RapidsDB

RapidsDB Installation and Management Guide

Release 4.2.3

Version 2

Table of Contents

1 Overview

1.1 Changes

1.1.1 Changes from 4.2.3

- Updated the section on ssh setup (section 5.5) to deal with private key files that are not in the expected format
- Added section 6.3.3, “Trial License Limitations”, which documents the limitations when using a trial license
- Removed “url” option for Generic JDBC Connector (section 7.10.10) and replaced with “connectionstring”. Support for “url” will be added in a future release

1.1.2 Changes from 4.2.2

- This is the GA release. There are some documentation updates but no new features.

1.1.3 Changes from 4.2.1

- The Hadoop Connector now supports the DATE datatype
- The Hadoop Connector now supports the Hive FLOAT and DOUBLE datatypes

1.1.4 Changes from 4.2

- The user can now create multiple MOXE Connectors and thereby have multiple MOXE schemas in the same RapidsDB Cluster.
- A MOXE Connector can now be configured to run on a subset of the nodes in a RapidsDB Cluster
- New options have been added when creating any of the following Connectors:
    - MySQL
    - MemSQL
    - Oracle
    - Postgres
    - JDBC
- The behavior of the REFRESH command has changed so that a refresh will automatically get invoked in the background whenever any of the following activities occur:
    - Startup of the RapidsDB Cluster
    - CREATE or DROP table requests
    - CREATE CONNECTOR
- RapidsDB now supports Audit Logging which provides the ability to dynamically record what a user, or set of users, are doing within the RapidsDB cluster so that the actions can be audited for security purposes at a later stage.

1.1.5 Changes from 4.1

- The RapidsSE Connector is not supported for this release
- The DATE datatype is now supported
- A new MySQL Connector has been added
- The Hadoop Connector no longer supports the SCHEMA_METADATA option

- The user can cancel a running query from any of the supported interfaces
- Two new system metadata tables, QUERIES and QUERY_STATS, have been added to support the tracking of active queries
- Added support for the auditing of user commands

1.1.6 Changes from 4.03

- Licensing has been enabled
- User authentication has been enabled

1.1.7 Changes from Release 4.0.1

- The Hadoop Connector supports two new options, NAMESERVICE and NAMENODES, for use with Hadoop HA configurations.

1.1.8 Changes from Release 3.6

- Licensing has been added to this release but has not been activated for this initial release
- User authentication has been disabled for the initial R4.0 release and will be re-enabled in a R4.0 update
- Addition of a new, internal, integrated memory storage engine for RapidsDB named MOXE
- The Hadoop Connector now supports accessing delimited tables from the Hive metastore
- Changes to the Hadoop Connector in the way that HDFS files are referenced
- The cluster.config and zk.config files are no longer located in the cfg installation directory, they are now located in the parent installation directory
- Removed support for VoltDB
- The internal communication fabric has been changed from using JeroMQ to an internally developed socket-based fabric named the Plex
- The RapidsSE and Stream connectors are not available for this release, and will be supported in a release update

1.1.9 Changes from Release 3.5

- RapidsDB now requires users to be authenticated when connecting to the system.
- Added CREATE/DROP/ALTER commands for USERS and AUTHENTICATORS.
- RapidsDB supports user authentication via passwords and Kerberos.
- Added metadata tables to manage users and authenticators.
- Updated JDBC driver to support user authentication.
- A custom SSH port number can now be specified in the cluster configuration.
- The Hadoop Connector now supports the reading and writing of Parquet files.
- The Hadoop Connector now supports reading the schema metadata for Parquet tables from the Hive metastore.
- The Hadoop Connector now supports, CREATE, DROP and TRUNCATE table.
- The Hadoop Connector now supports a new option, READONLY, which controls whether tables can be changed via an INSERT, TRUNCATE or DROP command.

1.1.10 Changes from Release 3.4.2

- The Hadoop Connector now supports Kerberos authentication
- The Hadoop Connector now supports the option to specify the character set encoding for HDFS files
- The Hadoop Connector now supports an option for setting the user name to be used when accessing HDFS
- The catalog name for the schema associated with a Hadoop Connector is now the Connector name and not “HADOOP”

1.1.11 Changes from Release 3.4.1

- Some corrections to examples

1.1.12 Changes from Release 3.4

The following lists the major changes from Release 3.4:

- The Hadoop Connector now supports writing data to HDFS files, using either an INSERT or INSERT … SELECT statement.
- Support has been added for TRUNCATE TABLE for RapidsSE.

1.1.13 Changes from Release 3.3.2

The following lists the major changes from Release 3.3.2:

- The MemSQL, Oracle, Postgres and Generic JDBC Connectors now support the SCHEMA_METADATA option which allows the user to dynamically update the set of tables being accessed by the Connector
- The user can now refresh the metadata for a single Connector using the “refresh connector” command
- The handling of the following options has been changed in the RapidsSE Loader and the Hadoop Connector:
    - ENCLOSED_BY:
        - The default has been changed to no enclosing characters
        - All data types can now be optionally enclosed
        - ENCLOSED_BY has been extended to allow for two characters, where the first character will define the start of the enclosing, and the second character will define the end of the enclosing
    - DATE_FORMAT – this is a new option to configure how the date portion of timestamps are formatted
    - The handling of NULL values has been updated
- CREATE …. AS SELECT – support has been added to allow the user to automatically create a table based on the result set of a query and to have that result set inserted into the table
- The performance of RapidsSE has been significantly improved for the both indexed and non-indexed access

- The performance of INSERT … SELECT has been improved for the MemSQL, Oracle, Postgres and Generic JDBC Connectors
- A new version of the rapids-shell is now supported that uses the RapidsDB JDBC Driver for communicating with the RapidsDB Cluster. This new version can also be installed on any platform that supports Java

1.1.14 Changes from Release 3.3

The following lists the major changes from Release 3.3:

- Support for the following options when loading delimited data into RapidsSE:
  - DELIMITER
  - TERMINATOR
  - ENCLOSED_BY
  - ESCAPE_CHAR
- Support for the following options for the Hadoop Connector:
  - DELIMITER
  - TERMINATOR
  - ENCLOSED_BY
  - ESCAPE_CHAR
  - IGNORE_HEADER
- Support for Hive-style partitioning in the Hadoop Connector
- Changed the name for Hadoop csv Connector to Hadoop Connector to reflect the fact that the Connector will support other formats in the future.
- Added support for disabling and enabling Connectors

1.1.15 Changes from Release 3.1

The following list summarizes the major changes from Release 3.1:

- Addition of Connectors for Greenplum, Oracle and Postgres (refer to Sections 3.2 and 6.5 for more details)
- JDBC Connector – the JDBC Connector is a new Connector that supports access to data sources that provide a JDBC Driver (refer to Sections 3.3 and 6.5 for more details).
- Case-insensitive naming – RapidsDB will now support case-insensitive naming across all of the Connectors (refer to Section 2 for more details).
- VIEWS supported – the Connectors for MemSQL, Greenplum, Oracle, Postgres, VoltDB, and the JDBC Connector, will include views along with tables as part of the metadata collected from the source system.
- RapidsSE:
  - Support for multi-column partitioning keys (refer to Section 6.4.5 for more details)
  - Support for multi-column, non-unique indexes (refer to Section 6.4.6 for more details)
  - Support for up to 8 indexes per table

    - Support for up to 8 columns per index
    - Support for range queries (lower and upper bound). The RapidsDB Execution Engine will now be able to request a range of column values from a RapidsSE table instead of having to request all of the rows and then filter the data in the RapidsDB Execution Engine. RapidsSE will make use of an index to satisfy the range query when there is an index defined on the column referenced in the range query (refer to section 3.5 for more details).
    - Support for exact key queries. This is a special case of range queries where only a single value is being requested from the RapidsDB Execution Engine (refer to section 3.5 for more details).
    - Column validation. RapidsSE will check that the data for each column matches the data type and precision specified for that column.
    - Support for LOAD and EXPORT commands from the rapids-shell or JDBC as ESCAPE commands (refer to Sections 6.4.8 and 6.4.9 for more details)
- Hadoop csv Connector – added the following new features:
    - Multi-file Support – The user can use wildcard characters in the file name component for the file location such that when multiple files match the wildcard specification, all of the matching files will be accessed as a single table (refer to section 6.5.12.3 for more details).
    - Dynamic Schema Metadata – Allows the user to dynamically change the schema being used by the Connector so that new tables can be defined, existing table definitions updated, or existing table definitions can be removed, without requiring the user to drop and recreate the Connector with the new schema (refer to section 6.5.12.4 for more details).
- ssh – updated section 5.6 that describes how to configure ssh.
- Index metadata – Connectors will now return information about indexes if they are supported by the underlying data store, such as MemSQL

1.2 What is RapidsDB?

RapidsDB is a fully parallel, distributed, in-memory federated query system that is designed to support complex analytical SQL queries running against a set of different data stores. Figure 1 below shows the major components of the RapidsDB system:

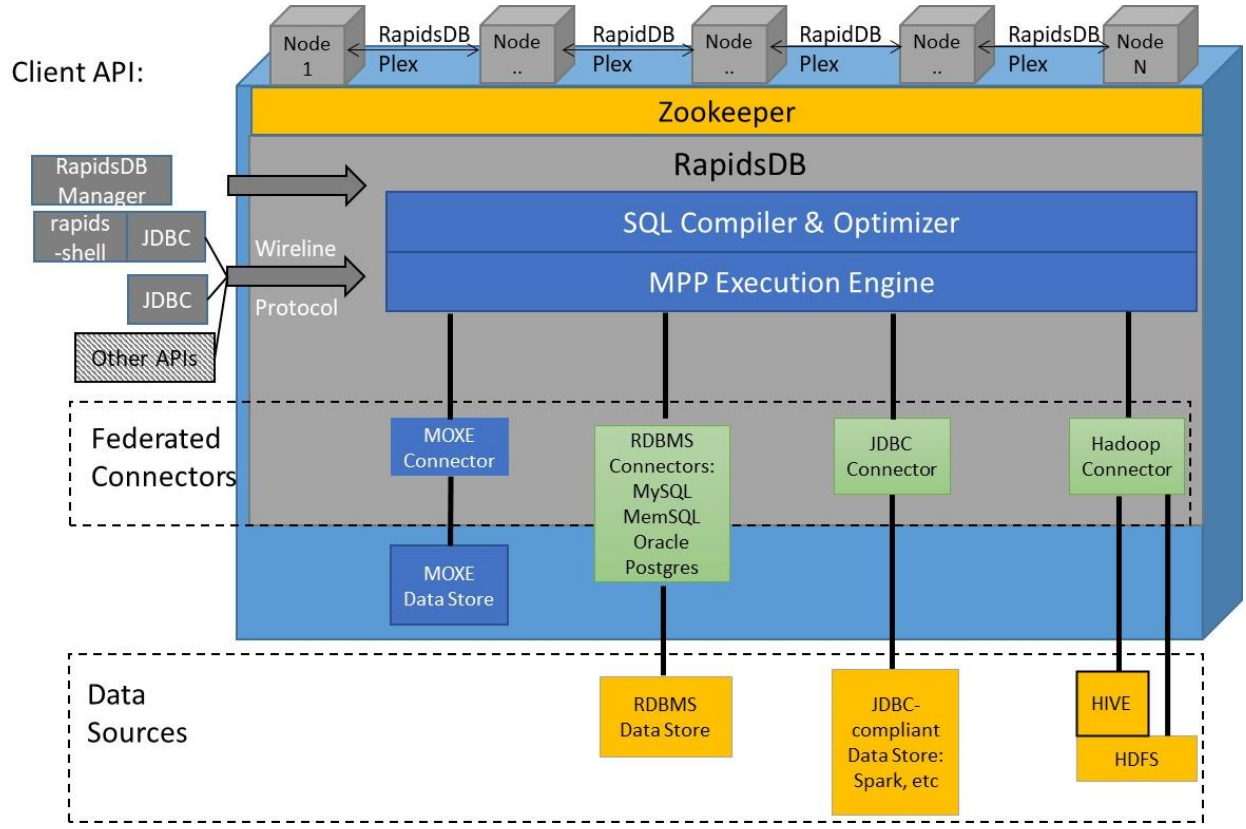

Figure 1. RapidsDB Architecture

RapidsDB provides unified SQL access to a wide variety of data sources, which can include relational and non-relational data sources. Data can be joined across all of the data sources. For this release, RapidsDB supports access to the following data sources:

- MOXE – internal in-memory data store for RapidsDB
- The following RDBMS data sources:
  - MySQL
  - MemSQL
  - Oracle
  - Postgres
  - Greenplum
- Any data source that supports access via JDBC not including the RDBMS data sources above, for example, Spark
- Hadoop/HDFS delimited, ORC and Parquet files either through the Hive Metastore or directly against the files in HDFS

1.3 RapidsDB Components

1.3.1 RapidsDB Plex

The RapidsDB Plex is the internal communication fabric that is used between nodes in a RapidsDB cluster.

1.3.2 SQL Compiler and Optimizer

RapidsDB has an advanced, cost-based, SQL Compiler & Optimizer that is responsible for taking a user's SQL query and building the optimum query execution plan for that query. The query plan will then get passed to the MPP Execution Engine for execution of the query.

1.3.3 MPP Execution Engine

RapidsDB has its own fully parallel, MPP Execution Engine that is responsible for execution of the query plan generated by the RapidsDB SQL Compiler & Optimizer. The MPP Execution Engine is responsible for the execution of the query plans in concert with the Federated Connectors that provide the access to the underlying data sources. The Execution Engine is responsible for executing those parts of the query execution plan that cannot get pushed down to the underlying data source (see Federated Connectors below for details on query pushdown), and for delivering the final results of the query to user. For example, if the query involves a JOIN between a MOXE table and a Hive table, then the Execution Engine will perform the JOIN by getting the data from the MOXE and Hive tables using the MOXE and Hive Connectors.

1.3.4 Federated Connectors

The RapidsDB Federated Connectors are a set of dynamically, pluggable Connectors that control access to the underlying data stores that make up the federated database. The Connectors manage the metadata for the objects (typically tables or files) in the remote data store and present that metadata to the RapidsDB query execution engine as an ANSI-based SQL schema, thereby allowing the user to view the objects from the entire set of data sources as a single, federated SQL database.

The Connectors are responsible for managing data type conversion between the native data store and the RapidsDB query system which allows for the uniform handling of data types across all of the data stores. The Connectors present the data to the RapidsDB Execution Engine as rows and columns, which allows the data to be queried using standard ANSI SQL, and provides a uniform query interface regardless of the data source.

In order to optimize performance we generally want to have the underlying data source perform as much of the query as possible to reduce the amount of data that has to be processed by the RapidsDB Execution Engine, rather than simply pulling all of the data from the data source and processing it within RapidsDB. For example, when the data source is a relational database such as Oracle, that data source is typically capable of executing a join on the tables in that data source, or it can filter the data based on predicates in the query.

RapidsDB supports an optimization feature called "Adaptive Query Pushdown" to deal with this. With Adaptive Query Pushdown, each Connector involved in the execution of a query plan analyzes the query

plan and decides which parts of the query plan it can push down to the data source or and for the remaining parts of the query the Connector will determine the optimum way to retrieve the data required to complete those parts of the query plan.

1.3.5 MOXE

MOXE (in-Memory Operational eXtreme Engine) is a parallel, fully distributed, internal in-memory data store that is co-resident with RapidsDB, sharing the same process space as the other RapidsDB components. MOXE uses hash partitioning to distribute the data across multiple partitions, and each partition operates in parallel when delivering data to the Execution Engine. The system can support multiple MOXE Connectors, each of which manages a single schema.

1.3.6 Client API

RapidsDB provides the following interfaces for accessing RapidsDB:

1.3.6.1 RapidsDB Manager

The RapidsDB Manager is a web-based management console for configuring and managing the RapidsDB cluster.

1.3.6.2 rapids-shell

The rapids-shell is a command line interface for configuring connectors and submitting queries. The rapids-shell uses the RapidsDB Unified JDBC Driver to communicate with the RapidsDB Cluster. Refer to the Rapids-shell User Guide for more information.

1.3.6.3 JBDC

RapidsDB supports the JDBC programmatic interface via the RapidsDB Unified JDBC Driver. The Unified RapidsDB JDBC Driver uses the RapidsDB Wireline Protocol (see below). Refer to the Unified JDBC Driver manual for more information.

1.3.6.4 Wireline Protocol

The RapidsDB Wireline Protocol is a platform-independent, Thrift-based, protocol that can be used for programmatically accessing RapidsDB from many different programming languages. It is a specification of the messages that flow between the client and server and sequencing of those messages. The Thrift-based API is a low-level interface that enables higher-level APIs to be developed to support almost any programming language. Refer to the RapidsDB Wireline Protocol Specification for more information.

1.3.7 Zookeeper

RapidsDB uses Zookeeper (version 3.4.6 or later) for configuration management across the RapidsDB cluster.

1.4 RapidsDB Cluster Topology

Figure 2 below shows the topology of a RapidsDB Cluster, in this example there are 5 nodes in the RapidsDB Cluster.:

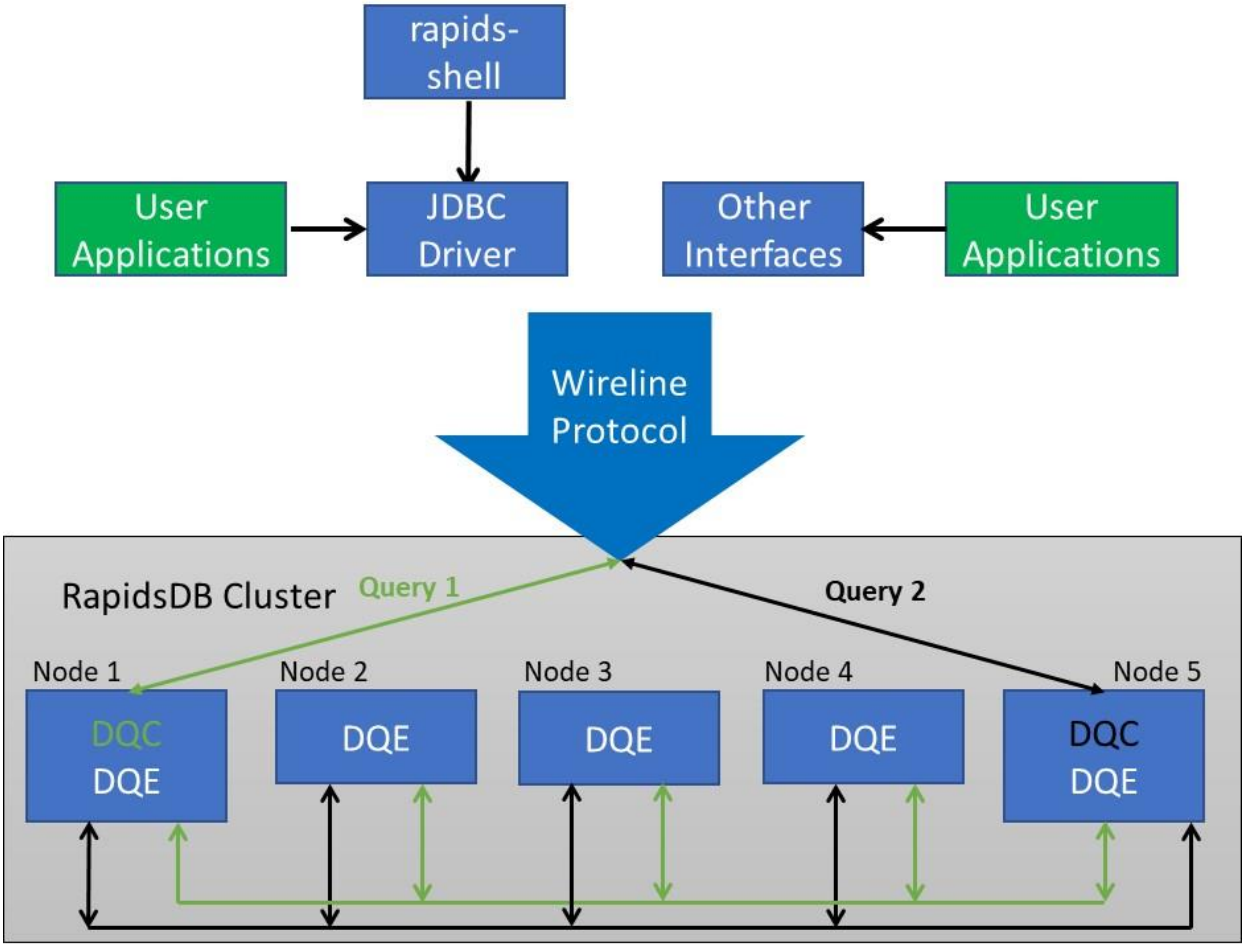

Figure 2. RapidsDB Cluster Topology

- rapids-shell – provides the command line interface for sending queries to the RapidsDB Cluster. The rapids-shell uses the RapidsDB JDBC Driver to communicate with the RapidsDB Cluster.
- User Applications can communicate with the RapidsDB Cluster using either the RapidsDB JDBC Driver or any other interfaces built on top of the RapidsDB Wireline Protocol.
- DQC (Distributed Query Coordinator) – queries can be submitted for execution over the Wireline Protocol to any node in the RapidsDB Cluster. The node where the query is submitted is called the DQC node, and this node is responsible for coordinating the execution of the query.

  In the example above, two queries are submitted to the RapidsDB Cluster, Query 1 is submitted to Node 1, which becomes the DQC for that query, and Query 2 is submitted to Node 5, which becomes the DQC for that query.

  In addition, when configuring the RapidsDB Cluster, one node must be configured as a DQC node, and this is the node where the RapidsDB cluster is installed from and where the RapidsDB cluster is stopped and started.

- DQE (Distributed Query Executor) – the nodes where the query execution is performed are called the DQE nodes. A DQE node might have a Connector running on it that is participating in the query. For example, if the query includes a MOXE table, then each node in the RapidsDB cluster will participate in the query. For other Connectors participating in a query, the Connector configuration will determine the nodes that participate in the query.

2 Federations, Connectors and Naming

2.1 Overview

In RapidsDB, a Federation is a logical grouping of a set of zero or more Connectors. Federations are named, and RapidsDB has a default Federation named "DEFAULTFED". At this time, the DEFAULTFED Federation is the only Federation available. In a future release support will be provided for using multiple Federations.

2.2 Connectors

The Connectors manage the metadata for the objects (typically tables or files) in the remote data store and present that metadata to the RapidsDB query execution engine as an ANSI-based SQL schema, thereby allowing the user to view the objects from the entire set of data sources as a single, federated SQL database.

The Connectors are responsible for managing data type conversion between the native data store and the RapidsDB query system which allows for the uniform handling of data types across all of the data stores. The Connectors present the data to the RapidsDB Execution Engine as rows and columns, which allows the data to be queried using standard ANSI SQL, and provides a uniform query interface regardless of the data source.

In order to optimize performance we generally want to have the underlying data source perform as much of the query as possible to reduce the amount of data that has to be processed by the RapidsDB Execution Engine, rather than simply pulling all of the data from the data source and processing it within RapidsDB. For example, when the data source is a relational database such as Oracle, that data source is typically capable of executing a join on the tables in that data source, or it can filter the data based on predicates in the query.

RapidsDB supports an optimization feature called "Adaptive Query Pushdown" to deal with this. With Adaptive Query Pushdown, each Connector involved in the execution of a query plan analyzes the query plan and decides which parts of the query plan it can push down to the data source, and then only the query results from the pushed down query need to be streamed to the Execution Engine to complete the query execution. For the remaining parts of the query the Connector will determine the optimum way to retrieve the data required to complete those parts of the query plan, and then stream that data to the Execution Engine.

2.3 Table Naming

All tables (and views) in RapidsDB are identified using an ANSI standard 3-part name of <catalog>.<schema>.<table>. The 3-part name has to be unique within a Federation, if the name is ambiguous an error will be returned when the name is referenced in a query. Each Connector is responsible for managing its own 3-part name space. Catalogs, schemas and tables can be imported under an alternate name to prevent name conflicts (see section 2.9 below). Where the underlying data source supports 3-part names, the Connector will use that 3-part name by default to identify each table . Where the underlying data source does not support a catalog and/or schema name, the Connector will provide the missing part(s) of the name. The table below shows the default assignment of catalog names and schema names for the different Connectors supported in this release:

Connector Type	Catalog Name	Schema Name
MOXE	Connector name	Connector name
MemSQL	Connector name	MemSQL database name (specified using the "DATABASE" option as part of the Connector definition)
MySQL	Connector name	MySQL database name
Oracle	Connector name	Oracle schema name
Postgres	Database name (specified using the "DATABASE" option as part of the Connector definition)	Postgres schema name
Greenplum	Database name (specified in JDBC connection url as part of the Connector definition)	Greenplum schema name
Hadoop	Connector name	PUBLIC
Hadoop with Hive Metastore	Connector name	Hive database name
JDBC	If the data source supports 3-part naming then this will be the data source catalog name, otherwise it will be the Connector name	If the data source supports 3-part naming then this will be the data source schema name, otherwise it will be the catalog or schema name returned by the JDBC Driver for that data source, whichever is not NULL.

The table below shows some examples for a Postgres and Oracle Connector:

```
CREATE CONNECTOR PG1 TYPE POSTGRES WITH DATABASE= 'dw1', USER='adm', PASSWORD='admpsw'
NODE NODE1 SCHEMA "production";
```

Catalog Name	Schema Name	Table Name
dw1	production	history
dw1	production	items
dw1	production	parts

```
CREATE CONNECTOR ORA TYPE ORACLE WITH USER='dba', PASSWORD='dba123', HOST='apollo',
SID='orcl' NODE NODE5 SCHEMA "customers";
```

Catalog Name	Schema Name	Table Name
ORA	customers	ORDERS
ORA	customers	CUSTOMER
ORA	customers	HISTORY

2.4 Retrieval and Storage of Schema Metadata

A Connector is responsible for retrieving the schema metadata (catalogs, schemas, tables, views) from the underlying data store that the Connector connects to, and for subsequently making that information available for use by the RapidsDB Query Planner. A Connector will save the schema metadata using the character case provided by the underlying data store. For example, by default Postgres returns all of the schema metadata information in lower case, whereas MemSQL will return the metadata for the table name in the case that was used when the table was created. The next section will describe how this schema metadata information is used.

The following table shows the schema metadata as stored by MemSQL and Postgres and the metadata stored by RapidsDB:

MEMSQL			RAPIDSDB		
	DATABASE	TABLE	CATALOG	SCHEMA	TABLE
	WEST	customer_west	MEM1	WEST	customer_west

	WEST	ORDERS	MEM1	WEST	ORDERS
POSTGRES			**RAPIDSDB**		
DATABASE	**SCHEMA**	**TABLE**	**CATALOG**	**SCHEMA**	**TABLE**
sales	east	customer_east	sales	east	customer_east
sales	east	orders	sales	east	orders

As can be seen from the table above, the schema information is stored in RapidsDB in exactly the same case as the underlying data store, with “MEM1” being used as the Catalog name for MemSQL because MemSQL does not support catalogs, and that is the name of the MemSQL Connector in this example.

The user can check on the schema metadata by querying the RapidsDB System Metadata tables in the RAPIDS.SYSTEM schema:

- CATALOGS
- SCHEMAS
- TABLES
- COLUMNS

Below are some sample queries against the System Metadata tables for tables managed by MOXE:


```
rapids > select * from tables where schema_name='MOXE';
CATALOG_NAME   SCHEMA_NAME   TABLE_NAME     IS_PARTITIONED PROPERTIES
------------   -----------   ----------     -------------- ----------
MOXE           MOXE          SUPPLIER                true
MOXE           MOXE          ORDERS                  true
MOXE           MOXE          LINEITEM                true
MOXE           MOXE          PARTSUPP                true
MOXE           MOXE          PART                    true
MOXE           MOXE          REGION                 false
MOXE           MOXE          CUSTOMER                true
MOXE           MOXE          NATION                 false

8 row(s) returned (0.12 sec)
rapids > select column_name, data_type, is_partition_key, is_nullable from columns
       > where schema_name='MOXE' and table_name='LINEITEM';
COLUMN_NAME       DATA_TYPE     IS_PARTITION_KEY   IS_NULLABLE
-----------       ---------     ----------------   -----------
L_ORDERKEY        INTEGER                  true         false
L_PARTKEY         INTEGER                 false         false
L_SUPPKEY         INTEGER                 false         false
L_LINENUMBER      INTEGER                 false         false
L_QUANTITY        DECIMAL                 false         false
L_EXTENDEDPRICE   DECIMAL                 false         false
L_DISCOUNT        DECIMAL                 false         false
L_TAX             DECIMAL                 false         false
L_RETURNFLAG      VARCHAR                 false         false
L_LINESTATUS      VARCHAR                 false         false
L_SHIPDATE        TIMESTAMP               false         false
L_COMMITDATE      TIMESTAMP               false         false
L_RECEIPTDATE     TIMESTAMP               false         false
L_SHIPINSTRUCT    VARCHAR                 false         false
L_SHIPMODE        VARCHAR                 false         false
L_COMMENT         VARCHAR                 false         false

16 row(s) returned (0.12 sec)
```

2.5 Object Name Resolution and Case Sensitivity

When the user submits a query, the RapidsDB Query Planner will need to determine which Connector is responsible for each of the tables referenced in a SQL query. Each table name can optionally be qualified with a catalog and/or schema name. In order to determine which Connector is responsible for a table, the RapidsDB Query Planner will send a request to all of the Connectors with the name of the table , optionally qualified with the catalog and/or schema name, and each Connector will then look up the table name in its schema metadata information. By default the RapidsDB Query Planner follows SQL conventions, converting all table names to upper case before sending the request to the Connectors, unless any part of the name (catalog, schema, or table) is enclosed in identifier delimiters (back-ticks or double quotes), in which case that portion of the name will be sent using the case specified in the query. The table below shows some examples:

Original Query	**Object Name Sent to Connectors for Resolution**
Select * from customer;	CUSTOMER
Select * from west.customer;	WEST.CUSTOMER
Select * from “customer”;	customer
Select * from “west”.”customer”;	west.customer

2.6 Connector Lookup of Object Names (Default)

The following describes the default way that Connectors will do lookup of the object names sent from the Query Planner. Each Connector maintains its own metadata information and controls the matching of names used in queries to names in the underlying data store. What the following will show is that **the user can specify the object names as case-insensitive names, even when the underlying data store, such as MemSQL, is case-sensitive.**

By default, each active Connector will do a case-insensitive lookup of the object name provided by the Query Planner, informing the Query Planner if it has a match for that object name. In the event that there are two or more matches for the object name, then an error will be returned to the user indicating that the object name is ambiguous. Assuming that the object name is unique, when the Connector builds the query to be submitted to the back-end data store, it will use the case as seen in the object metadata retrieved by the Connector (see Retrieval and Storage of Schema Metadata above). This means that for the vast majority of queries, the RapidsDB user can specify object names without regard for case. The Connector will ensure that the case used for the object names will match what the underlying data store expects. The only time that this would be a problem is when the underlying data store uses case-sensitive names (eg MemSQL) and the user has used the same object name but with different cases (eg customer and CUSTOMER). This situation is dealt with in section 2.7 below.

The following examples show how this name resolution can be applied to two Connectors, MemSQL (which is case-sensitive), and Postgres (by default, the object metadata information is stored in lower case).

The following table shows the database/schema and table names as stored by MemSQL and Postgres, and the names as stored by the MemSQL and Postgres Connectors:

MEMSQL		**RAPIDSDB**	
DATABASE	**TABLE**	**SCHEMA**	**TABLE**
WEST	customer_west	WEST	customer_west
WEST	ORDERS	WEST	ORDERS
POSTGRES		**RAPIDSDB**	
SCHEMA	**TABLE**	**SCHEMA**	**TABLE**
east	customer_east	east	customer_east
east	orders	east	orders

As described in section 2.5 above, Connectors preserve the case of names retrieved from the underlying data store but allow case-insensitive matching of names.

The following examples will go through the name resolution process using the metadata above:

1. Select * from customer_west;

   The Query Planner would ask the Connectors for the object name “CUSTOMER_WEST”, and the Connectors would do a case-insensitive match on that name, and the MemSQL Connector would have a match.

   The MemSQL Connector would then build (condense) the following query to be sent to MemSQL:

   ```
   select * from WEST.customer_west;
   ```

2. Select * from CUSTOMER_WEST;

   The Query Planner would ask the Connectors for the object name “CUSTOMER_WEST”, and the Connectors would do a case-insensitive match on that name, and the MemSQL Connector would have a match.

   The MemSQL Connector would then build (condense) the following query to be sent to MemSQL:

   ```
   select * from WEST.customer_west;
   ```

   Note that in this case the query as specified by the user had the table name in upper case, but when the query was sent to MemSQL the table name was converted to lower case to match what was provided by MemSQL when the schema metadata was retrieved.

3. Select * from WEST.CUSTOMER_WEST;

   The Query Planner would ask the Connectors for the object name “WEST.CUSTOMER_WEST”, and the Connectors would do a case-insensitive match on that name, and the MemSQL Connector would have a match.

   The MemSQL Connector would then construct (condense) the following query to be sent to MemSQL using the correct case for any table names and doing any necessary conversion for SQL syntax differences:

   ```
   select * from WEST.customer_west;
   ```


Note that in this case the query as specified by the user had the schema and table names in upper case, but when the query was sent to MemSQL the table name was converted to lower case to match what was provided by MemSQL when the schema metadata was retrieved.

4. select * from orders;

   The Query Planner would ask the Connectors for the object name “ORDERS”, and the Connectors would do a case-insensitive match on that name, and both the MemSQL and Postgres Connectors would have a match, and so the query would be rejected with an ambiguous name error. The user would have to specify the schema name to disambiguate the query as shown in example 5 below.

5. select * from west.orders;

   The Query Planner would ask the Connectors for the object name “WEST.ORDERS”, and the Connectors would do a case-insensitive match on that name, and the MemSQL Connector would have a match.

   The MemSQL Connector would then construct (condense) the following query to be sent to MemSQL using the correct case for any table names and doing any necessary conversion for SQL syntax differences:

   ```
   select * from WEST.orders;
   ```

   Note that in this case the query as specified by the user had the schema and table names in lower case, but when the query was sent to MemSQL the schema name was converted to upper case to match what was provided by MemSQL when the schema metadata was retrieved.

What is important to note is that the user can specify the object names as **case-insensitive** names, even when the underlying data store, such as MemSQL, is case-sensitive.

The following table shows another set of table names as stored by MemSQL and the names as stored by MemSQL Connector:

MEMSQL		**RAPIDSDB**	
DATABASE	**TABLE**	**SCHEMA**	**TABLE**
WEST	customer_west	WEST	customer_west
WEST	CUSTOMER_WEST	WEST	CUSTOMER_WEST

In this example, we have the same table name but they are specified using different cases.

The following examples will go through the name resolution process using the metadata above:

1. Select * from CUSTOMER_WEST;

   The Query Planner would ask the Connectors for the object name “CUSTOMER_WEST”, and the Connectors would do a case-insensitive match on that name, and the MemSQL Connector would now have two matches, and so the query would be rejected with an ambiguous name error.

2. Select * from “customer_west”;

   The Query Planner would ask the Connectors for the object name “customer_west”, and the Connectors would do a case-insensitive match on that name, and the MemSQL Connector would still have two matches, and so the query would be rejected with an ambiguous name error. Even when the user specified a quoted name, because the Connector did a case-insensitive lookup, both names matched.

The MemSQL and JDBC Connectors provide an option to handle this rare situation, IGNORE_CASE, which is described in section 2.7 below.

2.7 CASE_SENSITIVE_MATCH Lookups

In the very rare case that the underlying data store uses case-sensitive naming (eg MemSQL) **AND** the user has used the same name but with different cases for two tables in the same schema, then the IGNORE_CASE option can be set to FALSE to instruct the Connector to use **case-sensitive** matching of names provided by the Query Planner.

The following table shows the database/schema and table names as stored by MemSQL and Postgres, and the names as stored by RapidsDB:

MEMSQL		**RAPIDSDB**	
DATABASE	**TABLE**	**SCHEMA**	**TABLE**
WEST	customer_west	WEST	customer_west
WEST	CUSTOMER_WEST	WEST	CUSTOMER_WEST
WEST	orders	WEST	orders
POSTGRES		**RAPIDSDB**	
SCHEMA	**TABLE**	**SCHEMA**	**TABLE**
east	customer_east	east	customer_east

east	orders	east	orders

The following examples will go through the name resolution process using the metadata above, with the IGNORE_CASE option set to 'FALSE' for the MemSQL Connector:

1. Select * from customer_west;

   In this case this might not result in what the user expected because the original query had the table name in lower case, but because the table name was not enclosed within double quotes, the Query Planner will convert the name to upper case. In order to access the table name "customer_west" the table name must be quoted and specified in lower case as in the example below.

   In this example, the Query Planner would ask the Connectors for the object name "CUSTOMER_WEST", and the MemSQL Connector would do a case-sensitive match on that name, and the MemSQL Connector would match the MemSQL table named "CUSTOMER_WEST".

   The MemSQL Connector would then construct (condense) the following query to be sent to MemSQL:

   ```
   select * from WEST.CUSTOMER_WEST;
   ```

2. Select * from "customer_west";

   The Query Planner would ask the Connectors for the object name "customer_west", and the MemSQL Connector would do a case-sensitive match on that name, and the MemSQL Connector would match the MemSQL table named "customer_west".

   The MemSQL Connector would then build (condense) the following query to be sent to MemSQL:

   ```
   select * from WEST.customer_west;
   ```

3. Select * from west.orders;

   The Query Planner would ask the Connectors for the object name "WEST.ORDERS", and the MemSQL Connector would do a case-sensitive match on that name, and the MemSQL Connector would not find a match because both the schema name and table do no match (due to the case).

