

1.0.0

Managing Apache Impala

Date published: 2020-11-30

Date modified: 2024-07-26

CLOUDERA

<https://docs.cloudera.com/>

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

ACID Operation.....	4
Concepts Used in FULL ACID v2 Tables.....	4
Key Differences between INSERT-ONLY and FULL ACID Tables.....	5
Compaction of Data in FULL ACID Transactional Table.....	6
Using HLL Datasketch Algorithms in Impala.....	6
Using KLL Datasketch Algorithms in Impala.....	10
Automatic Invalidation/Refresh of Metadata.....	13

READ Support for FULL ACID ORC Tables

FULL ACID v2 transactional tables are readable in Impala without modifying any configurations. You must have Cloudera Runtime 7.2.2 or higher and have connection to Hive Metastore server in order to READ from FULL ACID tables.

There are two types of transactional tables available with Hive ACID.

- INSERT-ONLY
- FULL ACID

Until this release, Impala in CDP supported INSERT-ONLY transactional tables allowing both READ and WRITE operations. The latest version of Impala in CDP now also supports READ of FULL ACID ORC tables.

By default tables created in Impala are INSERT-ONLY managed tables whereas the default tables in Hive are managed tables that are FULL-ACID.

Limitations

- Impala cannot CREATE or WRITE to FULL ACID transactional tables yet. You can CREATE and WRITE FULL ACID transactional tables with transaction scope at the row level via HIVE and use Impala to READ these tables.
- Impala does not support ACID v1.

Concepts Used in FULL ACID v2 Tables

Before beginning to use FULL ACID v2 tables you must be aware of these new concepts like transactions, WriteIds, rowIDs, delta delete directories, locks, etc. that are added to FULL ACID tables to achieve ACID semantics.

Write IDs

For every transaction, both read and write, Hive will assign a globally unique ID. For transactional writes like INSERT and DELETE, it will also assign a table-wise unique ID, a write ID. The write ID range will be encoded in the delta and delete directory names. Results of a DML transactional query are allocated to a location under partition/table. This location is derived by Write ID allocated to the transaction. This provides Isolation of DML queries and such queries can run in parallel without interfering with each other.

New Sub-directories

New data files resulting from a DML query are written to a unique location derived from WriteId of the transaction. You can find the results of an INSERT query in delta directories under partition/table location. Depending on the operation type there can be two types of delta directories:

- Delta Directory: This type is created for the results of INSERT statements and is named `delta_<writeId>_<writeId>` under partition/table location.
- Delete Delta Directory: This delta directory is created for results of DELETE statements and is named `delete_delta_<writeId>_<writeId>` under partition/table location.

UPDATE operations create both delete and delta directories.

Row IDs

rowId is the auto-generated unique ID within the transaction and bucket. This is added to each row to identify each row in a table. RowID is used during a DELETE operation. When a record is deleted from a table, the rowId of the deleted row will be written to the delete_delta directory. So for all subsequent READ operations all rows will be read except these rows.

Schematic differences between INSERT-ONLY and FULL ACID tables

INSERT-ONLY tables do not have a special schema. They store the data just like plain original files from the non-ACID world. However, their files are organized differently. For every INSERT statement the created files are put into a transactional directory which has transactional information in its name.

Full ACID tables do have a special schema. They have row identifiers to support row-level DELETES. So a row in Full ACID format looks like this:

```
{
  "operation": 0,
  "originalTransaction": 1,
  "bucket": 536870912,
  "rowId": 0,
  "currentTransaction": 1,
  "row": {"i": 1}
}
```

- The green columns are the hidden/system ACID columns.
- Field “row” holds the user data.
- operation 0 means INSERT, 1 UPDATE, and 2 DELETE. UPDATE will not appear because of the split-update technique (INSERT + DELETE).
- originalTransaction is the write ID of the INSERT operation that created this row.
- bucket is a 32-bit integer defined by BucketCodec class.
- rowId is the auto-generated unique ID within the transaction and bucket.
- currentTransaction is the current write ID. For INSERT, it is the same as currentTransaction. For DELETE, it is the write ID when this record is first created.
- row contains the actual data. For DELETE, row will be null.

