

Cloudera Runtime 7.0.0

Managing Apache Hive

Date published: 2019-08-21

Date modified:

CLOUdera

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

Legal Notice

© Cloudera Inc. 2025. 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 operations.....	4
Configure partitions for transactions.....	4
View transactions.....	4
View transaction locks.....	5
 Data compaction.....	 6
Compaction prerequisites.....	6
Enable automatic compaction.....	7
Start compaction manually.....	8
View compaction progress.....	9
Disable automatic compaction.....	9
Compactor properties.....	10
 Query vectorization.....	 12
Configuring query vectorization.....	13
Check query execution.....	14

ACID operations

You can perform ACID (atomicity, consistency, isolation, and durability) v2 transactions at the row level without any configuration. By default, managed tables are ACID tables. You cannot disable transactions.

The Data Warehouse service in CDP Public Cloud includes mature versions of Hive ACID transaction processing and low latency analytical processing (LLAP). LLAP is not available in CDP Private Cloud Base. Application development and operations are simplified with strong transactional guarantees and simple semantics for SQL commands. You do not need to bucket ACID v2 tables, so maintenance is easier. With improvements in transactional semantics, advanced optimizations, such as materialized view rewrites and automatic query cache, are available. With these optimizations, you can deploy new Hive application types.

A Hive operation is atomic. The operation either succeeds completely or fails; it does not result in partial data. A Hive operation is also consistent: After an application performs an operation, the results are visible to the application in every subsequent operation. Hive operations are isolated. Your operations do not cause unexpected side effects for other users. Finally, a Hive operation is durable. A completed operation is preserved in the event of a failure.

Hive operations are atomic at the row level instead of the table or partition level. A Hive client can read from a partition at the same time another client adds rows to the partition. Transaction streaming rapidly inserts data into Hive tables and partitions.

Configure partitions for transactions

To include INSERT, UPDATE, and DELETE statements in your transaction applications, you must set your dynamic partition mode to nonstrict.

Before you begin

Change the following parameters to meet your needs:

- `hive.exec.max.dynamic.partitions`
- `hive.exec.max.dynamic.partitions.pernode`

Procedure

1. In Cloudera Manager Clusters select the Hive service. Click Configuration, and search for `hive-site.xml`.
2. In HiveServer2 Advanced Configuration Snippet (Safety Valve) for `hive-site.xml`, click + and add the `hive.exec.dynamic.partition.mode` property.
3. Set the value to `nonstrict`.
4. Save the changes and restart the Hive service.

Related Information

[Hive Configuration Properties documentation on the Apache wiki](#)

View transactions

As Administrator, you can view a list of open and aborted transactions.

Procedure

Enter a query to view transactions.

```
SHOW TRANSACTIONS
```

The following information appears in the output:

- Transaction ID

- Transaction state
- Hive user who initiated the transaction
- Host machine or virtual machine where transaction was initiated

View transaction locks

As a Hive administrator, you can get troubleshooting information about locks on a table, partition, or schema.

About this task

Hive transactions, enabled by default, disables Zookeeper locking. DbLockManager stores and manages all transaction lock information in the Hive Metastore. Heartbeats are sent regularly from lock holders and transaction initiators to the Hive Metastore to prevent stale locks and transactions. The lock or transaction is aborted if the metastore does not receive a heartbeat within the amount of time specified by the `hive.txn.timeout` configuration property.

Before you begin

Check that transactions are enabled (the default).

Procedure

1. Enter a Hive query to check table locks.

```
SHOW LOCKS mytable EXTENDED;
```

2. Check partition locks.

```
SHOW LOCKS mytable PARTITION(ds='2018-05-01', hr='12') EXTENDED;
```

3. Check schema locks.

```
SHOW LOCKS SCHEMA mydatabase;
```

The following information appears in the output unless ZooKeeper or in-memory lock managers are used.