4. Select * from "WEST"."orders";

The Query Planner would ask the Connectors for the object name “WEST.orders”, and the MemSQL Connector would do a case-sensitive match on that name, and the MemSQL Connector would find a match for the table named “orders”.

The MemSQL Connector would then build (condense) the following query to be sent to MemSQL:

```
select * from WEST.orders;
```

It is highly recommended that all object names managed by a Connector with the IGNORE_CASE option set to ‘FALSE’ are specified as quoted names with the case of the object names set to match the case used by the underlying data store.

2.8 Importing Tables

When configuring Connectors (see section 7.10), the user can specify which tables should be “imported” and made available to the user for querying. The term “importing” in this context means getting the metadata associated with a table from the associated data source . The Import operation only applies to metadata, no actual table data will get imported. The actual table data will be accessed as part of a query that references that table. By default, a Connector will “import” all the metadata for all of the tables that can be accessed from the underlying data source. The user can restrict which tables will be imported when configuring a Connector by explicitly listing the catalogs, schemas and table names to be imported (see section 7.10).

2.9 Mapping Catalog, Schema and Table Names

When configuring a Connector, the user can map one or more parts of the 3-part name to local names that are used when querying through RapidsDB. The Connector maps these local names back to the original names used by the underlying data source. Figure 3 below shows two MemSQL Connectors, MEM1 and MEM2, each with the table “CUSTOMERS”, but in different schemas named “EAST” and “WEST”:

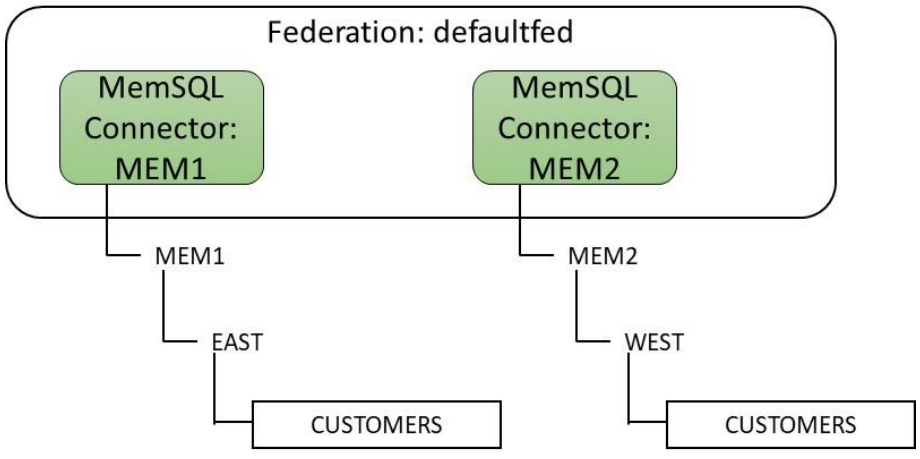

Figure 3. Mapping Catalog, Schema and Table Names

The query 'SELECT * FROM customers' would fail with an error indicating that the table name was ambiguous. To disambiguate the name the query would have to be changed to 'SELECT * FROM east.customers' to access the CUSTOMERS table managed by Connector MEM1.

When configuring the MemSQL Connectors the user could map the table named "EAST.CUSTOMERS" to "EASTCUSTOMERS" and the table named "WEST.CUSTOMERS" to "WESTCUSTOMERS", and then the table names would be unique (for each Connector) and would not have to be qualified using the schema name, eg.

```
CREATE CONNECTOR MEMSQL1
    TYPE MEMSQL WITH host='192.168.1.1', port='3306', user='user1', database='EAST'
    NODE NODE2
   CATALOG *
   SCHEMA *
   TABLE CUSTOMERS  as EAST_CUSTOMERS;
```

SELECT * FROM east_customers WHERE …

Refer to section 7.10.2 for details on how to do name mapping.

3 Operational Considerations for Connectors

Each of the RapidsDB Connectors exhibits different operational aspects which are described in the following sections.

3.1 MOXE Connector

The MOXE Connector operates as a multi-partitioned fully distributed Connector. When creating a MOXE Connector (see 7.10.4) the user can specify which nodes in the RapidsDB Cluster that Connector is to run on, the number of partitions used by MOXE on each node in the RapidsDB cluster, and the maximum amount of memory to be used on each node. MOXE supports both distributed and replicated tables. For distributed tables, each table managed by MOXE will be partitioned across all of the nodes on which the Connector is active and then distributed across all of the partitions on each node. The distribution of data across the partitions will be achieved by hashing the value of the partitioning column(s) specified for the table (as part of the CREATE TABLE – see section 8.1).

The user can create multiple MOXE Connectors on a system, each Connector will have its own catalog and schema which will match the Connector name.

When providing the data to Execution Engine for query processing, MOXE only passes pointers to the data to the Execution Engine, no physical copying of data occurs. The diagram below shows a two node RapidsDB cluster with a query being sent to Node 1 which acts as the query coordinator (DQC). The query results in all partitions of the table being scanned in parallel on both nodes and a set of row pointers (called row references) will be streamed to the Execution Engine on each node, and for each

row reference the Execution Engine will ask for the value for the column O_ORDERPRIORITY, which can be accessed using the row reference, and the predicate filter on O_ORDERPRIORITY will be applied. If the O_ORDERPRIORITY value satisfies the predicate, then the Execution Engine will request the values for the O_ORDERKEY and O_ORDERSTATS columns for that row and then the results from Node 2 will be pipelined to Node 1 where the data from Node 2 will be merged with the data from Node 1 and the final result set will be returned to the client:

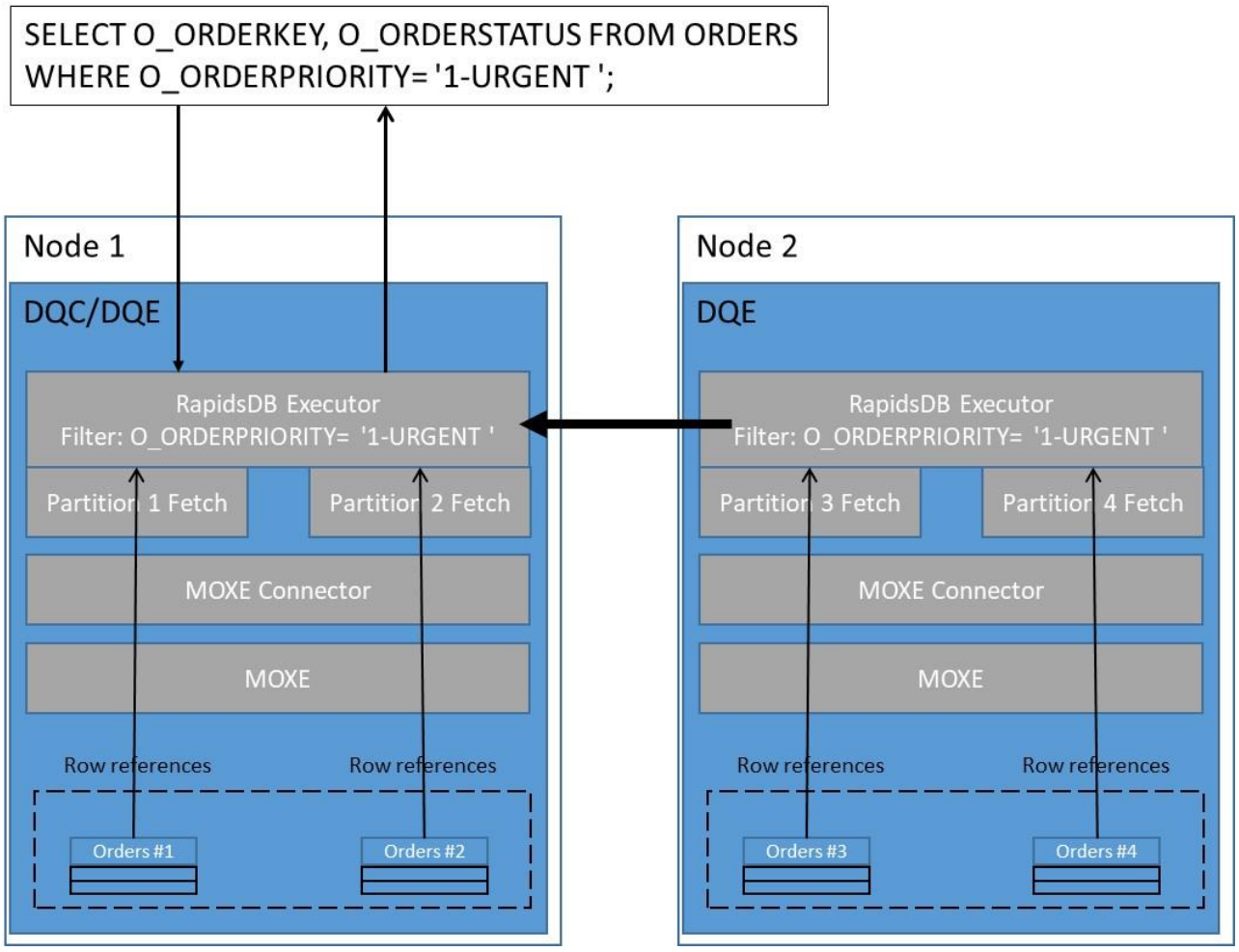

Figure 4. SELECT query processing with MOXE

For replicated tables, a full copy of each table will be replicated to each node in the RapidsDB cluster. Replicated Tables are typically used for storing smaller dimension tables so that when doing a JOIN with a larger table, the JOIN can be completed on each node without having to send any data from the replicated table across the network. In the example below, the NATION table is replicated and so the join between the NATION and SUPPLIER tables will be executed in parallel on both nodes, and the results from Node 2 will be pipelined to Node 1 where the data from Node 2 will be merged with the data from Node 1 and the final result set will be returned to the client:

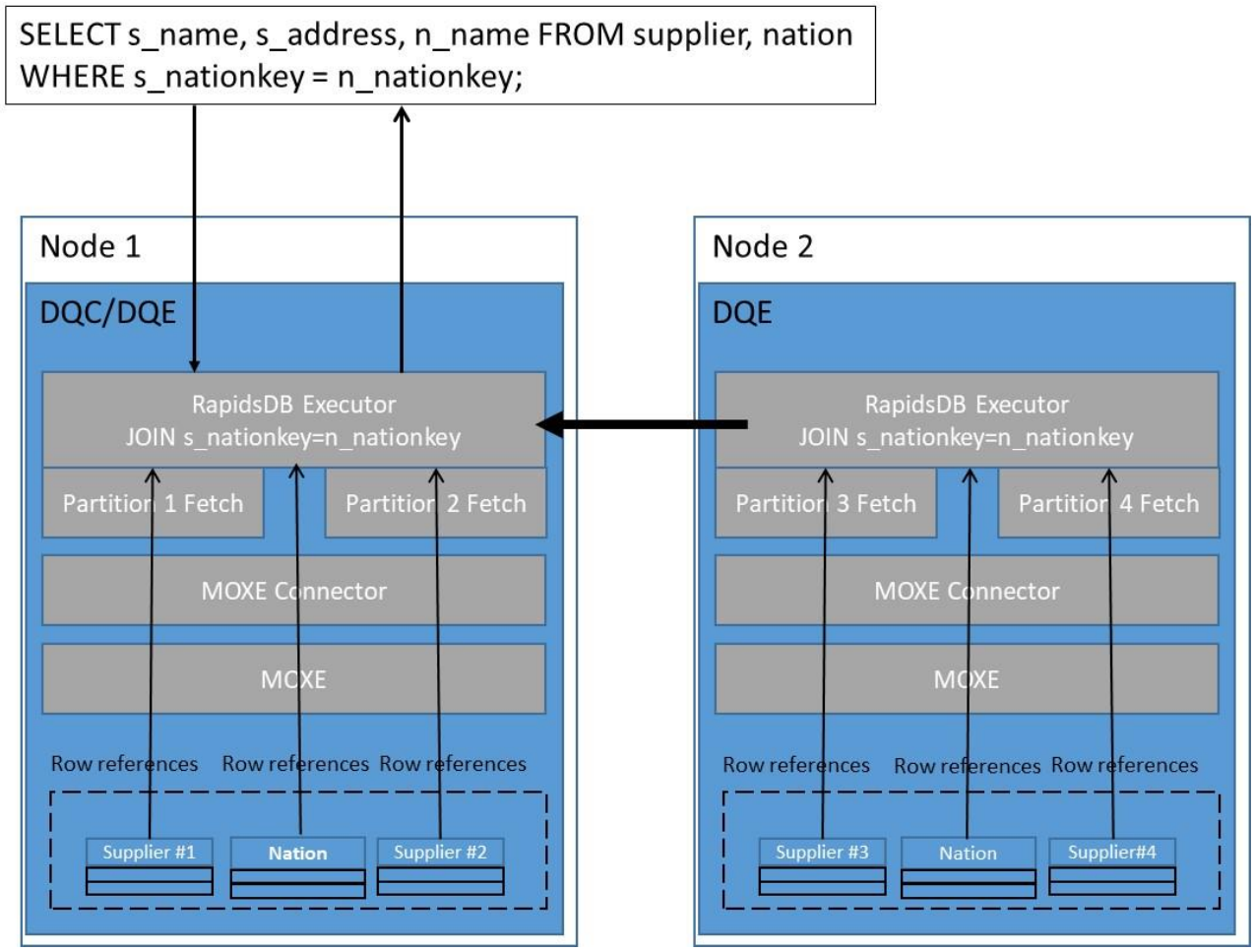

Figure 5. SELECT query processing with MOXE Replicated Table

MOXE also provides support for backing up either a complete database or individual tables to disk and then the user can subsequently restore the backups. Refer to section 8.5 for further details.

3.2 RDBMS Connectors

The RDBMS (MySQL, MemSQL, Oracle, Postgres and Greenplum) Connectors operate as non-partitioned, single node Connectors. When configuring an RDBMS Connector the user can select any node in the RapidsDB Cluster for the Connector to run on (this can be different from the node where the underlying data source is running). Where possible, the user's query will be pushed down to the underlying RDBMS database and only the result of the query will get returned, and in this case the query execution will take advantage of any parallelism in the underlying RDBMS database. In the event that all or part of a query cannot be pushed down (for example, a federated query involving two or more data sources), then the data will be fetched from the RDBMS database by the Connector, and pipelined to the RapidsDB Execution Engine for further processing. Even in this case, any predicates will get applied to the data being fetched, to minimize the amount of data being retrieved. The data pipelining allows the query to proceed as soon as the data starts to arrive from the data source, and in conjunction with the buffering performed by the Connector when retrieving the data, the net result is that only a small subset

of the entire dataset being retrieved will be in memory at any one time. Figure 6 below shows a MemSQL Connector running on the same node as the MemSQL Aggregator that the MemSQL Connector will be using.

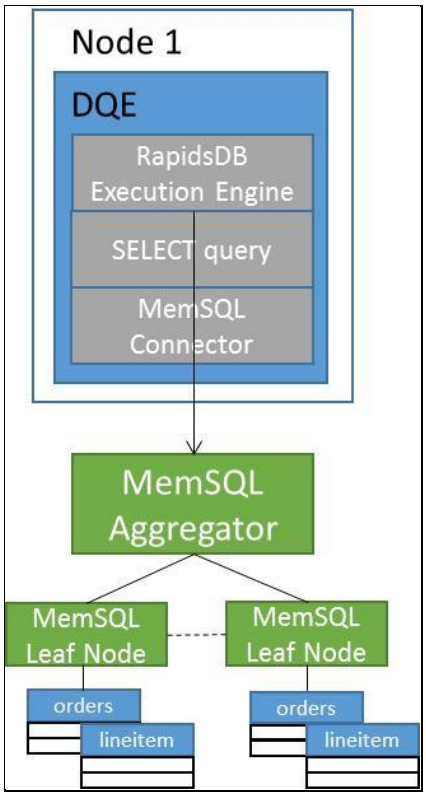

Figure 6. SELECT query processing with MemSQL Connector

3.3 JDBC Connector

The JDBC Connector operates as a non-partitioned, single node Connector, in exactly the same way as the RDBMS Connectors (see 3.2). When configuring a JDBC Connector the user can select any node in the RapidsDB Cluster for the Connector to run on (see 7.10.10).

3.4 Hadoop Connector

3.4.1 Partitioning

The Hadoop Connector operates as a multi-partitioned, fully distributed Connector. The user can specify which nodes in the RapidsDB cluster the Hadoop Connector is to run on, and how many partitions are supported per node. The partitioning with the Hadoop Connector is logical partitioning with all of the partitioning being done within the Connector, not at the physical file level. This means that the user can change the partitioning, such as changing the number of partitions per node, without having to change the physical data in HDFS.

3.4.1.1 Delimited Files

When reading data from an HDFS delimited file, the Hadoop Connector will allocate a portion of the HDFS file to each node assigned to that Connector, and then on each node where Hadoop Connector is running the data will be split evenly across a set of partition readers so that the data will be read in parallel across the RapidsDB cluster. If there are n nodes assigned to a Hadoop Connector and m partitions allocated per node, then there will be n*m parallel reads against the HDFS file. The data being read is also buffered, with the records being pipelined to the RapidsDB Execution Engine for filtering and other processing. This means that it is possible for the Hadoop Connector to read files that are larger than the available memory on the nodes in the RapidsDB Cluster. Figure 7 below shows a two-node Hadoop Connector with m partitions per node:

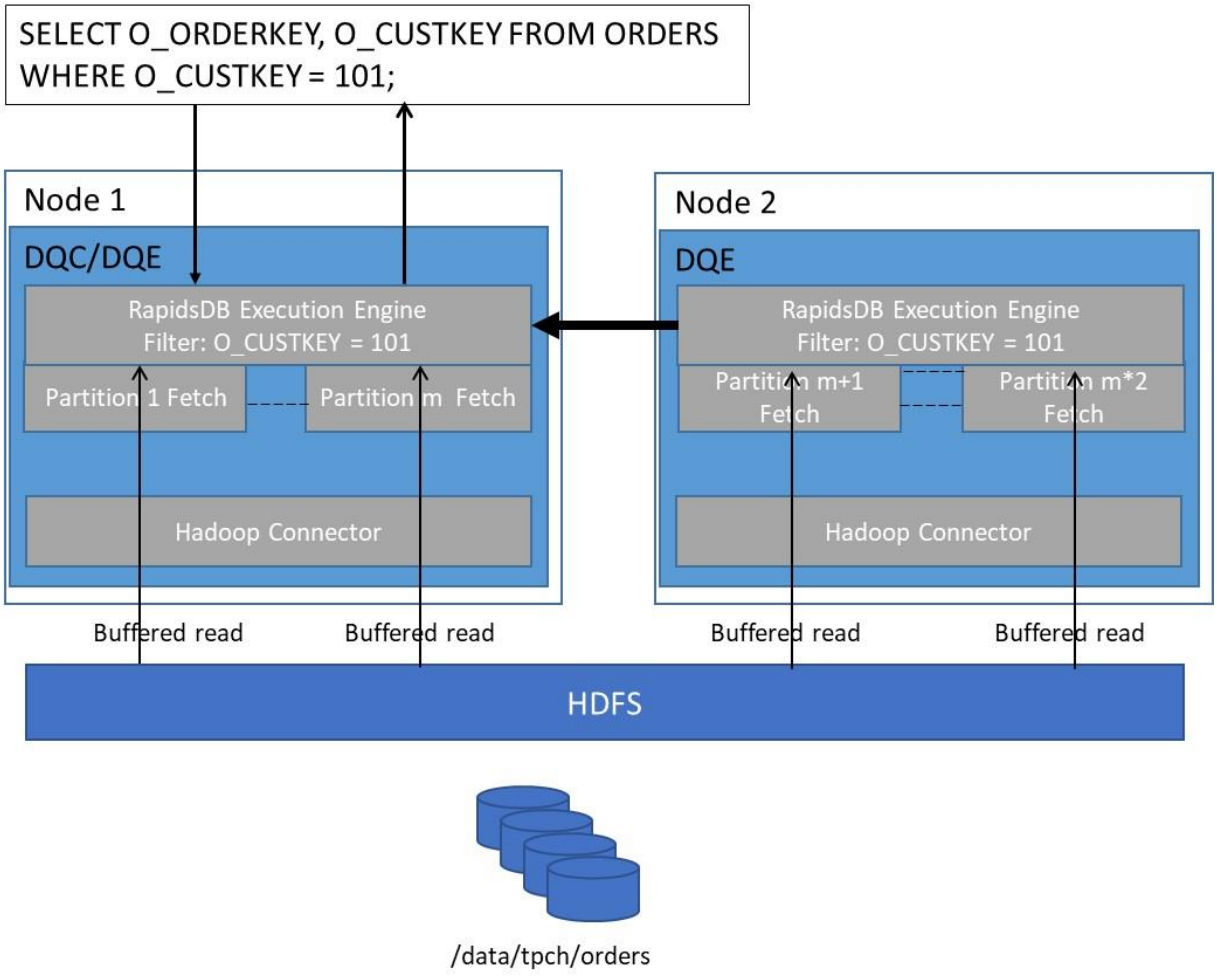

Figure 7. SELECT query processing with Hadoop Connector with delimited files

In this example, all of the columns from the data files in /data/orders/tpch directory will be returned to the Execution Engine, which will then apply the predicate O_CUSTKEY=101 to filter the data and then only return the requested columns to the client.

When writing data to delimited files, if the data is being written in parallel, then the data being written will be distributed round-robin over all of the partitions and then written out to the target files. Refer to section 7.10.11.5 for more details.

3.4.1.2 ORC and Parquet Files

When reading ORC and Parquet files, the Hadoop Connector will calculate the number of HDFS blocks across all of the files to be read, and then divide up the blocks across the partition readers on each node where the Hadoop Connector is running. For example, if there are 10 files to be read and each file has 3 HDFS blocks, then there are 30 blocks to be read and if the Hadoop Connector was running on 6 nodes with 4 partitions per node, then the 30 blocks would be split across the 24 partition readers.

When reading ORC and Parquet files only the columns required to process the query from the associated table will be read. In addition, any column predicates will get used to restrict the data being read from the Parquet files. The example below illustrates this:

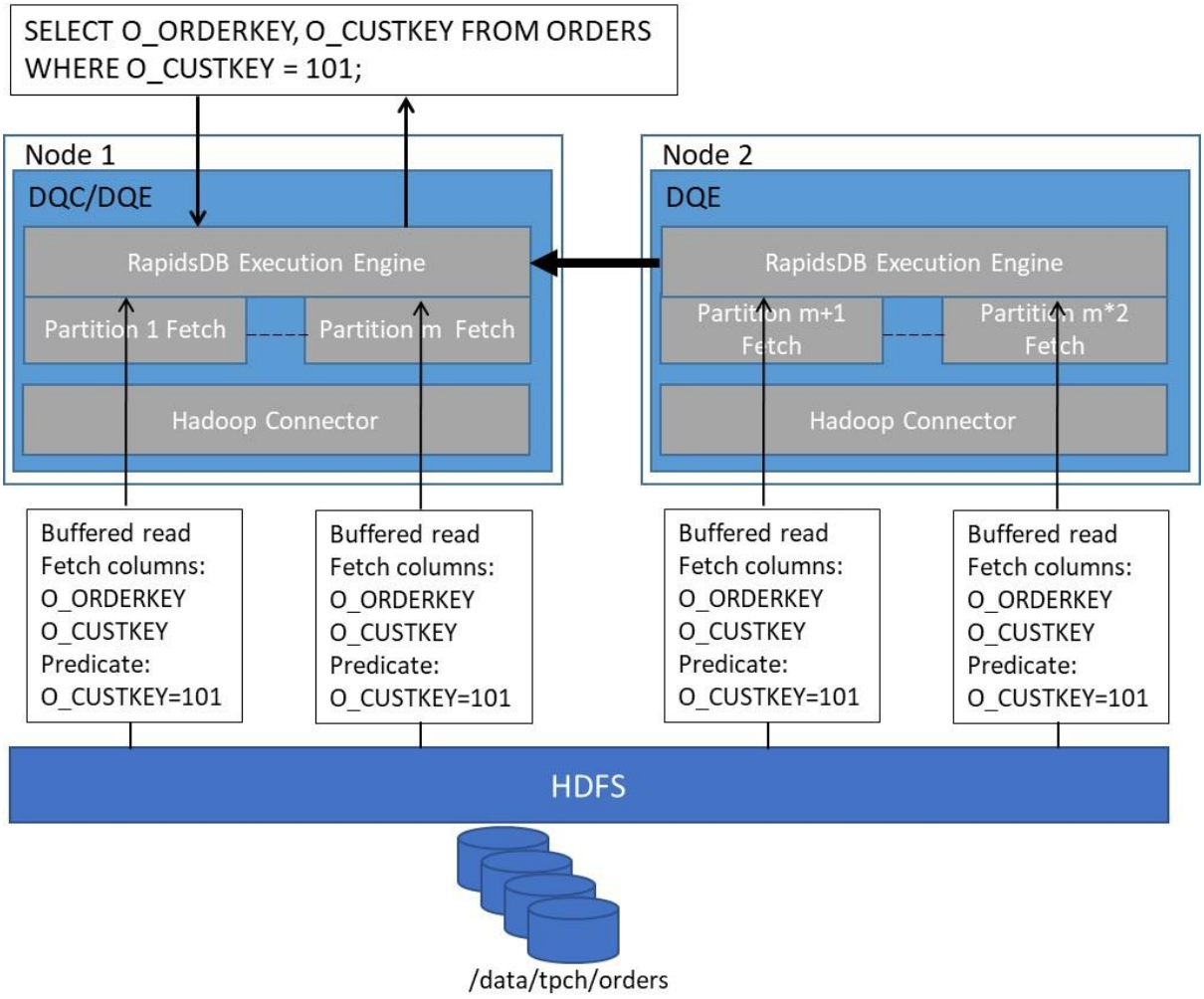

Figure 8. SELECT query processing with Hadoop Connector with Parquet files

In this example, only the data for the columns O_ORDERKEY and O_CUSTKEY will be read from the Parquet file(s) in the directory /data/tpch/orders, and only for those rows where the predicate O_CUSTKEY=101 is satisfied.

3.4.2 Hive-style Partitioning

The Hadoop Connector also supports Hive-style partitioning where the data stored in HDFS is arranged in directories where the directory names match the values for columns in the table. For example, in the following HDFS file structure below, the data is partitioned over the columns "region" and "country", and so the files under /data/user/region=North America/country=US would match with a region of "North America" and country of "US".

/data/user/region=North America/country=US
/data/user/region=North America/country=CA
/data/user/region=South America/country=BR
/data/user/region=South America/country=ME

When a query of the form SELECT <column list> FROM <table> WHERE REGION='North America' AND COUNTRY='US'; is submitted, the Hadoop Connector will use the predicate "REGION='North America' AND COUNTRY='US' " to restrict the files to be read to those files in the directory /data/user/region=North America/country=US. Refer to section 7.10.12.8 for details on using the PARTITION BY clause to specify this type of partitioning.

3.4.3 Writing Data to HDFS

The Hadoop Connector also supports writing the results of an INSERT or INSERT … SELECT statement to HDFS as described in section 7.10.11.11.

4 Users and Authentication

4.1 Overview

RapidsDB requires all client connections to authenticate prior to issuing commands. Similarly to how RapidsDB can federate many different data sources, it can also federate multiple authentication sources as well. RapidsDB users can be configured to be authenticated via a particular source. These are known as authenticators. RapidsDB currently supports authenticating users via passwords (validated internally), or via a 3rd party Kerberos installation.

This release does not include customizable authorization features. Instead, it limits certain operations to only be executed by the initial user, as explained below.

4.2 Initial User

When RapidsDB is first started it will create an initial user. The username for this user is RAPIDS and it will have the initial password set to "rapids". Users can authenticate as this initial user in order to configure the system and create additional users.

In this release, this initial user has more privileges than regular users, despite RapidsDB not supporting a full authorization implementation. The RAPIDS user is able to execute these commands that regular users are not:

- CREATE CONNECTOR
- DROP CONNECTOR
- ALTER CONNECTOR
- CREATE USER
- DROP USER (except regular users can drop themselves)
- ALTER USER (except regular users can alter themselves)
- CREATE AUTHENTICATOR
- DROP AUTHENTICATOR
- ALTER AUTHENTICATOR

The password for this initial user can be changed at any time by issuing the command:
ALTER USER rapids PASSWORD 'newPassword';

Please note that if you forget the new password that has been set then you will not be able to login to the system as this user.

It is not possible at this stage to change this initial user to use a different authenticator, such as Kerberos. This RAPIDS user can currently only be authenticated via password.

4.3 Authenticators

Authenticators are pluggable modules that control how a RapidsDB user is authenticated. Each user in the system is associated with at most only one instance of an Authenticator. When a user connects and tries to authenticate, the request and the handling of that authentication is performed by the authenticator instance.

RapidsDB currently supports an internal password-based authenticator as well as a Kerberos-based authenticator. An instance of an authenticator type needs to be created before it can be used, except for the internal password-based authenticator, which is automatically created by the system. Different authenticator types may require different information or configuration when they are created.

If users are associated with an authenticator and that authenticator is dropped, then those associated users can also be dropped simultaneously. If they are not dropped then they will no longer be able to login because there is no configured authenticator able to do this job.

Similarly, if an authenticator is disabled then all authentication requests it receives will be rejected, thus denying all users associated with that authenticator instance from being able to login. Note that the internal password-based authenticator cannot be dropped or disabled.

Refer to section 7.3 and beyond for how to add and manipulate authenticators.

4.4 Users

In this release, regular users are considered to be all those users that are not the initial RAPIDS user. Unlike the RAPIDS user, regular users can only alter themselves or drop themselves. They cannot affect other users or authenticators.

Regular users can be individually enabled or disabled to control their access to RapidsDB. A user that is disabled will not be able to login, even if they provide valid credentials.

RapidsDB stores user accounts in Zookeeper. Consideration should be given to ensuring that the Zookeeper data is appropriately backed up and can be restored in the event of failure.

Refer to section 7.6 and beyond for how to add and manipulate users.

4.5 Authentication Process

When a client connects to RapidsDB it sends RapidsDB a message indicating how it intends to authenticate and with the appropriate information to support that authentication method. This could be with a username and password, or via a token based method such as Kerberos. The message also contains an identifier for the user. This identifier may be a RapidsDB username, or it may be a user identifier from an external authentication system (e.g. a Kerberos principal, or an LDAP distinguished name).

When RapidsDB receives this initial authentication message it needs to determine which RapidsDB user is trying to connect and then look up the authenticator for that user so the authenticator can complete the request. Initially, RapidsDB will take the user identifier given to it in the initial authentication message and try to look up the RapidsDB user under that name. If there is a user whose username matches the user identifier given then RapidsDB will look up the authenticator for that user and pass the request on to that particular authenticator.

If there are no users whose username matches the identifier given then RapidsDB will see if it has a mapping that can translate this identifier to a RapidsDB username. Refer to section 4.7 for more details on user name mapping. First, RapidsDB will look for a username mapping. This is a direct 1:1 mapping from an external identifier to a RapidsDB username. For example, given an external Kerberos principal of "`john@EXAMPLE.COM`", this might be mapped to the RapidsDB username `JOHN`. These username mappings can be added and removed manually or, if the external identifier is specified in the CREATE USER statement (see 7.6), it may be setup automatically when the user is created and dropped. There can be multiple mappings from different external identifiers to the a given RapidsDB username.

If there is a matching username mapping found then RapidsDB will use the associated username and authenticate the user as per above. If there is a mapping to a user that does not exist then authentication will fail.

If there are no mappings that match the external identifier given then RapidsDB will consult a table of pattern mappings to see if there are any general patterns specified that can be used to map an external identifier to a RapidsDB username. Refer to section 7.9 for more details on setting pattern mapping.

Pattern mappings may be employed if all external identifiers are consistent in their format and it is easier to write a regular expression to match this format and transform it into a valid RapidsDB username.

If an external identifier matches against a pattern then the transform is applied to yield a RapidsDB username. If a user with that username exists then authentication will proceed as described above. If not then authentication will fail.

If the external identifier does not match against a pattern then subsequent patterns are tried either until one of them matches or there are no more remaining patterns. At that point authentication will fail and the client will be denied access to RapidsDB.

When authentication fails, RapidsDB will wait a few seconds before returning the failure message so as to minimize potential brute-force credential guessing attacks.

When a client fails to authenticate, RapidsDB will return a generic error message to them to avoid giving clues as to why the authentication failed. However, an admin can look at the log file on the node that they were authenticating to and this will give extra information as to why the user could not be authenticated.

4.6 Authenticator Considerations

4.6.1 RDP Authenticator

The internal RDP Authenticator is the default authenticator used when creating users. It is a password-based authenticator that uses the scrypt Key Derivation Function (KDF) to transform password strings into hashes.

The scrypt KDF is configured so that it is applied repeatedly a number of times in order to be computationally expensive. While this means that it takes a small but noticeable amount of time to calculate the user's hash from their password, it also means that it also takes an attacker an equally significant amount of time to make a guess on the user's password. This, combined with the fact that scrypt requires a reasonable amount of memory to do so, means that it is significantly harder for an attacker to brute force a user's password in reasonable time because it requires significant computing power in order to parallelize the attack.

In addition, RapidsDB will generate a new salt for a user each time their password is changed. This means that an attacker cannot pre-compute a range of passwords and their hashes and compare these to other user's hashed passwords. Since each user will have a different salt the attacker would need to recompute these rainbow tables for every user in the system, and then re-do them every time the user changed their password (because the salt would be changed).

By default, the scrypt KDF has been configured to take between 0.125 and 0.250 seconds to compute the hash of the password when a client authenticates, across a range of computers.

RapidsDB currently stores the password hash, salt and KDF configuration along with the user's account in Zookeeper. Therefore this information is only as secure as Zookeeper is kept. Administrators should therefore ensure that access to the contents of Zookeeper is restricted to trusted individuals. This storage location could change in the future, however it won't affect how this authenticator is used.

RapidsDB does not store plain text passwords.

4.6.2 Kerberos Authenticator

When the Kerberos authenticator is being used, it needs to know two pieces of information:

1. What is the Kerberos principal of the current node?
2. What is the path of the local file that has the encryption keys for this principal (the keytab)?

By default, the Kerberos authenticator will use a default format for the name of the principal of each node. This is of the form "`rdp/<hostname>@<realm>`", where *<hostname>* is the fully qualified hostname of the current node (including the domain name) and *<realm>* is the Kerberos realm that the principal belongs to. The pattern of this principal can be specified when the Kerberos authenticator is created, or it can be provided in a file on each node.

Each node in the cluster should have a different Kerberos principal that matches its hostname as Kerberos may be configured to cross check the principal with the reverse DNS lookup of the machine's IP address.

The keytab file is the file that contains the encryption keys for the principal that represents the RapidsDB service on a given node. The security of this file is very important because with this file anyone would be able to represent themselves as this node's principal. This file should only be readable by the OS account used to run RapidsDB, i.e. it should have permissions of 600.

For convenience, it is possible to add the encryption keys for all RapidsDB service principals to a single keytab file and then distribute this keytab file to all nodes. This avoids having to manage a different keytab file for each node, but it also comes at the disadvantage of being less secure. If one of these keytab files are stolen then an attacker could impersonate any of the RapidsDB principals included.

Each node in the cluster must also be configured for Kerberos. This includes ensuring that the Kerberos client libraries are installed and configured correctly. Refer to Appendix C for information on installing Kerberos.

4.7 Mapping External User IDs

User ID mapping is important when a client wishes to authenticate with an external authentication system (e.g. Kerberos or LDAP) and the client has an external identifier for this external authentication system but does not know the username of the RapidsDB account it wishes to be authenticated as. An example of this is mapping the Kerberos principal name of a user to an RDP username.

RapidsDB includes two methods for mapping an external user identifier (e.g. a Kerberos principal or an LDAP distinguished name) to a RapidsDB username. The process by which these mappings are used is explained in section 4.5 (Authentication Process).

This user ID mapping is typically not needed when RapidsDB users use the internal authenticator. It is most likely to be used when clients wish to authenticate using an external authentication system such as Kerberos or LDAP.

 Please note that mapping user IDs does not change how a user is authenticated. E.g. if a Kerberos principal is mapped to a RapidsDB user that uses the internal authenticator and expects a username and password, then authentication will always fail for that user even with correct Kerberos credentials.

4.7.1 Username Mappings

Username mappings are 1:1 mappings between an external identifier and a RapidsDB username. Typically, one external identifier would map to a single RapidsDB username, however it is possible for multiple different external identifiers to map to the same RapidsDB username. It is not possible for an external identifier to map to more than one RapidsDB username.

Username mappings are created automatically if the CREATE USER statement includes the external identifier as part of the WITH clause. E.g. for a user being authenticated by a Kerberos authenticator it would look like this:

CREATE USER john AUTH 40ilma40os WITH principal = 'john@EXAMPLE.COM';

Mappings that are created this way are also automatically removed when the user is dropped.

Username mappings can also be manipulated manually by an admin with the commands:

```
ADD USERNAME MAPPING '<external_id>' TO <username>;

REMOVE USERNAME MAPPING '<external_id>';
```

4.7.2 Pattern Mappings

Pattern mappings are regular expression search/replace patterns that can be used whenever there are many users with external identifiers that all follow a common format. Instead of specifying individual username mappings for each user, an admin could instead specify a pattern that covers all of the external identifiers and transforms them into RapidsDB usernames in a common way.