Key Differences between INSERT-ONLY and FULL ACID Tables

Before beginning to use FULL ACID v2 tables you must be aware of the key differences between the INSERT-ONLY and FULL-ACID tables.

This table highlights some of the differences between the INSERT-ONLY and FULL ACID tables.

	INSERT-ONLY	FULL ACID
Schema	There is no special data schema. They store the data just like plain original files from the non-ACID world.	Data is in special format, i.e. there are synthetic columns with transactional information in addition to actual data.
Transactional information	Transactional information is encoded in directory names.	Full ACID tables also use the same directory structure as INSERT-only tables. Transactional information is encoded in the directory names. Directory name and filename are the source of transactional information.
Table properties	'transactional'='true', 'transactional_properties'='insert_only'	'transactional'='true'
Supported operations	INSERT-ONLY tables only support insertion of data. UPDATES and DELETES are not supported. These tables also provide CREATE TABLE, DROP TABLE, TRUNCATE, INSERT, SELECT operations.	FULL ACID ORC tables can be READ using IMPALA. These tables also provide UPDATE and DELETE operations at the row level using HIVE. This is achieved using transactions like Insert-Only Tables along with changes in ORC Reader to support deletes.
WRITE operation	WRITE operations are atomic and the results of the insert operation are not visible to other query operations until the operation is committed.	WRITE operations are atomic - The operation either succeeds completely or fails; it does not result in partial data.

	INSERT-ONLY	FULL ACID
INSERT operation	For every INSERT statement the created files are added to a transactional directory which has transactional information in its name.	INSERT operation is done through HIVE and this statement is executed in a single transaction. This operation creates a delta directory containing information about this transaction and its data.
DELETE operation	N/A	DELETE operation is done through HIVE and this event creates a special “delete delta” directory.
UPDATE operation	N/A	UPDATE operation is done through HIVE. This operation is split into an INSERT and DELETE operation. This operation creates a delta dir followed by a delete dir.
READ operation	READ operations always read a consistent snapshot of the data.	READ operations always read a consistent snapshot of the data.
Supported file format	Supports any file formats.	Supports only ORC.
Compactions	Minor and major compactions are supported.	Minor compactions can be created, which means several delta and delete directories can be compacted into one delta and delete directory. Major compactions are also supported.

File structure of FULL ACID transactional table

Hive 3 achieves atomicity and isolation of operations on transactional tables by using techniques in write, read, insert, create, delete, and update operations that involve delta files, which can provide query status information and help you troubleshoot query problems.

Compaction of Data in FULL ACID Transactional Table

As administrator, you need to manage compaction of delta files that accumulate during data ingestion. Compaction is a process that performs critical cleanup of files.

Hive creates a set of delta files for each transaction that alters a table or partition and stores them in a separate delta directory. When the number of delta and delete directories in the table grow, the read performance will be impacted, since reading is a process of merging the results of valid transactions. To avoid any compromise on the read performance, occasionally Hive performs compaction, namely minor and major. This process merges these directories while preserving the transaction information.

Using HLL Datasketch Algorithms in Impala

You can use Datasketch algorithms (HLL) for queries that take too long to calculate exact results due to very large data sets (e.g. number of distinct values).

You may use data sketches (HLL algorithms) to generate approximate results that are much faster to retrieve. HLL is an algorithm that gives approximate answers for computing the number of distinct values in a column. The value returned by this algorithm is similar to the result of COUNT(DISTINCT col) and the NDV function integrated with Impala. However, HLL algorithm is much faster than COUNT(DISTINCT col) and the NDV function and is less memory-intensive for columns with high cardinality.