- Database name
- Table name
- Partition, if the table is partitioned
- Lock state:
 - Acquired - transaction initiator hold the lock
 - Waiting - transaction initiator is waiting for the lock
 - Aborted - the lock has timed out but has not yet been cleaned
- Lock type:
 - Exclusive - the lock cannot be shared
 - Shared_read - the lock cannot be shared with any number of other shared_read locks
 - Shared_write - the lock may be shared by any number of other shared_read locks but not with other shared_write locks
- Transaction ID associated with the lock, if one exists
- Last time lock holder sent a heartbeat
- Time the lock was acquired, if it has been acquired
- Hive user who requested the lock
- Host machine or virtual machine on which the Hive user is running a Hive client
- Blocked By ID - ID of the lock causing current lock to be in Waiting mode, if the lock is in this mode

Related Information

[Apache wiki transaction configuration documentation](#)

Data compaction

As administrator, you need to manage compaction of delta files that accumulate during data ingestion. Compaction is a process that performs critical housekeeping 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. By default, Hive automatically compacts delta and base files at regular intervals. Compaction is a consolidation of files. You can configure automatic compactions, as well as perform manual compactions of base and delta files. To submit compaction Jobs, Hive uses Tez as the execution engine, and uses MapReduce algorithms in the Stack. Compactions occur in the background without affecting concurrent reads and writes. The compactor initiator should run on only one HMS instance.

There are two types of compaction:

- **Minor**
Rewrites a set of delta files to a single delta file for a bucket.
- **Major**
Rewrites one or more delta files and the base file as a new base file for a bucket.

Related Information

[Apache Wiki transactions and compaction documentation](#)

Compaction prerequisites

To prevent data loss or an unsuccessful compaction, you must meet the prerequisites before compaction occurs.

Exclude compaction users from Ranger policies

Compaction causes data loss if Apache Ranger policies for masking or row filtering are enabled and the user hive or any other compaction user is included in the Ranger policies.

1. Set up Ranger masking or row filtering policies to exclude the user hive from the policies.

The user (named hive) appears in the Users list in the Ranger Admin UI.

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users
7	all - hiveservice	--	Enabled	Enabled	--	--	hive rangerlookup impala

2. Identify any other compaction users from the masking or row filtering policies for tables as follows:
 - If the `hive.compaction.run.as.user` property is configured, the user runs compaction.
 - If a user is configured as owner of the directory on which the compaction will run, the user runs compaction.
 - If a user is configured as the table owner, the user runs compaction
3. Exclude compaction users from the masking or row filtering policies for tables.

Failure to perform these critical steps can cause data loss. For example, if a compaction user is included in an enabled Ranger masking policy, the user sees only the masked data, just like other users who are subject to the Ranger masking policy. The unmasked data is overwritten during compaction, which leads to data loss of unmasked content as only the underlying tables will contain masked data. Similarly, if Ranger row filtering is enabled, you do not see, or have access to, the filtered rows, and data is lost after compaction from the underlying tables.

The worker process executes queries to perform compaction, making Hive data subject to data loss ([HIVE-27643](#)). MapReduce-based compactons are not subject to the data loss described above as these compactons directly use the MapReduce framework.

Related Information

[Row-level filtering and column masking in Hive with Ranger policies](#)

Enable automatic compaction

Several properties in the Hive and Hive metastore service configurations must be set to enable automatic compaction. You need to check that the property settings are correct and to add one of the properties to the Hive on Tez service. Automatic compaction will then occur at regular intervals, but only if necessary.

About this task

The following properties must be set in Hive metastore (Hive-1) and Hive on Tez services as follows:

- `hive.compactor.initiator.on` = true (default)
- `hive.compactor.worker.threads` = <a value greater than 0> (default and recommended value = 5)
- `hive.metastore.runworker.in` = hs2 (default)

Before you begin

Tables or partitions you are compacting must be full ACID or insert-only ACID tables.