There can be multiple patterns defined, and they are tried in a priority order. The individual patterns contain the following 3 fields:

1. Priority (integer)

2. Search (string)
3. Replace (string)

The priority field identifies the order that an individual pattern will be tried (higher values are tried first). The search field is used to match against the external ID. If it matches then the replace field will be used to generate the RapidsDB username.

These pattern maps are loaded into RapidsDB from a text file on one of its nodes. Once the file has been loaded, the patterns are shared with all RapidsDB nodes in the cluster and persisted into Zookeeper so they do not need to be loaded again unless they are being changed. Pattern maps can be loaded and changed by executing the command:

```
SET PATTERN MAP FILE [ 'path/to/file' | NULL ] ;
```

Specifying a null path will result in the pattern map being cleared.

Refer to section 7.9.4 for more details and examples of pattern maps.

5 Installation Overview

The following instructions describe how to set up the RapidsDB system on a cluster of nodes running CentOS (or RHEL) version 6.5 or later. The setup may be different for other Linux variants such as Ubuntu. For the purpose of this document the hardware setup will be done on 3 nodes:

- NODE1:192.168.1.1
- NODE2:192.168.1.2
- NODE3:192.168.1.3

The instructions are adaptable to any number of nodes, and to other environments, such as Linux Containers and AWS.

5.1 Basic System Setup

The following assumes that there is no way to create a system image (eg AWS AMI) that can be applied to each node in the cluster, and so the following instructions need to be followed for each node in the cluster.

5.2 Minimum System Requirements

The minimum system requirements for running the RapidsDB software are:

- Minimum of 4 cores
- Minimum of 32GB of memory
- Linux CentOS or RHEL release 6.5 or later. Other Linux distributions, such as Ubuntu, can be used, but the setup instructions may differ.

NOTE:

1. **RapidsDB requires bash to be installed on the system.**
2. **It is preferable, though not necessary, that the user account that RapidsDB runs under has bash as its default shell.**
3. **If installing on Ubuntu make certain that the symbolic link /bin/sh is a symbolic link to /bin/bash**

5.3 Create the rapids user

The following creates the operating system userid that will be used when running the RapidsDB software. Any userid can be used, in this case we are using the userid "rapids":

1. Log in as root, or use sudo to gain root privileges
2. Create the user:
   useradd rapids
3. Set up the password for the user:
   passwd rapids , then enter the user password

5.4 Install Java

1. Make sure you are on Java version 1.8 jdk. If not, you will need to upgrade to the Java 1.8 jdk. To check the Java version, type java –version <enter>. If it is not 1.8, execute the following when running as a user with root privileges:

   yum install java-1.8.0-openjdk-devel

2. The Java Runtime Environment (JRE) must be available on the default system path. This typically includes directories such as /bin and /usr/bin. If Java is not in this path then please check your java installation or try adding a symbolic link from /usr/bin/java to wherever Java is installed.

5.5 Configure ssh Access

RapidsDB requires passwordless ssh across all of the nodes in the RapidsDB cluster. The requirement is that you should be able to SSH from any account where the RapidsDB bootstrapper (see section 6.4) executes to all of the nodes in the RapidsDB cluster using a password-less public/private key authentication only. Additionally, each node in the RapidsDB should be able to SSH to itself using a public/private key without requiring a password. This will allow the bootstrapper to start and stop nodes across the cluster without needing to store or prompt for passwords.

The RapidsDB cluster is configured using the cluster.config file as described in section 6.1.2. Included in the cluster.config file is the following entry:

```
"sshPathToIdentityFile" : "~/.ssh/id_rsa",
```

This entry specifies the path name to private key file for the rapids user, which by default is "~/.ssh/id_rsa". If you set up the private key in a different file then the cluster.config file must be changed to reflect the correct the path name to the private key file.

NOTE:

Verify the private key file is in the correct format by checking the prefix line:

```
 $ head -1 ~/.ssh/id_rsa
-----BEGIN RSA PRIVATE KEY
```

This is the correct result.

If the result starts with OPENSSH PRIVATE KEY, then the private key format will need to be changed to a compatible private key format.

```
$ head -1 ~/.ssh/id_rsa

-----BEGIN OPENSSH PRIVATE KEY-----
```

This is an incompatible private key format for bootstrapper.

To convert a user's private key format, use the ssh-keygen change passphrase option "-p" and specify the "pem" format. You will be asked to enter a new passphrase, typically just the enter key (empty passphrase.) The actual key and previously created public key will not change.

```
ssh-keygen -p -f ~/.ssh/id_rsa -m pem
```

The resulting format can again be verified.

```
$ head -1 ~/.ssh/id_rsa

-----BEGIN RSA PRIVATE KEY-----
```

Appendix D provides a set of recommendations for how to configure ssh for use with RapidsDB.

5.6 Set up RapidsDB Installation Directories

Create the RapidsDB Cluster installation directory on each node in the cluster. It is recommended that the installation directory be /opt/rdp:

1. As root, logon on to the node and create the installation directory:
    - mkdir /opt/rdp
2. Change the ownership of this directory to the rapids user:
    - chown rapids:rapids rdp

5.7 Download the RapidsDB Cluster Software

You must choose one of the nodes in the RapidsDB cluster to act as the DQC node, which will be the node where all management operations against the cluster (install, start, stop) are performed.

The RapidsDB Cluster software only needs to be downloaded to the DQC node, which in this example is NODE1.

The software will be downloaded into the RapidsDB installation directory just set up (see 5.6):

1. Become the rapids user before you scp the RapidsDB installer.
    - su rapids –
2. Change to the /opt/rdp directory, and scp the latest RapidsDB Installer file from the RapidsDB repository. Contact your local RapidsDB support person to get the details for downloading the file.
    - cd /opt/rdp
    - scp rapids@.../rdp-installer-4.0-0.run .
3. Change the file settings on the Installer file:
    - chmod +x rdp-installer-4.0-0.run
4. Unpack the installation file:
    - ./rdp-installer-4.0-0.run

5.8 Install Zookeeper

Zookeeper version 3.4.6 (or a later stable version) is required. It is recommended that Zookeeper is installed into /opt, but it is not a requirement. The following instructions show how to install Zookeeper into /opt on the DQC node (any node can be used for Zookeeper). Apply the following from the node that will be acting as the RapidsDB DQC node:

1. As root (or a user with root privileges), create the directory /opt/zookeeper_data and assign ownership to the rapids user:
    - mkdir /opt/zookeeper_data
    - chown rapids:rapids zookeeper_data
2. Download the zookeeper-3.4.6.tar.gz file from the zookeeper website (http://www.apache.org/dyn/closer.cgi/zookeeper/) to the /opt directory.
3. Extract the tar file:
    - tar -zxvf zookeeper-3.4.6.tar.gz
4. Rename the directory zookeeper-3.4.6 to zookeeper:
    - mv zookeeper-3.4.6 zookeeper
5. Rename conf/zoo_sample.cfg to conf/zoo.cfg:
    - mv conf/zoo_sample.cfg conf/zoo.cfg
6. Edit the file zoo.cfg and change the entry for dataDir to /opt/zookeeper_data

- dataDir=/opt/zookeeper_data

7. Change the ownership of these directories to rapids:
    - chown -R rapids:rapids zookeeper

5.9 Open up TCP/IP Ports

Open the required ports. It is assumed that the reader is familiar with how to open ports under Linux. If not, refer to the Appendix: Configuring Ports Under Linux. The following port numbers must be opened on all nodes in the RapidsDB Cluster. If the default port numbers are changed when configuring RapidsDB (see 6.1.2) then the appropriate port numbers must be opened:

Port #	Usage
2181	Default port used by the RapidsDB Zookeeper instance
4333	Default port used by the JDBC Driver for communicating with a node in the RapidsDB Cluster
4334	Default port used by RapidsDB for communication between RapidsDB nodes

5.10 Install Kerberos

If Kerberos authentication is required with RapidsDB, please refer to Appendix C for details.

6 Installing RapidsDB

6.1 Configuring the RapidsDB Cluster Software

6.1.1 Setting up Zookeeper configuration: zk.config

The Zookeeper configuration file used by the RapidsDB Cluster is stored in the file zk.config which must be set up in the RapidsDB installation directory on the DQC node (see 5.7). A sample file, named zk.config.sample, is provided with the RapidsDB software in the directory in the cfg directory. The file has two entries,

nodeList	Tells RapidsDB the hostname/IP addresses and port numbers where it can find Zookeeper nodes. The Zookeeper client within RapidsDB will connect to one of these and if that node goes down then it will automatically connect to another node in the node list. The format is a comma-separated list.
rootDir	Tells RapidsDB the root path within Zookeeper of where to store all its data. This allows for different configurations to be saved within the same Zookeeper server without interfering with each other. The default root path is /rdp.

Below is the zk.config.sample file:


```
# A list of zookeeper servers in a cluster to connect to.
# The format is a comma-separated set of “host:port” entries.
# e.g.:
# nodeList = 192.168.1.1:2181, fred:2181, 46ilma:2181, bambam:2181
nodeList = localhost:2181

# The path within Zookeeper where all data is kept for this Rapids cluster.
# This separates data from multiple separate Rapids clusters that all share
# the same Zookeeper cluster.
# The value must start with a forward slash.
rootDir = /rdp
```

This file only needs to be modified if Zookeeper was installed on a node other than the DQC node, or there are multiple Zookeepers being used or if the root directory to be used in not /rdp. The file then needs to be copied from the cfg directory to the installation directory, such as /opt/rdp, and stored as zk.config:

- cd /opt/rdp
- cp current/cfg/zkconfig.sample zk.config

6.1.2 Setting up RapidsDB configuration: cluster.config

RapidsDB uses a file named cluster.config to define the cluster configuration. The cluster.config file must be set up in the RapidsDB installation directory on the DQC node (see 5.7). A sample file, named cluster.config.sample, is provided with the RapidsDB software in the current/cfg directory and this file must be copied to the file cluster.config in the installation directory and then edited to reflect the cluster configuration that will be used for this installation.

The cluster.config file uses JSON to store the configuration settings. The file is split into two main sections, commonNodeConfig and nodeConfigs. Any settings provided in the commonNodeConfigs are applicable to all nodes in the cluster unless they have been explicitly overridden in the nodeConfigs section. Each subfield of the commonNodeConfig section is also applicable in the nodeConfigs section.

The fields in the cluster.config file are described below:

Field	Sub-field	Value(s)	Description
commonNodeConfig			Defines the default settings that will apply to all nodes in the cluster. Any default field value can be overridden for a node in the cluster by setting a different value for the field in the nodeConfig section for a node.
commonNodeConfig	enabled	true false	Node is active Node is inactive

commonNodeConfig	role	DQC DQE	DQC node DQE node
commonNodeConfig	hostname	<host name> or <ip address>	Host name or ip address for node
commonNodeConfig	clientPort	valid port number	Port number to be used by the JDBC Driver for communicating with the RapidsDB cluster. The default is 4333.
commonNodeConfig	clusterPort	valid port number	Port number to be used for internal communication between nodes in the RapidsDB cluster. The default is 4334.
commonNodeConfig	seEnabled	true false	RapidsSE is enabled on that node RapidsSE is disabled on that node The default is false
commonNodeConfig	seArgs	string	A string of optional command line arguments to be passed to RapidsSE when it is started. This could include things like memory segment size, region prefix name, etc. Refer to section xxxx for the supported command line arguments. No default
commonNodeConfig	sshPort	valid port number	The port number that the SSH daemon is listening on. The default is 22.
commonNodeConfig	sshUsername	<user name>	User name that RapidsDB software is running under
commonNodeConfig	sshPathToIdentity File	~/.ssh/id_rsa	Path to ssh private key file
commonNodeConfig	installationDir	<directory name>	Fully qualified path name to the installation directory eg /opt/rdp
commonNodeConfig	workingDir	<directory name>/current	Fully qualified path name to the current release of the RapidsDB software eg. /opt/rdp/current
commonNodeConfig	startupCommand	sh ./startDqx.sh	Command to start the node
commonNodeConfig	shutdownComma nd	sh ./stopDqx.sh	Command to stop the node
nodeConfigs			Defines the setup of the RapidsDB cluster. There will be one set of values for each node in the cluster.
nodeConfigs	enabled	true false	Node is active Node is inactive
nodeConfigs	name	<node name>	The name to be used by the RapidsDB cluster to identify this node. This name can be any string and does not have to match the Linux node name.

			This field must be specified for each node in the cluster
nodeConfigs	role	DQC DQE	DQC node DQE node This field must be specified for each node in the cluster
nodeConfigs	hostname	<host name> or <ip address>	Host name or ip address for node This field must be specified for each node in the cluster
nodeConfigs	clientPort	valid port number	Port number to be used by the JDBC Driver for communicating with the RapidsDB cluster. The default is 4333.
nodeConfigs	clusterPort	valid port number	Port number to be used for internal communication between nodes in the RapidsDB cluster. The default is 4334.
nodeConfigs	seEnabled	true false	RapidsSE is enabled on that node RapidsSE is disabled on that node The default is false **RapidsSE is not supported for this release so this option must not be changed.**
nodeConfigs	seArgs	string	A string of optional command line arguments to be passed to RapidsSE when it is started. This could include things like memory segment size, region prefix name, etc. Refer to section 7.4 for the supported command line arguments. **RapidsSE is not supported for this release so this option must not be changed.**
nodeConfigs	sshPort	valid port number	The port number that the SSH daemon is listening on. The default is 22.
nodeConfigs	sshUsername	<user name>	User name that RapidsDB software is running under
nodeConfigs	sshPathToIdentityFile	~/.ssh/id_rsa	Path to ssh private key file
nodeConfigs	installationDir	<directory name>	Fully qualified path name to the installation directory eg /opt/rdp
nodeConfigs	workingDir	<directory name>/current	Fully qualified path name to the current release of the RapidsDB software eg. /opt/rdp/current

nodeConfigs	startupCommand	sh ./startDqx.sh	Command to start the node
nodeConfigs	shutdownCommand	sh ./stopDqx.sh	Command to stop the node

Below is the content for the cluster.config.sample file:

```
{
  "commonNodeConfig": {
    "enabled"               : true,
    "role"                  : "DQE",
    "clientPort"            : 4333,
    "clusterPort"           :  4334,
    "seEnabled"             : false,
    "seArgs"                : "",
    "sshPort"               : 22,
    "sshUsername"           : "rapids",
    "sshPathToIdentityFile" : "~/.ssh/id_rsa",
    "installationDir"       : "/opt/rdp",
    "workingDir"            : "/opt/rdp/current",
    "startupCommand"        : "sh ./startDqx.sh",
    "shutdownCommand"       : "sh ./stopDqx.sh"
  },

  "nodeConfig": [
    {
      "name"         : "NODE1",
      "role"         : "DQC",
      "hostname"     : "192.168.1.1"
    },
    {
      "name"         : "NODE2",
      "hostname"     : "192.168.1.2"
    },
    {
      "name"         : "NODE3",
      "hostname"     : "192.168.1.3"
    }
  ]
}
```

The following will configure the RapidsDB Cluster with the DQC on NODE1 and two DQE nodes on NODE2 and NODE3. Note that the entry, "role" : "DQC", was added to the cluster.config file for NODE1 to indicate that this is the DQC node. All of the other nodes in the cluster default to being DQE nodes as specified by the `"role" : "DQE"` setting in the `"commonNodeConfig"` section.

1. Log in to the DQC node as the rapids user
2. Copy the sample cluster.config file to cluster.config:

- cd /opt/rdp
- cp current/cfg/cluster.config.sample cluster.config

3. Edit cluster.config and set up the ip addresses for the DQC and DQE nodes:
    - Change the hostname entry for the DQC node to 192.168.1.1
    - Change the hostname entry for the first DQE node to 192.168.1.2
    - Change the hostname entry for the second DQE node to 192.168.1.3

The resulting file should be as shown below:

```
{
  "commonNodeConfig": {
    "enabled"               : true,
    "role"                  : "DQE",
    "clientPort"            : 4333,
    "clusterPort"           : 4334,
    "seEnabled"             : false,
    "seArgs"                : "",
    "sshUsername"           : "rapids",
    "sshPathToIdentityFile" : "~/.ssh/id_rsa",
    "installationDir"       : "/opt/rdp",
    "workingDir"            : "/opt/rdp/current",
    "startupCommand"        : "sh ./startDqx.sh",
    "shutdownCommand"       : "sh ./stopDqx.sh"
  },

  "nodeConfig": [
    {
      "name"         : "NODE1",
      "role"         : "DQC",
      "hostname"     : "192.168.1.1"
    },
    {
      "name"         : "NODE2",
      "hostname"     : "192.168.1.2"
    },
    {
      "name"         : "NODE3",
      "hostname"     : "192.168.1.3"
    }
  ]
}
```

Alternatively, the host names could be used instead of the ip addresses for the nodes as shown in the example below:


```
{
  "commonNodeConfig": {
    "enabled"               : true,
    "role"                  : "DQE",
    "clientPort"            : 4333,
    "clusterPort"           : 4334,
    "seEnabled"             : false,
    "seArgs"                : "",
    "sshUsername"           : "rapids",
    "sshPathToIdentityFile" : "~/.ssh/id_rsa",
    "installationDir"       : "/opt/rdp",
    "workingDir"            : "/opt/rdp/current",
    "startupCommand"        : "sh ./startDqx.sh",
    "shutdownCommand"       : "sh ./stopDqx.sh"
  },

  "nodeConfig": [
    {
      "name"         : "NODE1",
      "role"         : "DQC",
      "hostname"     : "boray01"
    },
    {
      "name"         : "NODE2",
      "hostname"     : "boray02"
    },
    {
      "name"         : "NODE3",
      "hostname"     : "boray03"
    }
  ]
}
```

In the example above the host name boray01 is for ip address 192.168.1.1, and boray02 is for ip address 192.168.1.2, etc. It is also possible to use the host name as the name for the RapidsDB node, for example, instead of "NODE1" the "name" field could have been set to "boray01".

6.2 Starting Zookeeper

From the DQC node (or whichever node(s) Zookeeper was installed on) log in as the rapids user and execute the following:

- cd /opt/zookeeper/bin
- ./zkServer.sh start

6.3 Licensing

6.3.1 License Files

In order to use the RapidsDB software you must have a valid license file that was provided to you from Boray Data. Contact your local Boray Data Sales office if you have questions about getting a valid license. A license file contains the following information:

- license.version – the version number of the license file
- license.type – the type of license file, either commercial or trial (see below for more details)
- user.name – the company name that the RapidsDB cluster is licensed to.
- user.id – the customer id number.
- rapidsdb.version – the maximum version number of the RapidsDB software that can be used with this license. A value of "-1" indicates that the license is valid for all versions of RapidsDB software.
- allow.expiration_date – the expiration date for the license. A date of 9999-12-31 indicates that the license has no expiration date.
- allow.number_of_cpu – the total number of cores across the cluster that the RapidsDB cluster is licensed to run on.
- signature - the License signature that is generated when the license is created and is used to validate that the license file has not been altered.

There are two types of license files:

- Commercial License – this is the typical license and allows the user full access to the system. Typically, a commercial license will be issued for a 12 month period and the license will then need to be renewed and the new license installed (see section 6.3.5 below).
- Trial License – this is issued to allow a customer to evaluate the RapidsDB software and the license will have a one month expiration date. A trial license can be replaced at any time with another trial license or with a new commercial license (see section 6.3.5 below).

Below are sample Commercial and Trial License files.

Sample commercial license file:


```
# RapidsDB License
# (c) Boray Data Technology Co, Ltd

# License file version
license.version=1.0

# License type
license.type=commercial

# Licensee information
user.name=ABC Company
user.id=1479342161

# Maximum supported RapidsDB version
rapidsdb.version=-1

# License expiry date
allow.expiration_date=2020-10-14

# Maximum number of logical CPU cores in the cluster
allow.number_of_cpu=64

signature=K8b8OjtOVo7tJM1EHeq3UeHF8XthR7Rcuz5cqSBgHR42rrSviyiUGYXPKu2d/xQ/iTJZV5brf6iZixownJ5TgCkatkcIfIyAU685/
+IAKRoCOc+zVgqzZMuu1dePLwusHrgYLwY1NB+H6NUOjPh4KalpHMotrxEZ5hqeAhL9jec=
```

Sample trial license file:

```
# RapidsDB License
# (c) Boray Data Technology Co, Ltd

# License file version
license.version=1.0

# License type
license.type=trial

# Licensee information
user.name=Trial Company
user.id=2105851459

# Maximum supported RapidsDB version
rapidsdb.version=-1

# License expiry date
allow.expiration_date=2019-11-14

# Maximum number of logical CPU cores in the cluster
allow.number_of_cpu=24

signature=AiwrJKzwLY4CqOOzCLkPI8Z5tnniGaOcRfhuQGYithOQWwVlmoquUkV+5lTKTtelFryZm8QvYvmTH+/pbUqHljwuY6fNDYjwGLpfc
q+cSWld5VHJipCmMgWvTxhQcbKdRrkWdvSMF/JehnOY4vYU9J+jJNU9Q8uFF55SyIeuTQY=
```

6.3.2 License Checking

The table below describes the licensing checks performed by the RapidsDB software and the actions taken based on the results of the license checks. There are two types of action that can be taken by the RapidsDB system:

- Warning - results in a message being logged to the dqx.log file in the RapidsDB current directory (eg. /opt/rdp/current), but the RapidsDB cluster continues running
- Fatal – results in the RapidsDB cluster either failing to start if the error occurs during startup, or causes the RapidsDB cluster to be stopped if the error occurs after the RapidsDB cluster was successfully started.

In the table below, the action applies to both the Commercial and Trial licenses unless indicated otherwise.

Type of error	RapidsDB action when encountered on start up	RapidsDB action when encountered after cluster has started
No license file can be found.	Fatal	Warning (commercial) Fatal (trial)
RapidsDB cannot gather necessary prerequisites to be compared to license values (e.g., current number of CPUs in the cluster).	Fatal	Warning
The number of cores on the node exceeds the licensed number of CPU cores in the cluster.	Warning	Warning
The license has expired	Warning (commercial) Fatal (trial)	Warning (commercial) Fatal (trial)
The version of the license file is greater than that which the RapidsDB software can support.	Fatal	Warning (commercial) Fatal (trial)
The license and signature do not match.	Fatal	Warning (commercial) Fatal (trial)
The version of RapidsDB is greater than the maximum RapidsDB version specified in the license.	Fatal	Warning (commercial) Fatal (trial)
A trial license has previously been used on a future date than the current date, potentially indicating clock tampering. This error is not applicable to commercial licenses.	Fatal (trial)	Fatal (trial)
An internal error occurs while trying to validate the license.	Fatal	Warning

Table 1- Licensing errors and how they are handled by RDP.

The license checking will be performed at system startup and every hour after the RapidsDB cluster has been successfully started.

6.3.3 Trial License Limitations

The following limitations apply when using a trial license:

1. The only Connector available is the MOXE Connector, all other Connectors are disabled
2. The maximum size of a MOXE database is 4GB per node (set using the mem_per_node option – refer to section 7.10.4 for more information)

6.3.4 Installing a License

The license file is installed as part of the RapidsDB installation process as described in section 6.4.1.

6.3.5 Updating a License File

A new license file can be installed on a running RapidsDB cluster by using the COPY_LICENSE action of the bootstrapper as described in the steps below:

1. Copy the new license file to the node where the RapidsDB DQC is running. The new license file can be copied to any directory on the DQC node. For example, the license file was copied to the location /opt/rdp/license_2019_10.lic
2. On the DQC node, run the bootstrapper COPY_LICENSE action as shown below (assuming that the RapidsDB installation directory is /opt/rdp):
    - cd /opt/rdp
    - ./bootstrapper.sh -a COPY_LICENSE -l /opt/rdp/license_2019_10.lic

The COPY_LICENSE action takes on parameter "-l <path to new license file>"

The new license file will get installed into the RapidsDB installation directory on each of the RapidsDB cluster with the file name "rapids.lic".

Below is an example log of the COPY_LICENSE action:

```
[rapids@zeus current]$ ./bootstrapper.sh -a COPY_LICENSE -l /opt/rdp/license_2019_10.lic
2019-10-17 20:31:48,331 [bootstrapper.Bootstrapper] INFO : Retrieving cluster definition...
2019-10-17 20:31:49,279 [bootstrapper.Bootstrapper] INFO : Validating the license file at:
/opt/rdp/license_2019_10.lic
2019-10-17 20:31:50,821 [license.LicenseService] INFO : License check successful: /opt/rdp/license_2019_10.lic
2019-10-17 20:31:50,822 [license.LicenseService] INFO : This product is licensed to ABC Company <ID:
1479342161>
2019-10-17 20:31:50,822 [cluster.Cluster] INFO : Copying license file /opt/rdp/license_2019_10.lic to all nodes
as rapids.lic.
```

6.4 Performing the Installation Process

6.4.1 Initial Installation

Log into the DQC node where RapidsDB is installed and navigate to the directory under which it was installed. This is usually /opt/rdp/current.

- cd /opt/rdp/current

Located in this directory are the files required to install the RapidsDB Cluster software. RapidsDB writes a number of files into this directory as a part of the installation process that are required for successful operation:

- boostrapper.sh
  This is an executable shell script that performs many of the startup, shutdown and installation functions. It should not be deleted or tampered with.
- rapids-shell.sh
  This is an executable shell script that brings up a rapids> prompt thereby allowing you to interact with the system.
- version.txt
  This file records the version number of the RapidsDB software and is used by the licensing check process. If this file is missing during a licensing check, an error message will be printed to the dqx.log. It should not be deleted or tampered with.
- startDqx.sh

This is an executable shell script that is called by the boostrapper to start up a node in the cluster during the cluster start-up process. It is referenced in the cluster.config in the key “startupCommand”. This file should not be deleted or tampered with.

- stopDqx.sh
  This is an executable shell script that is called by the bootstrapper to stop a node in the cluster during the shutdown process. It is referenced in the cluster.config in the key “shutdownDommand”. This file should not be deleted or tampered with.

The following command will write the cluster.config file to zookeeper, check whether the user rapids can ssh into every node, copies the run file to the directory indicated in the cluster.config, explodes the run file creating a new directory and creates the symbolic link to current. To perform these steps, the following command is used:

- ./bootstrapper.sh -a install -i <path to installer file> -l <path to license file>

Where

<path to install run file> is the path to the RapidsDB Installer file (see 5.7)

<path to license file> is the path to the license file for this RapidsDB Cluster (see 6.3)

For example:

- ./bootstrapper.sh -a install -i ../rdp-installer-4.0-0.run -l ../license_2019_10.lic

All of these steps require the rapids user to have ssh and write permissions on the target nodes, as listed in the cluster.config, to be able to access the nodes, write files and create directories.

The most common errors that can occur with this command are either caused by not having passwordless-ssh configured correctly or by not having the correct file permissions on the target installation directory. Problems in these areas can be eliminated by doing the following:

1. Attempt to ssh to each node in the RapidsDB cluster as defined by the "hostname" setting in the cluster.config file. Using the first example cluster.config file from section 6.1.2
   - ssh 192.168.1.1
   - exit
   - ssh 192.168.1.2
   - exit
   - ssh 192.168.1.3
   - exit

   If these tests fail then there is either a problem with the passwordless-ssh setup or the "hostname" is incorrect in the cluster.config file for one or more the nodes.

   Replace the ip addresses above with host names if host names were used in the cluster.config file.

2. If the above were successful then check the file permissions on the target installation directory as specified by the "installationDir" setting in the cluster.config file for each of the other nodes:
    - ssh 192.168.1.2 'mkdir /opt/rdp/test'
    - ssh 192.168.1.2 'rmdir /opt/rdp/test'
    - ssh 192.168.1.3 'mkdir /opt/rdp/test'
    - ssh 192.168.1.3 ‘rmdir /opt/rdp/test’

    If these tests fail then there was either a problem with the file permissions on the target installation directory or the target installation directory does not exist or the "installationDir" setting in the cluster.config file is not correct.

    Replace the ip addresses above with host names if host names were used in the cluster.config file.

If all of the above tests are successful then check the following:

1. Check that the path name to the ssh private key file as specified by the "sshPathToIdentityFile" setting in the cluster.config file is correct (see 6.1.2)

6.4.2 Updating the RapidsDB Cluster Software

Updating a RapidsDB cluster follows a similar process to an initial install with a few important exceptions. The RapidsDB software cannot be updated while RapidsDB is running. Therefore, it must be shutdown first.

Follow these steps to update the RapidsDB Cluster software:

1. Stop the RapidsDB Cluster as described in section 6.6
2. Download the Installer file as described in section 5.7
3. Install the new software as described in section 6.4.1
4. If you are running any Connectors that required JDBC Drivers to be installed on any nodes in the RapidsDB Cluster, then those JDBC Driver jar files must be copied from the lib directory from the prior release to the lib directory for the current release. This needs to be done on each node in the RapidsDB Cluster where there was a JDBC Driver jar file previously installed.

During an update installation, a new directory with the new version number will be created and the symbolic link to "current" will point to the new directory. All data, such as logs, will be preserved in the previous directory.

6.5 Starting the RapidsDB Cluster

6.5.1 Start Command

To start the RapidsDB cluster, a user must be logged in as the rapids user and be in the “current” directory which is by default located at /opt/rdp/current on the node that is designated the DQC in the cluster config. To navigate to this directory, issue the following command:

- cd /opt/rdp/current

Start the RapidsDB Cluster by executing the following command that will start up all nodes in the cluster:

- ./bootstrapper.sh -a start [-q <limit>] [--jvm_settings VAL]

Where,

- <limit>: is the limit on the number of queries that can be running concurrently. Any queries submitted beyond the query limit will be queued. The default value is 10.
- --jvm-settings can be used to pass in run-settings to RapidsDB. The VAL parameter specifies a specific run setting. At this time, the only run-setting that is supported is
  - "-DrefreshTimeout=<timeout value in milliseconds>"

When issuing the start command, a number of checks occur as the RapidsDB Cluster is brought up:

- A check is made to ensure there is a valid cluster.config available in zookeeper. (The cluster.config is written to zookeeper during install – see 6.4.1)
- A ssh check occurs to ensure that all nodes in the cluster can communicate with each other and with themselves
- A check for a valid license is done on each node in the cluster. If a valid license is not found, an error message will be returned and RapidsDB cluster will not start up until the issue is resolved (see 6.3).
- A check that each node in the cluster has the same version of the RapidsDB software. If it is found that a node does not have the same version as the DQC node, an error message will be returned and the user must resolve the issue (usually by running the installation process again).
- A check to ensure that an instance of RapidsDB is not already running. If it is, an error message will be returned and the user will need to shutdown the running instance before starting a new one (see 6.6).

If the above checks pass, the bootstrapper will start a RapidsDB instance on each node listed in the cluster.config.

If any of the above checks fail, an exception message will print to the console and the start process will be halted.

A successful start will produce output to the console that looks similar to below:

```
2019-10-14T19:45:34,416 [cluster.Cluster      ] INFO : Checking for a valid license on each node in the
cluster...
2019-10-14T19:45:37,364 [cluster.Cluster      ] INFO :       NODE1  [     OK]  Path: rapids.lic
2019-10-14T19:45:37,448 [cluster.Cluster      ] INFO :       NODE2  [     OK]  Path: rapids.lic
2019-10-14T19:45:37,576 [cluster.Cluster      ] INFO :       NODE3  [     OK]  Path: rapids.lic
2019-10-14T19:45:37,578 [cluster.Cluster      ] INFO : Licenses have been validated on each node.
2019-10-14T19:45:39,546 [cluster.Cluster      ] INFO : All nodes have the same software version "4.0-0".
```



```
2019-10-14T19:45:44,027 [cluster.Cluster       ] INFO : Processes for 3 DQX nodes in the cluster are now loading.
Waiting for nodes to start . . . . .
2019-10-14T19:45:49,030 [bootstrapper.Bootstrapper] INFO : Now running a health check to see if all nodes started up correctly.
2019-10-14T19:45:49,063 [brick.works          ] INFO : JVM ea=false max=27305MB ver=1.8.0_171 vm=25.171-b10
Status of NODE1                                [OK]
Status of NODE2                                [OK]
Status of NODE3                                [OK]
Status of NODE4                                [OK]
Health check complete.
All nodes are responding.
```

There is one more file that is created during the installation process, the dqx.log file. This is a log of all significant actions taken by the software for not only installation and start-up but also for normal operation. This log allows users to see the state of the software, what version is currently being run, and any errors that are logged.

6.5.2 Diagnosing Startup Problems

6.5.2.1 Network Error

If a network error is reported follow these steps:

1. For any node that has an error reported verify that the node can be reached by doing an ssh command to that node. For example, using the cluster.config from section 6.1.2. the boostrapper reported a problem with NODE2:
    a. ssh 192.168.1.2 'ls opt/rdp/current'
    If there is a problem reaching the target node, then fix the problem, and then stop (see 6.6) the RapidsDB Cluster and then restart the RapidsDB Cluster.
2. If the above was successful then verify that the "clusterPort" in the cluster.config file is correct:
    a. If the port number is incorrect then correct the port number in the cluster.config file and update the RapidsDB configuration (see 6.7), and start the RapidsDB Cluster.
3. If the "clusterPort" is correct verify that the port is not currently open on that node:
    a. netstat -a | grep <port number> where <port number> is the "clusterPort" from the cluster.config file
4. If the port number is currently in use, then check to make certain that the RapidsDB DQC/DQE instance for that node is not currently running:
    a. cat /opt/rdp/current/dqx.pid
    If this file exists, then this indicates that the DQC/DQE instance for that node is still running, so follow the steps to stop the RapidsDB cluster (see 6.6) and then start the RapidsDB Cluster.

b. If the file does not exist then check which process is using this port and resolve the conflict: netstat -ltnp | grep -w ':<port number>'
After resolving the problem,

5. If none of the above then check the dqx.log file on the node(s) which failed to start for more details on the error.

6.5.2.2 Non-network Error

Refer to the error message and the dqx.log file on the node that failed for more information on the error.

6.5.3 Changing the Password for rapids user

After successfully starting the RapidsDB cluster it is highly recommended that the password for the "rapids" user is changed from the default of "rapids". The password can be changed from the rapids-shell after logging in as the rapids user and then issuing an ALTER USER command (see 7.8):

Below is an example showing this.

First, we log in using the default password of 'rapids':

```
[rapids@boray01 current]$ ./rapids-shell.sh -p 7335
Please enter a username > rapids
Please enter the password for user 'RAPIDS' >
rapids>
```

Next, we change the password:

```
rapids > alter user rapids password 'rapids123';
0 row(s) returned (0.60 sec)
```

Now we exit and try to log back in using the original password, which fails:

```
rapids > exit
[rapids@boray01 current]$ ./rapids-shell.sh -p 7335
Please enter a username > rapids
Please enter the password for user 'RAPIDS' >
java.sql.SQLException: Authentication failed.
```

Finally, we use the changed password:

```
Please enter a username > rapids
Please enter the password for user 'RAPIDS' >
rapids > alter user rapids password 'rapids';
rapids>
```


NOTE: After the password to the rapids user is changed there is no way to recover the password if it is lost, and so it is critical that the password is stored somewhere secure otherwise there will be no way to log into the system using the rapids userid, which is currently required for doing certain administrative operations.

6.6 Stopping the RapidsDB Cluster

To stop the RapidsDB cluster, a user must be logged in as the rapids user and be in the "current" directory which is by default located at /opt/rdp/current on the node that is designated the DQC in the cluster config. To navigate to this directory, issue the following command:

- cd /opt/rdp/current

To stop the RapidsDB cluster, issue the following command:

- ./bootstrapper.sh -a stop

The bootstrapper will send a stop message to every node in the cluster using the instructions indicated in the cluster.config shutdownCommand entry. The acutal instructions for stopping the RapidsDB Cluster can be found in the stopDQX.sh file in the current directory.

Here is a sample output from the stop command:

```
[rapids@boray01 current]$ ./bootstrapper.sh -a stop
2019-10-30 14:00:07,108 [cluster.Cluster] INFO : Stopped 3 nodes in the cluster.
```

NOTE:

Shutting down the cluster will result in data loss if that data is held in memory. If data is held in the MOXE internal data store, it can be persisted to disk using the unload/reload options (see section 8.5 for more details.)

6.7 Updating the RapidsDB Configuration

6.7.1 Adding or Removing Nodes

If nodes are added or removed from the RapidsDB Cluster, then the following steps must be followed:

1. Stop the RapidsDB Cluster if it is currently running (see 6.6)
2. Update the cluster.config file to reflect the new node configuration as described in section 6.1.2)
3. Repeat the installation process (see 6.4.1)

4. The new RapidsDB Cluster can now be started

 If nodes are added or removed from the RapidDB Cluster, then any MOXE "unload" files (see 8.5.1) that were created to backup the data in MOXE tables will no longer be valid and the data will be lost. In this case, any data to be persisted will need to be persisted to an external data source (such as HDFS) and then recreated (using INSERT … SELECT) after the cluster is reconfigured.

