RapidsDB SQL Syntax Guide

Release 4.3.3

Table of Contents

1 SQL Syntax

A SQL statement is a series of words describing a database operation. This section describes the SQL syntax supported by RapidsDB.

1.1 Lexical Structure

A SQL command is composed of a sequence of tokens, terminated by a semicolon (";"). Which tokens are valid depends on the syntax of the particular command.

A token can be a keyword, an identifier, a quoted identifier, a literal (or constant), or a special character symbol. Additionally, comments can occur in SQL input. Comments are not tokens, they are effectively equivalent to whitespace.

Here is an example of syntactically valid SQL command:

SELECT * FROM CUSTOMER WHERE CUSTOMER_NAME = 'Smith';

Refer to Appendix B for details of the SQL grammar supported by RapidsDB.

1.1.1 Identifiers and Keywords

Tokens such as SELECT or WHERE are examples of keywords, that is, words that have a fixed meaning in the SQL language. In the example query above, the tokens CUSTOMER and CUSTOMER_NAME are examples of identifiers. They identify names of tables, columns, or other database objects, depending on the command they are used in. Therefore they are sometimes simply called "names". Keywords and identifiers have the same lexical structure, meaning that one cannot know whether a token is an identifier or a keyword without knowing the language.

SQL identifiers and keywords must begin with a letter (a-z, but also letters with diacritical marks and non-Latin letters) or an underscore (_). Subsequent characters in an identifier or keyword can be letters, underscores, or digits (0-9).

RapidsDB supports identifiers up to a maximum length of 32,000 characters, but underlying data systems may reject very long identifiers in CREATE TABLE statements.

By default, SQL keywords and identifiers are converted internally to uppercase. Therefore:

SELECT * FROM CUSTOMER WHERE CUSTOMER_NAME = 'Smith';

can equivalently be written as:

Select * FroM customer Where Customer_name = 'Smith';

A convention often used is to write keywords in upper case and names in lower case, e.g.:

SELECT * FROM customer WHERE customer_name = ‘Smith’;

There is a second kind of identifier: the delimited identifier or quoted identifier. It is formed by enclosing an arbitrary sequence of characters in back-ticks (`) or double-quotes ("). A delimited identifier is always recognized as an identifier, never a keyword. So "select" could be used to refer to a column or table named "select", whereas an unquoted select would be taken as a keyword and would therefore provoke a parse error when used where a table or column name is expected. The example can be written with quoted identifiers like this:

SELECT * FROM “CUSTOMER” WHERE “CUSTOMER_NAME” = ‘Smith’;

Quoted identifiers can contain any character, except the character with code zero. (To include a double quote, write two double quotes.) This allows constructing table or column names that would otherwise not be possible, such as ones containing spaces or ampersands. Quoted identifiers are case sensitive.

1.1.2 Constants

There are three kinds of implicitly-typed constants: strings, booleans and numbers. Constants can also be specified with explicit types. These alternatives are discussed in the following subsections.

1.1.2.1 String Constants

A string constant is an arbitrary sequence of characters bounded by single quotes ('), for example 'This is a string'. To include a single-quote character within a string constant, write two adjacent single quotes, e.g., 'Dianne''s horse'. Note that this is not the same as a double-quote character (").

1.1.2.2 Boolean Constants

A boolean constant can either be the 4 character string true, with no enclosing quotes, or the 5 character string false, with no enclosing quotes.

e.g.

```
rapids > select true=true;
  [1]
  ---
 true
```

1.1.2.3 Numeric Constants

Numeric constants are accepted in these general forms:

```
digits
digits.[digits][e[+-]digits]
[digits].digits[e[+-]digits]
digitse[+-]digits
```

where digits is one or more decimal digits (0 through 9). At least one digit must be before or after the decimal point, if one is used. At least one digit must follow the exponent marker (e), if one is present.