Procedure

1. In Cloudera Manager, select the Hive metastore service: Clusters Hive-1 Configuration .
2. Search for compact.

The screenshot shows the Cloudera Manager interface for the HIVE-1 service configuration. The 'Configuration' tab is active, and a search bar at the top contains the text 'compact'. On the left, there are two filter sections: 'SCOPE' and 'CATEGORY'. The 'SCOPE' section lists 'HIVE-1 (Service-Wide)' with a count of 0, 'Gateway' with 0, 'Hive Metastore Server' with 3, 'HiveServer2' with 0, and 'WebHCat Server' with 0. The 'CATEGORY' section lists 'Advanced' with 3 and 'Cloudera Navigator' with 0. The main configuration area on the right shows three settings: 'Turn on compactor initiator thread.' which is checked and associated with 'Hive Metastore Server Default Group'; 'Number of Threads Used by Compactor' which is set to 5 and associated with 'Hive Metastore Server Default Group'; and 'Run compactor on Hive Metastore or HiveServer2.' which is set to 'hs2' (selected with a radio button) and associated with 'Hive Metastore Server Default Group'.

3. Check that Turn on Compactor Initiator Thread (`hive.compactor.initiator.on`), Number of Threads Used by Compactor (`hive.compactor.worker.threads`), and Run Compactor on Hive Metastore or HiveServer2 (`hive.metastore.runworker.in`) are set to the values shown above.
4. Save the changes.
5. In Cloudera Manager, select the Hive metastore service: Clusters HIVE_ON_TEZ-1 Configuration .

6. Search for compact.

The screenshot shows the Cloudera Manager configuration interface for the service HIVE_ON_TEZ-1. The 'Configuration' tab is active. A search bar at the top contains the text 'compact'. On the left, a 'Filters' sidebar is visible, showing 'SCOPE' with 'HIVE_ON_TEZ-1 (Service-Wide)' selected, and 'CATEGORY' with 'HiveServer2' selected. The main content area displays two configuration items: 'Number of Threads Used by Compactor' (hive.compactor.worker.threads) with a value of 5, and 'Run compactor on Hive Metastore or HiveServer2' (hive.metastore.runworker.in) with a radio button selection for 'hs2'.

7. Check that the Number of Threads Used by Compactor (hive.compactor.worker.threads), and Run compactor on Hive Metastore or HiveServer2 (hive.metastore.runworker.in) is set to hs2.
8. Save the changes and restart the Hive on Tez and Hive (HIVE-1) metastore services at an appropriate time.

Related Information

[Example of using the Cloudera Manager Safety Valve](#)

Start compaction manually

You manually start compaction when automatic compaction fails for some reason. You can start compaction by running a Hive statement.

About this task

Carefully consider the need for a major compaction as this process can consume significant system resources and take a long time. Start a major compaction during periods of low traffic. Base and delta files for a table or partition are compacted.

Start compaction using a query

You use the following syntax to issue a query that starts compaction:

```
ALTER TABLE tablename [PARTITION (partition_key='partition_value' [...])]
COMPACT 'compaction_type'
```

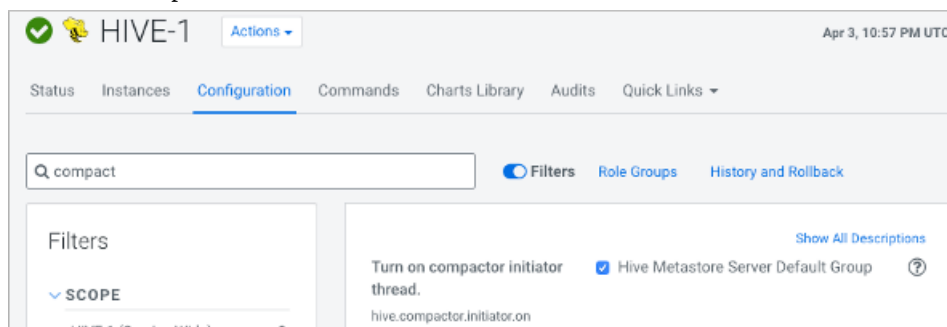
Before you begin

Tables or partitions you are compacting must be full ACID or insert-only ACID tables.