6.7.2 Changing Configuration Settings

If a change is required in the cluster config (for example, ip addresses are changed, node names are changed, port numbers are changed) then the following steps must be followed:

1. Stop the RapidsDB Cluster (see 6.6)
2. Navigate to the RapidsDB Installation directory as defined by the "installationDir" in the cluster.config file (see 6.1.2). By default this would be /opt/rdp:
    a. cd /opt/rdp
3. Edit the cluster.config file and make the required changes
4. Navigate to the "current" directory:
    a. cd current
5. Run the following command to update the RapidsDB configuration, which will cause the new configuration to be written to Zookeeper:
    a. ./bootstrapper.sh -a populate
6. Start the RapidsDB Cluster with the new configuration (see 6.5)

Below is an example:

```
[rapids@boray01 current]$ ./bootstrapper.sh -a populate
2019-10-30 14:32:07,237 [cluster.Cluster] INFO : {
  "commonNodeConfig": {
    "enabled"               : true,
    "role"                  : "DQE",
    "clusterPort"           : 7334,
    "clientPort"            : 7335,
    "sshUsername"           : "rapids",
    "sshPathToIdentityFile" : "~/.ssh/id_rsa",
    "installationDir"       : /opt/rdp",
    "workingDir"            : /opt/rdp/current",
    "startupCommand"        : "sh ./startDqx.sh",
    "shutdownCommand"       : "sh ./stopDqx.sh",
    "seArgs"                : "--port 7336 --size 8GB --
partitionsPerNode 8 --regionPrefix /SETest_R4",
    "seEnabled"             : false
  },
```



```
  "nodeConfig": [
    {
      "name"                : "boray01",
      "role"                : "DQC",
      "hostname"            : "boray01"
    },
    {
      "name"                : "boray02",
      "hostname"            : "boray02"
    }
  ]
}


2019-10-30 14:32:07,457 [bootstrapper.Bootstrapper] INFO : Config file
has been written to ZooKeeper.
```

6.8 Checking Status of RapidsDB Cluster

To check whether the nodes in the RapidsDB Cluster are running, you can do either of the following:

1. Run the bootstrapper healthcheck option. A user must be logged in as the rapids user and be in the "current" directory which is by default located at /opt/rdp/current on the node that is designated the DQC in the cluster config. To navigate to this directory, issue the following command:
    1. cd /opt/rdp/current

    Now issue the following command:

    - ./bootstrapper.sh -a healthcheck

    Below is the sample output:

```
[rapids@boray01 current]$ ./bootstrapper.sh -a healthcheck
2019-10-30 14:48:58,038 [brick.works ] INFO : JVM ea=false max=27305MB ver=1.8.0_121
vm=25.121-b13
Status of NODE1                                [OK]
Status of NODE2                                [OK]
Status of NODE3                                [OK]
Health check complete.
All nodes are responding.
```


2. Execute jps on each node in the RapidsDB Cluster. You should see an entry for a "Dqx" process which will be the RapidsDB instance running on that node along with the pid for that instance.

   Below is the sample output:

```
[rapids@boray01 current]$ jps
16435 DataNode
27459 Dqx
16612 NodeManager
27860 Jps
38748 QuorumPeerMain
```

If the RapidsDB Cluster is not running correctly then consult the RapidsDB Cluster log file, dqx.log, on each node in the cluster. The dqx.log file is located in the current directory, eg /opt/rdp/current/dqx.log

6.9 Stopping Zookeeper

Log in to the node where Zookeeper was installed (see 5.8) and execute the following commands:

a. cd /opt/zookeeper/bin
b. ./zkServer.sh stop

7 Managing RapidsDB

7.1 Command-line Interface: rapids-shell

The command-line interface to RapidsDB is provided through the rapids-shell.

7.2 Running the rapids-shell

7.2.1 Running the RapidsDB shell Locally

The RapidsDB shell, rapids-shell.sh, can be run from any node in the RapidsDB Cluster, and is run from the RapidsDB cluster installation directory (eg /opt/rdp/current):

1. cd /opt/rdp/current
2. ./rapids-shell.sh

You will be prompted for a username and password before seeing the rapids prompt:

```
[rapids@boray05 current]$ ./rapids-shell.sh
Please enter a username > rapids
```



```
Please enter the password for user 'RAPIDS' >
rapids >
```

3. You should then be able to execute SQL queries or any of the supported rapids-shell command (refer to the Rapids-shell User Guide for more details). For example:

```
rapids > select * from catalogs;
CATALOG_NAME
------------
HADOOP
RAPIDS
rapidsse

3 row(s) returned (0.01 sec)
```

7.2.2 Running the RapidsDB shell Remotely

The rapids-shell can also be run remotely from any node that has TCP/IP connectivity to the RapidsDB Cluster. To install the rapids-shell on a remote system use the following steps:

1. Copy the rapids-shell-<version>.zip file from the 'shell' directory located in the RapidsDB installation directory on any node in the RapidsDB Cluster to the target system
2. Unzip the rapids-shell-<version>.zip file
3. The rapids-shell can then be started by running either the rapids-shell.sh file on Linux or the rapids-shell.bat file on Windows.
4. Refer to the Rapids-shell User Guide for more information.

7.2.3 Authentication of the RapidsDB shell

When the RapidsDB shell is started normally it will interactively ask for the username and password to be used for authentication. When entering the username interactively, rapids-shell will treat the name like a SQL object identifier by folding the name to uppercase unless it is surrounded by double quotes, in which case the case will be preserved. Care must be taken if case-sensitive usernames are used.

The password being entered will be treated as case sensitive and does not require any quoting.

To avoid being prompted to enter a username and password when invoking rapids-shell, simply define the following shell or environment variables when starting rapids-shell:

- RDP_USERNAME
- RDP_PASSWORD

Shell variables can be defined on the same line used to invoke the rapids-shell. They will only exist for the rapids-shell process.

Example 1:


```
$ RDP_USERNAME=rapids RDP_PASSWORD=rapids ./rapids-shell.sh

rapids > select current_user from tables limit 1;

?1

--

RAPIDS


1 row(s) returned (0.02 sec)
```

Alternatively, the variables can be exported to be environment variables before the rapids-shell is invoked.

Example 2:

```
$ export RDP_USERNAME=rapids

$ export RDP_PASSWORD=rapids

$ ./rapids-shell.sh

rapids > select current_user from tables limit 1;

?1

--

RAPIDS


1 row(s) returned (0.02 sec)
```

When defining the shell or environment variables for the username, please note that it will be treated the same as if it was entered interactively. i.e., unquoted usernames will be folded to uppercase. However, entering double quotes around a shell/environment variable is not as straight forward as it seems because the unix shell will first interpret the quotes and remove them before they are seen by the rapids-shell. As a result, the double quotes need to be escaped by single quotes that will be removed by the unix shell.

Example 3:

```
$ export RDP_USERNAME='"john"'      ← note outer single quotes and inner
double quotes.
```



```
$ export RDP_PASSWORD=john

$ ./rapids-shell.sh

rapids > select current_user from tables limit 1;

?1

--

john                                ← note case sensitive name.


1 row(s) returned (0.02 sec)
```

Please refer to the Rapids-shell User Guide for more details.

To use Kerberos authentication with the RapidsDB shell, please refer to the Rapids-shell User Guide for details.

7.2.4 Cancelling Queries

There are two ways that a query can be cancelled from the rapids-shell:

1. Ctrl-k - A long running query that was executed via rapids-shell can be interrupted by pressing Control-K. This will send a message to RapidsDB on a new connection to cancel the long running query, and return control to the user.

   Example:

```
rapids > select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price,
    >       sum(l_extendedprice*(1-l_discount)) as sum_disc_price, sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
    >       avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*) as count_order
    >  from LINEITEM
    > where l_shipdate <= timestamp '1998-12-01 00:00:00' - interval '90' day
    > group by l_returnflag, l_linestatus
    > order by l_returnflag, l_linestatus;
```

   Press Ctrl-k:

```
[CANCELLING QUERY]
Cancellation of query SSN_1@BORAY01#33 requested.
```



```
java.sql.SQLException: System exception:
com.rapidsdata.common.exceptions.RdpRemoteException:
com.rapidsdata.plan.exceptions.ExecCanceledException: Canceled locus=BORAY01
```

2. Using the CANCEL QUERY command

   The syntax for the CANCEL QUERY command is:

```
CANCEL QUERY [ IF EXISTS ] <queryId>;
```

- The <queryId> can be found from the QUERIES metadata table (see the RapidsDB User Guide for more information on the QUERIES table).
- Queries can be cancelled on remote nodes as well as the local node.

Example: In the example below the query is initiated on Session 1, and then cancelled from a different session, Session 2.

Session 1:

```
rapids > select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as
sum_base_price,
    >       sum(l_extendedprice*(1-l_discount)) as sum_disc_price, sum(l_extendedprice*(1-
l_discount)*(1+l_tax)) as sum_charge,
    >       avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as
avg_disc, count(*) as count_order
    >  from LINEITEM
    > where l_shipdate <= timestamp '1998-12-01 00:00:00' - interval '90' day
    > group by l_returnflag, l_linestatus
    > order by l_returnflag, l_linestatus;
```

Session 2:

```
rapids > select * from queries;
QUERY_ID        SESSION_ID     NODE     USERNAME   START_TIME            QUERY_TEXT
--------        ----------     ----     --------   ----------            ----------
SSN_3@BORAY01#7   SSN_3@BORAY01   BORAY01   RAPIDS    2020-07-30 21:17:11.608   select *
from queries;
SSN_1@BORAY01#32   SSN_1@BORAY01   BORAY01   RAPIDS    2020-07-30 21:17:09.838   select
l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price,
sum(l_extendedprice*(1-l_discount)) as sum_disc_price, sum(l_extendedprice*(1-
l_discount)*(1+l_tax)) as sum_charge, avg(l_quantity) as avg_qty, avg(l_extendedprice) as
avg_price, avg(l_discount) as avg_disc, count(*) as count_order from LINEITEM where l_shipdate
```



```
<= timestamp '1998-12-01 00:00:00' - interval '90' day group by l_returnflag, l_linestatus order by
l_returnflag, l_linestatus;

2 row(s) returned (0.21 sec)
rapids > cancel query SSN_1@BORAY01#32 ;
0 row(s) returned (0.70 sec)
```

Session 1 – this is the exception returned after the query is cancelled:

```
java.sql.SQLException: System exception:
com.rapidsdata.common.exceptions.RdpRemoteException:
com.rapidsdata.plan.exceptions.ExecCanceledException: Canceled locus=BORAY01
com.rapidsdata.common.exceptions.AnException:
com.rapidsdata.common.exceptions.RdpRemoteException:
com.rapidsdata.plan.exceptions.ExecCanceledException: Canceled locus=BORAY01
rapids>
```

7.3 Adding Authenticators – CREATE AUTHENTICATOR

7.3.1 Overview

The CREATE AUTHENTICATOR command is used to add a new authenticator instance to the system. Authenticators exist across all federations. The general format for the CREATE AUTHENTICATOR command is:

```
CREATE AUTHENTICATOR
    [ IF NOT EXISTS ]
    <name>
    TYPE <type>
    [ SET [ ENABLED | DISABLED ] ]
    [ WITH <key> = '<value>' [, … ] ] ;
```

RapidsDB will start with an internal authenticator already existing that can authenticate username and password credentials. This internal authenticator cannot be created or dropped. The CREATE AUTHENTICATOR command supports authenticators of these additional types:

- `kerberos`

The Kerberos authenticator is able to authenticate Kerberos principals.

Support for additional authenticator types (e.g. LDAP) will be added in the future.

When creating an authenticator, the name of the authenticator instance must be unique in the cluster.

If an authenticator is created in the disabled state then any users associated with this authenticator will not be able to login.

The WITH clause is used to specify authenticator-specific options, in key-value pairs. Some authenticator types may require extra information to be specified when they are created. Values in these key-value pairs are typically specified as single quoted strings to preserve case. Keys are often unquoted. Keys and values can also be unquoted so long as they don't contain spaces or other special characters, however unquoted keys and values will be folded to uppercase in accordance with RapidsDB's SQL object identifier handling.

Example:

```
rapids > create authenticator krb type kerberos with realm = 'RDP.COM';
0 row(s) returned (0.10 sec)
```

7.3.2 Creating a Kerberos Authenticator

Kerberos authenticators require additional information to be created successfully. Specifically, a Kerberos authenticator needs to know:

1. The name of the RapidsDB **service principal** for each node.
2. The path to the **keytab file** for this service principal.

This information applies to each node in the cluster. How this information is specified is detailed below.

7.3.2.1 Specifying the Service Principal

The service principal can be specified in the following ways:

1. By providing a local file on each node in the cluster that contains the name of the service principal for that particular node. RapidsDB will look for this file in the path `../rdp.principal` relative to the directory that RapidsDB executes in.

2. By providing a custom property in the `WITH` clause of the `CREATE AUTHENTICATOR` statement that specifies the pattern of the service principal:

   ```
   CREATE AUTHENTICATOR … WITH principal_pattern = '<pattern_string>' ;
   ```

   The `<pattern_string>` must contain the fully qualified principal including realm. Since this `<pattern_string>` is applied to all nodes in the cluster the escape sequence \H can be used to insert the fully qualified hostname of the current node (as displayed by the output of the terminal command `hostname -f`).
   Example:


```
rapids > create authenticator krb type kerberos with
principal_pattern = 'rapidsdb/\H@RDP.COM';
0 row(s) returned (0.33 sec)
rapids > select * from authenticators;
AUTHNAME   TYPE         ENABLED DDL
--------   ----         ------- ---
KRB        KERBEROS        true CREATE AUTHENTICATOR KRB TYPE
KERBEROS WITH PRINCIPAL_PATTERN = 'rapidsdb/\H@RDP.COM';
RDPAUTH    RDP             true CREATE AUTHENTICATOR RDPAUTH TYPE
RDP ;

2 row(s) returned (0.57 sec)
```

On a node with a fully qualified hostname of `myNode.myDomain`, this principal pattern would be turned into a service principal of: `rapidsdb/myNode.myDomain@EXAMPLE.COM`

3. By providing a custom property that specifies the Kerberos realm in the `WITH` clause of a `CREATE AUTHENTICATOR` statement, and letting RapidsDB fill in the service name and host parts of the principal. E.g.:

Example:

```
rapids > create authenticator krb type kerberos with realm =
'RDP.COM';
0 row(s) returned (0.05 sec)
rapids > select * from authenticators;
AUTHNAME   TYPE         ENABLED DDL
--------   ----         ------- ---
KRB        KERBEROS        true CREATE AUTHENTICATOR KRB TYPE
KERBEROS WITH REALM = 'RDP.COM';
RDPAUTH    RDP             true CREATE AUTHENTICATOR RDPAUTH TYPE
RDP ;

2 row(s) returned (0.64 sec)
```

When only the realm is provided the service principal for any given node will be formed using the pattern:

`'rdp/<fully_qualified_host_name>@<realm>'`

where `<fully_qualified_host_name>` is the name of the node including its domain name (e.g. refer to the output of the shell command `hostname -f`). E.g. On a node with a fully qualified hostname of `myNode.myDomain`, this principal pattern would be turned into a service principal of: `rdp/myNode.myDomain@EXAMPLE.COM`

The Kerberos authenticator will look for service principals on each node in this order:

a. `../rdp.principal` file.
b. `principal_pattern` custom property.
c. `realm` custom property.

The easiest way to specify the service principal is to specify the `realm` property when creating the connector and to ensure that the fully qualified name of each node matches the expected service principal name.

7.3.2.2 Specifying the Keytab file

By default, the Kerberos authenticator will look for a keytab file in the path `../rdp.keytab` on each node.

The keytab path can be overridden by specifying a custom property in the `WITH` clause when creating the authenticator.

Example:

```
rapids > create authenticator krb type kerberos with keytab =
'/home/rapids/rdptest/r4/rdp.keytab', realm = 'RDP.COM';
0 row(s) returned (0.04 sec)
rapids > select * from authenticators;
AUTHNAME   TYPE         ENABLED DDL
--------   ----         ------- ---
KRB        KERBEROS        true CREATE AUTHENTICATOR KRB TYPE KERBEROS
WITH KEYTAB = '/home/rapids/rdptest/r4/rdp.keytab', REALM = 'RDP.COM';
RDPAUTH    RDP             true CREATE AUTHENTICATOR RDPAUTH TYPE RDP
;

2 row(s) returned (0.77 sec)
```

The keytab file must exist in this path on each node in the cluster.

7.4 Dropping Authenticators – DROP AUTHENTICATOR

The `DROP AUTHENTICATOR` command is used to remove an instance of an authenticator from the system. The syntax for dropping an authenticator is:

```
DROP AUTHENTICATOR [ IF EXISTS ] <name> [ KEEPING USERS ] ;
```

By default, when an authenticator is dropped all users that are associated with that authenticator are also dropped, unless the `KEEPING USERS` clause is specified. In that case all associated user accounts are kept but those users will not be able to login because their accounts refer to an authenticator that now does not exist. If a new authenticator is created with the same name then those user accounts will

try to use that new authenticator. Alternatively, the user accounts can be altered to associated them with a different authenticator.

It is not possible to drop the internal authenticator.

Example:

```
rapids > create authenticator krb type kerberos with keytab =
'/home/rapids/rdptest/r4/rdp.keytab', realm = 'RDP.COM';
0 row(s) returned (0.04 sec)
rapids > select * from authenticators;
AUTHNAME   TYPE         ENABLED DDL
--------   ----         ------- ---
KRB        KERBEROS        true CREATE AUTHENTICATOR KRB TYPE KERBEROS
WITH KEYTAB = '/home/rapids/rdptest/r4/rdp.keytab', REALM = 'RDP.COM';
RDPAUTH    RDP             true CREATE AUTHENTICATOR RDPAUTH TYPE RDP
;

2 row(s) returned (0.77 sec)
rapids > drop authenticator krb;
0 row(s) returned (0.03 sec)
rapids > select * from authenticators;
AUTHNAME   TYPE     ENABLED DDL
--------   ----     ------- ---
RDPAUTH    RDP         true CREATE AUTHENTICATOR RDPAUTH TYPE RDP ;

1 row(s) returned (0.58 sec)
```

7.5 Altering Authenticators – ALTER AUTHENTICATOR

The ALTER AUTHENTICATOR command is used to change some properties of an authenticator instance that already exists. The syntax for altering an authenticator is:

```
ALTER AUTHENTICATOR <name>
      [ SET [ ENABLED | DISABLED ] ]
      [ WITH <key> = <value>, … ] ;
```

An authenticator can be altered to enable or disable it. When an authenticator is disabled it will deny authentication to any user associated with this authenticator that tries.

The ALTER AUTHENTICATOR command can also be used to change the custom properties that are associated with an authenticator instance. When the WITH clause is specified in an ALTER AUTHENTICATOR statement, any previous keys and values will be discarded and replaced by the new set of keys and values.

Values in these key-value pairs are typically specified as single quoted strings to preserve case. Keys are often unquoted. Keys and values can also be unquoted so long as they don't contain spaces or other

special characters, however unquoted keys and values will be folded to uppercase in accordance with RapidsDB's SQL object identifier handling.

It is not possible to change an authenticator's type with an ALTER AUTHENTICATOR statement. Instead, the authenticator must first be dropped and a new one created. Similarly, it is also not possible to change the name of an authenticator with an ALTER AUTHENTICATOR statement.

Example:

```
rapids > create authenticator krb type kerberos with realm =
'RDP.COM';
0 row(s) returned (0.06 sec)
rapids > select * from authenticators;
AUTHNAME   TYPE         ENABLED DDL
--------   ----         ------- ---
KRB        KERBEROS        true CREATE AUTHENTICATOR KRB TYPE KERBEROS
WITH REALM = 'RDP.COM';
RDPAUTH    RDP             true CREATE AUTHENTICATOR RDPAUTH TYPE RDP
;

2 row(s) returned (0.68 sec)
rapids > alter authenticator krb set disabled;
0 row(s) returned (0.03 sec)
rapids > select * from authenticators;
AUTHNAME   TYPE         ENABLED DDL
--------   ----         ------- ---
KRB        KERBEROS       false CREATE AUTHENTICATOR KRB TYPE KERBEROS
SET DISABLED WITH REALM = 'RDP.COM';
RDPAUTH    RDP             true CREATE AUTHENTICATOR RDPAUTH TYPE RDP
;

2 row(s) returned (1.73 sec)
```

7.6 Adding Users – CREATE USER

The CREATE USER command is used to add new user accounts into RapidsDB. Users exist across all federations. The syntax of the CREATE USER command is:

```
CREATE USER
    [ IF NOT EXISTS ]
    <username>
    [ AUTH <authenticator_name> ]
    [ PASSWORD '<password>' ]
    [ SET [ ENABLED | DISABLED ] ]
    [ WITH <key> = <value> [, … ] ] ;
```


When creating users, the username must be unique within the cluster. The username follows standard object identifier rules in that it will be folded to uppercase unless it is double quoted to preserve case sensitivity.

If an authenticator name is not specified then the user will be created to use the internal RDP authenticator, which requires users to authenticate with a username and password.

The password field is only applicable if the authenticator type requires a password. E.g. Kerberos authentication does not use a password, so it would be an error to specify a password if a Kerberos authenticator was also specified. Users that are created with the internal authenticator must set a password when the user is created.

The WITH clause is used to specify user-specific options, in key-value pairs. These options may be used by the authenticator to affect how it does its authentication. An example is that a user could be created that uses an external authentication system (e.g. Kerberos or LDAP) and the WITH clause is used to specify the external identifier used by the external authentication system for this RapidsDB user. The authenticator can make use of this to set up an automatic mapping from the external user identifier to the RapidsDB username so that when the client connects they only need to specify either the external identifier or their RapidsDB username, not both.

Example:

```
rapids > create user john password 'john';
0 row(s) returned (0.42 sec)
```

7.6.1 Adding Kerberos Users

When a client connects to RapidsDB and wishes to connect with Kerberos, that client will typically supply the Kerberos principal to be authenticated as but not the RapidsDB username. RapidsDB needs to find the username associated with this Kerberos principal by using username mapping or pattern mapping (refer to section 4.7). This mapping can be created by hand or it can be done automatically if the correct option is given when the user is created.

To create a RapidsDB user that uses a Kerberos authenticator and have the mapping between the user's Kerberos principal and their RapidsDB username automatically set up, simply ensure that you specify the user's principal in the WITH clause as follows:

```
CREATE USER <username>
TYPE <kerberos_authenticator_name>
WITH principal = '<kerberos_principal>' ;
```

When RapidsDB sees that this principal key has been set and the user is associated with a Kerberos authenticator then it will automatically add a mapping from the Kerberos principal specified to the username. This can be seen by querying the `username_maps` metadata table after the user has been created.

Example:

```
rapids > create authenticator krb type kerberos with realm =
'RDP.COM';
0 row(s) returned (0.32 sec)
rapids > CREATE USER dave AUTH krb WITH PRINCIPAL='dave@RDP.COM';
0 row(s) returned (0.20 sec)
rapids > select * from username_maps;
ID              USERNAME
--              --------
dave@RDP.COM    DAVE

1 row(s) returned (0.07 sec)
```

Example: in this example the user is initially disabled:

```
rapids > CREATE USER craig AUTH krb SET DISABLED WITH
PRINCIPAL='craig@RDP.COM';
0 row(s) returned (0.06 sec)
rapids > select * from users where USERNAME='CRAIG';
USERNAME     ENABLED AUTHNAME
--------     ------- --------
CRAIG          false KRB

1 row(s) returned (0.04 sec)
```

7.7 Dropping Users – DROP USER

The DROP USER command is used to remove a user from the system. The syntax for dropping a user is:

```
DROP USER [ IF EXISTS ] <username> ;
```

Dropping a user does not affect the authenticator that the user is associated with (if any).

Currently only the initial RAPIDS user can drop other user accounts. Users can drop themselves though.

If a mapping from an external identifier was automatically setup when the user was created (e.g. from Kerberos principal to the user's username) then this mapping will also be deleted when the user is dropped.

Example:

```
rapids > select * from users;
USERNAME    ENABLED AUTHNAME
--------    ------- --------
CRAIG         false KRB
```



```
DAVE          true KRB
RAPIDS        true RDPAUTH

3 row(s) returned (0.02 sec)
rapids > drop user craig;
0 row(s) returned (0.29 sec)
rapids > select * from users;
USERNAME     ENABLED AUTHNAME
--------     ------- --------
DAVE          true KRB
RAPIDS        true RDPAUTH

2 row(s) returned (0.50 sec)
```

7.8 Altering Users – ALTER USER

The ALTER USER command is used to change some properties of an existing user account, such as its password. The syntax for ALTER USER is:

```
ALTER USER <username>
      [ AUTH <authenticator_name> ]
      [ PASSWORD <password> ]
      [ SET [ ENABLED | DISABLED ] ]
      [ WITH <key> = <value>, … ] ;
```

An existing user account can be altered to change the name of the authenticator that it uses by specifying the AUTH clause. Specifying a different authenticator name will invalidate any existing custom keys and values that were previously set in the WITH clause.

If the user account is associated with an authenticator that authenticates with passwords then the PASSWORD clause can be specified to change it. Specifying the PASSWORD clause when the associated authenticator does not use passwords (e.g. Kerberos) will result in an error.

Users can also be altered to be enabled or disabled. A disabled user will not be able to authenticate with RapidsDB.

The ALTER USER command can also be used to change the custom properties that are associated with the user account. These custom properties can be used by the associated authenticator to control how they operate when this user is authenticated. When the WITH clause is specified in an ALTER USER statement, any previous keys and values will be discarded and replaced by the new set of keys and values.

Values in these key-value pairs are typically specified as single quoted strings to preserve case. Keys are often unquoted. Keys and values can also be unquoted so long as they don't contain spaces or other

special characters, however unquoted keys and values will be folded to uppercase in accordance with RapidsDB's SQL object identifier handling.

Example:

```
rapids > create user john password 'john';
0 row(s) returned (1.35 sec)
rapids > alter user john password 'new123';
0 row(s) returned (0.33 sec)
rapids > alter user john set disabled;
0 row(s) returned (0.01 sec)
rapids > select * from users where username='JOHN';
USERNAME     ENABLED AUTHNAME
--------     ------- --------
JOHN           false RDPAUTH

1 row(s) returned (1.22 sec)
```

7.9 User ID Mapping

7.9.1 Automatic User ID Mapping

If the correct properties and authenticator type are specified when the user is created then a mapping between an external user identifier and the user's username will be setup automatically. Similarly, when this user is dropped or altered then this mapping will be removed. How this external identifier is specified is dependent on each authenticator type. E.g. refer to section 7.6.1 for a Kerberos example of this.

These user ID mappings can also be manipulated manually, as explained below.

Example:

```
rapids > CREATE USER dave AUTH krb WITH PRINCIPAL='dave@RDP.COM';
0 row(s) returned (0.20 sec)
rapids > select * from username_maps;
ID             USERNAME
--             --------
dave@RDP.COM   DAVE

1 row(s) returned (0.07 sec)
```

7.9.2 Manually Adding a Username Mapping – ADD USERNAME MAPPING

A direct mapping from an external identifier to a RapidsDB username can be manually added with the command:


```
ADD USERNAME MAPPING '<external_id>' TO <username> ;
```

This will add a new mapping that is visable in the `username_maps` metadata table. The external identifiers are case sensitive, and a new mapping cannot be added if a mapping already exists with this external identifier.

Example:

```
rapids > CREATE USER dave AUTH krb;
0 row(s) returned (0.01 sec)
rapids > select * from username_maps;
0 row(s) returned (0.42 sec)
rapids > ADD USERNAME MAPPING 'dave@RDP.COM' TO dave;
0 row(s) returned (0.02 sec)
rapids > select * from username_maps where username='DAVE';
ID              USERNAME
--              --------
dave@RDP.COM    DAVE

1 row(s) returned (0.07 sec)
```

7.9.3 Manually Removing a Username Mapping – REMOVE USERNAME MAPPING

A direct mapping from an external identifier to a RapidsDB username can be manually removed with the command:

```
REMOVE USERNAME MAPPING '<external_id>' ;
```

The external identifier is case sensitive.

Example:

```
rapids > select * from username_maps where username='DAVE';
ID              USERNAME
--              --------
dave@RDP.COM    DAVE

1 row(s) returned (0.07 sec)
rapids > remove username mapping 'dave@RDP.COM';
0 row(s) returned (0.01 sec)
rapids > select * from username_maps where username='DAVE';
0 row(s) returned (0.78 sec)
```


7.9.4 Setting the Pattern Map – SET PATTERN MAP FILE

The pattern map is set by loading the mappings from a file. This will replace any existing pattern mappings in the cluster. Once loaded, the cluster will persist these pattern maps so they do not need to be reloaded when the cluster is restarted.

The pattern map can be loaded with the following command:

```
SET PATTERN MAP FILE 'path/to/file' ;
```

The pattern map file must exist on the RapidsDB node that this command is executed on.

The pattern map file is in CSV format with the following columns in this order:

1. Priority (integer)
2. Search (string)
3. Replace (string)

Because there can be multiple patterns specified, `priority` represents the importance of an individual mapping. A higher number means that that mapping is applied before a mapping with a lower priority.

The `search` field is a regex pattern to be matched against the external ID credential. If this pattern matches against the external identifier then the corresponding `replace` field is used as the substitute RapidsDB username.

By default this `search` pattern may match multiple times against the external identifier unless the pattern starts with a '^' and ends with a '$' to signify that it should match the entire record. This field should be quoted to preserve case.

The value of the `replace` field is used as the RapidsDB username when the corresponding `search` pattern matches against the external identifier. The replace field can contain regex capture group syntax (e.g. '$1' for the first capture group, '$2' for the second capture group, etc). If the `search` pattern matched against the external identifier multiple times then the value of this replace field will be applied against the external identifier multiple times also.

Please note that if the `replace` field is quoted then the case of the resulting username will be preserved. If it is not quoted then the username will be folded to uppercase as per RapidsDB object naming rules.

Examples:

Transform any Kerberos admin account to the username 'admin'.e.g. map
`craig/admin@EXAMPLE.COM` → `ADMIN`

```
100, "^(.+)/admin@EXAMPLE.COM$", admin
```

Map all Kerberos users to RDP usernames without any qualifiers and the realm. e.g. map
`craig/engineering@EXAMPLE.COM` → `CRAIG`


```
90, "^(.+?)(/[^@]*)?@EXAMPLE.COM$", $1
```

Replace all instances of the word 'boray' with 'rapids' instead:

```
80, "boray", "rapids"
```

In order to see if the pattern maps are working as expected, please refer to the RapidsDB dqx.log file on the node where a client connects to and tries to authenticate. The RapidsDB node will print log messages that indicate the initial identifier given and whether this is mapped either by direct username mapping or via pattern mapping.

Example:

```
[rapids@boray01 current]$ cat
/home/rapids/dave/R4_Tests/R3.6_Tests/Auth/Kerberos/pattern.map
100, "^(.+)/admin@RDP.COM$", ADMIN
90, "^(.+?)(/[^@]*)?@RDP.COM$", $1
80, "boray", "rapids"
rapids > SET PATTERN MAP FILE
'/home/rapids/dave/R4_Tests/R3.6_Tests/Auth/Kerberos/pattern.map';
0 row(s) returned (0.37 sec)
rapids > select * from pattern_maps order by priority asc;
  PRIORITY SEARCH                    REPLACE
  -------- ------                    -------
        80 boray                     rapids
        90 ^(.+?)(/[^@]*)?@RDP.COM$  $1
       100 ^(.+)/admin@RDP.COM$      ADMIN

3 row(s) returned (1.50 sec)
```

7.9.5 Clearing the Pattern Map

The pattern map can be cleared by either loading an empty file, or by executing the command:

```
SET PATTERN MAP FILE NULL ;
```

This will clear the pattern map across all cluster nodes.

Example:

```
rapids >  select * from pattern_maps order by priority asc;
  PRIORITY SEARCH                    REPLACE
  -------- ------                    -------
        80 boray                     rapids
        90 ^(.+?)(/[^@]*)?@RDP.COM$  $1
       100 ^(.+)/admin@RDP.COM$      ADMIN
```



```
3 row(s) returned (0.23 sec)
rapids > set pattern map file null;
0 row(s) returned (0.04 sec)
rapids >  select * from pattern_maps order by priority asc;
0 row(s) returned (0.12 sec)
```

7.10 Adding Connectors

7.10.1 CREATE CONNECTOR Command

The CREATE CONNECTOR command is used to add new Connectors to a Federation. The general format for the CREATE CONNECTOR command is:

CREATE CONNECTOR <name>
TYPE <Connector type> [WITH <key>='<value>' [,<key>='<value>']]
[NODE <node name> [WITH <key>='<value>' [,<key>='<value>'] [<further node clauses>]]
[<Import clause> [<Import clause>]]

The WITH clause is used to specify Connector-specific options (specified as key-value pairs), such as the host name/ip address for the underlying data source. The WITH clause requires that the <value> option is enclosed in single quotes. For example,

```
CREATE CONNECTOR ORA1 TYPE ORACLE WITH CONNECTIONSTRING=
'jdbc:oracle:thin:@localhost:1522:dev1', USER='rapids', PASSWORD='rdpuser',  NODE BORAY05;
```

This command would create an Oracle Connector named ORA1 running on node BORAY05.

After the Connector has been created, the Connector will automatically retrieve all of the table metadata for the schemas and tables associated that Connector.

7.10.2 Importing Tables

7.10.2.1 Connector-Level Definition

As described in section 2, when configuring a Connector the user can specify the tables to be imported and can also set up name mapping for the catalog, schema and/or tables that are imported. The following describes how to set up the import configuration that will be part of the Connector configuration command described later.

The <Import Clause> is used for the following:

1. To specify the tables that should be imported into the name space managed by the Connector.
2. To map the catalog, schema and/or table names

An Import Clause has the following format:
CATALOG {*|<catalog name>} [AS <catalog name]

[SCHEMA {*|<schema name>} [AS <schema name]]
[TABLE {*|<table name> [AS <table name>] [USING (<column defs>) [,<table name> …]}]
[TABLE {*|<table name> [AS <table name>] [USING (<column defs>) [,<table name> …]}] …

When the "AS" clause is used, the catalog, schema or table name will be mapped within the RapidsDB Connector to the specified name, and that is the name that the user will use when submitting queries to RapidsDB.

The USING clause applies to the Stream and Hadoop Connectors.

If no Import Clause is specified when a Connector is created, then the default will be:

CATALOG * SCHEMA * TABLE *

which will import all of the tables that the user is authorized to access.

The catalog, schema and table names are case-insensitive and do not have to be enclosed in double quotes unless the Connector option IGNORE_CASE is set to FALSE, and in this case all catalog, schema and table names are case-sensitive and must match the case used by the underlying data source, and must be enclosed in double quotes.

Below are some sample Import Clauses:

```
CATALOG ORA1
SCHEMA EAST
TABLE CUSTOMERS, ORDERS
```

This clause would import the tables CUSTOMERS and ORDERS from the schema named EAST. The fully qualified name for the CUSTOMERS table the user would be:

ORA1.EAST.CUSTOMERS

```
CATALOG ORA1
SCHEMA EAST
TABLE CUSTOMERS AS EASTCUSTOMERS, ORDERS AS EASTORDERS
```

This clause would import the tables CUSTOMERS and ORDERS from the schema named EAST, and map the CUSTOMERS and ORDERS table names to EASTCUSTOMERS and EASTORDERS. The fully qualified name for the CUSTOMERS table the user would be:

ORA1.EAST.EASTCUSTOMERS

```
CATALOG ORA1 AS USA
SCHEMA EAST
TABLE CUSTOMERS AS EASTCUSTOMERS, ORDERS AS EASTORDERS
```


This clause would import the tables CUSTOMERS and ORDERS from schema named EAST, and map the catalog name to USA and map the CUSTOMERS and ORDERS table names to EASTCUSTOMERS and EASTORDERS. The fully qualified name for the CUSTOMERS table the user would be:
USA.EAST.EASTCUSTOMERS

```
CATALOG ORA1 AS USA
SCHEMA EAST
TABLE CUSTOMERS AS EASTCUSTOMERS, ORDERS AS EASTORDERS
SCHEMA WEST
TABLE CUSTOMERS AS WESTCUSTOMERS, ORDERS AS WESTORDERS
```

This clause would import the tables CUSTOMERS and ORDERS from schemas named EAST, and WEST and map the catalog name to USA and map the CUSTOMERS and ORDERS table from the EAST schema to EASTCUSTOMERS and EASTORDERS and from the WEST schema to WESTCUSTOMERS and WESTORDERS. The user could then reference the table CUSTOMERS from the WEST schema as WESTCUSTOMERS. The fully qualified name would be:

USA.WEST.WESTCUSTOMERS

```
CATALOG MEMSQL
SCHEMA EAST
TABLE “CUSTOMERS” AS UPPER_CUSTOMERS, “customers” AS LOWER_CUSTOMERS
```

This example is for a MemSQL Connector where the user has specified two tables with the same names, but in different cases. This clause would import the tables “CUSTOMERS” and “customers” from the EAST schema, and map the “CUSTOMERS” table to UPPER_CUSTOMERS and the “customers” table to LOWER_CUSTOMERS. With this mapping the user can reference the tables UPPER_CUSTOMERS and LOWER_CUSTOMERS as case-insensitive names.

7.10.3 Handling of Decimal Datatypes

The RapidsDB Execution Engine supports a high performance internal data type for decimals, and by default a Connector will use this internal data type when passing decimal column values to the Execution Engine in order to provide the best performance. The internal data type has a precision of 17. However, if the data value from the source column cannot be represented using the internal RapidsDB decimal data type, or if an intermediate value that is computed as part of a query exceeds the size for the internal decimal data type, then the associated query will fail with an error indicating that an overflow of some kind happened. If this happens, then the Connector can be configured to use the Java bigdecimal data type by setting the “BIGDECIMAL” option to ‘TRUE’. With this option set, queries using decimal values will be slower but are guaranteed to be able to handle all possible column values and intermediate query results.