These HLL algorithms create “sketches”, which are data structures containing an approximate calculation of some facet of the data and which are compatible regardless of which system reads or writes them. For e.g., This particular algorithm works in two phases. First it creates a sketch from the dataset provided for the algorithm and then as the second step in the process you can read the estimate from the sketch. Using this approach it’s also possible to create granular sketches for a specific dataset (e.g. one sketch per partition) and later on use them to provide answers for different count(distinct) queries without reading the actual data as they can also be merged together. Using HLL you can create sketches by Hive and read it with Impala for getting the estimate and vice versa.

Usage Notes

Query results produced by the HLL algorithm are approximate but within well defined error bounds. Because the number of distinct values in a column returned by this HLL algorithm is an estimate, it might not reflect the precise number of different values in the column, especially if the cardinality is very high.

Data Types

The following is a list of currently supported data types of the datasets that these HLL functions are compatible with.

Supported:

1. Numeric Types:

TINYINT, INT, BIGINT, FLOAT, DOUBLE

2. String types:

STRING, CHAR, VARCHAR

3. Other types:

BINARY

Unsupported:

1. Complex types

2. Some of the scalar types: DECIMAL, TIMESTAMP, BOOLEAN, SMALLINT, DATE

File Formats

The currently supported file formats are ORC and Parquet. Since Impala does not yet have read support for materialized views, it is required for Hive to write the sketches into regular tables (i.e. not into a materialized view), using the BINARY type. Then Impala can be used to read these sketches and calculate the estimate as Hive does.

Limitations:

You can write HLL sketches into a text table, however it may not work as the serialized sketches can contain characters that are special for text format.

Internal Configuration

The configuration is hard coded and the level of compression is HLL_4 and the number of buckets in the HLL array is k=12 (2 power of 12).

Using HLL Sketches

This datasketch algorithm works in two phases.

- First it creates a sketch from the dataset provided for the algorithm.
- As the second step in the process the function provides an estimate from the sketch.

The following example shows how to create a sketch and to create an estimate from the sketch.

Creating a sketch

This query creates a sketch using the `ds_hll_sketch` function from a column `date_string_col` containing the supported data type.

```
Select ds_hll_sketch(date_string_col) from tableName;
```

This query receives a primitive expression and returns a serialized HLL sketch that is not in a readable format.

Estimating the number of distinct values in a column

The following query returns a cardinality estimate (similarly to `ndv()`) for that particular column (`date_string_col`) by using the HLL function `ds_hll_estimate`.

```
Select ds_hll_estimate(ds_hll_sketch(date_string_col)) from tableName;
```

This query receives a primitive expression (column name) and then creates a sketch from the column, and then passes the sketch to the outer function that gives a cardinality estimate.

```
+-----+
| ds_hll_estimate(ds_hll_sketch(date_string_col)) |
+-----+
| 729 |
+-----+
Fetched 1 row(s) in 0.25s
```

The above result is similar to the results received using the `count(distinct col)` or the `ndv()` approach.

```
Select count(distinct date_string_col) from tableName;
+-----+
| count (distinct date_string_col) |
+-----+
| 730 |
+-----+
```

```
Select ndv(distinct date_string_col) from tableName;
+-----+
| ndv (date_string_col) |
+-----+
| 736 |
+-----+
```

Inserting the derived sketch into a table

You can create a sketch and save it for later use. If you save the sketches to a table or view with the same granularity (e.g. one sketch for each partition in a table) then later you can simply read the sketch and get the estimate almost for free as the real cost here is just the sketch creation.

The following example shows the creation of a sketch for the column (`date_string_col`) from the table “`tableName`” and saving the created sketch into another table called `sketch_table` and later reading the sketch and its estimate from the saved table called `sketch_table`.

```
select year, month, ds_hll_estimate(ds_hll_sketch(date_string_col)) from
tableName group by year, month;
```

year	month	ds_hll_estimate(ds_hll_sketch(date_string_col))
2010	5	31
2010	11	30
2010	6	30
2010	2	28
2010	10	31
2009	11	30
2009	7	31
2009	10	31

2010	11	31
2009	7	31
2010	10	31
2010	8	30
2010	5	31
2009	7	30
2010	4	31
2010	12	30
2009	9	31
2009	8	30
2010	6	28
2010	3	31
2010	4	31
2010	2	30
2009	1	31
2009	12	30