There cannot be any spaces or other characters embedded in the constant. Note that any leading plus or minus sign is not actually considered part of the constant; it is an operator applied to the constant.

These are some examples of valid numeric constants:

42
3.5
4.
.001
5e2
1.925e-3

A numeric constant that contains neither a decimal point nor an exponent has the data type INTEGER. A numeric constant containing a decimal point but no exponent has the data type DECIMAL. A numeric constant containing an exponent has the data type FLOAT. For more information on data types, see 1.2 below.

1.1.3 Operators

The following operators are supported:

'+'
'-'
'*'
'/'
'%'
'||'
'<'
'<='
'='
'!='
'<>'
'>='
'>'

1.1.4 Special Characters

Some characters that are not alphanumeric have a special meaning that is different from being an operator. Details on the usage can be found at the location where the respective syntax element is described. This section only exists to advise the existence and summarize the purposes of these characters.

Parentheses (()) have their usual meaning to group expressions and enforce precedence. In some cases parentheses are required as part of the fixed syntax of a particular SQL command.

Commas (,) are used in some syntactical constructs to separate the elements of a list.

The semicolon (;) terminates an SQL command. It cannot appear anywhere within a command, except within a string constant or delimited identifier.

The asterisk (*) is used in some contexts to denote all the columns of a table.

The period (.) is used in numeric constants, and to separate schema, table, and column names.

1.1.5 Comments

C-style block comments can be used where the comment begins with /* and extends to the matching occurrence of */. These block comments cannot nest.

A comment is removed from the input stream before further syntax analysis and is effectively replaced by whitespace.

Example:

SELECT /*Customer query */ * FROM customer /* Source table */ WHERE customer_name = 'Smith';

1.1.6 Operator Precedence

The table below shows the operator precedence rules:

Operator	Associativity	Description
.	Left	Table/column name separator
+ -	Right	Unary plus, unary minus
* / %	Left	Multiplication, division, modulo
+ -	Left	Addition, subtraction
BETWEEN IN LIKE		
< > = <= >= <>		Comparison operators
IS NULL, IS NOT NULL		
NOT	Right	
AND	Left	
OR	left	

1.2 Data Types and Type Specifiers

1.2.1 Data Types

Because the RapidsDB execution engine is implemented in the Java language and uses the Java Virtual Machine (JVM) for runtime support, RapidsDB data types are implemented internally as Java classes. Every data value handled by the system is an "instance" of a Java class. The Java class implements the behaviors and functionality for values of that class.

For simplicity and to facilitate type harmonization across different data systems, RapidsDB organizes these underlying Java classes into abstract "SQL types" that are similar in concept to the data types used in most SQL-based data management systems. A RapidsDB user will generally specify data types in terms of the SQL types, leaving the choice of Java class to the Connector and the execution engine. Connectors translate the user's requested SQL types to equivalent types in the associated DBMS or data store. When presenting data to the execution engine, the Connector will select a suitable Java class (unless the user has explicitly specified a class to be used).

The RapidsDB runtime includes a library of Java classes that implement the standard behaviors and capabilities for SQL types, including RapidsDB-SQL standard functions (see section 3) and type conversions. The system can also be extended with other Java classes ("User Defined Types") which offer extended or specialized capabilities. A User Defined Type may or may not correspond to any standard SQL type.

The table below shows the RapidsDB SQL types, along with the underlying Java class used by default in the execution engine and most Connectors. Note, however, that a given Connector may select a different Java class, for example to provide extended numeric precision or to handle data that doesn't correspond to a SQL type.

