Scheduled Queries

Intro

Executing statements periodically can be usefull in

- Pulling informations from external systems
- Periodically updating column statistics
- Rebuilding materialized views

Overview

- The metastore maintains the scheduled queries in the metastore database
- Hiveserver(s) periodically polls the metastore for a scheduled query to be executed
  - During execution informations about ongoing/finished executions are kept in the metastore

Scheduled queries were added in Hive 4.0 (HIVE-21884)

Hive has it's scheduled query interface built into the language itself for easy access:

Maintaining scheduled queries

Create Scheduled query syntax

CREATE SCHEDULED QUERY <scheduled_query_name>
<scheduleSpecification>
[<executedAsSpec>]
[<enableSpecification>]
<definedAsSpec>

Alter Scheduled query syntax

ALTER SCHEDULED QUERY <scheduled_query_name>
(<scheduleSpec>|<executedAsSpec>|<enableSpecification>|<definedAsSpec>|<executeSpec>);

Drop syntax

DROP SCHEDULED QUERY <scheduled_query_name>;

scheduleSpecification syntax

Schedules can be specified using CRON expressions or for common cases there is a simpler form; in any case the schedule is stored as Quartz cron expression.

CRON based schedule syntax

CRON <quartz_schedule_expression>

where quartz_schedule_expression is quoted schedule in the Quartz format

https://www.freeformatter.com/cron-expression-generator-quartz.html

For example the `CRON '0 */10 * * * ? *'` expression will fire every 10 minutes.

EVERY based schedule syntax

To give a more readable way to declare schedules EVERY can be used.

EVERY [<integer>] (SECOND|MINUTE|HOUR) [(OFFSET BY|AT) <timeOrDate>]

the format makes it possible to declare schedules in a more readable way:

EVERY 2 MINUTES
EVERY HOUR AT '0:07:30'
EVERY DAY AT '11:35:30'

ExecutedAs syntax

EXECUTED AS <user_name>

Scheduled queries are executed as the declaring user by default; but people with admin privileges might be able to change the executing user.

enableSpecification syntax

(ENABLE[D] | DISABLE[D])

Can be used to enable/disable a schedule.

⚠ For CREATE SCHEDULED QUERY statements the default behaviour is set by the configuration key **hive.scheduled.queries.create.as.enabled**

⚠ In case there are in-flight scheduled executions at the time when the corresponding schedule is disabled - the already running executions will still finish. But no more executions will be triggered.

Defined AS syntax

[DEFINED] AS <hiveQuery>

The “query” is a single statement expression to be scheduled for execution.

executeSpec syntax

EXECUTE

Changes the schedules next execution time to be now. Could be useful during debugging/development.

System tables/views

Informations about scheduled queries/executions can be obtain by using the *information_schema* or the *sysdb* - recommended way is to use the information_schema; *sysdb* is tables are there to build the *information_schema* level views - and for debugging.

information_schema.scheduled_queries

Suppose we have a scheduled query defined by:

create scheduled query sc1 cron '0 */10 * * * ? *' as select 1;

Let's take a look at it in the **information_schema.scheduled_queries** table by using

select * from information_schema.scheduled_queries;

I will transpose the resultset to describe each column

scheduled_query_id	1	Internally, every scheduled query also has a numeric id
schedule_name	sc1	The name of the schedule
enabled	true	True if the schedule is enabled
cluster_namespace	hive	The namespace thes scheduled query belongs to
schedule	0 */10 * * * ? *	The schedule described in QUARTZ cron format
user	dev	The owner/executor of the query
query	select 1	The query being scheduled
next_execution	2020-01-29 16:50:00	Technical column; shows when the next execution should happen

⚠ **(schedule_name,cluster_namespace)** is unique

information_schema.scheduled_executions

This view can be used to get information about recent scheduled query executions.

select * from information_schema.scheduled_executions;

One record in this view has the following informations:

scheduled_execution_id	**13**	Every scheduled query execution has a unique numeric id
schedule_name	**sc1**	The schedule name to which this execution belongs
executor_query_id	**dev_20200131103008_c9a39b8d-e26b-44cd-b8ae-9d054204dc07**	The query id assigned by the execution engine for the given scheduled execution
state	**FINISHED**	State of the execution; can be
start_time	**2020-01-31 10:30:06**	Start time of execution
end_time	**2020-01-31 10:30:08**	End time of execution
elapsed	**2**	(computed) end_time-start_time
error_message	**NULL**	In case the query is FAILED the error message is shown here

last_update_time	**NULL**	During execution the last update time the executor provided informations about the state

Execution states

INITED	The scheduled execution record is created at the time an executor is assigned to run it; The INITED state is retained until the first update from the executor comes in.
EXECUTING	Queries in executing state are being processed by the executor; during this phase the executor reports the progress of the query in intervals defined by: **hive.scheduled.queries.executor.progress.report.interval**
FAILED	The query execution stoped by an error code(or an exception) when this state is set the **error_message** is also filled.
FINISHED	The query finished without problems
TIMED_OUT	An execution is considered timed out when it's being executed for more than **metastore.scheduled.queries.execution.timeout**. The scheduled queries maintenance task checks for any timed out executions.