Insert the created sketch into another table:

```
insert into sketch_table select year, month, ds_hll_sketch(date_string_col) from tableName group by year, month;
```

Verify the number of rows modified using count function in the table:

```
select count(1) from sketch_table;
```

You can also verify it by getting the estimates from the sketches stored in the table using the following query:

```
select year, month, ds_hll_estimate(sketch_col) from sketch_table;
```

Combining Sketches

You can combine sketches to allow complex queries to be computed from a few number of simple sketches.

```
select ds_hll_estimate(ds_hll_union(sketch_col)) from sketch_table;
```

Non-deterministic vs Deterministic Results

Based on the characteristics of a query, DataSketches HLL might return results in a non-deterministic manner. During the cache populating phase it returns exact results but in the approximating phase the algorithm can be sensitive to the order of data it receives. As a result you might see the results differ in consecutive runs but all of them will be within the error range of the algorithm.

Due to the characteristics of the algorithm, and to how it was implemented in Impala, the results might be non-deterministic when the query is executed on one node and the input data set is big enough that the algorithm starts approximating instead of giving exact results.

Inter-opreability between Hive and Impala

From the above examples you will notice that the NDV function is much faster than the Datasketches. But NDV is used only by Impala and not by Hive whereas the Datasketches are implemented in Hive as well as in Impala. In

order to maintain the inter-operability between Hive and Impala we recommend to use Datasketches to calculate the estimate. A sketch created by Hive using Datasketches can be fed to Impala to produce an estimate.

Using KLL Datasketch Algorithms in Impala

You can use the stochastic streaming algorithm (KLL) that uses the percentile/quantile functions to statistically analyze the approximate distribution of comparable data from a very large stream.

Use the `ds_kll_quantiles()` or its inverse functions the Probability Mass Function `ds_kll_PMF()` and the Cumulative Distribution Function `ds_kll_CDF()` to obtain the analysis. Query results produced by this KLL algorithm are approximate but within well defined error bounds.

Data Types and File Formats

KLL function currently supports only floats. If you sketch an int data the KLL function converts it into float.

You can sketch data from any file formats. However it is recommended to save the sketches into parquet tables. It is not recommended to save the sketches to text format.

Using KLL Sketches

This datasketch algorithm works in two phases.

- First it creates a sketch from the dataset provided for the algorithm.
- As the second step in the process you can read the estimate from the sketch.

Using this approach you can create granular sketches for a specific dataset (e.g. one sketch per partition) and later use them to provide answers for different quantile queries without reading the actual data as they can also be merged together. Using KLL you can create sketches by Hive and read it with Impala for getting the estimate and vice versa.

Creating a sketch

The following example shows how to create a sketch using the `ds_kll_sketch` function from a column (`float_col`) containing the supported data type. This created sketch is a data structure containing an approximate calculation of some facet of the data and which are compatible regardless of which system reads or writes them.

```
select ds_kll_sketch(float_col) from tableName;
```

where `ds_kll_sketch()` is an aggregate function that receives a float parameter (e.g. a float column of a table) and returns a serialized Apache DataSketches KLL sketch of the input data set wrapped into `STRING` type. Since the serialized data might contain unsupported characters this query may error out. So another approach to this method is to insert the derived sketch into a table or a view and later use for quantile approximations.

Inserting the derived sketch into a table

If you save the sketches to a table or view with the same granularity (e.g. one sketch for each partition in a table) then later you can simply read the sketch for quantile approximation.

```
insert into table_name select ds_kll_sketch(float_col) from tableName;
```