Procedure

1. In Cloudera Manager, select the Hive metastore service: Clusters Hive-1 Configuration .

2. Search for compact.



3. Check that the Hive Metastore Server Default Group is selected (hive.compactor.initiator.on=true).

4. Execute a query to start a major compaction of a table.

```
ALTER TABLE mytable COMPACT 'major'
```

ALTER TABLE compacts tables even if the NO_AUTO_COMPACTION table property is set.

Related Information

[Example of using the Cloudera Manager Safety Valve](#)

View compaction progress

You view the progress of compactons by running a Hive query.

Procedure

Enter the query to view the progress of compactons.

```
SHOW COMPACTIONS;
```

- Unique internal ID
- Database name
- Table name
- Partition name
- Major or minor compaction
- Compaction state:
 - Initiated - waiting in queue
 - Working - currently compacting
 - Ready for cleaning - compaction completed and old files scheduled for removal
 - Failed - the job failed. Details are printed to the metastore log.
 - Succeeded
 - Attempted - initiator attempted to schedule a compaction but failed. Details are printed to the metastore log.
- Thread ID
- Start time of compaction
- Duration
- Job ID - ID of the submitted MapReduce job

Disable automatic compaction

You can disable automatic compaction of a particular Hive table by setting the NO_AUTO_COMPACTION property in TBLPROPERTIES of a table. Disabling automatic compaction does not prevent you from performing manual compaction.

Procedure

Start the Hive shell, and in the database of the target table, alter the TBLPROPERTIES.

```
ALTER TABLE my_t SET TBLPROPERTIES ( 'NO_AUTO_COMPACTION'='true' );
```

Compactor properties

You check and change a number of Apache Hive properties to configure the compaction of delta files that accumulate during data ingestion. You need to know the defaults, valid values, and where to set these properties: Cloudera Manager, TBLPROPERTIES, hive-site.xml, or core-site.xml. When properties do not appear in Cloudera Manager search of configuration properties for a runtime service, you add the property to hive-site or core-site using the Cloudera Manager Safety Valve.

Basic compactor properties

hive.compactor.initiator.on

Default=false

Whether to run the initiator and cleaner threads on this metastore instance or not.

hive.compactor.worker.threads

Default=0

Set this to a positive number to enable Hive transactions, which are required to trigger transactions. Worker threads spawn jobs to perform compactions, but do not perform the compactions themselves. Increasing the number of worker threads decreases the time that it takes tables or partitions to be compacted. However, increasing the number of worker threads also increases the background load on the CDP cluster because they cause more jobs to run in the background.

hive.metastore.runworker.in

Default=HS2

Specifies where to run the Worker threads that spawn jobs to perform compactions. Valid values are HiveServer (HS2) or Hive metastore (HMS).

hive.compactor.abortedtxn.threshold

Default=1000

The number of aborted transactions involving a given table or partition that will trigger a major compaction.

Advanced compactor properties

hive.compactor.worker.timeout

Default=86400s

Expects a time value using d/day, h/hour, m/min, s/sec, ms/msec, us/usec, or ns/nsec (default is sec). Time in seconds after which a compaction job will be declared failed and the compaction re-queued.

hive.compactor.check.interval

Default=300s

A valid value is a time with unit (d/day, h/hour, m/min, s/sec, ms/msec, us/usec, ns/nsec), which is sec if not specified.

