Scheduled queries in Hive

Users right now have options to use external tools to schedule queries in Hive like cron.
With materialized views the need is increased to have something to help in maintaining these views - to keep them at optimal performance level.

1 How other databases handle this

- Oracle

    - DBMS_SCHEDULER
- Postgresql
    - No built-in solution
    - pgAgent
        - Runs externally as a process; configured thru pgadmin3
    - pg_cron extension
        - Runs as an extension internally
- DB2
    - Provided ADMIN_ADD_TASK
- MySql
    - Event scheduler

2 Use cases

- Compaction
  - alter table t compact ‘major’
- Periodically rebuild materialized views
  - ALTER MATERIALIZED VIEW mv REBUILD
- Run ingestion from kafka
  - Recommended way to use a single-statement multi insert ; to ensure that the data/offsets are in sync
  - https://docs.hortonworks.com/HDPDocuments/HDP3/HDP-3.1.0/integrating-hive/content/hive_ingest_kafka_data_into_hive.html
-

3 Objectives

- support multiple compute clusters
  - I’m not sure; if multiple HS2 instances connecting to the same HMS qualifies as multiple cluster support; then yes.
- make it easy for other engines to support the query schedules (impala, spark)
  - Proposal is to move the scheduling into hms and hs2 implements only execution
  - Hms api is designed to be narrow
- allow more complex "scripts" at some point.
  - As of now it seems like “single-statement” will be ok; the most complicated use case; the kafka ingestion needs a multi insert statement to work reliably
- should be able to use sql to administrate (create schedules, read schedules, status)

4 Notes

- Persistence
  - store it in metastore
- Make status visible to user
  - expose metastore tables as sysdb
  - Store query status in that table
- Execute a scheduled query exactly once after at least one trigger time have passed
  - HS2 asks Metastore which scheduled query should be running
- Temporary off switch
  - After say 10 minutes of a new HS2 startup; scheduled queries are not launched (to prevent possible meltdown)
  - Make it possible to disable this in a HS2

- Limit number of parallel scheduled queries
    - Total number can be enforced by the metastore
    - Per HS2 count can be enforced there
- Execute only once in a given timeframe
    - Some rpc ping-pong between hs2 and metastore
- Executed query status
    - Store it in metastore table; but only the return code
- No restart before termination
    - ok
- Per query enablement
    - must
- Security?
    - First: only admin may create it; but may run the query as someone else
    - Letting anyone create a scheduled query may cause troubles…
- Schedule description format
    - Use some “cron” alike one; as it seems like pretty common
- Timeframe bound schedule support: from/until
    - Not planned only the cases which are possible with cron
- AT schedule (fire once)
    - Not planned only the cases which are possible with cron
- Detailed info about scheduled query execution:
    - Some info about the execution state; probably errorstring
    - Hive’s query id

5 SQL level access options

5.1 UDF alike implementation as in pg/ora

- Pro:
  - May provide better flexibility in some cases; since using a select also enable constant folding/etc
  - Handling codes could placed inside a single UDF ; more concise
- Con:
  - Might need to check if it's not a full fledged execution; and prevent running it as a real udf (outside of HS2)
  - I don't think we can place them to a package or something; so every function is prefixed
- Pro/Con:
  - The query needs to be escaped

5.1.1 Approach - separate UDFs

Separate udfs for create/alter/drop to move the type of operation into the UDFs name

```
* create
  select create_scheduled_query('Q1', 'SELECT 1', '1 2 * * *');
* change schedule
  select alter_scheduled_query('Q1', 'schedule', '1 2 * * *');
* change query
  select alter_scheduled_query('Q1', 'query', 'select 1');
* disable
  select alter_scheduled_query('Q1', 'disable');
* enable
  select alter_scheduled_query('Q1', 'enable');
* list status
  select * from sysdb.scheduled_queries;
* drop
  select drop_scheduled_query('Q1', 'enable');
```

5.1.2 Approach - single UDF

In this version I really like the cleary understandable logic behind the UDF which might aid the user
Users could get the full doc about how to use it at the command line (desc function scheduled_query)

```
* create
  select scheduled_query('Q1', ‘create’, 'SELECT 1', '1 2 * * *');
  select scheduled_query('Q1', ‘create’, $$SELECT 1$$, '1 2 * * *');
* change schedule
  select scheduled_query('Q1', 'schedule', '1 2 * * *');
* change query
  select scheduled_query('Q1', 'query', 'select 1');
* disable
  select scheduled_query('Q1', 'disable');
* enable
  select scheduled_query('Q1', 'enable');
* list status
  select * from sysdb.scheduled_queries;
* drop
  select scheduled_query('Q1', 'drop’);
```