How long are execution informations are retained?

The scheduled query maintenance task removes older than **metastore.scheduled.queries.execution.max.age** entries.

Configuration

Hive metastore related configuration

- **metastore.scheduled.queries.enabled (default: true)**
  Controls the metastore side support for scheduled queries; forces all HMS scheduled query related endpoints to return with an error
- **metastore.scheduled.queries.execution.timeout (default: 2 minutes)**
  In case a scheduled execution is not updated for at least this amount of time; it's state will be changed to **TIMED_OUT** by the cleaner task
- **metastore.scheduled.queries.execution.maint.task.frequency (default: 1 minute)**
  Interval of scheduled query maintenance task. Which removes executions above max age; and marks executions as TIMED_OUT if the condition is met
- **metastore.scheduled.queries.execution.max.age (default: 30 days)**
  Maximal age of a scheduled query execution entry before it is removed.

HiveServer2 related configuration

- **hive.scheduled.queries.executor.enabled (default: true)**
  Controls whether HS2 will run scheduled query executor.
- **hive.scheduled.queries.namespace (default: "hive")**
  Sets the scheduled query namespace to be used. New scheduled queries are created in this namespace; and execution is also bound to the namespace
- **hive.scheduled.queries.executor.idle.sleep.time (default: 1 minute)**
  Time to sleep between querying for the presence of a scheduled query.
- **hive.scheduled.queries.executor.progress.report.interval (default: 1 minute)**
  While scheduled queries are in flight; a background update happens periodically to report the actual state of the query.
- **hive.scheduled.queries.create.as.enabled (default: true)**
  This option sets the default behaviour of newly created scheduled queries.
- **hive.security.authorization.scheduled.queries.supported (default: false)**
  Enable this if the configured authorizer is able to handle scheduled query related calls.

Examples

Example 1 – basic example of using schedules

```
create table t (a integer);

-- create a scheduled query; every 10 minute insert a new row
create scheduled query sc1 cron '0 */10 * * * ? *' as insert into t values (1);
-- depending on hive.scheduled.queries.create.as.enabled the query might get create in disabled mode
-- it can be enabled using:
alter scheduled query sc1 enabled;

-- inspect scheduled queries using the information_schema
select * from information_schema.scheduled_queries s where schedule_name='sc1';
+-----------------------+------------------+------------+----------------------+-------------------+---------
+-----------+----------------------+
| s.scheduled_query_id  | s.schedule_name  | s.enabled  | s.cluster_namespace  |    s.schedule     | s.user  |
s.query  |   s.next_execution   |
+-----------------------+------------------+------------+----------------------+-------------------+---------
+-----------+----------------------+
| 1                     | sc1              | true       | hive                 | 0 */10 * * * ? *  | dev     |
select 1  | 2020-02-03 15:10:00  |
+-----------------------+------------------+------------+----------------------+-------------------+---------
+-----------+----------------------+

-- wait 10 minutes or execute by issuing:
alter scheduled query sc1 execute;

select * from information_schema.scheduled_executions s where schedule_name='sc1' order by
scheduled_execution_id desc limit 1;
+---------------------------+------------------+-----------------------------------------------------+-----------
+----------------------+----------------------+------------+------------------+---------------------+
| s.scheduled_execution_id  | s.schedule_name  |                s.executor_query_id                  |  s.state
|     s.start_time     |      s.end_time      | s.elapsed  | s.error_message  | s.last_update_time  |
+---------------------------+------------------+-----------------------------------------------------+-----------
+----------------------+----------------------+------------+------------------+---------------------+
| 496                       | sc1              | dev_20200203152025_bdf3deac-0ca6-407f-b122-c637e50f99c8 |
FINISHED  | 2020-02-03 15:20:23  | 2020-02-03 15:20:31  | 8          | NULL             | NULL                |
+---------------------------+------------------+-----------------------------------------------------+-----------
+----------------------+----------------------+------------+------------------+---------------------+
```