Debugging the created sketch

To troubleshoot the sketch created by the `ds_kll_sketch` function, use `ds_kll_stringify` on the sketch. `ds_kll_stringify()` receives a string that is a serialized Apache DataSketches sketch and returns its stringified format by invoking the related function on the sketch's interface.

```
select ds_kll_stringify(ds_kll_sketch(float_col)) from tableName;
```

```

+-----+
| ds_kll_stringify(ds_kll_sketch(float_col)) |
+-----+
| ### KLL sketch summary:                    |
|      K                                     : 200 |
|      min K                                : 200 |
|      M                                     : 8   |
|      N                                     : 100 |
|      Epsilon                             : 1.33%|
|      Epsilon PMF                         : 1.65%|
|      Empty                               : false|
|      Estimation mode                     : false|
|      Levels                              : 1   |
|      Sorted                              : false|
|      Capacity items                      : 200 |
|      Retained items                      : 100 |
|      Storage bytes                       : 432 |
|      Min value                           : 0   |
|      Max value                           : 9   |
| ### End sketch summary                    |
+-----+

```

Determining the number of datasets sketched

To determine the number of records sketched into this sketch, use `ds_kll_n` function.

```
select ds_kll_n(ds_kll_sketch(float_col)) from tableName;
```

```

+-----+
| ds_kll_n(ds_kll_sketch(float_col)) |
+-----+
| 100                                |
+-----+

```

Calculate Quantile

This function `ds_kll_quantile()` function receives two parameters, a `STRING` parameter that contains a serialized KLL sketch and a `DOUBLE` (0.5) that represents the rank of the quantile in the range of [0,1]. E.g. rank=0.5 means the approximate median of all the sketched data.

```

+-----+
| ds_kll_quantile(ds_kll_sketch(float_col), 0.5) |
+-----+
| 4.0                                           |
+-----+

```

This query returns a data (4.0) that is bigger than the 50% of the data.

Calculating quantiles

To calculate the quantiles for multiple rank parameters, use the function `ds_kll_quantiles_as_string ()` that is very similar to `ds_kll_quantile()` but this function receives any number of rank parameters and returns a comma separated string that holds the results for all of the given ranks.

```
select ds_kll_quantiles_as_string(ds_kll_sketch(float_col), 0.5, 0.6, 1)
from tableName;
```

```
+-----+
| ds_kll_quantiles_as_string(ds_kll_sketch(float_col), 0.5, 0.6, 1) |
+-----+
| 4, 5, 9 |
+-----+
```

Calculate Rank

This rank function `ds_kll_rank()` receives two parameters, a STRING that represents a serialized DataSketches KLL sketch and a float to provide a probing value in the sketch.

```
select ds_kll_rank(ds_kll_sketch(float_col), 4) from tableName;
```

```
+-----+
| ds_kll_rank(ds_kll_sketch(float_col), 4) |
+-----+
| 0.48 |
+-----+
```

This query returns a DOUBLE that is the rank of the given probing value in the range of [0,1]. E.g. a return value of 0.48 means that the probing value given as parameter is greater than 48% of all the values in the sketch.



Note: This is only an approximate calculation.

Calculate Probability Mass Function (PMF)

This Probabilistic Mass Function (PMF) `ds_kll_pmf_as_string()` receives a serialized KLL sketch and one or more float values to represent ranges in the sketched values. In the following example, the float values [1, 3, 4, 8] represent the following ranges: (-inf, 1), [1, 3), [3, 4), [4, 8) [8, +inf)



Note: The input values for the ranges have to be unique and monotonically increasing.

```
select ds_kll_pmf_as_string(ds_kll_sketch(float_col), 1, 3, 4, 8) from table
Name
```

```
+-----+
| ds_kll_pmf_as_string(ds_kll_sketch(float_col), 1, 3, 4, 8) |
+-----+
| 0.12, 0.24, 0.12, 0.36, 0.16 |
+-----+
```

This query returns a comma separated string where each value in the string is a number in the range of [0,1] and shows what percentage of the data is in the particular ranges.

