval field	Starts
classicmodels	EVN_record_customer_waiting_time_every_1_minute_for_1_hour	aomondi@%	SYSTEM	RECURRING	NULL	1	MINUTE	2020-06-18 17:49:01



Ends	Status	Originator	character_set_client	collation_connection	Database Collation
2020-06-18 18:46:01	DISABLED	1	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci

Confirm that the event is no longer changing data in the ticket relation by executing the following command about 3 minutes after you disabled the scheduled event:

```
SELECT * FROM classicmodels.customer_service_ticket;
```



```
SELECT * FROM classicmodels.customer_service_ticket;
```

	customer_servic	customer_service_ticket_resolved	customer_service_ticket_raise_time	customer_service_total_wait_time_mi	customer_service_ticket_last_update	customerNumber
▶	1	0	2020-06-18 17:39:31	13	The last 1-minute recurring update was made at 2020-06-18 17:53:01	145
	2	0	2020-06-18 17:40:22	12	The last 1-minute recurring update was made at 2020-06-18 17:53:01	496
	3	0	2020-06-18 17:42:38	10	The last 1-minute recurring update was made at 2020-06-18 17:53:01	168
*	NULL	NULL	NULL	NULL	NULL	NULL

#	Time	Action	Message	Duration / Fetch
1	17:58:21	SELECT * FROM classicmodels.customer_service_ticket LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

STEP 11. Set the status of the issue raised via ticket number 2 to "resolved"

Execute the following command to update the status of the issue resolved via ticket number 2 to "resolved":

```
UPDATE `classicmodels`.`customer_service_ticket` SET
`customer_service_ticket_resolved` = '1' WHERE
(`customer_service_ticket_ID` = '2');
```

STEP 12. Enable the scheduled event

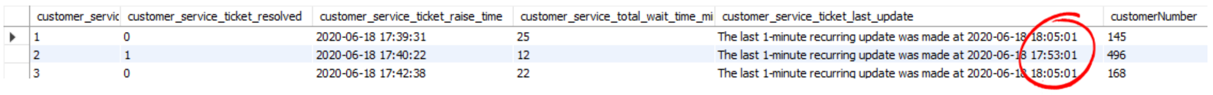
Execute the following command to enable the scheduled event again:

```
ALTER EVENT EVN_record_customer_waiting_time_every_1_minute_for_1_hour
ENABLE;
```

STEP 13. View the data in the ticket relation after the status of the issue raised via ticket number 2 has been set to "resolved"

Execute the following command to view the data:

```
SELECT * FROM `classicmodels`.`customer_service_ticket`;
```



	customer_servic	customer_service_ticket_resolved	customer_service_ticket_raise_time	customer_service_total_wait_time_mi	customer_service_ticket_last_update	customerNumber
▶	1	0	2020-06-18 17:39:31	25	The last 1-minute recurring update was made at 2020-06-18 18:05:01	145
	2	1	2020-06-18 17:40:22	12	The last 1-minute recurring update was made at 2020-06-18 17:53:01	496
	3	0	2020-06-18 17:42:38	22	The last 1-minute recurring update was made at 2020-06-18 18:05:01	168

This is the data as at 6:05 pm. Note that the waiting time recorded under the attribute "customer_service_total_wait_time_minutes" is no longer being updated for ticket number 2. This is because of how the UPDATE statement was written in the scheduled event, i.e.

```
UPDATE `classicmodels`.`customer_service_ticket`
SET
    `customer_service_total_wait_time_minutes` =
     TIMESTAMPDIFF(MINUTE,
        `customer_service_ticket_raise_time`,
        CURRENT_TIMESTAMP),
    `customer_service_ticket_last_update` = CONCAT('The last 1-minute
     recurring update was made at ', CURRENT_TIMESTAMP)
WHERE
    `customer_service_ticket_resolved` = 0;
```

STEP 14. Update the scheduled event

A DBA can also update a scheduled event without disabling it first. Execute the following 2 commands to update the scheduled event:

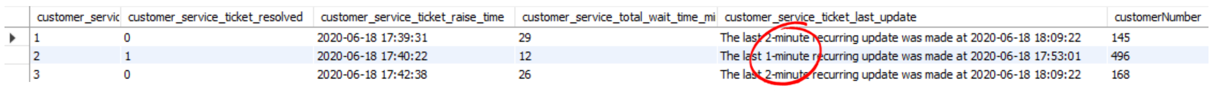
```
ALTER EVENT EVN_record_customer_waiting_time_every_1_minute_for_1_hour
RENAME TO EVN_record_customer_waiting_time_every_2_minutes_for_1_hour;

ALTER EVENT EVN_record_customer_waiting_time_every_2_minutes_for_1_hour
ON SCHEDULE EVERY 2 MINUTE
STARTS CURRENT_TIMESTAMP + INTERVAL 3 MINUTE
ENDS CURRENT_TIMESTAMP + INTERVAL 1 HOUR
ON COMPLETION PRESERVE
DO
UPDATE `classicmodels`.`customer_service_ticket`
SET
    `customer_service_total_wait_time_minutes` = TIMESTAMPDIFF(MINUTE,
        `customer_service_ticket_raise_time`,
        CURRENT_TIMESTAMP),
    `customer_service_ticket_last_update` = CONCAT('The last 2-minute
recurring update was made at ', CURRENT_TIMESTAMP)
WHERE
    `customer_service_ticket_resolved` = 0;
```

- `RENAME TO EVN_record_customer_waiting_time_every_2_minutes_for_1_hour` – Specifies the new name of an event
- `ON SCHEDULE EVERY 2 MINUTE` – Used to change the time interval from every 1 minute to every 2 minutes
- `STARTS CURRENT_TIMESTAMP + INTERVAL 3 MINUTE` and `ENDS CURRENT_TIMESTAMP + INTERVAL 1 HOUR` – If you do not specify the start and end times when updating an event, the event will not use the start and end times that were set when it was being created.
- `` `customer_service_ticket_last_update` = CONCAT('The last 2-minute recurring update was made at ', CURRENT_TIMESTAMP) `` – Specifies a different message to be updated, i.e. "The last 2-minute recurring update was…" instead of "The last 1-minute recurring update was…"

STEP 15. Confirm that the updated scheduled event is being fired every 2 minutes

Execute the following statement to view the data in the ticketing relation:

```
SELECT * FROM `classicmodels`.`customer_service_ticket`;
```



customer_servic	customer_service_ticket_resolved	customer_service_ticket_raise_time	customer_service_total_wait_time_mi	customer_service_ticket_last_update	customerNumber
1	0	2020-06-18 17:39:31	29	The last 2-minute recurring update was made at 2020-06-18 18:09:22	145
2	1	2020-06-18 17:40:22	12	The last 1-minute recurring update was made at 2020-06-18 17:53:01	496
3	0	2020-06-18 17:42:38	26	The last 2-minute recurring update was made at 2020-06-18 18:09:22	168

Notice the message being posted for the unresolved tickets.

STEP 16. Delete the scheduled event

Execute the following command to delete the scheduled event:

```
DROP EVENT EVN_record_customer_waiting_time_every_2_minutes_for_1_hour;
```

Note that the name of the scheduled event had been updated. At this point, we are referring to the updated name of the scheduled event which is, "EVN_record_customer_waiting_time_every_2_minutes_for_1_hour"

STEP 17. Delete the relation that stores the ticketing details

Execute the following command to delete the "customer_service_ticket" relation:

```
DROP TABLE `classicmodels`.`customer_service_ticket`;
```

This allows you, if you wish, to go through the lab manual from STEP 1 again.

Further Reading and References

Modifying MySQL Events. (n.d.). *MySQL Tutorial*. Retrieved 18 June 2020, from https://www.mysqltutorial.org/mysql-triggers/modifying-mysql-events/

Oracle (2020). *MySQL :: MySQL 8.0 Reference Manual: 24.4 Using the Event Scheduler*. https://dev.mysql.com/doc/refman/8.0/en/event-scheduler.html

Working with MySQL Scheduled Event. (n.d.). *MySQL Tutorial*. Retrieved 18 June 2020, from https://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/