5.2 Build support into the language

- Pro:
  - May seem more readable (at first at least)
- Con:
  - Needs to be implemented deeper; embedded into the parser/etc
  - I would be afraid that users won't be able to remember it...and had to go to the docs every time they need to interact with it

5.2.1 Language native with escaping

- Non-escaped query
  - may make the users believe that multi statement is also supported
  - could be later extended to support multiple statements

```
* create
  create scheduled query Q1 executed by joe  cron '1 1 * * *' defined as 'update ...';
* change schedule
  alter scheduled query Q1 cron '2 2 * * *'
* change query
  alter scheduled query Q1 defined as 'select 2'
* disable
  alter scheduled query Q1 set disabled
* enable
  alter scheduled query Q1 set enabled
* list status
  select * from sysdb.scheduled_queries;
* drop
  drop scheduled query Q1
```

5.2.2 Language native without escaping

- Clearly communicates that only 1 statement is supported

```
* create
  create scheduled query Q1 executed as joe scheduled '1 1 * * *’ as update t set a=1;
* change schedule
  alter scheduled query Q1 cron '2 2 * * *'
* change query
  alter scheduled query Q1 defined as select 2
* disable
  alter scheduled query Q1 set disabled
* enable
  alter scheduled query Q1 set enabled
* list status
  select * from sysdb.scheduled_queries;
* drop
  drop scheduled query Q1
```

5.3 Build around a “table”

- Pro:
  - Every operation could be a plain sql operation
- Con:
  - Would probably need support for DO-INSTEAD for table insert/delete/etc; which right now don’t have / problematic to do - so it’s most probably not an option...

5.3.1 Sample

```
* create
  insert into sysdb.scheduled_queries ('id','query','schedule') values ('Q1', 'SELECT 1', '1 2 * *
*');
* change schedule
  update sysdb.scheduled_queries set schedule='1 2 * * *'
  where id='Q1';
* change query
  update sysdb.scheduled_queries set query='select 2'
  where id='Q1';
* disable
  update sysdb.scheduled_queries set enabled=false
  where id='Q1';
* enable
  update sysdb.scheduled_queries set enabled=true
  where id='Q1';
* list status
  select * from sysdb.scheduled_queries;
* drop
  delete from sysdb.scheduled_queries
  where id='Q1';
```

6 Actual dialect

Currently it seems like the language native; single statement one will be it (without quotation).

7 Persisted data

7.1 SCHEDULED_QUERIES

- **Id:** integer id
- **clusterId:** the cluster the scheduled query belongs to
- **catalog:** the catalog this schedule belongs to
- **scheduleId**: every query should be named by the user; to make it easier to both reference and identify
- **enabled:** boolean to enable/disable the query
- **schedule**: cron expression; probably use cron-utils library to work with it
- **user**: the user who should be executing the query
- **query:** the actual query which is scheduled
- **nextExecution**: timestamp of next pending execution

7.2 SCHEDULED_EXECUTIONS

- **id**: primary key
- **scheduledQueryId**: the queryId this execution belongs to
- **executorQueryId:** the queryid present in the logs
- **state**: some summarized state of the execution:
    - EXECUTING / TIMED_OUT / FAILED / OK
- **startTime**: the start time of the execution
- **endTime:** the end time
- **nextDeadline**: auxilary timestamp used to identify queries in TIMED_OUT state
- **errorString**: give some info about what went wrong

8 Actual scheduling of a query

8.1 Metastore based coordination

- The actual handling of the schedules/etc is on the metastore side, which makes it easier at the execution side to use this feature.
- Seamlessly supports multi hms and multiple hs2 or other clients
- Doesn't put the implementation of the actual scheduling to the executor

8.1.1 Executor side(HS2 or other):

Execute-periodically the following:

1. Poll the metastore for pending query to execute
2. Start executing the query
    a. During execution report back periodically that the scheduled query is in progress
3. Report exit status back to the metastore

8.1.2 Metastore side

8.1.2.1 scheduled_query_poll(catalog)

Executor ask for outstanding scheduled query to execute

1. Atomically update the value of **nextExecution** in ***scheduled_queries*** which has **min(nextExecution)**
2. Insert a new row into ***schdeuled_executions*** and use the execution id to track this execution
3. Return the following to the executor
    a. query
    b. executionId

8.1.2.2 scheduled_query_progress(executionId, state, errorString)

1. Update the row in ***schdeuled_executions*** ; update **nextDeadline** to now() + timeout

8.1.2.3 HouseKeeping

- There should be a logic to mark executions which are not updated for a while as TIMED_OUT
- The ***schdeuled_executions*** table should be trimmed from time to time

Questions

- May we permit the same named “schedule” in multiple namespaces and/or catalogs?
    - It will be permitted…

Optional improvements

- Alter scheduled query asd set next execution?
- Report the time until next execution during poll; so that frequently scheduled queries will run more accurately without doing frequent polls
-