Calculate Cumulative Distribution Function (CDF)

This Cumulative Distribution Function (CDF) `ds_kll_cmf_as_string()` receives a serialized KLL sketch and one or more float values to represent ranges in the sketched values. In the following example, the float values [1, 3, 4, 8] represents the following ranges: (-inf, 1), (-inf, 3), (-inf, 4), (-inf, 8), (-inf, +inf)



Note: The input values for the ranges have to be unique and monotonically increasing.

```
select ds_kll_cdf_as_string(ds_kll_sketch(float_col), 1, 3, 4, 8) from table
Name;
```

```
+-----+
| ds_kll_cdf_as_string(ds_kll_sketch(float_col), 1, 3, 4, 8) |
+-----+
| 0.12, 0.36, 0.48, 0.84, 1 |
+-----+
```

This query returns a comma separated string where each value in the string is a number in the range of [0,1] and shows what percentage of the data is in the particular ranges.

Calculate the Union

To take advantage of the UNION function, create granular sketches for a specific dataset (one sketch per partition), write these sketches to a separate table and based on the partition you are interested in, you can UNION the relevant sketches together to get an estimate.

```
insert into sketch_tbl select ds_kll_sketch(float_col) from tableName;
```

```
select ds_kll_quantile(ds_kll_union(sketch_col), 0.5) from sketch_tbl
where partition_col=1 OR partition_col=5;
```

This function `ds_kll_union()` receives a set of serialized Apache DataSketches KLL sketches produced by `ds_kll_sketch()` and merges them into a single sketch.

Automatic Invalidation/Refresh of Metadata

In this release, you can invalidate or refresh metadata automatically after changes to databases, tables or partitions render metadata stale. You control the synching of tables or database metadata by basing the process on events. You learn how to access metrics and state information about the event processor.

When tools such as Hive and Spark are used to process the raw data ingested into Hive tables, new HMS metadata (database, tables, partitions) and filesystem metadata (new files in existing partitions/tables) are generated. In previous versions of Impala, in order to pick up this new information, Impala users needed to manually issue an `INVALIDATE` or `REFRESH` commands.

When automatic invalidate/refresh of metadata is enabled,, the Catalog Server polls Hive Metastore (HMS) notification events at a configurable interval and automatically applies the changes to Impala catalog.

Impala Catalog Server polls and processes the following changes.

- Invalidates the tables when it receives the `ALTER TABLE` event.
- Refreshes the partition when it receives the `ALTER`, `ADD`, or `DROP` partitions.
- Adds the tables or databases when it receives the `CREATE TABLE` or `CREATE DATABASE` events.

- Removes the tables from catalogd when it receives the DROP TABLE or DROP DATABASE events.
- Refreshes the table and partitions when it receives the INSERT events.

If the table is not loaded at the time of processing the INSERT event, the event processor does not need to refresh the table and skips it.

- Changes the database and updates catalogd when it receives the ALTER DATABASE events. The following changes are supported. This event does not invalidate the tables in the database.
 - Change the database properties
 - Change the comment on the database
 - Change the owner of the database
 - Change the default location of the database

Changing the default location of the database does not move the tables of that database to the new location. Only the new tables which are created subsequently use the default location of the database in case it is not provided in the create table statement.

This feature is controlled by the `##hms_event_polling_interval_s` flag. Start the catalogd with the `##hms_event_polling_interval_s` flag set to a positive integer to enable the feature and set the polling frequency in seconds. We recommend the value to be less than 5 seconds.

The following use cases are not supported:

- When you bypass HMS and add or remove data into table by adding files directly on the filesystem, HMS does not generate the INSERT event, and the event processor will not invalidate the corresponding table or refresh the corresponding partition.

It is recommended that you use the LOAD DATA command to do the data load in such cases, so that event processor can act on the events generated by the LOAD command.

- The Spark API that saves data to a specified location does not generate events in HMS, thus is not supported. For example:

```
Seq((1, 2)).toDF("i", "j").write.save("/user/hive/warehouse/spark_etl.db/
customers/date=01012019")
```

Disable Event Based Automatic Metadata Sync

When the `##hms_event_polling_interval_s` flag is set to a non-zero value for your catalogd, the event-based automatic invalidation is enabled for all databases and tables. If you wish to have the fine-grained control on which tables or databases need to be synced using events, you can use the `impala.disableHmsSync` property to disable the event processing at the table or database level.