Time in seconds between checks to see if any tables or partitions need to be compacted. This value should be kept high because each check for compaction requires many calls against the NameNode. Decreasing this value reduces the time it takes to start compaction for a table or partition that requires it. However, checking if compaction is needed requires several calls to the NameNode for

each table or partition involved in a transaction done since the last major compaction. Consequently, decreasing this value increases the load on the NameNode.

hive.compactor.delta.num.threshold

Default=10

Number of delta directories in a table or partition that triggers a minor compaction.

hive.compactor.delta.pct.threshold

Default=0.1

Percentage (fractional) size of the delta files relative to the base that triggers a major compaction. (1.0 = 100%, so the default 0.1 = 10%.)

hive.compactor.max.num.delta

Default=500

Maximum number of delta files that the compactor attempts to handle in a single job.

hive.compactor.wait.timeout

Default=300000

The value must be greater than 2000 milliseconds.

Time out in milliseconds for blocking compaction.

hive.compactor.initiator.failed.compacts.threshold

Default=2

A valid value is between 1 and 20, and must be less than `hive.compactor.history.retention.failed`.

The number of consecutive compaction failures (per table/partition) after which automatic compactions are not scheduled any longer.

hive.compactor.cleaner.run.interval

Default=5000ms

A valid value is a time with unit (d/day, h/hour, m/min, s/sec, ms/msec, us/usec, ns/nsec), which is msec if not specified.

The time between runs of the cleaner thread.

hive.compactor.job.queue

Specifies the Hadoop queue name to which compaction jobs are submitted. If the value is an empty string, Hadoop chooses the default queue to submit compaction jobs.

Providing an invalid queue name results in compaction job failures.

hive.compactor.compact.insert.only

Default=true

The compactor compacts insert-only tables, or not (false). A safety switch.

hive.compactor.crud.query.based

Default=false

Performs major compaction on full CRUD tables as a query, and disables minor compaction.

hive.split.grouping.mode

Default=query

A valid value is either query or compactor.

This property is set to compactor from within the query-based compactor. This setting enables the Tez SplitGrouper to group splits based on their bucket number, so that all rows from different bucket files for the same bucket number can end up in the same bucket file after the compaction.

hive.compactor.history.retention.succeeded

Default=3

A valid value is between 0 and 100.

Determines how many successful compaction records are retained in compaction history for a given table/partition.

hive.compactor.history.retention.failed

Default=3

A valid value is between 0 and 100.

Determines how many failed compaction records are retained in compaction history for a given table/partition.

hive.compactor.history.retention.attempted

Default=2

A valid value is between 0 and 100.

Determines how many attempted compaction records are retained in compaction history for a given table/partition.

hive.compactor.history.reaper.interval

Default=2m

A valid value is a time with unit (d/day, h/hour, m/min, s/sec, ms/msec, us/usec, ns/nsec), which is msec if not specified.

Determines how often compaction history reaper runs.

Related Information

[Example of using the Cloudera Manager Safety Valve](#)

Query vectorization

You can use vectorization to improve instruction pipelines and cache use. Vectorization enables certain data and queries to process batches of primitive types on entire column rather than one row at a time.

Default vectorized query execution

CDP enables vectorization by default and the value of `hive.vectorized.execution.enabled` is set to `true`. Vectorized query execution processes Hive data in batch, channeling a large number of rows of data into columns, foregoing intermediate results. This technique is more efficient than the MapReduce execution process that stores temporary file.

Unsupported functionality on vectorized data

Some functionality is not supported on vectorized data:

- DDL queries
- DML queries other than single table, read-only queries
- Formats other than Optimized Row Columnar (ORC)

Supported functionality on vectorized data

The following functionality is supported on vectorized data:

- Single table, read-only queries
 - Selecting, filtering, and grouping data is supported.
- Partitioned tables

- The following expressions:
 - Comparison: >, >=, <, <=, =, !=
 - Arithmetic plus, minus, multiply, divide, and modulo
 - Logical AND and OR
 - Aggregates sum, avg, count, min, and max

Supported data types

You can query data of the following types using vectorized queries:

- tinyint
- smallint
- int
- bigint
- date
- boolean
- float
- double
- timestamp
- stringchar
- varchar
- binary

Configuring query vectorization

You can manage query vectorization by setting HiveServer properties in Cloudera Manager.