7.10.4 Adding a MOXE Connector

To add a MOXE Connector use the following command

```
CREATE CONNECTOR <name> TYPE MOXE [WITH <key>='<value>' [,<key>='<value>']]
[NODE * | NODE <node name> [NODE <node name>] [<further node names>]]
```

The table below shows the possible settings for the key and value fields for a MOXE Connector:

Key:	Default Value	Value syntax	Description
mem_per_node	1	<nnn>GB where, <nnn> is an integer value > 0	Specifies the maximum amount of memory in GB that MOXE can use on each node in the RapidsDB cluster.
partitions_per_node	2	Integer value > 0	Specifies the number of partitions per node

Example 1:

```
CREATE CONNECTOR MOXE_16 TYPE MOXE WITH mem_per_node= '16GB ';
```

This command would create a MOXE Connector with a maximum of 16GB per node and with 2 partitions per node, and the Connector would run on all nodes in the RapidsDB cluster. The catalog and schema name associated with this Connector would be “MOXE_16”.

Example 2:

```
CREATE CONNECTOR MOXE2 TYPE MOXE WITH mem_per_node= '16GB ',
partitions_per_node='8' NODE NODE2 NODE NODE3;
```

This command would create a MOXE Connector to run on two nodes (NODE2 and NODE3) in the RapidsDB Cluster with a maximum of 16GB per node and with 8 partitions per node. The catalog and schema name associated with this Connector would be “MOXE2”.

Refer to section 8 for details on managing MOXE tables

7.10.5 Adding a MemSQL Connector

To add a MemSQL Connector use the following command

```
CREATE CONNECTOR <name>
TYPE MEMSQL [WITH <key>='<value>' [,<key>='<value>']]
NODE <node name>
[<Import clause> [<Import clause>]]
```


For the MemSQL Connector, the user must specify the node on which the MemSQL Connector will run, and there can only be one node specified. Ideally, this node should be the same as the MemSQL Aggregator node.

The table below shows the possible settings for the key and value fields:

Key:	Default Value	Value syntax	Description
host	'localhost'	Non-empty, non-whitespace string.	Specifies the hostname or IP address that the MemSQL Connector should use for establishing a socket connection to MemSQL.
port	3306	Integer value > 0 and < 65536	Specifies the port number that the MemSQL Connector should use for establishing a socket connection to MemSQL. **This must be specified**
user		Non-empty, non-whitespace string	The user name for accessing MemSQL **This must be specified**
password		Non-empty, non-whitespace string	The password for the user for accessing MemSQL
database		Non-empty, case-sensitive, non-whitespace string	The MemSQL database to be used. The case must match the case used by MemSQL. **This must be specified**
batch_size	100	Integer value > 0 and < 1000	Specifies how the result set should be batched when writing the result set back to the MemSQL database. Each batch will contain the number of rows specified by the batch_size.
bigdecimal	'FALSE'	'TRUE' or 'FALSE'	Specifies whether decimal data types should be mapped to the internal RapidsDB decimal data type (for optimal performance) or whether the standard Java bigdecimal data type should be used. Refer to section 7.10.3 for more information.
classpath		A list of absolute classpaths, with each classpath separated	Specifies the classpaths to be searched first for any jar files to be included with this Connector.

		using either the colon ":" character for Linux, or the semicolon ";" character for Windows	The Connector will then use the regular RapidsDB classpath to complete any searches.
fetch_size	100	Integer value > 0 and < 1000	Specifies how the data being retrieved from the MemSQL database should be batched. Each batch will contain the number of rows specified by the fetch_size.
ignore_case	'TRUE'	'TRUE' or 'FALSE'	Specifies whether the Connector should do case-insensitive matching for table names. **NOTE:** This setting would **only** be set to 'FALSE' if the MemSQL database included tables with the same names in the same schema that were specified in different cases.
server_timezone	'UTC'	Non-empty, non-whitespace string specifying a valid timezone	Override detection/mapping of time zone. Used when the time zone from server doesn't map to the Java time zone.
_`<property key>` =<property value>		Non-empty, non-whitespace string. By default, the <property key> will be converted to upper case, in to maintain the case for the <property key> it must be enclosed in back-ticks (`)	Specifies a key value pair that will be set as part of the Connection Properties object for the connection being established to the MemSQL database. For example: _useCompression=true This would result in the following key value pair being set up in the Properties object: "USECOMPRESSION", "true" For MemSQL, the Properties keys are case-sensitive and so this Property would get ignored. In order to preserve the case for the key it must be enclosed in back ticks as shown below: _`useCompression`=true

			This would result in the following key value pair being set up in the Properties object: “useCompression”, “true” **NOTE:** The use of this option could result in unpredictable behavior for the Connector and should only be used in situations where the user is very confident that there will be no unexpected side effects from setting this Property.

Refer to section 6.5.2 for details on how to specify the <Import clause>.

NOTES:

1. For a MemSQL Connector, there is only one catalog which is named MEMSQL, and the schema name is the same as the MemSQL database name.

Example commands:

```
1. CREATE CONNECTOR MEMSQL1
   TYPE MEMSQL WITH host='192.168.1.1', user='user1', database='tpch'
    NODE NODE2;
```

This command would create a MemSQL Connector on node NODE2 and it would connect via JDBC to MemSQL using the default port number (3306) using ip address 192.168.1.1. The Connnector would import all of the tables available from the MemSQL data source for the database named “tpch”.

```
2. CREATE CONNECTOR MEMSQL1
   TYPE MEMSQL WITH host='192.168.1.1', user='user1', database='tpch'
   NODE NODE2
   CATALOG *
   SCHEMA *
   TABLE customers, orders;
```

This command sets up the nodes as for the previous example, but this time only the customers and orders tables from the database named “tpch” will be imported.

3.
```
CREATE CONNECTOR MEMSQL1
TYPE MEMSQL WITH host='192.168.1.1', user='user1', database='tpch',
ignore_case='false'
NODE NODE2
CATALOG *
SCHEMA *
TABLE “customers” ,“CUSTOMERS”;
```

This command sets up the nodes as before, but this time the option “ignore_case” has been set to FALSE because there are two tables both named the same but with different cases. In this case the table names have to be specified in double quotes, and they would also have to be specified in double quotes when querying. For example:

SELECT * FROM “customers”;

4.
```
CREATE CONNECTOR MEMSQL1
TYPE MEMSQL WITH host='192.168.1.1', port='3306', user='user1', database='tpch',
ignore_case='false'
NODE NODE2
CATALOG *
SCHEMA *
TABLE “customers” as LOWER_CUSTOMERS, “CUSTOMERS” AS UPPER_CUSTOMERS;
```

This command is the same as the previous command but this time the tables “customers” and “CUSTOMERS” are mapped to case-insensitive names so avoid the user having to enclose the table names in double quotes. For example, the following queries would both reference the same “customers” table:

SELECT * FROM lower_customers;
SELECT * FROM LOWER_CUSTOMERS;

5.
```
CREATE CONNECTOR MEMSQL1
TYPE MEMSQL WITH host='192.168.1.1', port='3306', user='user1', database='tpch',
_`useCompression`='true', bigdecimal='true'
NODE NODE2;
```

This command includes the setting of the connection property, useCompression, and instructs the Connector to use the Java bigdecimal data type for all decimal values.

7.10.6 Adding a MySQL Connector

To add a MySQL Connector use the following command

```
CREATE CONNECTOR <name>
TYPE MYSQL [WITH <key>='<value>' [,<key>='<value>']]
NODE <node name>
<Import clause> [<Import clause>]]
```

For the MySQL Connector, the user must specify the node on which the MySQL Connector will run, and there can only be one node specified

The table below shows the possible settings for the key and value fields:

Key:	Default Value	Value syntax	Description
host	'localhost'	Non-empty, non-whitespace string.	Specifies the hostname or IP address that the Connector should use for establishing a socket connection to MySQL.
port	3306	Integer value > 0 and < 65536	Specifies the port number that the Connector should use for establishing a socket connection to MySQL.
database	'public'	Non-empty, case-sensitive, non-whitespace string	The MySQL database to be used. The case must match the case used by MySQL.
user		Non-empty, non-whitespace string	The user name for accessing MySQL **This must be specified**
password		Non-empty, non-whitespace string	The password for the user for accessing MySQL
batch_size	1000	Integer value >= 100 and <= 1000	Specifies how the result set should be batched when writing the result set back to the MySQL database. Each batch will contain the number of rows specified by the batch_size.
bigdecimal	'FALSE'	'TRUE' or 'FALSE'	Specifies whether decimal data types should be mapped to the internal RapidsDB decimal data type (for optimal performance) or whether the standard

			Java bigdecimal data type should be used. Refer to section 7.10.3 for more information.
classpath		A list of absolute classpaths, with each classpath separated using either the colon ":" character for Linux, or the semicolon ";" character for Windows	Specifies the classpaths to be searched first for any jar files to be included with this Connector. The Connector will then use the regular RapidsDB classpath to complete any searches.
fetch_size	see Description	Integer value > 0 and < 1000	Specifies how the data being retrieved from the MySQL database should be batched. Each batch will contain the number of rows specified by the fetch_size. **Note:** the default value has been set to provide the optimal fetch performance and it is not recommended that this default be overridden.
ignore_case	'TRUE'	'TRUE' or 'FALSE'	Specifies whether the Connector should do case-insensitive matching for table names. **NOTE:** This setting would **only** be set to 'FALSE' if the MySQL database included tables with the same names in the same schema that were specified in different cases.
server_timezone	'UTC'	Non-empty, non-whitespace string specifying a valid timezone	Specifies the timezone that the MySQL database is using. The value specified will get passed to the MySQL Connector-J as part of the connection url. Refer to the MySQL Connector-J documentation for more information

use_ssl	'FALSE'	'TRUE' or 'FALSE'	Specifies whether ssl should be used when connecting to the MySQL database. The value specified will get passed to the MySQL Connector-J as part of the connection url. Refer to the MySQL Connector-J documentation for more information
_`<property key>` =<property value>		Non-empty, non-whitespace string. By default, the <property key> will be converted to upper case, in to maintain the case for the <property key> it must be enclosed in back-ticks (`)	Specifies a key value pair that will be set as part of the Connection Properties object for the connection being established to the MySQL database. For example: _useCompression=true This would result in the following key value pair being set up in the Properties object: “USECOMPRESSION”, “true” For MySQL, the Properties keys are case-sensitive and so this Property would get ignored. In order to preserve the case for the key it must be enclosed in back ticks as shown below: _`useCompression`=true This would result in the following key value pair being set up in the Properties object: “useCompression”, “true” **NOTE:** The use of this option could result in unpredictable behavior for the Connector and should only be used in

			situations where the user is very confident that there will be no unexpected side effects from setting this Property.

Refer to section 7.10.2 for details on how to specify the <Import clause>.

NOTES:

1. For a MySQL Connector, there is only one catalog which is named MYSQL, and the schema name is the same as the MySQL database name.

Example commands:

```
CREATE CONNECTOR MYSQL1
TYPE MYSQL WITH host='192.168.1.1', user='user1', database='tpch'
NODE NODE2;
```

This command would create a MySQL Connector on node NODE2 and it would connect to MySQL using MySQL Connector-J using the default port number,3306, and ip address 192.168.1.1. The Connnector would import all of the tables available from the MySQL data source for the database named “tpch”.

```
CREATE CONNECTOR MYSQL1
TYPE MYSQL WITH host='192.168.1.1', port='3307', user='user1', database='tpch',
use_ssl='true', bigdecimal='true'
NODE NODE2;
```

This command would create a MySQL Connector on node NODE2 and it would connect to MySQL using MySQL Connector-J using port number 3307 and ip address 192.168.1.1, and it would also pass the “use_ssl” option to MySQL Connector-J . The Connector would also use Java bigdecimal for all decimal values. The Connnector would import all of the tables available from the MySQL data source for the database named “tpch”.

```
CREATE CONNECTOR MYSQL1
TYPE MYSQL WITH host='192.168.1.1', port='3307', user='user1', database='tpch',
_`useCompression`='true'
NODE NODE2;
```

This command would create a MySQL Connector on node NODE2 and it would connect to MySQL using MySQL Connector-J using port number 3307 and ip address 192.168.1.1, and it would also pass the connection property useCompression.


```
CREATE CONNECTOR MYSQL1
TYPE MYSQL WITH host='192.168.1.1', port='3307', user='user1', database='tpch',
_`useCompression`='true'
 NODE NODE2
 CATALOG *
 SCHEMA *
 TABLE customers, orders;
```

This command sets up the nodes as for the previous example, but this time only the CUSTOMERS and ORDERS tables from the database named “tpch” will be imported.

```
CREATE CONNECTOR MYSQL1
TYPE MYSQL WITH host='192.168.1.1', port='3307', user='user1', database='tpch',
ignore_case='false'
NODE NODE2
CATALOG *
SCHEMA *
TABLE “customers” ,“CUSTOMERS”;
```

This command sets up the nodes as before, but this time the option “ignore_case” has been set to FALSE because there are two tables both named the same but with different cases. In this case the table names have to be specified in double quotes, and they would also have to be specified in double quotes when querying. For example:

SELECT * FROM “customers”;

```
CREATE CONNECTOR MYSQL1
TYPE MYSQL WITH host='192.168.1.1', port='3307', user='user1', database='tpch'
ignore_case='false'
NODE NODE2
CATALOG *
SCHEMA *
TABLE “customers” as LOWER_CUSTOMERS, “CUSTOMERS” AS UPPER_CUSTOMERS;
```

This command is the same as the previous command but this time the tables “customers” and “CUSTOMERS” are mapped to case-insensitive names so avoid the user having to enclose the table names in double quotes. For example, the following queries would both reference the same “customers” table:

SELECT * FROM lower_customers;

SELECT * FROM LOWER_CUSTOMERS;

NOTE:

The MySQL Connector should not be used for connecting to a MemSQL database because it can result in failures and unpredictable results. For example, the following error could be reported in the dqx.log file:

rapidsdata.database.exceptions.DbSubException: java.sql.SQLException: Unknown system variable 'performance_schema'
2020-08-04T23:02:57,983 [StdErr] ERROR: java.util.concurrent.CompletionException: com.rapidsdata.database.exceptions.DbSubException: java.sql.SQLException: Unknown system variable 'performance_schema'

When connecting to a MemSQL database always use the MemSQL Connector (see 7.10.5)

7.10.7 Adding an Oracle Connector

To add an Oracle Connector use the following command

```
CREATE CONNECTOR <name>
TYPE ORACLE [WITH <key>='<value>' [,<key>='<value>']]
NODE <node name>
[<Import clause> [<Import clause>]]
```

For the Oracle Connector, the user must specify the node on which the Oracle Connector will run, and there can only be one node specified.

NOTE: The RapidsDB system includes the following jar file for the Oracle JDBC Driver: ojdbc7-12.1.0.jar. If the user wishes to use a different version of the Oracle JDBC Driver then the user must delete the ojdbc7-12.1.0.jar file from the lib directory of the RapidsDB installation directory on the node where the Oracle Connector is going to run and then copy the new Oracle JDBC Driver to the same lib directory.

The table below shows the possible settings for the key and value fields:

Key:	Default Value	Value syntax	Description
host	'localhost'	Non-empty, non-whitespace string.	Specifies the hostname or IP address that the Connector should use for establishing a socket connection to Oracle.

port	1521	Integer value > 0 and < 65536	Specifies the port number that the Connector should use for establishing a socket connection to Oracle.
sid		Non-empty, non-whitespace string	Oracle SID
user		Non-empty, non-whitespace string	The user name for accessing the data source. **This must be specified**
password		Non-empty, non-whitespace string	The password for the user for accessing the data source
batch_size	1000	Integer value >= 100 and <= 1000	Specifies how the result set should be batched when writing the result set back to the Oracle database. Each batch will contain the number of rows specified by the batch_size.
bigdecimal	'FALSE'	'TRUE' or 'FALSE'	Specifies whether decimal data types should be mapped to the internal RapidsDB decimal data type (for optimal performance) or whether the standard Java bigdecimal data type should be used. Refer to section 7.10.3 for more information.
classpath		A list of absolute classpaths, with each classpath separated using either the colon ":" character for Linux, or the semicolon ";" character for Windows	Specifies the classpaths to be searched first for any jar files to be included with this Connector. The Connector will then use the regular RapidsDB classpath to complete any searches.
fetch_size	10	Integer value > 0 and < 1000	Specifies how the data being retrieved from the Oracle database should be batched. Each batch will contain the number of rows specified by the fetch_size.
ignore_case	'TRUE'	'TRUE' or 'FALSE'	Specifies whether the Connector should do case-insensitive matching for table names. **NOTE:** This setting would **only** be set to 'FALSE' if the Oracle database included tables with the same names in the same schema

			that were specified in different cases.
_<property key> =<property value>		Non-empty, non-whitespace string.	Specifies a key value pair that will be set as part of the Connection Properties object for the connection being established to the Oracle database. **NOTE:** The use of this option could result in unpredictable behavior for the Connector and should only be used in situations where the user is very confident that there will be no unexpected side effects from setting this Property.

Refer to section 7.10.2 for details on how to specify the <Import clause>.

Examples:

1. The following is a sample Connector to an Oracle database with a SID of "dev1" that will import all of the schemas and tables accessible by the specified user:

```
CREATE CONNECTOR ORA1 TYPE ORACLE WITH SID= 'dev1', USER='rapids', PASSWORD='rdpuser'
NODE BORAY05;
```

2. The following is the same Connector as in example 1, plus it the Connector will use Java bigdecimal for all decimal values:

```
CREATE CONNECTOR ORA1 TYPE ORACLE WITH SID= 'dev1', USER='rapids',
PASSWORD='rdpuser', bigdecimal='true' NODE BORAY05;
```

3. The following is a sample Connector to an Oracle database which will import only the schemas named "orders" and "sales":

```
CREATE CONNECTOR ORA1 TYPE ORACLE WITH SID= 'dev1', USER='rapids', PASSWORD='rdpuser'
NODE BORAY05 CATALOG * SCHEMA "orders' SCHEMA "sales" TABLE *;
```

7.10.8 Adding a Postgres Connector

To add a Postgres Connector use the following command

```
CREATE CONNECTOR <name>
TYPE POSTGRES [WITH <key>='<value>' [,<key>='<value>']]
NODE <node name>
[<Import clause> [<Import clause>]]
```


For the Postgres Connector, the user must specify the node on which the Postgres Connector will run, and there can only be one node specified.

NOTE: The RapidsDB system includes the following jar file for the Postgres JDBC Driver: postgresql-42.1.4.jar. If the user wishes to use a different version of the Postgres JDBC Driver then the user must delete the postgresql-42.1.4.jar file from the lib directory of the RapidsDB installation directory on the node where the Postgres Connector is going to run and then copy the new Postgres JDBC Driver to the same lib directory.

The table below shows the possible settings for the key and value fields:

Key:	Default Value	Value syntax	Description
host	'localhost'	Non-empty, non-whitespace string.	Specifies the hostname or IP address that the Connector should use for establishing a socket connection to Postgres.
port	5432	Integer value > 0 and < 65536	Specifies the port number that the Connector should use for establishing a socket connection to Postgres.
database	'public'	Non-empty, non-whitespace string	The database to connect to.
user		Non-empty, non-whitespace string	The user name for accessing the data source. **This must be specified**
password		Non-empty, non-whitespace string	The password for the user for accessing the data source
batch_size	100	Integer value >= 100 and <= 1000	Specifies how the result set should be batched when writing the result set back to the Postgres database. Each batch will contain the number of rows specified by the batch_size.
bigdecimal	'FALSE'	'TRUE' or 'FALSE'	Specifies whether decimal data types should be mapped to the internal RapidsDB decimal data type (for optimal performance) or whether the standard Java bigdecimal data type should be used. Refer to section 7.10.3 for more information.
classpath		A list of absolute classpaths, with each classpath separated using either the colon ":" character for Linux, or the semicolon ";" character for Windows	Specifies the classpaths to be searched first for any jar files to be included with this Connector. The Connector will then use the regular RapidsDB classpath to complete any searches.
fetch_size	100	Integer value > 0 and < 1000	Specifies how the data being retrieved from the Postgres database should be batched. Each batch will contain the number of rows specified by the fetch_size.

ignore_case	'TRUE'	'TRUE' or 'FALSE'	Specifies whether the Connector should do case-insensitive matching for table names. **NOTE:** This setting would **only** be set to 'FALSE' if the Postgres database included tables with the same names in the same schema that were specified in different cases.
_<property key> =<property value>		Non-empty, non-whitespace string.	Specifies a key value pair that will be set as part of the Connection Properties object for the connection being established to the Postgres database. **NOTE:** The use of this option could result in unpredictable behavior for the Connector and should only be used in situations where the user is very confident that there will be no unexpected side effects from setting this Property.

Refer to section 7.10.2 for details on how to specify the <Import clause>.

When specifying schema or table names using the SCHEMA or TABLE clauses, the names must be enclosed in double quotes and match the case used by the Postgres database (the default is lower case).

Examples:

1. The following is a sample Connector to a Postgres database “dw1” that will import all of the schemas and tables accessible by the specified user, and the Connector will use Java datatypes for all numeric values:

```
CREATE CONNECTOR PG1 TYPE POSTGRES WITH host= '10.10.1.1', port='6432', database='dw1', USER='adm', PASSWORD='admpsw', TYPES='JAVA'  NODE BORAY05 CATALOG * SCHEMA “orders” SCHEMA “sales” ;
```

2. The following is a sample Connector to a Postgres database “dw1” which will import only the schemas named “orders” and “sales”:

```
CREATE CONNECTOR PG1 TYPE POSTGRES WITH host= '10.10.1.1', port='6432', database='dw1', USER='adm', PASSWORD='admpsw' NODE BORAY05 CATALOG * SCHEMA “orders” SCHEMA “sales” ;
```


NOTES:

1. If two Postgres Connectors are created specifying the same database name, and the Connectors reference the same tables, then any such table names will be duplicated in the RapaidsDB metadata tables because the fully qualified table names will have the same catalog and schema names. To ensure that this does not happen, the CATALOG name for one of the Postgres Connectors should be mapped to a different name as shown in the example below:

   CREATE CONNECTOR PG1 TYPE POSTGRES WITH USER='rapids', PORT='5432', PASSWORD='rapids', DATABASE='rdp4', HOST='192.168.10.12' NODE * CATALOG * SCHEMA * TABLE *;

   CREATE CONNECTOR PG2 TYPE POSTGRES WITH USER='rapids', PORT='5432', PASSWORD='rapids', DATABASE='rdp4', HOST='192.168.10.12' NODE * CATALOG "rdp4" AS "pg2" SCHEMA * TABLE *;

   The table names for the Connectors can now be disambiguated by using the catalog name as "rdp4" for the PG1 Connector or "pg2"for the PG2 Connector:
   - Select * from rdp4.public.t1;
   - Select * from pg2.public.t1;

7.10.9 Adding a Greenplum Connector

To add a Greenplum Connector use the following command

```
CREATE CONNECTOR <name>
TYPE POSTGRES [WITH <key>='<value>' [,<key>='<value>']]
NODE <node name>
[<Import clause> [<Import clause>]]
```

For the Greenplum Connector, the user must specify the node on which the Greenplum Connector will run, and there can only be one node specified.
NOTE: **The user must also copy the Greenplum JDBC Driver jar file to the lib directory of the RapidsDB installation directory on the node where the Greenplum Connector is going to run.**

The table below shows the possible settings for the key and value fields:

Key:	Default Value	Value syntax	Description
url		jdbc:pivotal:greenplum://<host>:<port>;DatabaseName=<database>[?<atttributes>]	Specifies the JDBC connection string (url) to be used for the Greenplum JDBC Driver.

			Example: jdbc:pivotal:greenplum://10.10.1.1:5432;DatabaseName=dwctr **This must be specified**
user		Non-empty, non-whitespace string	The user name for accessing the data source. **This must be specified**
password		Non-empty, non-whitespace string	The password for the user for accessing the data source
batch_size	1000	Integer value >= 100 and <= 1000	Specifies how the result set should be batched when sending the results back to the Connector. The result set will be batched using the specified number as the batch size.
bigdecimal	'FALSE'	'TRUE' or 'FALSE'	Specifies whether decimal data types should be mapped to the internal RapidsDB decimal data type (for optimal performance) or whether the standard Java bigdecimal data type should be used. Refer to section 7.10.3 for more information.
classpath		A list of absolute classpaths, with each classpath separated using either the colon ":" character for Linux, or the semicolon ";" character for Windows	Specifies the classpaths to be searched first for any jar files to be included with this Connector. The Connector will then use the regular RapidsDB classpath to complete any searches.
fetch_size	100	Integer value > 0 and < 1000	Specifies how the data being retrieved from the Greenplum database should be batched. Each batch will contain the number of rows specified by the fetch_size.
ignore_case	'TRUE'	'TRUE' or 'FALSE'	Specifies whether the Connector should do case-insensitive matching for table names. **NOTE:** This setting would **only** be set to 'FALSE' if the Greenplum database included tables with the same names in

			the same schema that were specified in different cases.
_<property key> =<property value>		Non-empty, non-whitespace string.	Specifies a key value pair that will be set as part of the Connection Properties object for the connection being established to the Greenplum database. **NOTE:** The use of this option could result in unpredictable behavior for the Connector and should only be used in situations where the user is very confident that there will be no unexpected side effects from setting this Property.

Refer to section 7.10.2 for details on how to specify the <Import clause>.

When specifying schema or table names using the SCHEMA or TABLE clauses, the names must be enclosed in double quotes and match the case used by the Greenplum database (the default is lower case).

Examples:

1. The following is a sample Connector to a Greenplum database “dwctr” that will import all of the schemas and tables accessible by the specified user:

```
CREATE CONNECTOR GP1 TYPE POSTGRES WITH
url= 'jdbc: pivotal:greenplum://10.10.1.1:5432;DatabaseName=dwctr', USER='adm',
PASSWORD='admpsw' NODE BORAY05;
```

2. The following is a sample Connector to a Greenplum database “dwctr” which will import only the schemas named “orders” and “sales”, and the Connector will use Java bigdecimal for all decimal values:

```
CREATE CONNECTOR GP1 TYPE POSTGRES WITH
CONNECTIONSTRING= 'jdbc: pivotal:greenplum://10.10.1.1:5432;DatabaseName=dwctr',
USER='adm', PASSWORD='admpsw', bigdecimal='true' NODE BORAY05 CATALOG * SCHEMA
“orders” SCHEMA “sales”;
```


7.10.10 Adding a JDBC Connector

7.10.10.1 Creating the JDBC Connector

To add a JDBC Connector use the following command

```
CREATE CONNECTOR <name>
TYPE JDBC [WITH <key>='<value>' [,<key>='<value>']]
NODE <node name>
[<Import clause> [<Import clause>]]
```

For the JDBC Connector, the user must specify the node on which the JDBC Connector will run, and there can only be one node specified. **The user must also copy the JDBC Driver for the associated data source to the /lib directory of the RapidsDB installation directory on the node where the JDBC Connector is going to run.**

The table below shows the possible settings for the key and value fields:

Key:	Default Value	Value syntax	Description
connectionstring		Non-empty, non-whitespace string.	Specifies the JDBC connection string (url) to be used for the JDBC Driver for the data source to be accessed. Refer to the documentation for the JDBC Driver being used for details on the format for the url option. Example: jdbc:hive2://localhost:10000/default **This must be specified**
user		Non-empty, non-whitespace string	The user name for accessing the data source. **This must be specified**
password		Non-empty, non-whitespace string	The password for the user for accessing the data source
batch_size	1000	Integer value >= 100 and <= 1000	Specifies how the result set should be batched when sending the results back to the Connector. The result set will be batched using the specified number as the batch size.
bigdecimal	'FALSE'	'TRUE' or 'FALSE'	Specifies whether decimal data types should be mapped to the internal RapidsDB decimal data type (for

			optimal performance) or whether the standard Java bigdecimal data type should be used. Refer to section 7.10.3 for more information.
classpath		A list of absolute classpaths, with each classpath separated using either the colon ":" character for Linux, or the semicolon ";" character for Windows	Specifies the classpaths to be searched first for any jar files to be included with this Connector. The Connector will then use the regular RapidsDB classpath to complete any searches.
fetch_size	100	Integer value > 0 and < 1000	Specifies how the data being retrieved from the source database should be batched. Each batch will contain the number of rows specified by the fetch_size.
ignore_case	'TRUE'	'TRUE' or 'FALSE'	Specifies whether the Connector should do case-insensitive matching for table names. **NOTE:** This setting would **only** be set to 'FALSE' if the target database included tables with the same names in the same schema that were specified in different cases.
server_timezone	'UTC'	Non-empty, non-whitespace string specifying a valid timezone	Override detection/mapping of time zone. Used when the time zone from server doesn't map to the Java time zone.
use_ssl	'FALSE'	'TRUE' or 'FALSE'	Specifies whether ssl should be used when connecting to the target database.
_<property key> =<property value>		Non-empty, non-whitespace string. By default, the <property key> will be converted to upper case, in to maintain the case for the <property key> it must be enclosed in back-ticks (`)	Specifies a key value pair that will be set as part of the Connection Properties object for the connection being established to the data source. For example: _useCompression=true This would result in the following key value pair being set up in the Properties object: "USECOMPRESSION", "true"

			For some data sources, the Properties keys are case-sensitive and so this Property would get ignored. In order to preserve the case for the key it must be enclosed in back ticks as shown below: _`useCompression`=true This would result in the following key value pair being set up in the Properties object: “useCompression”, “true” Refer to the documentation for the JDBC Driver being used for details on the case sensitivity for Properties keys. **NOTE:** The use of this option could result in unpredictable behavior for the Connector and should only be used in situations where the user is very confident that there will be no unexpected side effects from setting this Property.

Refer to section 7.10.2 for details on how to specify the <Import clause>.

Examples:

1. The following is a sample Connector to a SQL Server database:

```
CREATE CONNECTOR S1 TYPE JDBC WITH CONNECTIONSTRING='
jdbc:sqlserver://10.0.0.1:1433;databaseName=Sales', USER='sales', NODE BORAY05;
```

2. The following is a sample Connector to a SQL Server database, with a connection property included in the url:

```
CREATE CONNECTOR S1 TYPE JDBC WITH CONNECTIONSTRING ='
jdbc:sqlserver://10.0.0.1:1433;databaseName=Sales; loginTimeout=30', USER='sales', NODE
BORAY05;
```

3. The following is the same as the previous example, but this time the connection property is specified outside of the url:


```
CREATE CONNECTOR S1 TYPE JDBC WITH CONNECTIONSTRING ='
jdbc:sqlserver://10.0.0.1:1433;databaseName=Sales', USER='sales', _loginTimeout=30 NODE
BORAY05;
```

As connection properties for SQL Server are not case-sensitive there is no need to enclose the loginTimeout in back ticks.

4. The following is a sample Connector to a SQL Server database, with the option set to specify that Java bigdecimal should be used for all decimal values:

```
CREATE CONNECTOR S1 TYPE JDBC WITH CONNECTIONSTRING ='
jdbc:sqlserver://10.0.0.1:1433;databaseName=Sales', USER='sales', bigdecimal='true' NODE
BORAY05;
```

7.10.11 Adding a Hadoop Connector

7.10.11.1 Creating a Hadoop Connector

To add a Hadoop Connector use the following command:

```
CREATE CONNECTOR <name> TYPE HADOOP
  [WITH <key>='<value>' [,<key>='<value>'] [,<further key values>]]
   [NODE * | NODE <node name> [NODE <node name>] [<further node names>]]
   CATALOG [* | <name> [AS <catalog>]]
   SCHEMA [* | PUBLIC [AS <schema>] | <Hive Database> [WITH INCLUDES=<table list>]]
   [<table_specifier> [,<further table specifiers>]];

<table list>:
  <table name>[,<table name>[,<table name>…]] | <wildcard> | <regex>

<wildcard>:
  Any combination of characters and *'s
<regex>:
  Any valid regex expression

<table_specifier>:
  TABLE <table name>
  USING ( <column definition>, …)
  [PARTITION BY (<column name> [,<further column names>]) ]
  [WITH <key>='<value>' [,<key>='<value>'] [,<further key values>]]

column_definition:
   <column name> <data type>
```



```
<table name>:  the name of the table associated with this HDFS file

<data_type>:
   INTEGER[(<precision>)]
  | FLOAT
  | DECIMAL [(precision, scale)]
  | DATE
  | TIMESTAMP
  | VARCHAR
```

The WITH clause is used to specify Connector-specific options (specified as key-value pairs), such as the url to the HDFS name node. The WITH clause can be specified at the Connector (outermost) level, in which case it applies to all of the tables below, or it can be specified at the table level in which case it only applies to that table and overrides the setting at the Connector level. For example, in the following the field delimiter is defined as ',' at the Connector level, but for the table T2 the field delimiter is '|'.

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', delimiter=','
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 timestamp) WITH path='/data/sample/t1'
TABLE T2 USING  ( c1 integer, c2 timestamp) WITH path='/data/sample/t2', delimiter='|';
```

The NODE clause specifies which nodes in the RapidsDB Cluster the Hadoop Connector will run on. By default, if the NODE clause is not specified, then the Hadoop Connector will run on all of the nodes in the RapidsDB Cluster. The example below configures a Hadoop Connector to run on two nodes in the RapidsDB Cluster:

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', delimiter=','
NODE BORAY04 NODE BORAY05
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 timestamp) WITH path='/data/sample/t1'
TABLE T2 USING ( c1 integer, c2 timestamp) WITH path='/data/sample/t2', delimiter='| ';
```


The table below shows the possible settings for the key and value fields:

Key:	Default Value	Value syntax	Description
hdfs		Non-empty, non-whitespace string.	Specifies either the url to the HDFS name node, or the HDFS nameservice ID for an HDFS HA configuration. For example 'hdfs://192.168.10.15:8020' This should match what is in the core-site.xml file. Or 'hdfs://boray' This should match what is in the hdfs-site.xml file. **This must be specified when not using the Hive Metastore (see metastore option below)**
format		'delimited' | 'orc' | 'parquet'	Specifies the format of the file, which can either be 'delimited', 'orc' or 'parquet'. **This must be specified at the Connector level or table level.**
path		Non-empty, non-whitespace string.	Specifies the full path name to the HDFS file(s) associated with this table. This can only be specified as part of the table_specifier
charset	LANG setting	Non-empty, non-whitespace string.	Specifies the character set encoding for the associated HDFS file(s) See 7.10.11.8 for more details
delimiter	','	'<char>' Non-empty, single character string	Specifies the field delimiter. See 7.10.11.5.7 for more details
enclosed_by		'<char>[<char>]' or "" Non-empty, single or two character string	Specifies whether a field is optionally enclosed by a specified character. This is commonly used to specify that string fields are optionally

		If the enclosed_by is a single quote character then it must be specified using double quotes: """ The default is no enclosed_by	enclosed by either a single quote or double quote character and that character should not be included as part of the field data. If the same character is also included as part of the field data, then it must be escaped (see 7.10.11.5.3.1 for more details). See 7.10.11.5.8 for more details
escape_char	'\'	'<char>' Non-empty, single character string	Specifies the character to be used as an escape character. This will allow the user to include embedded field and record terminator characters in the data field as well embedded quotes in the event that the field is a string field that is enclosed within quote characters. See 7.10.11.5.9 for more details
ignore_header	'0'	Integer value >= 0	Specifies the number of header records to be skipped See 7.10.11.5.11 for more details
kerberos_keytab		Non-empty, non-whitespace string	Specifies the path name to the Linux file containing the Kerberos Keytab file for the user specified by the kerberos_user option
kerberos_user		Non-empty, non-whitespace string	Specifies the Kerberos principal (user) name
metastore		Non-empty, non-whitespace string.	Specifies the ip address and port number to be used for accessing the Hive Metastore. For example, 192.168.10.15:9083
namenodes		<node>:<port>,…	Specifies a comma-separated list of the name nodes in an HDFS HA Cluster. The name node can be specified as either: <host name>:<port> or <host ip address>:<port> For example, 'node1:8020, node2:8020' or

			'192.168.10.10:8020, 192.168.10.12:8020' This option must be specified when the 'hdfs' option specifies the nameservice ID (see 'hdfs' above).
nameservice		Non-empty, non-whitespace string.	The nameservice ID for an HDFS HA configuration. Example: 'boray' This option must be specified when the 'hdfs' option specifies the nameservice ID (see 'hdfs' above).
partitions_per_node	Number of cores available to the JVM	Integer value > 0 and < 128	The number of partitions allocated per node for this table.
readonly	False	True or False	Specifies whether the tables managed by this Connector are read-only. If the readonly option is set to true, then all write operations (INSERT, CREATE, DROP and TRUNCATE) will fail. By default, the READONLY option is set to FALSE.
terminator	'\n'	An optional single character, followed by one of the following characters: \n, \r\n or \r.	Allows the user to specify how records are terminated. See 7.10.11.5.10 for more details
use_datanode_hostname	'false'	'true' or 'false'	If this option is set to 'true', when the Connector accesses an HDFS data node, the data node id returned from the Name Node will be its hostname, instead of its IP address, which can help avoid the “cannot access data node” problem when HDFS is installed in a docker or similar environment
user		Non-empty, non-whitespace string	Specifies the name of the user to be used when accessing HDFS.

			See 7.10.11.3.1 for more details.