This feature can be turned off by setting the `##hms_event_polling_interval_s` flag set to 0.

When you add the DBPROPERTIES or TBLPROPERTIES with the `impala.disableHmsSync` key, the HMS event based sync is turned on or off. The value of the `impala.disableHmsSync` property determines if the event processing needs to be disabled for a particular table or database.

- If `impala.disableHmsSync='true'`, the events for that table or database are ignored and not synced with HMS.
- If `impala.disableHmsSync='false'` or if `impala.disableHmsSync` is not set, the automatic sync with HMS is enabled if the `##hms_event_polling_interval_s` global flag is set to non-zero.
- To disable the event based HMS sync for a new database, set the `impala.disableHmsSync` database properties in Hive as currently, Impala does not support setting database properties:

```
CREATE DATABASE <name> WITH DBPROPERTIES ('impala.disableHmsSync'='true');
```

- To enable or disable the event based HMS sync for a table:

```
CREATE TABLE <name> ... TBLPROPERTIES ('impala.disableHmsSync'='true' |
'false');
```

- To change the event based HMS sync at the table level:

```
ALTER TABLE <name> SET TBLPROPERTIES ('impala.disableHmsSync'='true' | 'false');
```

When both table and database level properties are set, the table level property takes precedence. If the table level property is not set, then the database level property is used to evaluate if the event needs to be processed or not.

If the property is changed from true (meaning events are skipped) to false (meaning events are not skipped), you need to issue a manual `INVALIDATE METADATA` command to reset event processor because it doesn't know how many events have been skipped in the past and cannot know if the object in the event is the latest. In such a case, the status of the event processor changes to `NEEDS_INVALIDATE`.

Metrics for Event Based Automatic Metadata Sync

You can use the web UI of the catalogd to check the state of the automatic invalidate event processor.

By default, the debug web UI of catalogd is at `http://impala-server-hostname:25020` (non-secure cluster) or `https://impala-server-hostname:25020` (secure cluster).

Under the web UI, there are two pages that presents the metrics for HMS event processor that is responsible for the event based automatic metadata sync.

- `/metrics#events`
- `/events`

This provides a detailed view of the metrics of the event processor, including min, max, mean, median, of the durations and rate metrics for all the counters listed on the `/metrics#events` page.

The `/metrics#events` page provides the following metrics about the HMS event processor.

Name	Description
events-processor.avg-events-fetch-duration	Average duration to fetch a batch of events and process it.
events-processor.avg-events-process-duration	Average time taken to process a batch of events received from the Metastore.
events-processor.events-received	Total number of the Metastore events received.
events-processor.events-received-15min-rate	Exponentially weighted moving average (EWMA) of number of events received in last 15 min. This rate of events can be used to determine if there are spikes in event processor activity during certain hours of the day.
events-processor.events-received-1min-rate	Exponentially weighted moving average (EWMA) of number of events received in last 1 min. This rate of events can be used to determine if there are spikes in event processor activity during certain hours of the day.
events-processor.events-received-5min-rate	Exponentially weighted moving average (EWMA) of number of events received in last 5 min. This rate of events can be used to determine if there are spikes in event processor activity during certain hours of the day.
events-processor.events-skipped	Total number of the Metastore events skipped. Events can be skipped based on certain flags are table and database level. You can use this metric to make decisions, such as: <ul style="list-style-type: none"> • If most of the events are being skipped, see if you might just turn off the event processing. • If most of the events are not skipped, see if you need to add flags on certain databases.

Name	Description
events-processor.status	<p>Metastore event processor status to see if there are events being received or not. Possible states are:</p> <ul style="list-style-type: none">• PAUSED The event processor is paused because catalog is being reset concurrently.• ACTIVE The event processor is scheduled at a given frequency.• ERROR The event processor is in error state and event processing has stopped.• NEEDS_INVALIDATE The event processor could not resolve certain events and needs a manual INVALIDATE command to reset the state.• STOPPED The event processing has been shutdown. No events will be processed.• DISABLED The event processor is not configured to run.