Vectorization properties

hive.vectorized.groupby.checkinterval

In vectorized group-by, the number of row entries added to the hash table before re-checking average variable size for memory usage estimation.

hive.vectorized.groupby.flush.percent

Ratio between 0.0 and 1.0 of entries in the vectorized group-by aggregation hash that is flushed when the memory threshold is exceeded.

hive.vectorized.execution.enabled

Enable optimization that vectorizes query execution by streamlining operations by processing a block of 1024 rows at a time.

hive.vectorized.execution.reduce.enabled

Whether to vectorize the reduce side of query execution.

hive.vectorized.use.vectorized.input.format

If enabled, Hive uses the native vectorized input format for vectorized query execution when it is available.

hive.vectorized.use.checked.expressions

To enhance performance, vectorized expressions operate using wide data types like long and double. When wide data types are used, numeric overflows can occur during expression evaluation in a different manner for vectorized expressions than they do for non-vectorized expressions. Consequently, different query results can be returned for vectorized expressions compared to results returned for non-vectorized expressions. When this configuration is enabled, Hive uses vectorized expressions that handle numeric overflows in the same way as non-vectorized expressions are handled.

hive.vectorized.adaptor.usage.mode

Vectorized Adaptor Usage Mode specifies the extent to which the vectorization engine tries to vectorize UDFs that do not have native vectorized versions available. Selecting the "none" option specifies that only queries using native vectorized UDFs are vectorized. Selecting the "chosen" option specifies that Hive chooses to vectorize a subset of the UDFs based on performance benefits using the Vectorized Adaptor. Selecting the "all" option specifies that the Vectorized Adaptor be used for all UDFs even when native vectorized versions are not available.

hive.vectorized.use.vector.serde.deserialize

If enabled, Hive uses built-in vector SerDes to process text and sequencefile tables for vectorized query execution.

hive.vectorized.input.format.excludes

Specifies a list of file input format classnames to exclude from vectorized query execution using the vectorized input format. Note that vectorized execution can still occur for an excluded input format based on whether row SerDes or vector SerDes are enabled.

Check query execution

You can determine if query vectorization occurred during execution by running the EXPLAIN VECTORIZATION query statement.

Procedure

1. Start Hive from Beeline.

```
$ hive
```

2. Set hive.explain.user to false to see vector values.

```
SET hive.explain.user=false;
```

3. Run the EXPLAIN VECTORIZATION statement on the query you want CDP to process using vectorization.

```
EXPLAIN VECTORIZATION SELECT COUNT(*) FROM employees where emp_no>10;
```

The following output confirms that vectorization occurs:

```
+-----+
|                               |
|                               | Explain                               |
|                               |                               |
+-----+-----+
| Plan optimized by CBO.      |
|                               |
| Vertex dependency in root stage |
| Reducer 2 <- Map 1 [CUSTOM_SIMPLE_EDGE] *vectorized* |
|   File Output Operator [FS_14] |
|   Group By Operator [GBY_13] (rows=1 width=12) |
|   Output:["_col0"],aggregations:["count(VALUE._col0)"] |
|   <-Map 1 [CUSTOM_SIMPLE_EDGE] vectorized |
|   PARTITION_ONLY_SHUFFLE [RS_12] |
|   Group By Operator [GBY_11] (rows=1 width=12) |
|   Output:["_col0"],aggregations:["count()"] |
|   Select Operator [SEL_10] (rows=1 width=4) |
|   Filter Operator [FIL_9] (rows=1 width=4) |
|   predicate:(emp_no > 10) |
|   TableScan [TS_0] (rows=1 width=4) |
|   default@employees,employees,Tbl:COMPLETE,Col:NONE,Outpu |
| tput:["emp_no"] |
|                               |
```

```
+-----+  
23 rows selected (1.542 seconds)
```