NOTES:

1. For a Hadoop Connector, the catalog name will be the name of the Connector
2. When connecting to the Hive Metastore, the schema name will match the Hive database name. When not connecting to the Hive Metastore, the schema name will be “PUBLIC”
3. The following standard SQL exclusions apply:
   - Varchars cannot have a length specification.
   - Null, not null designations are not supported. Nulls are allowed for all fields.
   - Primary keys are not supported

7.10.11.2 Setting up the Hadoop Connector for HDFS HA Configurations

When HDFS is configured for HA, the Hadoop Connector must have the following options set:

- If the "hdfs" option is used then it must be set to the nameservice ID
- The "nameservice" option must be set and it must specify the nameservice ID.
- The "namenodes" option must be set and it must specify a comma-separated list of the name nodes

Example 1:

```
CREATE CONNECTOR HDFS_TEST1 TYPE HADOOP WITH HDFS='hdfs://boray',
NAMESERVICE='boray', NAMENODES='node1:8020, node2:8020',
FORMAT='delimited', ENCLOSED_BY="'", USER='hdfs' NODE BORAY04 NODE BORAY05
TABLE …
```

Example 2, the same as example 1 except specifying the ip addresses for the name nodes:

```
CREATE CONNECTOR HDFS_TEST1 TYPE HADOOP WITH HDFS='hdfs://boray',
NAMESERVICE='boray', NAMENODES='192.168.10.10:8020, 192.168.10.12:8020',
FORMAT='delimited', ENCLOSED_BY="'", USER='hdfs' NODE BORAY04 NODE BORAY05
TABLE …
```

Example 3, using the Hive Metastore (see 7.10.11.10.1):


```
CREATE CONNECTOR HDFS_HIVEM TYPE HADOOP WITH METASTORE='192.168.10.14:9083',
NAMESERVICE='boray', NAMENODES='node1:8020, node2:8020',
NODE * CATALOG * SCHEMA TPCH_SF1 TABLE * ;
```

7.10.11.3 Setting up HDFS Access Privileges for the Hadoop Connector (non-Kerberos)

7.10.11.3.1 USER Option

By default, when accessing HDFS the Hadoop Connector will use a user id set to “anonymous”.

The USER option allows the user to specify the userid to be used when accessing the HDFS files specified for this Connector:

Syntax:

USER='<user name>'

Example:

```
CREATE CONNECTOR HDFS_TEST1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.202:8020', FORMAT='delimited', ENCLOSED_BY="'",
USER='hdfs' NODE BORAY04 NODE BORAY05
TABLE …
```

The Connector HDFS_TEST1 would access HDFS using the userid of 'hdfs'

7.10.11.3.2 SELECT Access

In order for the user to be able to run SELECT queries against any of the tables defined by a Hadoop Connector, the userid being used by the Hadoop Connector must be given read access to the underlying HDFS files associated with the tables.

7.10.11.3.3 INSERT Access

In order for the user to be able to run INSERT queries against any of the tables defined by a Hadoop Connector, the userid being used by the Hadoop Connector must be given write access to the underlying HDFS files associated with the tables.

7.10.11.3.4 TRUNCATE

In order to be able to run the TRUNCATE command against any of the tables defined by a Hadoop Connector, the userid being used by the Hadoop Connector must be given write access to the underlying HDFS files associated with the tables.

7.10.11.3.5 CREATE/DROP TABLE

In order to be able to create and drop tables from the Hive Metastore (see 7.10.12.8), the userid being used by the Hadoop Connector must be given write access to the underlying HDFS files associated with the tables.

7.10.11.4 Kerberos Authentication

7.10.11.4.1 Overview

The Hadoop Connector also supports the ability to authenticate users using Kerberos. When configuring the Hadoop Connector the kerberos_user and kerberos_keytab options (see 7.10.11.4.4) instruct the Hadoop Connector to use Kerberos authentication.

7.10.11.4.2 Setting up for Kerberos Configuration File, krb5.conf

In order for a Hadoop Connector to use Kerberos authentication the Kerberos configuration file used to configure the Kerberos Admin server, krb5.conf, must be present on each node of the RapidsDB Cluster where the Hadoop Connector is configured to run. There are several alternatives for where to locate the configuration file (see https://docs.oracle.com/javase/8/docs/technotes/guides/security/jgss/tutorials/KerberosReq.html for more information):

- If the system property java.security.krb5.conf is set, its value is assumed to specify the path and file name. In order to set this property, the following option must be specified when starting up the RapidsDB Cluster using the bootstrapper:
    - ./bootstrapper.sh -a start --jvm_settings "-Djava.security.krb5.conf=<path to krb5.conf file>"
      For example:
      ./bootstrapper.sh -a start --jvm_settings "-Djava.security.krb5.conf=/opt/rdp/krb5.conf"

- If that system property value is not set, then the configuration file is looked for in the directory

  <java-home>\lib\security (Windows)

  <java-home>/lib/security (Solaris and Linux)

  Here <java-home> refers to the directory where the JRE was installed. For example, if java-home is /user/java/default on Linux, the directory in which the configuration file is looked for is:

  /user/java/default/lib/security

- If the file is still not found, then an attempt is made to locate it as follows:

  /etc/krb5.conf (Linux)

7.10.11.4.3 Setting up /etc/hosts File

If there is no DNS server setup on the network to resolve the hostname of the Kerberos Admin Server, then the /etc/hosts file on each node in the RapidsDB cluster where the Hadoop Connector is configured to run must have an entry for the host name for the Kerberos Admin server which must match the admin_server entry from the krb5.conf file.

For example, if the following is from the krb5.conf file:


```
[realms]
 RDP.COM = {
  admin_server = kerberose1
  kdc = kerberose1
 }
```

then the /etc/hosts file should have an entry such as the following (where host name kerberose1 has an ip address of 192.168.10.202):

192.168.10.202 kerberose1

7.10.11.4.4 Configuring the Hadoop Connector to use Kerberos

The following two options must be specified in order to have the Hadoop Connector use Kerberos authentication:

1. KERBEROS_USER='<Kerberos principal name>' this option specifies the Kerberos principal to be used for authenticating access to HDFS. The name can optionally include the Kerberos Realm name, for example, dave@RDP.COM If the Realm is not specified then it will default to the setting from the krb5.conf file (see 7.10.11.4.2).

   Examples:
   KERBEROS_USER='dave'
   KERBEROS_USER='dave@RDP.COM'
   KERBEROS_USER='dave/hr'
   KERBEROS_USER='dave/hr@RDP.COM'

2. KERBEROS_KEYTAB='<path to keytab file>' this option specifies the path to the Linux file that contains the Kerberos keytab file for the user specified by the KERBEROS_USER option above.
   NOTE: the keytab file must be copied to the same location on each node in the RapidsDB cluster where the Hadoop Connector is configured to run, and must be configured for read access by the userid used to start up the RapidsDB cluster. It is highly recommended for security reasons that the keytab file ONLY be secured for read access by the userid used to startup the RapidsDB Cluster.

   Example:
   KERBEROS_KEYTAB='/opt/rdp/dave.keytab'

7.10.11.4.5 Example Connectors Configured to use Kerberos

```
CREATE CONNECTOR KTEST TYPE HADOOP
WITH hdfs='hdfs://192.168.10.202:8020', FORMAT='delimited', ENCLOSED_BY="'",
KERBEROS_USER='dave', KERBEROS_KEYTAB='/home/rapids/rdptest/dave.keytab'
NODE BORAY04 NODE BORAY05
TABLE …
```


The above Connector will authenticate with Kerberos using the Kerberos principal named 'dave', with the Kerberos keytab file /home/rapids/rdptest/dave.keytab (which must be located on nodes boray04 and boray05).

```
CREATE CONNECTOR KTEST TYPE HADOOP
WITH hdfs='hdfs://192.168.10.202:8020', FORMAT='delimited', ENCLOSED_BY="'",
KERBEROS_USER='dave@RDP.COM', KERBEROS_KEYTAB='/home/rapids/rdptest/dave.keytab'
NODE BORAY04 NODE BORAY05
TABLE …
```

This Connector is equivalent to the first Connector assuming that the default Kerberos Realm is "RDP.COM".

7.10.11.5 Delimited File Formatting

7.10.11.5.1 Specifying Delimited Format

The Hadoop Connector supports the reading and writing of delimited files. To specify that files are using the delimited, the "format" option must be set to 'DELIMITED '. For example, at the Connector level

```
CREATE CONNECTOR PAR1 TYPE HADOOP WITH HDFS='hdfs://192.168.10.15:8020',
FORMAT='delimited' NODE * CATALOG * SCHEMA *
TABLE …
```

Or at the table level:

```
CREATE CONNECTOR PAR1 TYPE HADOOP WITH HDFS='hdfs://192.168.10.15:8020', NODE *
CATALOG * SCHEMA *
TABLE T1 USING (…) WITH format='delimited';
```

7.10.11.5.2 Delimited Format Options

The Hadoop Connector provides the following set of options as part of the Connector definition for controlling how the data in delimited files is formatted:

Key:	Default Value	Value syntax	Description
delimiter	','	'<char>' Non-empty, single character string	Specifies the field delimiter. See 7.10.11.5.7 for more details
enclosed_by		'<char>[<char>]' or "'"	Specifies whether a field is optionally enclosed by a specified character. This is

		Non-empty, single or two character string If the enclosed_by is a single quote character then it must be specified using double quotes: """" The default is no enclosed_by	commonly used to specify that string fields are optionally enclosed by either a single quote or double quote character and that character should not be included as part of the field data. If the same character is also included as part of the field data, then it must be escaped (see 7.10.11.5.3.1 for more details). See 7.10.11.5.8 for more details
escape_char	'\'	'<char>' Non-empty, single character string	Specifies the character to be used as an escape character. This will allow the user to include embedded field and record terminator characters in the data field as well embedded quotes in the event that the field is a string field that is enclosed within quote characters. See 7.10.11.5.9 for more details
ignore_header	'0'	Integer value >= 0	Specifies the number of header records to be skipped See 7.10.11.5.11 for more details
terminator	'\n'	An optional single character, followed by one of the following characters: \n, \r\n or \r.	Allows the user to specify how records are terminated. See 7.10.11.5.10 for more details

7.10.11.5.3 Text Handling

7.10.11.5.3.1 ESCAPE Sequences

There are a set of special characters that only come into effect when the escape character is specified (by default the escape character is set to a backslash). If the ESCAPE_CHAR is set to '' (empty string) then the following are just treated as regular text.

In the following table, the ESCAPE_CHAR is set to the backslash character. The data stored will be the ASCII Character with the exception of NULL, which is stored as a null value:

Escape Sequence	ASCII Character
\b	A backspace character <x08>
\f	A form feed character <x0F>
\n	A newline (linefeed) character <x0A>
\r	A carriage return character <x0D>
\t	A tab character <x09>
\Z	ASCII 26 (Control+Z) <x1A>
\N	NULL[1]
\\	\
\<DELIMITER>	<DELIMITER>[2]
\<ENCLOSED_BY>	<ENCLOSED_BY>[3]
\<character>	<character>[4]

Notes:

1. The escape sequence \N is only treated as the null value when that escape sequence is the only content of the input field. If the input field contains any other characters then \N will not be treated as an escape sequence, and will just be the character "N" (see 4 below)
2. If the data field is not enclosed, and if the data field includes the DELIMITER character, then the DELIMITER character must be escaped
3. If the ENCLOSED_BY is set, and if the data field includes the ENCLOSED_BY character then the ENCLOSED_BY character must be escaped. If the ENCLOSED_BY is 2 characters, then the escaping ONLY applies to the second character specified by the ENCLOSED_BY
4. For any other 2-character sequence, the escape character will be stripped and the following character is taken as the input. For example, \J will be stored as the single character "J"

Examples:

Example 1:

ENCLOSED_BY is not set (this is the default)

ESCAPE_CHAR='\' (this is the default)

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited',
CATALOG *
SCHEMA *
TABLE T1 USING (c1 varchar) WITH path='/data/sample/test/t1';
```

In the following table, <xnn> refers to the hexadecimal value stored for the character

INPUT	C1
\N	<null value>
'\N'	'N'
\N not on its own	N not on its own
\\N not on its own	\N not on its own
A tab \t	A tab <x09>
Addr 1\nAddr 2	Addr 1<x0A>Addr 2
Other chars \A\B	Other chars AB
Part 1\, past 2	Part 1, part 2
Dave’s house	Dave’s house

Example 2:

ENCLOSED_BY="'"

ESCAPE_CHAR='\' (this is the default)

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', enclosed_by="'"
CATALOG *
SCHEMA *
TABLE T1 USING (c1 varchar) WITH path='/data/sample/test/t1';
```

In the following table, <xnn> refers to the hexadecimal value stored for the character

INPUT	C1
\N	<null value>
'\N'	<null value>
\N not on its own	N not on its own
\\N not on its own	\N not on its own
A tab \t	A tab <x09>
Addr 1\nAddr 2	Addr 1<x0A>Addr 2
Other chars \A\B	Other chars AB
'Part 1, part 2'	Part 1, part 2
'Dave\'s house'	Dave's house

Example 3:

ENCLOSED_BY='{}'

ESCAPE_CHAR='\' (this is the default)

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', enclosed_by='{}'
```



```
CATALOG *
SCHEMA *
TABLE T1 USING (c1 varchar) WITH path='/data/sample/test/t1';
```

In the following table, <xnn> refers to the hexadecimal value stored for the character

INPUT	C1
\N	<null value>
{\N}	<null value>
{\N not on its own}	N not on its own
{\\N not on its own}	\N not on its own
{A tab \t}	A tab <x09>
{ Addr 1\nAddr 2}	Addr 1<x0A>Addr 2
{Other chars \A\B}	Other chars AB
{Part 1, part 2}	Part 1, part 2
{ Dave's house}	Dave's house
{{My home\}}	{My home}

7.10.11.5.3.2 Handling of Leading and Trailing Blanks

Leading and trailing space characters are considered part of a VARCHAR column. NOTE: When the ENCLOSED_BY is set, the leading and trailing space characters are ONLY those characters contained within the enclosed string (see examples below for more on this), any space characters outside of the enclosing characters are ignored.

Examples:

Example 1:

ENCLOSED_BY is not set (this is the default)

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar, c3 integer) WITH path='/data/sample/test/t1';
```

In the following <x20> is used to signify an ASCII space character (hex value 20)

INPUT	C1	C2	C3
1,<x20><x20><x20>3 leading,3	1	<x20><x20><x20>3 leading	3
1,<x20><x20><x20>'3 leading',3	1	<x20><x20><x20>'3 leading'	3
1,<x20><x20><x20>'3 leading\,2 trailing' ,3	1	<x20><x20><x20>'3 leading,2 trailing'<x20><x20>	3
1,'<x20><x20><x20>3 leading\,2 trailing ',3	1	'<x20><x20><x20>3 leading,2 trailing<x20><x20>'	3

Example 2:

ENCLOSED_BY="'"

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', enclosed_by="'"
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar, c3 varchar) WITH path='/data/sample/test/t1';
```

INPUT	C1	C2	C3
1,<x20><x20><x20>3 leading,3	1	<x20><x20><x20>3 leading	3
1,<x20><x20><x20>'3 leading',3	1	3 leading	3
1,<x20><x20><x20>'3 leading,2 trailing' ,3	1	3 leading,2 trailing	3
1,'<x20><x20><x20>3 leading,2 trailing ',3	1	<x20><x20><x20>3 leading,2 trailing<x20><x20>	3

Example 3:

ENCLOSED_BY='{}'

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', enclosed_by='{}'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar, c3 integer) WITH path='/data/sample/test/t1';
```


INPUT	C1	C2	C3
1,<x20><x20><x20>3 leading,3	1	<x20><x20><x20>3 leading	3
1,<x20><x20><x20>{3 leading},3	1	3 leading	3
1,<x20><x20><x20>{3 leading,2 trailing} ,3	1	3 leading,2 trailing	3
1,{<x20><x20><x20>3 leading,2 trailing },3	1	<x20><x20><x20>3 leading,2 trailing<x20><x20>	3

7.10.11.5.3.3 EMPTY STRINGS

An empty (zero-length) string is defined as a field with two adjacent enclosed_by characters (see 7.10.11.5.8 for more information on enclosed_by characters). For example, the second field in the sample record below would be interpreted as an empty string assuming that the enclosed_by character is the single quote character:

```
1,'',C2 is an empty string
```

The statement select char_length(c2) from hadoop.public.test; would return the value zero for the record above after it was loaded.

NOTE – this is different from an empty field, where no value is specified, which is interpreted as a NULL value (see 7.10.11.5.6 for more information on nulls) as shown in the example below:

```
1,,C2 is a NULL
```

7.10.11.5.4 DATE_FORMAT (DATES and TIMESTAMPS)

The user can specify the format for date strings for Dates and Timestamps. Timestamps consist of a date portion followed by a time portion. The format for the date portion can be specified using the DATE_FORMAT option (see below), whereas the format for the time portion is fixed as HH:MM:SS[.NNNNNN]. If the time component is missing then the time will be set as 00:00:00.

As for all data fields, dates and timestamps can be optionally enclosed using the ENCLOSED_BY character(s) (see7.10.11.5.8).

The date format can be specified as any combination of the following along with any specified separator character:

- YYYY
- MM
- DD

Where

- YYYY can be entered as either 4 digits or 2 digits. In the case of 2 digits, 2000 will be added to the year. For example, an input of 18 would be treated as 2018
- MM can be entered as 1 or 2 digits in the range 1-12.
- DD can entered as 1 or 2 digits in the range 1-31 (with applicable rules applied for validating the correct number of days in a month)

Some example formats:

- DATE_FORMAT='MM-DD-YYYY'
- DATE_FORMAT='YYYY.MM.DD'
- DATE_FORMAT='DD/MM/YYYYY'
- DATE_FORMAT='YYYY-DD-MM'

The default for DATE_FORMAT is 'YYYY-MM-DD'

The table below shows some examples:

Data Type	DATE_FORMAT	INPUT	TREATED AS
Date	'MM-DD-YYYY'	'4-30-18' '10-31-1998'	04-30-2018 10-31-1998
Timestamp	'MM-DD-YYYY'	'4-30-18 09:00:00' '10-31-1998'	04-30-2018 09:00:00 10-31-1998 00:00:00
Timestamp	'YYYY.MM.DD'	'2018.03.31' '18.05.30 09:00:00.123456'	2018.03.31 00:00:00 2018.05.30 09:00:00.123456
Timestamp		'2018-03-31'	2018-03-31 00:00:00

Timestamps with a fractional scale of more than 6 digits will get truncated to 6 digits. For example:

- 2018-01-01 09:00:00.12345678 would get stored as 2018-01-01 09:00:00.123456

Example:

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', date_format='YYYY.MM.DD'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 date, c3 timestamp) WITH path='/data/sample/test/t1';
```

This example will set the date format for table t1 to be YYYY.MM.DD

7.10.11.5.5 BOOLEANS

The table below specifies the valid input values for booleans:

Column value	Possible Inputs
FALSE	0 f F False[1]
TRUE	>0 t T True[1]

Notes:

1. The string false or true can be specified in mixed case, for example False, false. FALSE and fALSE are all valid.

7.10.11.5.6 NULL Handling

There are three ways to specify a null value for a field:

1. Using the keyword NULL – a 4-character field with just the 4 characters null (case independent). NOTE, a null value cannot be specified as an enclosed field using the ENCLOSED_BY (see 6.5.12.4.6) character(s). For example, if ENCLOSED_BY is set to a single quote then the input field 'null' would be stored as the 4-character string null and not as a null value.
2. An empty field – an empty field is defined as two adjacent delimiters, or a delimiter followed immediately by the record terminator.
3. \N – a 2-character field with just \N

Example 1:

ENCLOSED_BY is not set (this is the default)

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar, c3 integer) WITH path='/data/sample/test/t1';
```


INPUT	C1	C2	C3
NULL,Null,nULL	<null value>	<null value>	<null value>
null,\N,NULL	<null value>	<null value>	<null value>
,,	<null value>	<null value>	<null value>
1,'null',3	1	'null'[1]	3
1, null,3	1	<x20>null[2]	3
1,'\N',null	1	'N'[3]	<null value>
1,\N is not a null,3	1	N is not a null[4]	
1,'\\N',null	1	'\N'[5]	<null value>
1,The word null,3	1	The word null	3
1,'The word null',3	1	‘The word null‘	3
1,'null is not a null',3	1	'null is not a null'	3

Notes:

1. The reason that column C2 is the 6-character string 'null' is because there is no enclosed_by character which means that the first character of the field is the single quote character, and so this is a 6-character field (to match the keyword “null” the field has to be a 4-character field).
2. In this example, the second field has a leading space character followed by the string “null”. Due to the fact that leading blanks are significant (see 7.10.11.5.3.2), this is a 5-character field due to the leading space, and so it will not match the keyword “null” because to do so requires the field to only have the 4 characters “null”. The string <x20> is used to signify the ASCII SPACE character (hex 20).
3. The reason that column C2 is the string 'N' is because there is no enclosed_by character which means that the first character of the field is the single quote character, and so the escaping for \N as null does not apply as the field does not contain just the two character string “\N”, and so this is not an escape sequence, it will be treated as the single character N
4. The reason that column C2 is the string 'N is not a null' is because the sequence \N is not the only content for the field, and so the string “\N” will not be treated as an escape sequence, it will be treated as the single character N. This is similar to note 3 above.
5. The string “\\” is a valid escape sequence for the backslash character

Example 2:

ENCLOSED_BY= """

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', enclosed_by="'"
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar, c3 integer) WITH path='/data/sample/test/t1';
```


INPUT	C1	C2	C3
NULL,Null,nULL	<null value>	<null value>	<null value>
null,\N,NULL	<null value>	<null value>	<null value>
,,	<null value>	<null value>	<null value>
1, 'null',3	1	null[1]	3
1, null,3	1	<x20>null[2]	3
1,'\N',null	1	N[3]	<null value>
1,\N is not a null,3	1	N is not a null[4]	3
1, '\\N',null	1	\N[5]	<null value>
1,The word null,3	1	The word null	3
1, 'The word null',3	1	The word null	3
1,'null is not a null',3	1	null is not a null	3

Notes:

1. The reason that column C2 is the string null and not a null value is because the ENCLOSED_BY character is set, and null values cannot be enclosed.
2. In this example, the second field has a leading space character followed by the string “null”. Due to the fact that leading blanks are significant (see 6.5.12.4.1.2), this is a 5-character field due to the leading space. The string <x20> is used to signify the ASCII SPACE character (hex 20).
3. The reason that column C2 is the character N is because the ENCLOSED_BY character is set, and null values cannot be enclosed, and so the sequence \N is not treated as a special character sequence, it is treated as the single character N.
4. The reason that column C2 is the string “N is not a null” is because the sequence \N is not the only content for the field, and so \N will not be treated as an escape sequence, it will be treated as the single character N
5. The string “\\” is a valid escape sequence for the backslash character

Example 3:

ENCLOSED_BY='{}'

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', enclosed_by='{}'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar, c3 integer) WITH path='/data/sample/test/t1';
```


INPUT	C1	C2	C3
NULL,Null,nULL	<null value>	<null value>	<null value>
null,\N,NULL	<null value>	<null value>	<null value>
,,	<null value>	<null value>	<null value>
1, {null},3	1	null[1]	3
1, {\N},null	1	N[2]	<null value>
1,{\N is not a null},3	1	N is not a null[3]	3
1, {\\N},null	1	\N[4]	<null value>
1,The word null,3	1	The word null	3
1, {The word null},3	1	The word null	3
1, {null is not a null},3	1	null is not a null	3

Notes:

1. The reason that column C2 is the string null and not a null value is because the ENCLOSED_BY character is set, and null values cannot be enclosed.
2. The reason that column C2 is the character N is because the ENCLOSED_BY character is set, and null values cannot be enclosed, and so the sequence \N is not treated as a special character sequence, it is treated as the single character N.
3. The reason that column C2 is the string "N is not a null" is because the sequence \N is not the only content for the field, and so \N will not be treated as an escape sequence, it will be treated as the single character N
4. The string "\\" is a valid escape sequence for the backslash character

7.10.11.5.7 DELIMITER='<char> | \t'

Specifies the field delimiter character. The field delimiter can be a single character or the tab character (\t).

Default: ',' (comma)

Example:

The input file has the fields delimited by the dollar character '$'.

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', delimiter='$'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar, c3 varchar) WITH path='/data/sample/test/t1';
```

Input data:


```
1$This is a text field$Second text field
```

The resulting data returned from a select * from t1 would be

```
C1 C2                   C3
-- --                   --
1  This is a text field  Second text field
```

7.10.11.5.8 ENCLOSED_BY='<char>[<char>]' | ""

Specifies whether an input field is optionally enclosed by the specified character or by two characters where the first character will define the start of the enclosing, and the second character will define the end of the enclosing. This is commonly used to specify that character fields are enclosed by either a single quote or double quote character and that character should not be included as part of the field data.

NOTES

1. To explicitly specify a single quote as the delimiter, you must enclose the single quote inside double quotes, all other characters are specified using single quotes.
2. Use of the ENCLOSED_BY for character fields is optional, and so an input record could include some fields using the enclosed_by character with other character fields not using the enclosed_by character as shown in the example below.
3. If the ENCLOSED_BY character(s) is also included as part of the field data, then the character(s) must be escaped (see ESCAPE_CHAR 7.10.11.5.9).

Default: no enclosing character(s)

Examples:

ENCLOSED_BY	INPUT	DATA TO BE STORED	DATA TYPE	VALID?
	'DAVE's DATA'	'DAVE's DATA'	VARCHAR	Y
	null	<null value>	VARCHAR	Y
	'null'	'null'	VARCHAR	Y
	'9'	INVALID	INTEGER	N
	'9'	INVALID	DECIMAL	N
	'9'	INVALID	FLOAT	N
	'30-04-2018 09:00:00'	INVALID	TIMESTAMP	N
	'T'	INVALID	BOOLEAN	N
	DAVE's DATA	DAVE's DATA	VARCHAR	Y
	9	9	INTEGER	Y
	9	9	DECIMAL	Y
	9	9	FLOAT	Y
	30-04-2018 09:00:00	30-04-2018 09:00:00	TIMESTAMP	Y
	T	T	BOOLEAN	Y

ENCLOSED_BY="'"	'DAVE\'s DATA'	DAVE's DATA	VARCHAR	Y
	'DAVE's DATA'	INVALID	VARCHAR	N
	'null'	null	VARCHAR	Y
	null	<null value>	VARCHAR	Y
	'9'	9	INTEGER	Y
	'9'	9	DECIMAL	Y
	'9'	9	FLOAT	Y
	'30-04-2018 09:00:00'	30-04-2018 09:00:00	TIMESTAMP	Y
	'T'	T	BOOLEAN	Y
	DAVE's DATA	DAVE's DATA	VARCHAR	Y
	9	9	INTEGER	Y
	9	9	DECIMAL	Y
	9	9	FLOAT	Y
	30-04-2018 09:00:00	30-04-2018 09:00:00	TIMESTAMP	Y
	T	T	BOOLEAN	Y
ENCLOSED_BY='[]'	[DAVE's DATA]	DAVE's DATA	VARCHAR	Y
	{null}	null	VARCHAR	Y
	null	<null value>	VARCHAR	Y
	[9]	9	INTEGER	Y
	[9]	9	DECIMAL	Y
	[9]	9	FLOAT	Y
	[30-04-2018 09:00:00]	30-04-2018 09:00:00	TIMESTAMP	Y
	[T]	T	BOOLEAN	Y
	[WITH \[\]]	WITH []	VARCHAR	Y
	[WITH[]]	INVALID	VARCHAR	N
	'DAVE's DATA'	'DAVE's DATA'	VARCHAR	Y
	'9'	INVALID	INTEGER	N
	'9'	INVALID	DECIMAL	N
	'9'	INVALID	FLOAT	N
	'30-04-2018 09:00:00'	INVALID	TIMESTAMP	N
	'T'	INVALID	BOOLEAN	N
	DAVE's DATA	DAVE's DATA	VARCHAR	Y
	9	9	INTEGER	Y
	9	9	DECIMAL	Y
	9	9	FLOAT	Y
	30-04-2018 09:00:00	30-04-2018 09:00:00	TIMESTAMP	Y
	T	T	BOOLEAN	Y

Example 1:

The input records below contains fields which are enclosed in double quotes.


```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', enclosed_by= '"'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 varchar, c2 varchar) WITH path='/data/sample/test/t1';
```

Input data:

```
"Record 1", "Some text"
"Record 2", "Some more text"
```

The resulting data returned from a select * from t1 would be:

```
C1         C2
--         --
Record 1   Some text
Record 2   Some more text
```

Example 2:

The input records below contains fields which are enclosed in single quotes. The second record includes the enclosed_by character as part of the second field, and so the enclosed_by character has to be escaped (see 7.10.11.5.8). Also note that the first field does not use the enclosed_by character which is allowed because the enclosed_by character, when specified, is optional for any given field.

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', enclosed_by= "'"
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar) WITH path='/data/sample/test/t1';
```

Input data:

```
1, 'This is an example of a field that includes the delimiter character, a comma'
2, 'This field includes the enclosed_by character \''
```

The resulting data returned from a select * from t1 would be:


```
C1 C2
-- --
 1 This is an example of a field that includes the delimiter character, a comma
 2 This field includes the enclosed_by character '
```

Example 3:

The input records below contains fields which are enclosed in {}.

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', enclosed_by= '{}'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar) WITH path='/data/sample/test/t1';
```

Input data:

```
1, {This is an example of a field that includes the delimiter character, a comma}
2, {This field includes the enclosed_by characters: \{\}}
```

The resulting data returned from a select * from t1 would be:

```
C1 C2
-- --
 1 This is an example of a field that includes the delimiter character, a comma
 2 This field includes the enclosed_by characters:  {}
```

7.10.11.5.9 ESCAPE_CHAR='<char>'

Specifies the character to be used as the escape character.

Default: '\' (backslash)

Example:

The input record below contains a character field that is not enclosed in quotes. In this example we are using the character '^' as the escape character. The character field includes both the field separator (comma) and a newline character (note that the newline character uses the escape_char).


```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', escape_char='^'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar) WITH path='/data/sample/test/t1';
```

Input data:

```
1,Field with special chars: ^,^nThis is the second line,Text should be on 2 lines
```

The resulting data returned from a select * from t1 would be:

```
C1 C2
-- --
 1 Field with special chars: ,
This is the second line   Text should be on 2 lines
```

7.10.11.5.10 TERMINATOR='[<char>]\n' | '[<char>]\r\n' | '[<char>]\r']

Specifies that records are terminated by an optional character followed by one of the following character sequences: \n, \r\n or \r.

Default: '\n'

Example 1:

The input file is a delimited file where each record is terminated by \r\n.

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', terminator='\r\n'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar, c3 varchar) WITH path='/data/sample/test/t1';
```

Input data: in the example input data below the sequence <\r><\n> indicates the ANSI characters for \r and \n:

```
1,This is a text field,This is a second text field<\r><\n>
```

The resulting data returned from a select * from t1 would be


```
C1 C2                    C3
-- --                    --
 1 This is a text field This is a second text field
```

Example 2:

The input file is a delimited file where the delimiter is '|' and each record is terminated by '|\n'.

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', delimiter='|', terminator='|\n'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar) WITH path='/data/sample/test/t1';
```

Input data: in the example input data below the sequence <\n> indicates the ANSI character for \n:

```
1|This is a text field|This is a second text field|<\n>
```

The resulting data returned from a select * from t1 would be

```
C1 C2                    C3
-- --                    --
 1 This is a text field This is a second text field
```

7.10.11.5.11 IGNORE_HEADER

Specifies the number of header records to be ignored at the start of the file.

Example:

The input record below contains a header record followed by a record with three fields separated by a comma (the default delimiter).

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', ignore_header='1'
CATALOG *
SCHEMA *
TABLE T1 USING ( c1 integer, c2 varchar, c3 varchar) WITH path='/data/sample/test/t1';;
```


Input data:

```
C1 INTEGER,C2 VARCHAR,C3 VARCHAR
1,This is the first field,This is the second text field
```

The resulting data returned from a select * from t1 would be:

```
C1 C2                    C3
-- --------------------- ----------------------------
1  This is the first field This is the second text field
```

7.10.11.5.12 ERROR HANDLING

In the event that the Hadoop Connector is unable to process the data due to a problem with the format of the data, the Hadoop Connector will one of the following errors and stop processing the query.

- The TERMINATOR definition is not consistent with data file
  This error occurs when the record terminator does not match the TERMINATOR clause. For example, if the data file was from Windows and terminated with '\r\n', then the TERMINATOR clause must be to TERMINATOR='\r\n'
- Invalid data in Hadoop : definition of table has 5 columns while the data record has 1 fields
  This error occurs when the DELIMITER used in the data file does not match the DELIMITER specified by the DELIMITER clause.
- Invalid data in Hadoop : Timestamp format must be yyyy-mm-dd hh:mm:ss[.fffffffff]
  erroneous line : aa,99.99,'2012-01-01 09:00:00',true,valid data
  In the event that the data value does not match the data type for the associated column then the Hadoop Connector will report an error similar to the error message above, and the "erroneous line" will show the record in the data file that was being processed when the error occurred.

7.10.11.6 ORC Format

7.10.11.6.1 Specifying ORC Format

The Hadoop Connector supports the reading and writing of files using the ORC format. To specify that files are using ORC the "format" option must be set to 'ORC'. For example, at the Connector level

```
CREATE CONNECTOR PAR1 TYPE HADOOP WITH HDFS='hdfs://192.168.10.15:8020',
FORMAT='orc' NODE * CATALOG * SCHEMA *
TABLE …
```

Or at the table level:


```
CREATE CONNECTOR PAR1 TYPE HADOOP WITH HDFS='hdfs://192.168.10.15:8020', NODE *
CATALOG * SCHEMA *
TABLE T1 USING (…) WITH format='orc';
```

7.10.11.6.2 ORC Format Options

The Hadoop Connector provides the following set of options as part of the Connector definition for controlling how the data in ORC files is formatted:

Key:	Default Value	Value syntax	Description
compression	'zlib'	'lz4' | 'snappy' | 'zlib' | 'zstd'	Specifies the type of compression used See 7.10.11.7.3 for more details
stripe_size	'262144000'	Integer value >= 0	Specifies the number of bytes in each stripe The default is 250MB

7.10.11.6.3 Compression

The Hadoop Connector supports reading and writing compressed ORC files. The Hadoop Connector supports lz4, snappy zlib, and zstd compression. The option "compression" is provided to allow the user to specify the compression being used:

- compression='lz4' specifies lz4 compression
- compression='snappy' specifies snappy compression
- compression='zlib' specifies zlib compression
- compression='zstd' specifies zstd compression

The default compression is zlib.

Example:

```
CREATE CONNECTOR PAR1 TYPE HADOOP WITH HDFS='hdfs://192.168.10.15:8020', FORMAT='orc' ,
COMPRESSION='lz4' NODE * CATALOG * SCHEMA *
TABLE …
```

The above Connector specifies that by default all of the files associated with the tables owned by this Connector are using the ORC file format with lz4 compression.

7.10.11.7 Parquet Format

7.10.11.7.1 Specifying Parquet Format

The Hadoop Connector supports the reading and writing of files using the Parquet format. To specify that files are using Parquet the "format" option must be set to 'PARQUET'. For example, at the Connector level

```
CREATE CONNECTOR PAR1 TYPE HADOOP WITH HDFS='hdfs://192.168.10.15:8020',
FORMAT='parquet' NODE * CATALOG * SCHEMA *
TABLE …
```

Or at the table level:

```
CREATE CONNECTOR PAR1 TYPE HADOOP WITH HDFS='hdfs://192.168.10.15:8020', NODE *
CATALOG * SCHEMA *
TABLE T1 USING (…) WITH format='parquet';
```

7.10.11.7.2 Parquet Format Options

The Hadoop Connector provides the following set of options as part of the Connector definition for controlling how the data in parquet files is formatted:

Key:	Default Value	Value syntax	Description
blocksize	HDFS block size	Integer value >= 0	Specifies the Parquet file block size
compression		'gzip' | 'lz4' | 'snappy'	Specifies the type of compression used See 7.10.11.7.3 for more details
dictionary_pagesize	'65536'	Integer value >= 0	Specifies the Parquet dictionary page size
pagesize	'65536'	Integer value >= 0	Specifies the parquet file page size

7.10.11.7.3 Compression

The Hadoop Connector supports reading and writing compressed Parquet files. The Hadoop Connector supports gzip, snappy and lz4 compression. The option "compression" is provided to allow the user to specify the compression being used:

- compression= 'gzip' specifies gzip compression
- compression= 'snappy' specifies snappy compression
- compression='lz4' specifies lz4 compression

Example:


```
CREATE CONNECTOR PAR1 TYPE HADOOP WITH HDFS='hdfs://192.168.10.15:8020', FORMAT='parquet' ,
COMPRESSION='gzip' NODE * CATALOG * SCHEMA *
TABLE …
```

The above Connector specifies that by default all of the files associated with the tables owned by this Connector are using the Parquet file format with gzip compression.

7.10.11.8 Configuring Character Set

By default, the Hadoop Connector will read (and write) HDFS files in the character encoding specified by the LANG setting for the Linux userid that was used to start the RapidsDB Cluster. The user can specify a specific character encoding to be used when reading and writing HDFS files using the following option:

- CHARSET= '<character set encoding>'

Where, <character set encoding> is the identifier for the character set used by Java (https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html), for example GBK, or GB18030.