Example 2 – analyze external table periodically

Suppose you have an external table - the contents of it is slowly changing...which will eventually lead that Hive will utilize outdated statistics during planning time

```
-- create external table
create external table t (a integer);

-- see where the table lives:
desc formatted t;
[...]
| Location:                    | file:/data/hive/warehouse/t                       |
NULL                                          |
[...]

-- in a terminal; load some data into the table directory:
seq 1 10 > /data/hive/warehouse/t/f1

-- back in hive you will see that
select count(1) from t;
10
-- meanwhile basic stats show that the table has "0" rows
desc formatted t;
[...]
|                              | numRows                                           |
0                                             |
[...]

create scheduled query t_analyze cron '0 */1 * * * ? *' as analyze table t compute statistics for columns;

-- wait some time or execute by issuing:
alter scheduled query t_analyze execute;


select * from information_schema.scheduled_executions s where schedule_name='ex_analyze' order by
scheduled_execution_id desc limit 3;
+----------------------------+------------------+-----------------------------------------------------
+------------+----------------------+----------------------+------------+------------------
+----------------------+
| s.scheduled_execution_id  | s.schedule_name  |                s.executor_query_id                |  s.
state   |     s.start_time     |      s.end_time      | s.elapsed  | s.error_message  |  s.last_update_time  |
+----------------------------+------------------+-----------------------------------------------------
+------------+----------------------+----------------------+------------+------------------
+----------------------+
| 498                        | t_analyze       | dev_20200203152640_a59bc198-3ed3-4ef2-8f63-573607c9914e |
FINISHED   | 2020-02-03 15:26:38  | 2020-02-03 15:28:01  | 83         | NULL             | NULL
|
+----------------------------+------------------+-----------------------------------------------------
+------------+----------------------+----------------------+------------+------------------
+----------------------+

-- and the numrows have been updated
desc formatted t;
[...]
|                              | numRows                                           |
10                                            |
[...]


-- we don't want this running every minute anymore...
alter scheduled query t_analyze disable;
```

Example 3 – materialized view rebuild

```
-- some settings...they might be there already
```

```
set hive.support.concurrency=true;
set hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;
set hive.strict.checks.cartesian.product=false;
set hive.stats.fetch.column.stats=true;
set hive.materializedview.rewriting=true;

-- create some tables
CREATE TABLE emps (
  empid INT,
  deptno INT,
  name VARCHAR(256),
  salary FLOAT,
  hire_date TIMESTAMP)
STORED AS ORC
TBLPROPERTIES ('transactional'='true');

CREATE TABLE depts (
  deptno INT,
  deptname VARCHAR(256),
  locationid INT)
STORED AS ORC
TBLPROPERTIES ('transactional'='true');

-- load data
insert into emps values (100, 10, 'Bill', 10000, 1000), (200, 20, 'Eric', 8000, 500),
  (150, 10, 'Sebastian', 7000, null), (110, 10, 'Theodore', 10000, 250), (120, 10, 'Bill', 10000, 250),
  (1330, 10, 'Bill', 10000, '2020-01-02');
insert into depts values (10, 'Sales', 10), (30, 'Marketing', null), (20, 'HR', 20);

insert into emps values (1330, 10, 'Bill', 10000, '2020-01-02');

-- create mv
CREATE MATERIALIZED VIEW mv1 AS
  SELECT empid, deptname, hire_date FROM emps
    JOIN depts ON (emps.deptno = depts.deptno)
    WHERE hire_date >= '2016-01-01 00:00:00';

EXPLAIN
SELECT empid, deptname FROM emps
JOIN depts ON (emps.deptno = depts.deptno)
WHERE hire_date >= '2018-01-01';

-- create a schedule to rebuild mv
create scheduled query mv_rebuild cron '0 */10 * * * ? *' defined as
  alter materialized view mv1 rebuild;


-- from this expalin it will be seen that the mv1 is being used
EXPLAIN
SELECT empid, deptname FROM emps
JOIN depts ON (emps.deptno = depts.deptno)
WHERE hire_date >= '2018-01-01';

-- insert a new record
insert into emps values (1330, 10, 'Bill', 10000, '2020-01-02');

-- the source tables are scanned
EXPLAIN
SELECT empid, deptname FROM emps
JOIN depts ON (emps.deptno = depts.deptno)
WHERE hire_date >= '2018-01-01';

-- wait 10 minutes or execute
alter scheduled query mv_rebuild execute;

-- run it again...the view should be rebuilt
EXPLAIN
SELECT empid, deptname FROM emps
JOIN depts ON (emps.deptno = depts.deptno)
WHERE hire_date >= '2018-01-01';
```

Example 4 – Ingestion

```
drop table if exists t;
drop table if exists s;

-- suppose that this table is an external table or something
-- which supports the pushdown of filter condition on the id column
create table s(id integer, cnt integer);

-- create an internal table and an offset table
create table t(id integer, cnt integer);
create table t_offset(offset integer);
insert into t_offset values(0);

-- pretend that data is added to s
insert into s values(1,1);

-- run an ingestion...
from (select id==offset as first,* from s
join t_offset on id>=offset) s1
insert into t select id,cnt where first = false
insert overwrite table t_offset select max(s1.id);

-- configure to run ingestion every 10 minutes
create scheduled query ingest every 10 minutes defined as
from (select id==offset as first,* from s
join t_offset on id>=offset) s1
insert into t select id,cnt where first = false
insert overwrite table t_offset select max(s1.id);

-- add some new values
insert into s values(2,2),(3,3);

-- pretend that a timeout have happened
alter scheduled query ingest execute;
```