Examples:

1. The Connector HDFS_GBK_TEST below is configured by default to have all files encoded in the GBK character set:

```
CREATE CONNECTOR HDFS_GBK_TEST TYPE HADOOP WITH HDFS='hdfs://192.168.10.15:8020',
FORMAT='delimited', ENCLOSED_BY="'", CHARSET='GBK' NODE BORAY03 NODE BORAY04 NODE
BORAY05 CATALOG * SCHEMA *
TABLE …
```

2. For the HDFS_TEST2 Connector below, the file associated with table GBK_TEST_1K is encoded using the GBK character set, and the file associated with the table GB18030_TEST_1K is encoded using the GB18030 character set:

```
CREATE CONNECTOR HDFS_TEST2 TYPE HADOOP WITH HDFS='hdfs://192.168.10.15:8020',
FORMAT='delimited', ENCLOSED_BY="'" NODE BORAY03 NODE BORAY04 NODE BORAY05
CATALOG * SCHEMA *
TABLE GBK_TEST_1K USING ( ROW_ID INTEGER, ASCII_COL VARCHAR, GBK_COL1 VARCHAR,
GBK_COL2 VARCHAR) WITH path='/user/rapids/dave/gbkdata/GBKTest_Data_1K',
CHARSET='GBK'
TABLE GB18030_TEST_1K USING ( ROW_ID INTEGER, ASCII_COL VARCHAR, GBK_COL1
VARCHAR, GBK_COL2 VARCHAR) WITH
path='/user/rapids/dave/gbkdata/GB18030Test_Data_1K', CHARSET='GB18030';
```


7.10.11.9 Hive-style Partitioning: PARTITION BY VALUE ON

The PARTITION BY VALUE ON clause allows the user to specify the columns to be used when the underlying files are organized using Hive-style partitioning. This feature is supported for both delimited and Parquet files. With Hive-style partitioning the data stored in HDFS is arranged in directories where the directory names match the values for columns in the table. For example, in the HDFS file structure below, the data is partitioned over the columns “region” and “country”, and so the files under /data/user/region=North America/country=US would match with a region of “North America” and country of “US”.

/data/user/region=North America/country=US
/data/user/region=North America/country=CA
/data/user/region=South America/country=BR
/data/user/region=South America/country=ME

When a query of the form SELECT <column list> FROM <table> WHERE REGION='North America' AND COUNTRY='US'; is submitted, the Hadoop Connector will use the predicate “REGION='North America' AND COUNTRY='US' “ to restrict the files to be read to those files in the directory /data/user/region=North America/country=US.

Below is an example Connector that is using Hive-style partitioning, where the partitioning columns are region and country:

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', delimiter='|'
CATALOG *
SCHEMA *
TABLE user
USING (
    userid     INTEGER,
    first_name VARCHAR,
    last_name  VARCHAR,
    address1   VARCHAR,
    address2   VARCHAR,
    city       VARCHAR,
    zip_code   VARCHAR,
    state      VARCHAR,
    region     VARCHAR,
    country    VARCHAR
) PARTITION BY VALUE ON (region, country) WITH path='/data/user';
```

NOTE:

1. The partitioning columns must be VARCHAR columns
2. The column names are case-insensitive and so the following two queries are equivalent:
   select * from user where region='North America' and country='US';
   select * from user where REGION='North America' and country='US';
3. The values specified for the partitioning columns are case-sensitive and must match the case specified in the HDFS directory naming, Based on the example Connector definition above, the second query below would fail to match on the country column and hence return zero rows:
   select * from user where region='North America' and country='US';
   select * from user where region='North America' and country='us';
4. While the columns region and country are defined as part of the table schema, the underlying data files **DO NOT** include the data for those columns. The values for the region and country columns are derived from the values associated with those columns in the HDFS directory structure. For example, in the following directory structure, the values for the region column would be "North America" and "South America", and for the country column would be "US", "CA", "BR" and "ME":

   /data/user/region=North America/country=US
   /data/user/region=North America/country=CA
   /data/user/region=South America/country=BR
   /data/user/region=South America/country=ME

   A query of the form select * from user where region = 'North America' and country = 'CA'; would result in the Hadoop Connector accessing all of the data files in the directory data/user/region=North America/country=CA, and for each row retrieved the value for the region column would be "North America" and the value for the country column would be "CA".

7.10.11.10 Hive Metatstore Integration

The Hadoop Connector supports accessing the Hive Metastore to get the schema information for any of the supported table types (see 7.10.11.10.2). The user can specify the Hive databases and tables to be accessed (see 7.10.11.10.1) and then issue SELECT, INSERT, TRUNCATE or DROP TABLE commands against those tables. In addition the user can create new tables (see 7.10.11.10.4) that will be registered in the specified Hive database, and can drop any tables from the Hive Metastore that the Hadoop Connector has access to.

7.10.11.10.1 Configuring Hive Metastore Access

To configure a Hadoop Connector to access the Hive Metastore, the METASTORE option must be specified and it must be set to the ip address and port number for the Hive Metastore. The SCHEMA option is then used to specify the Hive database(s) to be accessed and it can be qualified with the INCLUDES clause to restrict the tables to be accessed from the specified database.

Examples:


```
CREATE CONNECTOR CUSTOMER TYPE HADOOP WITH metastore='192.168.10.15:9083',
PARTITIONS_PER_NODE='1' NODE * CATALOG * SCHEMA customer;
```

In the above example, the CUSTOMER Connector would provide access to the tables defined in the Hive customer database.

```
CREATE CONNECTOR SALES TYPE HADOOP WITH metastore='192.168.10.15:9083',
PARTITIONS_PER_NODE='1' NODE * CATALOG * SCHEMA customer, sales;
```

In the above example, the SALES Connector would provide access to the tables defined in the Hive customer and sales databases.

7.10.11.10.1.1 Configuring Tables to be Accessed using INCLUDES

The INCLUDES clause supports three different ways to specify the names of the tables to be accessed as described in the following sections.

7.10.11.10.1.1.1 List of Table Names

The user can specify a comma-separated list of table names as shown in the example below:

```
CREATE CONNECTOR SALES TYPE HADOOP WITH metastore='192.168.10.15:9083',
PARTITIONS_PER_NODE='4', USER='rapids' NODE *  CATALOG *
SCHEMA customer WITH INCLUDES='customers_east, customers_west'
SCHEMA sales WITH INCLDUES='stores, products, inventory';
```

In the above example, the SALES Connector would provide access to the customers_east and customers_west tables from the Hive customer database, and from the stores, products and inventory tables from the Hive sales database.

7.10.11.10.1.1.2 Wildcarding

The user can specify a wildcarded string that will result in all of the tables that match the wildcard expression being included. Note that you can only specify one wildcard string, If you want to specify multiple wildcards then you can do the equivalent using a regex expression (see 7.10.11.10.1.1.3)

```
CREATE CONNECTOR SALES TYPE HADOOP WITH metastore='192.168.10.15:9083',
PARTITIONS_PER_NODE='4', USER='rapids' NODE * CATALOG *
SCHEMA customer WITH INCLUDES='cust*';
```

In the above example, the SALES Connector would provide access to all tables from the Hive customer database where the table names start with the string “cust”.

7.10.11.10.1.1.3 Regex

The user can specify any regex expression that will result in all of the tables that match the regex expression being included.

```
CREATE CONNECTOR SALES TYPE HADOOP WITH metastore='192.168.10.14:9083',
PARTITIONS_PER_NODE='4', USER='rapids' NODE * CATALOG *
```



```
SCHEMA sales WITH INCLUDES='{^(cust).*|^(orders).*}';
```

In the above example, the SALES Connector would provide access to all tables from the Hive customer database where the table names start with the string “cust” or the string “orders”.

7.10.11.10.2 Supported Hive Table Types

The Hadoop Connector supports access to the following Hive table types:

- Delimited (org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe), where the following Hive options apply:
  - ROW FORMAT DELIMITED
  - STORED AS TEXTFILE
  - For date and timestamp columns, only the following formats are supported:
    - 'yyyy-MM-dd'
    - 'yyyy-MM-ddHH:mm:ss'
  - The following additional options are also supported:
    - FIELDS TERMINATED BY
    - ESCAPED BY
    - NULL DEFINED AS
  - Compression is not supported
- ORC (org.apache.hadoop.hive.ql.io.orc.OrcSerde)
- Parquet (org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe)

The Hadoop Connector will ignore any tables that do not have the above table types, or any tables that include unsupported data types (see 7.10.11.10.3).

7.10.11.10.3 Mapping of Hive Data Types

Only tables with the following Hive data types will be supported, tables with any other data types will be ignored:

Hive Datatype	RDP Datatype
TINYINT	INTEGER(8)
SMALLINT	INTEGER(16)
INT	INTEGER(32)
BIGINT	INTEGER
BOOLEAN	BOOLEAN
FLOAT	FLOAT
DOUBLE	FLOAT
DOUBLE PRECISION	FLOAT
STRING	VARCHAR
TIMESTAMP	TIMESTAMP
DECIMAL	DECIMAL
DECIMAL(p,n)	DECIMAL(p,n)
DATE	DATE
VARCHAR(n)	VARCHAR

CHAR(n)	VARCHAR

7.10.11.10.4 CREATE TABLE

The user can create new tables to be registered in the Hive Metastore using the Hive database associated with the specified schema. The table formats supported are delimited, orc and parquet as defined in section 7.10.11.10.2.

7.10.11.10.4.1 Syntax

```
CREATE TABLE <tableReference>
(
  <columnDefinition>, ...
)
[ PARTITION [BY] (<expr>, ...)]
[WITH <key>='<value>' [,<key>='<value>'] [,<further key values>]]

where:

<tableReference> is:
  [catalog.][schema.]<table name>

<column definition> is:
  <columnName> <type> [[NOT] NULL]

<type> is:
            INTEGER [(precision)] |
            DECIMAL [(scale[, precision])] |
            FLOAT |
            VARCHAR [(size)] |
            BOOLEAN |
            TIMESTAMP |
             DATE

<column name> is: <SQL identifier>

<key>:
            See table below for list of possible keys
<value>:
            See table below for list of possible values
```


The table below shows the possible key-value pairs that can be specified using the WITH clause:

Key:	Default Value	Value syntax	Description
format		'delimited' | 'parquet' | 'orc'	Specifies the format of the file, which can either be 'delimited' 'orc' or 'parquet'. **If the format is not specified as part of the Connector definition then this must be specified**
path	If this option is not set, then the table will be created as a Hive-managed table where Hive will define the path.	Non-empty, non-whitespace string.	Specifies the full path name to the HDFS file(s) associated with this table. If this option is set, then this will result in the table being created as an EXTERNAL table in the Hive Metastore.
delimiter	','	'<char>' Non-empty, single character string	Specifies the field delimiter. Applies to delimited format only. See 7.10.11.5.7 for more details
escape_char	'\'	'<char>' Non-empty, single character string	Specifies the character to be used as an escape character. This will allow the user to include embedded field and record terminator characters in the data field as well embedded quotes in the event that the field is a string field that is enclosed within quote characters. Applies to delimited format only. See 7.10.11.5.9 for more details

7.10.11.10.4.2 Creating Hive-managed Tables

If the "path" key is not specified for the table then the table will be created in the Hive Metastore as a Hive-managed table, which means that Hive will assign the path for the files for the table, and a DROP TABLE request (see 7.10.11.10.5.1) will result in both the metadata and the data getting deleted.

Example:


```
rapids > CREATE CONNECTOR TEST_WRITE TYPE HADOOP WITH
PARTITIONS_PER_NODE='8', METASTORE='192.168.10.14:9083', USER='rapids'
NODE BORAY01 CATALOG * SCHEMA TEST_WRITE TABLE *;
0 row(s) returned (0.19 sec)
rapids > create table test_write.t_test(c1 integer, c2 timestamp) with
format='delimited', delimiter=',';
0 row(s) returned (1.53 sec)
rapids > describe table  test_write.t_test;
TABLE_NAME   COLUMN_NAME   DATA_TYPE     ORDINAL   IS_PARTITION_KEY
IS_NULLABLE   PRECISION   SCALE
----------   -----------   ---------     -------   ----------------
-----------   ---------   -----
t_test       c1            INTEGER            0              false
true          64    NULL
t_test       c2            TIMESTAMP          1              false
true        NULL    NULL

2 row(s) returned (0.22 sec)
```

The above example created the table t_test in the Hive database test_write with the following attributes:

- ROW FORMAT delimited
- STORED AS textfile
- FIELD TERMINATOR ','

7.10.11.10.4.3 Creating Hive EXTERNAL Tables

The user can also create tables that are registered in the Hive Metastore as EXTERNAL tables by specifying the path to be used for the files associated with the table using the "path" key in the WITH clause (see 7.10.11.10.4.1). For tables created as external tables, the DROP table command (see 7.10.11.10.5.2) will behave the same as DROP TABLE in Hive and it will only result in the metadata for the table getting dropped not the data.

Example:

```
rapids > CREATE CONNECTOR TEST_PAR TYPE HADOOP WITH USER='rapids',
PARTITIONS_PER_NODE='8', METASTORE='192.168.10.14:9083' NODE * CATALOG
* SCHEMA TEST_PAR TABLE *;
0 row(s) returned (0.16 sec)
rapids > create table test_par.external_t1(c1 integer, c2 varchar)
with format='delimited',
path='/user/rapids/dave/writetests/external_t1';
0 row(s) returned (2.21 sec)
rapids > describe table test_par.external_t1;
```



```
TABLE_NAME    COLUMN_NAME   DATA_TYPE     ORDINAL   IS_PARTITION_KEY
IS_NULLABLE   PRECISION   SCALE
----------    -----------   ---------     -------   ---------------
-----------   ---------   -----
external_t1   c1            INTEGER             0               false
true          64    NULL
external_t1   c2            VARCHAR             1               false
true         255    NULL

2 row(s) returned (0.19 sec)
```

The above example created the table external_t1 in the Hive database test_par. Below is the detailed information from Hive for this table:

```
| Detailed Table Information  | Table(tableName:external_t1, dbName:test_par, owner:rapids,
createTime:1571778049, lastAccessTime:0, retention:0,
sd:StorageDescriptor(cols:[FieldSchema(name:c1, type:bigint, comment:null), FieldSchema(name:c2,
type:varchar(255), comment:null)],
location:hdfs://192.168.10.15:8020/user/rapids/dave/writetests/external_t1,
inputFormat:org.apache.hadoop.mapred.TextInputFormat,
outputFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false,
numBuckets:0, serdeInfo:SerDeInfo(name:null,
serializationLib:org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe,
parameters:{serialization.format=,, field.delim=,}), bucketCols:[], sortCols:[], parameters:{},
skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}),
storedAsSubDirectories:false), partitionKeys:[], parameters:{transient_lastDdlTime=1571778049,
totalSize=6, EXTERNAL=TRUE, numFiles=1}, viewOriginalText:null, viewExpandedText:null,
tableType:EXTERNAL_TABLE, rewriteEnabled:false)  |        |
```

7.10.11.10.5 DROP TABLE

Syntax:

```
DROP TABLE [<catalog>.][<schema>.]<table>;
```

The Hadoop Connector supports DROP TABLE for any table currently being accessed from the Hive Metastore.

7.10.11.10.5.1 Dropping Hive-managed Tables

For Hive-managed tables (see 7.10.11.10.4.2) the table and all associated data will be dropped along with the table metadata from both the RapidsDB Metastore as well as the Hive Metastore.

Example:


```
rapids > CREATE CONNECTOR TEST_WRITE TYPE HADOOP WITH PARTITIONS_PER_NODE='8',
METASTORE='192.168.10.14:9083', USER='rapids' NODE * CATALOG * SCHEMA TEST_WRITE TABLE * ;
0 row(s) returned (0.32 sec)
rapids > create table test_write.t_test (c1 integer, c2 varchar) with format='delimited';
0 row(s) returned (1.59 sec)
rapids >  insert into test_write.t_test values(1,'abc');
0 row(s) returned (1.05 sec)
```

In the example above, the table t_test was created in the Hive database test_write as a Hive-managed table. Below is the data in HDFS for this table:

```
[rapids@boray01 ~]$ hdfs dfs -ls /user/rapids/warehouse/test_write.db/t_test
Found 1 items
-rw-r--r--   3 rapids supergroup          6 2019-10-23 07:20
/user/rapids/warehouse/test_write.db/t_test/7_0
```

The following DROP TABLE request will remove the table metadata as well as the data for this table:

```
rapids > drop table test_write.t_test;
0 row(s) returned (1.66 sec)
rapids > describe table test_write.t_test;
Table not found, or Catalog/Schema has been set for others.
0 row(s) returned (0.33 sec)
```

From HDFS the data file is no longer present:

```
[rapids@boray01 ~]$ hdfs dfs -ls /user/rapids/warehouse/test_write.db/t_test
ls: `/user/rapids/warehouse/test_write.db/t_test': No such file or directory
```

7.10.11.10.5.2 Dropping External Tables

For External tables (see 7.10.11.10.4.3) only the metadata associated with the table will get dropped, not the data associated with the table.

Example:

```
rapids > CREATE CONNECTOR TEST_PAR TYPE HADOOP WITH USER='rapids',
PARTITIONS_PER_NODE='8', METASTORE='192.168.10.14:9083' NODE * CATALOG * SCHEMA TEST_PAR
TABLE *;
0 row(s) returned (0.19 sec)
rapids > create table test_par.external_t1 (c1 integer, c2 varchar) with format='delimited',
path='/user/rapids/dave/writetests/external_t1';
0 row(s) returned (2.14 sec)
rapids > select * from external_t1;
```



```
 c1 c2
 -- --
  1 abc

1 row(s) returned (0.21 sec)
```

The above would create the table external_t1 in the Hive database test_par and register the table as an external table. The sequence below shows dropping the table, and the recreating the same table with the data still present from before:

```
rapids > drop table external_t1;
0 row(s) returned (1.96 sec)
rapids > set trace off;
rapids > select * from external_t1;
 com.rapidsdata.parser.exceptions.UsageException: Line 1 position 15: Unresolved table or stream
name: EXTERNAL_T1.. locus=BORAY01.
rapids > create table test_par.external_t1 (c1 integer, c2 varchar) with format='delimited',
path='/user/rapids/dave/writetests/external_t1';
0 row(s) returned (2.14 sec)
rapids > select * from external_t1;
 c1 c2
 -- --
  1 abc

1 row(s) returned (0.21 sec)
```

7.10.11.11 Writing to HDFS

7.10.11.11.1 INSERT

When writing data as the result of an INSERT request, the Hadoop Connector will write the data to files in the directory specified by the PATH option or to the directory as specified by the Hive Metastore for that table using the following naming convention:

<partition>_<file index>

Where,

- <partition> is the number of the partition writer
- <file index> is the ……

The Hadoop Connector supports writing the results of an INSERT or INSERT … SELECT statement to HDFS. The format of the data written to HDFS is controlled by the FORMAT option associated with a table as described below.

7.10.11.11.1.1 FORMAT='DELIMITED'

The format of the data written to HDFS will follow the delimited file options specified for the Connector (see 7.10.11.5). For example, if ENCLOSED_BY="'", then all character fields written out to HDFS will be enclosed in single quotes. See below for examples.

When writing to HDFS the data will either be appended to an existing file or will be written to new directories/files created by the Hadoop Connector depending on the definition in the Connector for the target table as described in the following sections.

Examples:

In the following example, the ENCLOSED_BY is not set:

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', delimiter=','
CATALOG *
SCHEMA *
TABLE t1 USING (c1 varchar, c2 integer, c3 timestamp) WITH path='/data/sample/test/t1';
```

INSERT Statement	Data record written to HDFS
INSERT INTO t1 values('abcdef',1,'2018-04-30 09:00:00');	abcdef,1,2018-04-30 09:00:00.0
INSERT INTO t1 values('Dave''s house',2, '2018-04-30 09:00:00.123');	Dave's house,2,2018-04-30 09:00:00.123
INSERT INTO t1 values('abc,def',3, '2018-04-30 09:00:00.12345678');	abc\,def,3,2018-04-30 09:00:00.123456[1]
INSERT INTO t1 values(null,4,null);	null,4,null

Note:

1. The fractional scale is truncated to 6 digits

In the following example the ENCLOSED_BY is set:

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', delimiter=',', enclosed_by= "'"
CATALOG *
SCHEMA *
TABLE t1 USING (c1 varchar, c2 integer, c3 timestamp) WITH path='/data/sample/test/t1';
```

INSERT Statement	Data record written to HDFS
INSERT INTO t1 values('abcdef',1,'2018-04-30 09:00:00');	'abcdef',1,'2018-04-30 09:00:00.0'
INSERT INTO t1 values('Dave''s house',2, '2018-04-30 09:00:00.123');	'Dave\'s house',2,'2018-04-30 09:00:00.123'

INSERT INTO t1 values('abc, def',3, '2018-04-30 09:00:00.12345678');	'abc,def',3,'2018-04-30 09:00:00.123456[1]'
INSERT INTO t1 values(null,4,null);	null,4,null

Note:

1. The fractional scale is truncated to 6 digits

In the following example the ENCLOSED_BY and DATE_FORMAT are set:

```
CREATE CONNECTOR HDFS1 TYPE HADOOP
WITH hdfs='hdfs://192.168.10.15:8020', format='delimited', delimiter=',', enclosed_by='"',
DATE_FORMAT='YYYY.MM.DD'
CATALOG *
SCHEMA *
TABLE t1 USING (c1 varchar, c2 integer, c3 timestamp) WITH path='/data/sample/test/t1';;
```

INSERT Statement	Data record written to HDFS
INSERT INTO t1 values('abcdef',1,'2018-04-30 09:00:00');	'abcdef',1,'2018.04.30 09:00:00.0'
INSERT INTO t1 values('Dave''s house',2, '2018-04-30 09:00:00.123');	'Dave\'s house',2,'2018.04.30 09:00:00.123'
INSERT INTO t1 values('abc, def',3, '2018-04-30 09:00:00.12345678');	'abc,def',3,'2018.04.30 09:00:00.123456[1]'
INSERT INTO t1 values(null,4,null);	null,4,null

Note:

1. The fractional scale is truncated to 6 digits

7.10.11.11.1.2 FORMAT='ORC'

The data will be written using the ORC format, where the format for the data types will follow the Hive format as shown in the table below:

Rapids Data Type	Hive Data Type
INTEGER	BIGINT
DECIMAL	DECIMAL
BOOLEAN	BOOLEAN
DATE	DATE
TIMESTAMP	TIMESTAMP
FLOAT(24)	FLOAT
FLOAT(53)	DOUBLE
VARCHAR	STRING

7.10.11.11.1.3 FORMAT='PARQUET'

The data will be written using the Parquet format, where the format for the data types will follow the Hive format as shown in the table below:

Rapids Data Type	Hive Data Type
INTEGER	BIGINT
DECIMAL	DECIMAL
BOOLEAN	BOOLEAN
DATE	DATE
TIMESTAMP	TIMESTAMP
FLOAT(24)	FLOAT
FLOAT(53)	DOUBLE
VARCHAR	STRING

7.10.11.11.2 TRUNCATE TABLE

When doing a TRUNCATE TABLE the Hadoop Connector will delete all of the files from the directory associated with that table, along with any files in sub-directories under the parent directory (for Hive-style partitioning). The metadata information associated with the table will not be deleted.

Example 1:

```
rapids > CREATE CONNECTOR HDFS1 TYPE HADOOP WITH hdfs='hdfs://192.168.10.15:8020',
format='delimited', USER='rapids' CATALOG * SCHEMA *
     > TABLE TEST1 USING (c1 integer, c2 varchar) WITH path='/user/rapids/dave/writetests/test1';
0 row(s) returned (0.14 sec)
rapids > insert into hdfs1.public.test1 values(1, 'abc');
0 row(s) returned (0.70 sec)
rapids > select * from hdfs1.public.test1;
 C1 C2
 -- --
  1 abc

1 row(s) returned (0.11 sec)
rapids > truncate hdfs1.public.test1;
0 row(s) returned (0.13 sec)
rapids > select * from hdfs1.public.test1;
0 row(s) returned (0.06 sec)
```

Example 2:

```
rapids > CREATE CONNECTOR TEST_PAR TYPE HADOOP WITH USER='rapids',
PARTITIONS_PER_NODE='8', METASTORE='192.168.10.14:9083' NODE * CATALOG * SCHEMA TEST_PAR
TABLE *;
0 row(s) returned (0.16 sec)
rapids > select * from test_par.test1;
 c1 c2
```



```
 -- --
  1 abc

1 row(s) returned (0.15 sec)
rapids > truncate test_par.test1;
0 row(s) returned (0.28 sec)
rapids > select * from test_par.test1;
0 row(s) returned (0.12 sec)
```

7.11 Refreshing Connector Metadata Information

7.11.1 REFRESH Command

Syntax: refresh [connector <connector name>];

The RapidsDB REFRESH command will cause all of the active Connectors, or just the specified Connector, in the RapidsDB Cluster to update their schema metadata information based on the Connector definitions. This command would be used when the schema metadata on any of the data sources has been changed outside of RapidsDB.

For example, if the following Oracle Connector was active, then the following command would result in the ORA1 Connector getting the latest schema metadata information for the HR schema:

```
CREATE CONNECTOR ORA1 TYPE ORACLE WITH CONNECTIONSTRING=
'jdbc:oracle:thin:@localhost:1522:dev1', USER='rapids', PASSWORD='rdpuser' NODE BORAY05 CATALOG
* SCHEMA hr TABLE *;
```

Example:

```
rapids > refresh connector ora1;

0 row(s) returned (0.18 sec)
```

If there were multiple Connectors to be updated then the following command will update the metadata for all of the active Connectors:

```
rapids > refresh;

0 row(s) returned (1.37 sec)
```

7.12 Altering Connectors

There is no ALTER CONNECTOR command, to alter the definition for a Connector the user must drop the Connector and then recreate the Connector with the changed definition. To aid in recreating the

Connector you can get the text of the CREATE CONNECTOR command that was used to create the Connector by querying the table rapids.system.connectors (see 6.7). NOTE: this query must be executed before dropping the Connector.

7.13 Dropping Connectors

To drop an existing Connector use the following command from the rapids-shell:

```
DROP CONNECTOR <name>;
```

Any queries that were in flight at the time that the Connector was dropped and which reference the Connector will run to completion, but any new queries that reference tables managed by that Connector will fail.

Example:

```
rapids > drop connector stream1;
0 row(s) returned
```

7.14 Disabling and Enabling Connectors

7.14.1 DISABLE Connector

In the event that there is a temporary problem with a Connector, for example, the associated data source is not available, then the Connector can be disabled and that Connector will no longer participate in any requests. The format for the command is:

```
DISABLE CONNECTOR <Connector Name>;
```

The CONNECTORS table in the RapidsDB System Metadata tables (see 6.7) includes the column "IS_ENABLED" which indicates whether the Connector is enabled or disabled.

Example – the following example queries the CONNECTORS table for the RapidsSE Connector, which indicates that it is enabled, and then the RapidsSE Connector is disabled and the query to the CONNECTORS table now shows that the Connector is disabled:

```
rapids > select IS_ENABLED from connectors where connector_name='SE';
 IS_ENABLED
 ----------
     true

1 row(s) returned
rapids > disable connector se;
0 row(s) returned
rapids > select IS_ENABLED from connectors where connector_name='SE';
```



```
 IS_ENABLED
 ----------
     false

1 row(s) returned
```

7.14.2 ENABLE Connector

If a Connector has been disabled (see 6.10.1), it can be enabled again using the ENABLE CONNECTOR command:

```
ENABLE CONNECTOR <Connector name>;
```

The CONNECTORS table in the RapidsDB System Metadata tables (see 6.7) includes the column "IS_ENABLED" which indicates whether the Connector is enabled or disabled.

Example – the following example queries the CONNECTORS table for the RapidsSE Connector, which indicates that it is disabled, and then the RapidsSE Connector is enabled and the query to the CONNECTORS table now shows that the Connector is enabled:

```
rapids > disable connector se;
0 row(s) returned
rapids > select IS_ENABLED from connectors where connector_name='SE';
 IS_ENABLED
 ----------
     false

1 row(s) returned
rapids > enable connector se;
0 row(s) returned
rapids > select IS_ENABLED from connectors where connector_name='SE';
 IS_ENABLED
 ----------
      true

1 row(s) returned
```

7.15 Checking the RapidsDB Cluster Version

The version number of the RapidsDB software is stored in the file version.txt in the current directory, eg/opt/rdp/current/version.txt. Below is a sample content for the version.txt file:

```
Build information:
 version    2.4-1-g4c16a20
 from branch master
```


by rapids
on zeus.local
at Sun Jan 31 11:58:09 PST 2016

8 Managing MOXE

In order to take advantage of using MOXE to manage in-memory tables, the user must first create a MOXE Connector as described in section 7.10.4. Having created a MOXE Connector the following sections describe how to create, drop, backup and restore MOXE tables.

8.1 CREATE TABLE

The user can create MOXE tables from the rapids-shell or from JDBC using the CREATE TABLE command:

```
CREATE TABLE [IF NOT EXISTS] <tableReference>
(
 <columnDefinition>, ...
)
[ PARTITION [BY] (<expr>, ...)]

where:

<tableReference> is:
 [<connector name>.]MOXE. tableName

<column definition> is:
 <columnName> <type> [[NOT] NULL]

<type> is:
        INTEGER [(precision)] |
        DECIMAL [(scale[, precision])] |
        FLOAT |
        VARCHAR [(size)] |
        BOOELAN |
        DATE |
        TIMESTAMP
```

8.1.1 PARTITION [BY]

The partition (by) clause allows the user to specify which column(s) will be used for partitioning the table. The specified columns will be hashed to form a key that will be used to distribute the data across the MOXE partitions in the RapidsDB cluster. If the PARTITION clause is omitted the table will be created as a reference(replicated) table, in which case the table will be replicated to each node where

MOXE is running in the RapidsDB cluster and any inserts will result in each copy of the table being updated with the new rows (refer to 8.1.3 for more information on reference tables).

Examples:

The following example creates a distributed table with the column s_suppkey as the partitioning column:

```
rapids > create table MOXE.SUPPLIER (
    >   s_suppkey integer not null,
    >   s_name varchar(25) not null,
    >   s_address varchar(40) not null,
    >   s_nationkey integer not null,
    >   s_phone varchar(15) not null,
    >   s_acctbal decimal(15,2) not null,
    >   s_comment varchar(101) not null
    > ) PARTITION (s_suppkey);
0 row(s) returned (0.31 sec)
```

The following example creates a replicated table that will be replicated to every RapidsDB node in the cluster:

```
rapids > create table MOXE.REGION (
    >   r_regionkey integer not null,
    >   r_name varchar(25) not null,
    >   r_comment varchar(152)
    > );
0 row(s) returned (0.27 sec)
```

8.1.2 Data Types

8.1.2.1 INTEGER

By default the precision is 19.

8.1.2.2 DECIMAL

By default the scale is 19 and precision is zero. The maximum precision is 19.

8.1.2.3 FLOAT

MOXE uses 64-bit double precision for floats.

8.1.2.4 VARCHAR

The maximum size for a VARCHAR is 65,536 bytes. The default size is also 65,536 bytes.

8.1.2.5 TIMESTAMP

The maximum precision for the fractional component of a timestamp is 6 digits, for example:

2018-01-01 09:00:00.123456

8.1.2.6 DATE

The format for a date is YYYY-MM-DD.

8.1.3 Reference Tables

Reference tables are tables that are replicated to each node in the RapidsDB cluster. Reference tables are typically used for small dimension tables which can result in improved query performance when doing JOINs because the JOINs to the reference tables can be completed locally on each node in the RapidsDB cluster avoiding any network overhead. Refer to section 8.1 for details on how to create a reference table.

8.2 CREATE TABLE AS SELECT

Allows the user to create a table automatically from the results of a query and then insert the query results into the table. This command can be used from the rapids-shell or from JDBC.

CREATE TABLE AS SELECT is a simple way to create a copy of an existing table or to create a materialized copy of a result set. It is similar to the INSERT…SELECT statements except that the INSERT…SELECT statement appends rows to a table that already exists. As such, CTAS is a quick and easy way to take a copy of a result set and save it in a separate table.

Syntax:

```
statement := CREATE TABLE [ IF NOT EXISTS ] <tableName>

[ ( <tableDefinition> ) ]

[ <partitionInformation> ]1

[ <tableProperties> ] 2

[AS] <subquery> [ WITH [NO] DATA ];

tableDefinition := <objectDefinition> [ , <objectDefinition> [, ...] ]

objectDefinition:= <columnDefinition>

columnDefinition := <columnName> [ <columnType> [ <columnConstraint> ] ]

columnConstraint := NOT NULL

subquery := <selectOrValuesQuery> | ( <selectOrValuesQuery> )

selectOrValuesQuery:= <selectQuery> | <valuesQuery>

selectQuery := SELECT <selectQueryExpression>

valuesQuery := VALUES ( <expression> [, <expression [, ...] ] ) [, ( ... ) ]
```

Examples:

In the following the MOXE Connector name is “moxe”.

```
CREATE TABLE moxe.t SELECT * FROM db.test.t;
```

Creates a table t with columns and data from table db.test.t.

```
CREATE TABLE moxe.t AS SELECT * FROM u;
```

Creates a table t with columns and data from u, using the optional AS clause.

```
CREATE TABLE moxe.t AS SELECT a, b, c FROM db.test.t;
```

Creates a table t with columns a, b, and c from table `db.test.t`.

```
CREATE TABLE moxe.t AS VALUES (1, 'abc', true);
```

Creates a table t with automatically named columns (“col1”, “col2”, “col3”) and one row of data from the VALUES clause. The data types of the columns are determined by how the literals are expressed in the VALUES clause, and in this example will be:

- Integer
- Varchar
- Boolean

Refer to the RapidsDB User Guide for a more complete description of the CREATE TABLE AS SELECT command.

8.3 DROP TABLE

The syntax for the DROP TABLE command is:

```
DROP TABLE [IF EXISTS] [[<catalog>.]<schema>.]<table name>;
```

NOTES:

1. The catalog and schema names are only needed when the <table name> is not unique. For MOXE the catalog and schema names are the Connector name.

Examples:

```
DROP TABLE.supplier;

DROP TABLE moxe_conn.supplier;
```

In the last example the MOXE Connector name is “moxe_conn”.

8.4 TRUNCATE TABLE

The user can remove all of the data from a MOXE table using the TRUNCATE TABLE command:

```
TRUNCATE TABLE [[<catalog>.]<schema>.]<table name>;
```

NOTES:

1. The catalog and schema names are only needed when the <table name> is not unique. For MOXE the catalog and schema names are the Connector name.

Example:

```
rapids > create table moxe.t1(c1 varchar, c2 integer, c3 timestamp);
0 row(s) returned (0.25 sec)
rapids > INSERT INTO moxe.t1 values('abcdef',1, '2018-04-30 09:00:00');
0 row(s) returned (0.16 sec)
rapids > select * from moxe.t1;
C1          C2 C3
--          -- --
abcdef       1 2018-04-30 09:00:00.0

1 row(s) returned (0.47 sec)
rapids > truncate table moxe.t1;
0 row(s) returned (0.18 sec)
rapids > select * from moxe.t1;
0 row(s) returned (0.14 sec)
rapids >
```

8.5 Backing up and Restoring MOXE Tables

MOXE provides support for backing up either the entire database or individual tables using the UNLOAD command (see 8.5.1 below), which can subsequently be restored using the RELOAD command (see 8.5.2). These commands allow the user to persist the in-memory data and then shut down the RapidsDB Cluster and restart the RapidsDB Cluster and reload the MOXE data from the backups, so that the system is recovered to the point at which the last backups were taken.

 If nodes are added or removed from the RapidDB Cluster, then any MOXE “unload” files (see 8.5.1) that were created to backup the data in MOXE tables will no longer be loadable. In the need to add or remove nodes, any data to be persisted will need to be persisted to an external data source (such as HDFS) and then recreated (using INSERT … SELECT) after the cluster is reconfigured.

8.5.1 UNLOAD

8.5.1.1 UNLOAD Command

Syntax:

```
ESCAPE CONNECTOR <MOXE connector name> UNLOAD WITH name='<id>' ,path='<path>'
[,FILTER='<regex>'];
```

Where,

<id> is the user-assigned name to identify this backup

<path> is the fully qualified Linux path name to the parent directory where the backup files will be written. Each backup will be uniquely identified using the <id> specified in the command, so typically there would only need to be one parent backup directory created for all backups.

<regex> can be any Java regular expression (see https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html), and it will be used to match for fully qualified table MOXE table names, or it can be a fully qualified MOXE table name . If this option is omitted the entire database will be backed up.

Example 1: backup up table named supplier using a fully qualified name:

```
rapids > escape connector moxe unload with filter='MOXE.MOXE.SUPPLIER',
name='supplier_2019_10_31', path='/home/rapids/dave/moxe';

0 row(s) returned (1.24 sec)
```

Example 2: backup entire MOXE database:

```
rapids > escape connector moxe unload with name='db_2019_10_31', path='/home/rapids/dave/moxe';
0 row(s) returned (0.33 sec)
```

Example 3: backup using a regex, which would do a case-insensitive match for any table being with "part", which in this example would be PART and PARTSUPP

```
rapids > show tables;
CATALOG_NAME   SCHEMA_NAME   TABLE_NAME
------------   -----------   ----------
MOXE           MOXE          CUSTOMER
MOXE           MOXE          LINEITEM
MOXE           MOXE          NATION
MOXE           MOXE          ORDERS
MOXE           MOXE          PART
MOXE           MOXE          PARTSUPP
MOXE           MOXE          REGION
```



```
MOXE        MOXE        SUPPLIER

8 row(s) returned (0.08 sec)
rapids > escape connector moxe unload with filter='(?i).*\.MOXE\.part.*', name='parts_2019_10_31',
path='/home/rapids/dave/moxe';

0 row(s) returned (0.58 sec)
```

Here are the backup files on NODE1:

```
[rapids@boray01 moxe]$ ls /home/rapids/dave/moxe/moxe.NODE1/parts_2019_10_31
meta.info            MOXE.MOXE.PART$1.casks  MOXE.MOXE.PARTSUPP$0.casks
MOXE.MOXE.PARTSUPP.json  MOXE.MOXE.PART$0.casks  MOXE.MOXE.PART.json
MOXE.MOXE.PARTSUPP$1.casks
```

8.5.1.2 UNLOAD Directory Structure

The parent directory is the directory specified by the path option in the UNLOAD command. Using the examples above, the parent directory is home/rapids/dave/moxe. A directory named "moxe.<node name>" will be created on each node in the RapidsDB Cluster and this directory will include all of the backups for the data that resides on that node. The <node name> is the name of this node as specified in the RapidsDB cluster.config file (see 6.1.2). Using the example from that section we would have the directory /home/rapids/dave/moxe.NODE1 on NODE1 and the directory /home/rapids/dave/moxe.NODE2 on NODE2 etc. Finally, within the node directory would be directories named using the <id> from the UNLOAD command. Using the examples from the previous section we would have the following directories on NODE1:

- /user/rapids/dave/moxe/moxe.NODE1/db_2019_10_31
- /user/rapids/dave/moxe/moxe.NODE1/supplier_2019_10_31

These directories contain the actual backup data. For each table there would be the following files:

- <Connector>.MOXE.<table>.json contains the metadata for this table
- <Connector>.MOXE.<table>$<n>.casks for distributed tables contains the compressed data for each partition of the table, where <n> is the partition number
- <Connector>.MOXE.<table> for replicated tables contains the compressed data for the table

Example:

Assuming that the MOXE Connector name as "MOXE", then or the supplier table on NODE1 we would have the following files:

- MOXE.MOXE.SUPPLIER.json
- MOXE.MOXE.SUPPLER$0.casks

- MOXE.MOXE.SUPPLER$1.casks

8.5.2 RELOAD

Syntax:

```
ESCAPE CONNECTOR <MOXE connector name> RELOAD WITH name='<id>' ,path='<path>';
```

Where,

<id> is the user-assigned name to identify which backup is to be reloaded

<path> is the fully qualified Linux path name to the parent directory where the backup files were written.

NOTES:

1. The MOXE Connector definition cannot be changed in any way from the MOXE Connector definition that was active when the backup was taken. For example, you could not reset the MOXE database (see 8.6.1), and then create a new MOXE Connector with a different number of partitions and then use the RELOAD command to reload from backup taken with the prior Connector definition.

Example 1: reload a MOXE database

```
rapids > set catalog moxe;
rapids > show tables;
0 row(s) returned (0.15 sec)
rapids > escape connector moxe reload  with name='db_2019_10_31', path='/home/rapids/dave/moxe';
0 row(s) returned (0.21 sec)
rapids > show tables;
CATALOG_NAME   SCHEMA_NAME   TABLE_NAME
------------   ------------   ----------
MOXE        MOXE        CUSTOMER
MOXE        MOXE        LINEITEM
MOXE        MOXE        NATION
MOXE        MOXE        ORDERS
MOXE        MOXE        PART
MOXE        MOXE        PARTSUPP
MOXE        MOXE        REGION
MOXE        MOXE        SUPPLIER

8 row(s) returned (0.17 sec)
```


Example: in this example we have dropped the PART and PARTSUPP tables and then reloaded them from a backup of those tables.

```
rapids > drop table part;
0 row(s) returned (0.15 sec)
rapids > drop table partsupp;
0 row(s) returned (0.11 sec)
rapids > show tables;
CATALOG_NAME   SCHEMA_NAME   TABLE_NAME
------------   -----------   ----------
MOXE           MOXE          CUSTOMER
MOXE           MOXE          LINEITEM
MOXE           MOXE          NATION
MOXE           MOXE          ORDERS
MOXE           MOXE          REGION
MOXE           MOXE          SUPPLIER

6 row(s) returned (0.16 sec)
rapids >  escape connector moxe reload  with name='parts_2019_10_31',
path='/home/rapids/dave/moxe';
0 row(s) returned (0.17 sec)
rapids > show tables;
CATALOG_NAME   SCHEMA_NAME   TABLE_NAME
------------   -----------   ----------
MOXE           MOXE          CUSTOMER
MOXE           MOXE          LINEITEM
MOXE           MOXE          NATION
MOXE           MOXE          ORDERS
MOXE           MOXE          PART
MOXE           MOXE          PARTSUPP
MOXE           MOXE          REGION
MOXE           MOXE          SUPPLIER

8 row(s) returned (0.16 sec)
```

8.6 Changing the MOXE Configuration

In order to change the MOXE configuration dynamically any existing database must be dropped using the RESET command (see 8.6.1). Before dropping the database the user should consider whether they want to take a backup of the existing database (see 8.5). The steps to change the MOXE Configuration are:

1. If needed do an UNLOAD (backup) of the current MOXE database. Remember that this backup cannot be used to reload the data after the Connector configuration is changed, unless the configuration is put back to the same configuration used for the backup.
2. If there is sufficient memory, and the user wants to retain the data from the current MOXE database then do the following:
    a. Create a new MOXE Connector with the desired configuration, for this example we will name that Connector MOXE2 (with the existing Connector being MOXE1)
    b. Copy across all of the tables from the MOXE1 database to the newly created MOXE2 database using CREATE moxe2.<table> AS SELECT * FROM moxe1.<table>; for each table to be copied
    c. Issue a RESET command against the MOXE1 database to drop the database (see 8.6.1)
    d. Drop the MOXE1 Connector
3. If the data is not being copied then do the following:
    a. Issue a RESET command to drop the existing MOXE database (see 8.6.1)
    b. Drop the existing Connector
    c. Create the new MOXE Connector

8.6.1 Drop Database – RESET

Syntax:

```
ESCAPE CONNECTOR <MOXE connector name> RESET;
```

This command will cause the MOXE database to be dropped and all memory allocated to the MOXE database will be returned for subsequent use.

Example:

```
rapids > set catalog moxe;
rapids > show tables;
CATALOG_NAME   SCHEMA_NAME   TABLE_NAME
------------   ------------   ----------
MOXE           MOXE           CUSTOMER
MOXE           MOXE           LINEITEM
MOXE           MOXE           NATION
MOXE           MOXE           ORDERS
MOXE           MOXE           PART
MOXE           MOXE           PARTSUPP
MOXE           MOXE           REGION
MOXE           MOXE           SUPPLIER
8 row(s) returned (0.16 sec)
0 row(s) returned (0.15 sec)
rapids > escape connector moxe reset;
0 row(s) returned (0.16 sec)
```



```
rapids > show tables;
0 row(s) returned (0.15 sec)
```

9 RapidsDB System Metadata Tables

RapidsDB provides a set of system metadata tables that can be used to provide detailed information about the Federations, Connectors and the underlying tables configured in the system. The following system metadata tables can be accessed from the rapids.system schema:

Table Name	Description
Nodes	A list of all nodes in the cluster and metadata about them.
Federations	A list of all the Federations
Connectors	A list of all of the Connectors in the current Federation
Catalogs	A list of the catalogs that can be accessed from the current Federation
Schemas	A list of the schemas that can be accessed from the current Federation
Tables	Metadata for the tables or views that can be accessed from the current Federation.
Columns	A list of all the columns that can be accessed from the current Federation
Table_Providers	A list of all of the tables from each Connector
Indexes	Metadata about any indexes defined on
Authenticators	A list of all authenticator instances that have been created in the system.
Authenticator_config	Lists any additional custom properties about the authenticator instances that have been created in the system.
Users	A list of all users that exist in the system.
User_config	Any additional custom properties about users that exist in the system.
Sessions	A list of all active sessions across the cluster.
Username_maps	A list of defined mappings from an external identifier to RapidsDB usernames.
Pattern_maps	A list of defined patterns for transforming an external identifier to a RapidsDB username.
Queries	A list of the currently active queries
Query_stats	Internal statistics for all active queries

The system metadata tables are treated the same as any other tables by RapidsDB, and as for any user tables, it is only necessary to include the catalog and/or schema name when there are multiple tables in the current Federation that use the same name. Assuming that system metadata table names are all unique within the current Federation, then the following queries will all be successful:

i) select * from rapids.system.tables;
ii) select * from system.tables;
iii) select * from tables;

Refer to the RapidsDB User Guide for more information.

10 Audit Logging

10.1 Overview

RapidsDB supports the ability to dynamically record what a user, or set of users, are doing within the RapidsDB cluster so that the actions can be audited for security purposes at a later stage. The audit logging can be configured to:

1. Log all commands from all users
2. Log all commands for one or more specific users
3. Dynamically turn logging on and off

The commands for controlling audit logging are described in the following sections.

All logging information will be written to a file named audit.log in the "current" directory (e.g. /opt/rdp/current) on the RapidsDB node where the command was submitted. If needed, all audit events can be written to a single, centralized file using the settings in the log4j2.dqx.xml file in the "cfg" directory of the "current" directory (e.g. /opt/rdp/current/cfg) (see

10.2 Audit Logging Commands

10.2.1 SET AUDIT ENABLED

This command can be used to dynamically enable auditing. This command can only be executed by the rapids user.

Syntax:

```
SET AUDIT ENABLED ;
```

Example:

```
rapids > set audit enabled;
0 row(s) returned (0.01 sec)
```

NOTE: When enabling audit the following rule applies:

1. If no users are currently being explicitly audited, using the ADD AUDIT command (see 10.2.3), then the system will audit all users.

10.2.2 SET AUDIT DISABLED

This command can be used to dynamically disable auditing. This command can only be executed by the rapids user.

Syntax:

```
SET AUDIT DISABLED ;
```


Example:

```
rapids > set audit enabled;
0 row(s) returned (0.01 sec)
```

10.2.3 ADD AUDIT ON USER

This command can be used to specify that a specific user is to be audited. This command can only be executed by the rapids user.

Syntax:

ADD AUDIT ON USER <user> ;

Example:

```
rapids > add audit on user dave;
0 row(s) returned (0.51 sec)
```

NOTE:

1. The act of adding one or more users to be audited will disable the auditing for all other users who have not been explicitly added to the list of users to be audited. In order to audit all users, simply remove all the auditing on any specific users using the REMOVE AUDIT command (see 10.2.4)

10.2.4 REMOVE AUDIT ON USER

This command can be used to specify that a specific user is to be audited. This command can only be executed by the rapids user.

Syntax:

REMOVE AUDIT ON USER <user> ;

Example:

```
rapids > remove audit on user dave;
0 row(s) returned (0.51 sec)
```

NOTE:

1. If the removal of the auditing for this user results in no specific users being audited then the system will start auditing all users. Disable auditing if you want to stop all auditing (see 10.2.2)

10.3 Audit Log File Format

Below is an example of entries in the audit.log file which reflect what was logged for the following commands:

```
rapids > create user dave password 'dave123';
0 row(s) returned (0.31 sec)
rapids > add audit on user rapids;
0 row(s) returned (0.47 sec)
rapids > add audit on user dave;
0 row(s) returned (0.51 sec)
rapids > select * from tables;
…
rapids > CREATE CONNECTOR PGP6 TYPE POSTGRES WITH PASSWORD='postgres',
URL='jdbc:postgresql://localhost/tpch', USER='postgres' NODE BORAY01 ;
0 row(s) returned (0.64 sec)
```

audit.log:

```
[INFO ] 2020-08-05 13:44:15 audit-log - RAPIDS,BORAY01,SSN_1@BORAY01,"CREATE USER DAVE "
[INFO ] 2020-08-05 13:45:38 audit-log - RAPIDS,BORAY01,SSN_1@BORAY01,"ADD AUDIT ON USER
DAVE;"
[INFO ] 2020-08-05 14:23:37 audit-log - RAPIDS,BORAY01,SSN_1@BORAY01,"select * from tables;"
[INFO ] 2020-08-05 14:25:23 audit-log - RAPIDS,BORAY01,SSN_1@BORAY01,"CREATE CONNECTOR PGP6
TYPE POSTGRES WITH PASSWORD='XXXX', URL='jdbc:postgresql://localhost/tpch', USER='XXXX' NODE
BORAY01;"
```

Note that the password for the user is not logged and that the user and password were X-ed out in the log for the create connector command.

10.4 Configuring the Audit Log

The audit logging is controlled using log4j2, and the configuration is specified in the log4j2.dqx.xml file in the cfg directory. By default the audit log will be written to a file named audit.log in the RapidsDB installation directory (e.g. /opt/rdp) on the node where the command originated.

Below is a copy of the file with the default settings for audit logging hilited:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Copyright 2018 Boray Data Co. Ltd.  All rights reserved. -->
```



```
<!-- "status" here refers to the log4j internal logging level.
    Refer to the property "rootLoggingLevel" to set the RapidsDB log level. -->
<Configuration status="warn" monitorInterval="10">

  <Properties>
    <!-- The main logging level for RapidsDB. -->
    <Property name="rootLoggingLevel">info</Property>

    <Property name="filename">./dqx.log</Property>
    <Property name="auditname">./audit.log</Property>

    <Property name="rollingFilePattern">./dqx-%i.log</Property>

    <!--
       The pattern that log messages are written out in.
       Ref: https://logging.apache.org/log4j/2.x/manual/layouts.html#PatternLayout
       %d{ISO8601} = yyyy-mm-ddThh:mm:ss,mmm
       [%-22c{2}]  = left justify with 22 character width and print the class name ignoring the
                     first two components of the package name. i.e. the "com.rapidsdata" part.
       %-5p:       = Left justify with width 5 and print the priority (WARN, INFO, DEBUG, etc).
       %m%n        = print the message associated with the log event, followed by the line separator.
    -->
    <Property name="patternFormat">%d{ISO8601} [%-22c{2}] %-5p: %m%n</Property>

  </Properties>

  <Appenders>
    <!-- Where to log to. Must be referenced by a logger below. -->
    <RollingFile
          name="DqxFile"
          fileName="${filename}"
          filePattern="${rollingFilePattern}"
          append="true">
       <PatternLayout pattern="${patternFormat}"/>
       <Policies>
          <SizeBasedTriggeringPolicy size="20MB"/>
       </Policies>
       <DefaultRolloverStrategy max="10"/>
    </RollingFile>
    <RollingFile
          name="AuditFile" fileName="${auditname}"
          filePattern="${log-path}/analytics-%d{yyyy-MM-dd}.log">
```



```
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss} %c{1} - %msg%n</pattern>
      </PatternLayout>
      <Policies>
         <SizeBasedTriggeringPolicy size="10MB"/>
      </Policies>
      <DefaultRolloverStrategy max="10"/>
      </RollingFile>

  </Appenders>

  <Loggers>
     <!--
        Define loggers for packages that we wish to use a different logging level for.
        This is typically because these packages are noisy and we want to suppress
        their annoying log messages unless they are really important.
        Don't change audit-log name!
     -->
     <Logger name="org.apache.zookeeper" level="error" additivity="false" />
     <Logger name="org.I0Itec.zkClient"  level="warn"  additivity="false" />
     <Logger name="org.apache.curator"   level="warn"  additivity="false" />
     <Logger name="com.jcraft.jsch"      level="warn"  additivity="false" />
     <Logger name="org.apache.kafka"     level="warn"  additivity="false" />
     <Logger name="audit-log" level="info" additivity="false">
        <AppenderRef ref="AuditFile" level="info"/>
     </Logger>

     <!-- The root logger, the main logging level and which appenders to log to. -->
     <Root level="${rootLoggingLevel}">
        <AppenderRef ref="DqxFile"/>
     </Root>
  </Loggers>
```

The settings and logging package being used for auditing can be changed as needed. Note that if the configuration is changed it will need to be changed for each node in the cluster.

By default the audit log will be written to a file named audit.log in the RapidsDB installation directory (e.g. /opt/rdp) on the node where the command originated. The audit log file location is controlled by the line:

```
<Property name="auditname">./audit.log</Property>
```


This can be changed to reference any file, including a mapped/shared file which would enable all of the logging to be written to a single, shared file. It is also possible to use different Appenders to have the audit log written elsewhere.

11 Troubleshooting

11.1 RapidsDB Logs

Each RapidsDB node maintains its own log file named dqx.log in the directory /opt/rdp/current. In the event that there is a failure in the RapidsDB cluster check this log file for details of the cause of the error. In the event that a query does not return for an extended period of time, first check the dqx.log file for the DQC, and if this does not indicate any problems, then check the dqx.log file for each of the DQE nodes.

Appendix A: Configuring Ports Under CentOS or RHEL

1. su root <enter>.
2. yum install system-config-firewall-tui <enter>
3. Launch the gui: system-config-firewall-tui <enter>.
4. You want to enable the firewall. Do so by hitting the spacebar key.
5. Using the arrow keys, navigate to the Customize option. Hit the enter key.
6. Navigate to the Forward option. Hit the enter key.
7. Navigate to the add option. Hit the enter key.
8. The next screen is where you enter the ports from the list. The protocol is always tcp.
9. After each entry, navigate to OK and hit enter.
10. You will see the entry you just made appear in the list on the previous screen.
11. Repeat items 8 – 11 for each port you need.
12. When you have finished, navigate to close and hit enter.

Appendix B: bootstrapper.sh

The bootstrapper.sh script is used to manage the RapidsDB cluster. The following table lists the options supported by the bootstrapper:

Option	Description
-a (--action) [SAMPLE | POPULATE | BACKUP | INSTALL | START | STOP | HEALTHCHECK | REMOVE_LOGS | COPY_LOGS | REMOVE_FED | COUNT_CPUS | CHECK_LOCAL_LICENSE | CHECK_LICENSE | COPY_LICENSE]	Action to be performed: SAMPLE generates a sample cluster.config file. POPULATE populates ZooKeeper from the cluster.config file. Automatically run as part of POPULATE. BACKUP backup the ZooKeeper configuration to a file. INSTALL install software from the installation file to all nodes specified in the ZooKeeper configuration. START starts all nodes from the ZooKeeper configuration. STOP stops all nodes, as determined from the ZooKeeper configuration. HEALTHCHECK tests to see if a connection can be established to all nodes. REMOVE_LOGS deletes all log files matching "*.log*" from all nodes. COPY_LOGS copies all log files matching "*.log*" from all nodes and writes them to a ./logs subdirectory. REMOVE_FED removes a federation definition from Zookeeper. COUNT_CPUS returns a count of the number of cpus on each node in the cluster. CHECK_LOCAL_LICENSE checks the status of the license for the current node. CHECK_LICENSE checks the status of the license on each node in the cluster. COPY_LICENSE copies the license file to each node in the cluster.
--jvm_settings VAL	When used with the INSTALL action, this writes the JVM settings specified by the string VAL into the startup script for each DQX node so that they become default. When used with the START action then all DQX nodes are started with these settings (in addition to any defaults) temporarily.

	Example settings: allows the user to specify the timeout value, in milliseconds, for the refresh command.
-c FILE	Path to the JSON cluster configuration file used to populate ZooKeeper when the INSTALL or POPULATE action is used, or the file to write out the configuration data when the BACKUP action is used (default value is cfg/cluster.config).
-h N	The size of the java heap (in gigabytes) to be used. The default value is 16GB.
-i FILE	Path to the installation file to use when coupled with the INSTALL action.
-o FILE	Path of the output file to write to. Used for the SAMPLE and BACKUP actions.
-q N	Maximum number of concurrent client queries. Set to 0 for unlimited concurrent client queries. The default value is 10. The recommended setting is 3.
-z FILE	Path to the config file identifying which ZooKeeper server to use. (default value is "cfg/zk.config").

Bootstrapper messages

The following lists the messages reported from the bootstrapper:

- **Invalid action type**
  The action specified using the -a option was not one of POPULATE, BACKUP, INSTALL, START, STOP or HEALTHCHECK.

- **The -z option must be used to specify the ZooKeeper config file**
  When the POPULATE action is requested, a ZooKeeper configuration file must be specified using the -z option.

- **The ZooKeeper config file <file> does not exist**
  The ZooKeeper configuration file specified using the -z option could not be found.

- **The --jvm_settings option is only applicable when the INSTALL or START actions are specified**

- **The -h heap size option is only applicable when the INSTALL or START actions are specified**

- **The -h heap size option cannot be negative**

- **The -q query maximum option is only applicable when the INSTALL or START actions are specified**

- **The -q client query maximum queue size must be positive or zero**

- **The -c option must be used to specify the config file path**
  The requested action requires a Rapids DB cluster configuration file.

- **The config file <file> does not exist**

- **The -o option must be used to specify the output file**
  When the BACKUP or SAMPLE actions are requested, the path to the output file must be specified.

- **The -i option must be used to specify the installation file**
  When executing the INSTALL action, the location of the installation file must be specified using the -i option.

- **The installation file <file> does not exist**
  The installation file specified using the -i option could not be found.

- **<n> out of <n> nodes are not responding**
  When the HEALTHCHECK action is requested, this message indicates that one or more nodes in the cluster did not respond as expected.

Configuration and startup messages.

Note: these messages can be issued by any of the DQS component programs.

The ZooKeeper config file <file> does not exist

The file specified using the -z option could not be found.

ZooKeeper hostname must be specified

The ZooKeeper hostname must be specified in the ZooKeeper configuration file.

ZooKeeper port value is out of bounds

The ZooKeeper port number must be in the range 0 - 65535.

ZooKeeper root directory must be specified

The ZooKeeper configuration must specified a root location ("root Z-Node") within the ZooKeeper database where the RapidsDB configuration information is to be stored.

Error connecting to ZooKeeper server at <hostname:port>

RapidsDB could not connect to the ZooKeeper server. Check that the ZooKeeper configuration is correct and that the ZooKeeper server is started.

Could not connect to ZooKeeper server at <hostname:port>

RapidsDB could not connect to the ZooKeeper server. Check that the ZooKeeper configuration is correct and that the ZooKeeper server is started.

ZooKeeper configuration is empty

The RapidsDB information stored in ZooKeeper is missing. Initialize the information using the POPULATE action in the bootstrapper.

Error trying to create the directory <name>

An error occurred when RapidsDB tried to create information in the ZooKeeper database. Check the ZooKeeper configuration and permissions.

The JSON cluster config file <file> does not exist

RapidsDB was unable to locate the cluster configuration file.

Could not deserialize json configuration

The cluster configuration file is improperly formatted or the RapidsDB information stored in ZooKeeper is improperly formatted. Check that the configuration file is correctly formatted and run the POPULATE action in the bootstrapper if necessary.

Could not connect to the DQX node: <hostname:port>

RapidsDB was unable to connect to the named node. Check that the cluster configuration is correct and that the RapidsDB software is installed and running on all nodes.

Error trying to ping the DQX node: <hostname:port>

RapidsDB was unable to communicate with the named node. Check that the cluster configuration is correct, that the RapidsDB software is installed and running on all nodes and that all nodes are reachable on the network.

All nodes in the cluster must have the same version

RapidsDB has detected that some nodes have a different version of the RapidsDB software. Reinstall the desired version on all nodes.

SSH command failed for node: <hostname:port>

RapidsDB was unable to run a command on a remote node using the SSH protocol. Check the SSH configuration and the file and folder permissions for RapidsDB-related files and folders on the named host.

Unable to read the file <file> on node: <hostname:port>

RapidsDB was unable to access a file on a remote node using the SSH protocol. Check the SSH configuration and the file and folder permissions for RapidsDB-related files and folders on the named host.

Error writing the file <file> on node: <hostname:port>

RapidsDB was unable to write to a file on a remote node using the SSH protocol. Check the SSH configuration and the file and folder permissions for RapidsDB-related files and folders on the named host.

Could not get the software version for node: <hostname:port>

RapidsDB could not retrieve the RapidsDB software version from the named host. Check that the RapidsDB software is properly installed on the named host and that the host is reachable on the network.

Could not parse the version information from file <file>

RapidsDB could not retrieve the RapidsDB software version from the named host. Check that the RapidsDB software is properly installed on the named host. Reinstall the RapidsDB software if necessary.

Error occurred creating SSH session

RapidsDB encountered an error attempting to communicate using the SSH protocol. Check the SSH configuration and keys. Check that that the SSH server is running on all nodes in the cluster.

Error occurred connecting or authenticating SSH session

RapidsDB encountered an error attempting to communicate using the SSH protocol. Check the SSH configuration and keys.

Check that that the SSH server is running on all nodes in the cluster.

SSH command execution failed

RapidsDB encountered an error attempting to execute a command on a remote node. Check the SSH configuration and keys.

Check that that the SSH server is running on all nodes in the cluster.

Error sending file <file> to remote host: <hostname>

RapidsDB was unable to send a file to a remote node using the SSH protocol. Check the SSH configuration and the file and folder permissions for RapidsDB-related files and folders on the named host. Check that the SSH server is running on the named host.

Error copying file <file> from remote host: <hostname>

RapidsDB was unable to retrieve a file from a remote node using the SSH protocol. Check the SSH configuration and the file and folder permissions for RapidsDB-related files and folders on the named host. Check that the SSH server is running on the named host.

Error setting permissions on remote file <hostname> : <file>

RapidsDB was unable to set permissions on a file from a remote node using the SSH protocol. Check the SSH configuration and the file and folder permissions for RapidsDB-related files and folders on the named host. Check that the SSH server is running on the named host.

Appendix C: Kerberos Setup

If Kerberos is to be used for authenticating to RapidsDB then both the clients and the RapidsDB nodes need to be setup correctly.

Basic Kerberos Server Setup

This section describes how to install and setup a basic Kerberos Key Distribution Centre (KDC) server. This setup was performed on CentOS 7 and will be different for different operating systems.

1. Install NTP to ensure that the clocks on the Kerberos KDC server, client, and RapidsDB nodes are all synced.
   a. `sudo yum install ntp`
   b. `sudo systemctl enable ntpd`
   c. `sudo systemctl start ntpd`
   d. `ntpstat` or `ntpq -p` (verifies that it is running correctly)

2. Install the Kerberos binaries on the server:
   `sudo yum install krb5-server krb5-workstation krb5-libs`

3. Ensure that the Kerberos ports are opened up in the firewall on the machine running Kerberos:
   a. `sudo firewall-cmd --zone=public --add-service=kerberos --add-service=kpasswd --add-service=kadmin`
   b. `sudo firewall-cmd --zone=public --add-service=kerberos --add-service=kpasswd --add-service=kadmin --permanent`

4. As root, edit the file: `/etc/krb5.conf`
   a. Edit the `[realms]` section and copy the example realm definition below, changing the example values as appropriate. In this example, the name of the realm being defined is `EXAMPLE` and the KDC is running on a machine with a fully qualified hostname of `myKdcHost.myDomain`:

```
[realms]
EXAMPLE = {
  kdc = myKdcHost.myDomain
  admin_server = myKdcHost.myDomain
  kpasswd_server = myKdcHost.myDomain
}
```

   b. Edit the `[domain_realm]` section. This maps domains and hostnames to Kerberos realms. In this example any machines in the `.myDomainName` domain will be mapped to the `EXAMPLE` realm. You can enter hostnames here, or domain name wildcards which will start with a dot.

```
[domain_realm]
```



```
.myDomainName = EXAMPLE
myKdcHost = EXAMPLE
```

c. Edit the `[libdefaults]` section and uncomment the `default_realm` key. Set the value for this key to the name of the realm. Also comment out the line that sets the key `default_ccache_name`.

5. As root, edit the file: `/var/kerberos/krb5kdc/kdc.conf`
   a. Change the realm from `EXAMPLE.COM` to match the realm defined in `/etc/krb5.conf`.

6. Create the Kerberos database by executing:
```
sudo kdb5_util create -s
```

When you do this it will ask you to enter a master password for the database. **Do not forget this password.**

7. Edit the Kerberos ACL file `/var/kerberos/krb5kdc/kadm5.acl`. It should have a line that looks like this:
```
*/admin@EXAMPLE.COM      *
```

Change the name of the realm from EXAMPLE.COM to whatever realm name you've given. e.g.:
```
*/admin@HOME      *
```

This example gives all privileges to the Kerberos database (with the exception of the privilege to extract principal keys) to every principal that has an instance of.

8. Now we need to create some principals in the system. Let's start by adding an admin user with the kadmin.local command. Replace <username> with any desired admin username:

```
sudo kadmin.local -q "addprinc <username>/admin"
```

Ignore the warning that it returns about there being no policy specified for this principal. It is a warning about there being no password-aging and strength-checking policy.

9. Now start Kerberos and enable it to run at boot up:
```
sudo systemctl start krb5kdc
sudo systemctl enable krb5kdc
sudo systemctl start kadmin
sudo systemctl enable kadmin
```

Running `sudo systemctl status krb5kdc` will show that Kerberos is running.

Creating Kerberos Principals

1. To create additional Kerberos principals for clients, use kadmin (not kadmin.local) from any computer that has been setup for Kerberos. Authenticate as an admin user in order to have

permission to create additional principals

```
kadmin -p <admin_username>/admin
addprinc <new_principal_name>
```

You will then be asked to enter the password for the new client principal.

2. To check that the new client principal has been created successfully, exit out of the kadmin tool and try to initialize the Kerberos ticket cache as that new principal:

```
kinit <new_principal_name>
```

If the authentication was successful then the klist command will show a Ticket Granting Ticket in the local ticket cache, as per the example below:

```
[craig@copernicus ~]$ kinit
Password for craig@HOME:
[craig@copernicus ~]$ klist
Ticket cache: KEYRING:persistent:1000:1000
Default principal: craig@HOME

Valid starting     Expires            Service principal
17/12/18 17:17:36  18/12/18 17:17:36  krbtgt/HOME@HOME
[craig@copernicus ~]$ kdestroy
[craig@copernicus ~]$ klist
klist: Credentials cache keyring 'persistent:1000:1000' not found
```

3. To export the client's encryption keys to a file (e.g. for use in scripting a client application's connection to RapidsDB) simply login to kadmin as an admin user and execute the command:

```
ktadd -norandkey -k <path/to/output/file> <principal>
```

Kerberos Client Setup

This setup applies to all nodes wishing to authenticate to RapidsDB using Kerberos, either via the rapids-shell or directly over the wireline protocol.

1. On each linux client that will be accessing Kerberos, install the workstation libraries:
```
sudo yum install krb5-workstation krb5-libs
```

2. Ensure that the clocks on the client and Kerberos servers are all synced via NTP.
```
sudo yum install ntp
sudo systemctl enable ntpd
sudo systemctl start ntpd
ntpstat or ntpq -p (verifies that it is running correctly)
```


3. From the Kerberos server, copy the file `/etc/krb5.conf` file to the local client system.

4. Now let's verify that Kerberos is issuing tickets. Run `kinit` to obtain a ticket and store it in your ticket cache. Then use `klist` to view the list of tickets in your cache and `kdestroy` to destroy the ticket cache and all cached credentials:

```
[craig@hubble ~]$ kinit
Password for craig@HOME:
[craig@hubble ~]$ klist
Ticket cache: KEYRING:persistent:1000:1000
Default principal: craig@HOME

Valid starting     Expires            Service principal
17/12/18 17:17:36  18/12/18 17:17:36  krbtgt/HOME@HOME
[craig@ hubble ~]$ kdestroy
[craig@ hubble ~]$ klist
klist: Credentials cache keyring 'persistent:1000:1000' not found
```

RapidsDB Node Kerberos Setup

When Kerberos is being used for authenticating clients to RapidsDB, each RapidsDB node should have a separate service principal. This is because Kerberos service principals include the fully qualified hostname of the machine that they run on and Kerberos clients can be configured to check that the name of the service principal matches the hostname of the machine.

Unlike client principals which may be authenticated by using kinit and interactively entering a password, RapidsDB nodes need a keytab file that contains the encryption keys for their service principal. To do this:

1. Login to kadmin as an admin user:
   `kadmin <admin_username>/admin`

2. Create the service principal for each RapidsDB node and give it a random key:
   `add_principal -randkey <rdp_principal>`

3. Export the encryption keys for this principal to a file:
   `ktadd -norandkey -k <path/to/output/file> <rdp_principal>`

4. Exit `kadmin` and copy the keytab file to the RapidsDB node and rename it to the filename `rdp.keytab`. Place it in the parent of the RapidsDB working directory. E.g.

   ```
   scp rdp.node1.keytab node1:/opt/rdp/rdp.keytab
   ```


Appendix D: Setting up passwordless ssh

The following are a set of recommendations for configuring ssh for use with RapidsDB. Repeat the following for each node in the cluster:

1. Log in as a user with root privileges
2. Add the following lines to the beginning of the SSH configuration file, /etc/ssh/ssh_config, to bypass host key checking for the subnet that the cluster is running on (192.168.1 in this example):
   ```
   Host 192.168.1.*
         StrictHostKeyChecking no
         UserKnownHostsFile=/dev/null
   ```
3. The known_hosts file for the rapids user needs to exist, and to ensure that this file exists, login as the rapids user and enter the following command:
   ```
   touch /home/rapids/.ssh/known_hosts
   ```

Once the basic setup has been completed on all of the machines, the next step is to configure ssh access across all of the nodes in the cluster. The objective is that you should be able to SSH from any account where the bootstrapper executes to all of the nodes in the cluster definition using a password-less public/private key authentication only. Additionally, each cluster node should be able to SSH to itself using a public/private key without requiring a password. This will allow the bootstrapper to start and stop nodes across the cluster without needing to store or prompt for passwords.

Background:

Consider NodeA trying to establish an SSH session with NodeB using public/private key authentication. When NodeA is told to establish the SSH session to NodeB, it uses the following basic information:

- The hostname or IP address of NodeB;
- The username on NodeB to authenticate as;
- The path to the private key on NodeA to use during the authentication.

NodeA connects to NodeB, establishes a secure connection, and then tells NodeB that it wishes to authenticate as the given username using a public key authentication. NodeA gives NodeB an ID of the public/private key pair it holds and NodeB uses this ID to look up the corresponding public key in its authorized_keys file (located at ~/.ssh/authorized_keys on NodeB).

NodeB then generates a random number, encypts it using the public key found in its authorized_keys file and sends it to NodeA. NodeA then decrypts the number using its private key and sends proof to NodeB that it was able to decrypt it properly. If NodeB verifies this proof then NodeA has successfully authenticated as the given user on NodeB.

In this public key authentication scheme, public keys can only be used for encrypting information and private keys can only be used for decrypting information.

Approaches:

There are two approaches that can be taken to allow a user on a node to SSH to all the other nodes in the cluster using a password-less key-pair exchange:

1. Each node where the bootstrapper can run (including all cluster nodes) has their own unique private and public keys. The public key from each of these nodes is put into the ~/.ssh/authorized_keys file for every node in the cluster.

   This approach is not scalable because the number of public keys that need to be shared is at least the square of the number of nodes in the system (i.e. it is $O(n^2)$). For a 2 node system, NodeA will need to share its public key with itself and with NodeB. NodeB will need to share its public key with itself and NodeA.

2. Each node where the bootstrapper can run (including all cluster nodes) has a common shared private key and all cluster nodes have the corresponding common public key in their ~/.ssh/authorized_keys file.

   This approach is less secure because it grants access to anyone who happens to have the private key, but it also makes it far easier to setup when the size of the cluster is large. Each cluster node simply needs to have the same common public key in its ~/.ssh/authorized_keys file, and any node that can execute the bootstrapper (including all cluster nodes) simply needs to have the common private key saved locally too.

Because the second approach is more scalable with respect to the size of the cluster, we shall describe how to set it up here. However, a system administrator who is familiar with SSH is free to configure the system with unique private and public keys per node, as per approach #1. The key constraint that needs to be acknowledged with this approach is this:

> The **path** to the private key used to authenticate with a specific cluster node is a constant regardless of where you are connecting from.

This is because the cluster.config file contains an entry called “pathToSshIdentityFile” which can be configured independently for each node in the cluster. The interpretation of this is that when NodeA is connecting via SSH to NodeB it looks up in the cluster.config file the value of “pathToSshIdentityFile” **for the target node, NodeB**. If this key returned the value “`~/.ssh/nodeBPrivate.key`” then NodeA would look up that key in its local filesystem and attempt to use it to authenticate with NodeB.

So if each connecting node has a private key to allow it to connect and authenticate with a given target node then that private key must exist under a common name and path on each connecting node.

Setup:

This section assumes that each cluster node will share the same public and private key pair.

On the first node in the cluster:

1. Login as the user rapids (or the desired username for running RapidsDB).
2. If an SSH key pair does not exist under ~/.ssh (e.g., ~/.ssh/id_rsa and ~/.ssh/id_rsa.pub) then generate one using the following commands:
    a. Run the command: ssh-keygen
    b. Decide what you want to name the keys. This will need to be entered into the cluster.config file if you choose not to go with the default names of id_rsa (private key) and id_rsa.pub (public key).
    c. Do not enter a password for the private key.
    d. Accept any other default values.

On each node of the cluster (including the first node):

1. Login as the user rapids (or the desired username for running RapidsDB).
2. Copy both the private and public keys generated on the first node to the directory ~/.ssh on the current node (ignore this step if you are on the first node).
3. Ensure the copied keys have a permissions mask of 600 (i.e., rw- --- ---).
4. Append the newly copied public key into the authorized_keys file on the current node. E.g. cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
5. Ensure the authorized_keys file has a permission mask of 600 (i.e., rw- --- ---).
6. Ensure a ~/.ssh/known_hosts file exists by running: touch ~/.ssh/known_hosts
7. Test that you can ssh to the same node without needing to enter a password by using the copied private key:
   ssh –i *<path_to_private_key>* localhost
   …replacing *<path_to_private_key>* with the path to the private key copied from the first node. The *<path_to_private_key>* value used here should be the same value that appears in the cluster.config for this current node.
8. Test that you can ssh to a previous node that you have setup with the SSH public/private keys:
   ssh –i *<path_to_private_key>* *<username>*@*<hostname>*
   …replacing *<path_to_private_key>* with the path to the private key copied from the first node and set *<username>* and *<hostname>* appropriately. The *<path_to_private_key>* value used here should be the same value that appears in the cluster.config for the node you are connecting to.

When all nodes are done, you should be able to SSH from wherever you intend to run the bootstrapper to any of the cluster nodes by looking up the value of “sshUsername” and “sshPathToIdentityFile” in the cluster.config for the node you are connecting to and using those as the values for *<username>* and *<path_to_private_key>* in the command:
ssh –i *<path_to_private_key>* *<username>*@*<hostname>*