SQL Type	Default internal Java class	Description
INTEGER	com.rapidsdata.stdlib.FastInteger	64-bit signed integer, nullable[1]
DECIMAL	com.rapidsdata.stdlib.FastDecimal[2]	64-bit decimal (17 digits precision), nullable
FLOAT	com.rapidsdata.stdlib.FastFloat	64-bit IEEE floating point, nullable[1]
DATE	com.rapidsdata.stdlib.FastDate	64-bit date, range 0000-01-01 to 9999-12-31, nullable
TIMESTAMP	com.rapidsdata.stdlib.FastTimestamp	64-bit microsecond timestamp, nullable
BOOLEAN	com.rapidsdata.stdlib.FastBoolean	Boolean, nullable
VARCHAR	com.rapidsdata.stdlib.FastString[3]	Up to 32k UTF-16 characters, nullable

Notes:

1. The FastInteger and FastFloat types reserve the lowest possible numeric value to represent NULL

2. Some Connectors may use java.math.BigDecimal

3. Some Connectors may use java.lang.String

1.2.2 Type Specifiers

A RapidsDB type specifier specifies a data type and desired precision. Type specifiers are used in the CAST operator (see 1.2.3) and also the column definitions in CREATE TABLE statements (see 6.1) and the USING clause of the CREATE CONNECTOR statement (see Installation and Management Guide).

The interpretation of size, precision and scale values in a RapidsDB type specifier depends on the data type. In this release the interpretations are as follows:

Type	**Default interpretation of size / scale / precision**
INTEGER	Precision in decimal digits (if unspecified: 17)
DECIMAL	Precision in decimal digits, scale in decimal digits (if unspecified: 17, 2)
FLOAT	Size of mantissa in binary digits (if unspecified: 53)
VARCHAR	Maximum length in characters (if unspecifed: limited by Java class)

NOTES:

1. The value for precision of a FLOAT is interpreted as the number of binary digits in the mantissa. This is per ANSI SQL. A 53-bit mantissa corresponds to a standard 64-bit IEEE double precision floating point value. Expressed in decimal, the precision is 22 digits.
2. The type specifier may optionally be followed by a USING clause to explicitly specify a Java class to be used internally.

1.2.3 Use in CAST

In principle, CAST should precisely convert a value to the specified data type and precision or generate an error if this is not possible. So, for example, CAST(myColumn AS INTEGER(3)) should produce a value between -999 and +999 or generate an overflow exception if the value of myColumn lies outside that range. In practice, the behavior may deviate in certain cases, depending on whether the CAST operation is pushed down to the underlying data store (in particular, some data stores fail to generate overflow exceptions where expected).

1.2.4 Use in Column Definitions

When used in a column definition (CREATE TABLE or CREATE CONNECTOR WITH TABLE USING), a type specifier specifies the minimum requirement for a column. The underlying system may optionally substitute a definition of greater size or precision. It may not, however, substitute a definition of lesser size or precision. So, for example, INTEGER(4) specifies a column that can store values in the range -9999 to +9999 (and obeys the rules for integer math). An underlying data store may create the corresponding physical database column as, for example, a 16-bit integer.

If a column definition specifies a capability that exceeds the maximum capability of either the RapidsDB Execution Engine or an underlying Data Store, the definition will be rejected. For example, INTEGER(22) will be rejected by RapidsDB because the maximum precision for integers in RapidsDB is 19 decimal digits.

1.2.5 System Metadata

Column information in the COLUMNS table in the RapidsDB System Metadata (see 9.9) reflects the type, size, precision and scale of columns as reported by the underlying system and interpreted by RapidsDB. The information may differ from the definitions used to create the tables. The column data types are shown as standardized RapidsDB types. Size, precision and scale may exceed the values specified in a RapidsDB CREATE TABLE statement as noted above.

1.2.6 Internal Precision

The type specifiers affect storage, reading, writing and conversion of data values but do not control the precision of calculations on those values during query execution. Query calculations are performed by the RapidsDB Execution Engine (or the query engines of underlying Data Stores) with precision no less than the following:

- For integer values, 64-bit signed integers.
- For floating point values, 64-bit double precision (Java IEEE 754).
- For character values, Java String values of up to 2GB (using UTF-16 encoding).
- For binary values, Java byte arrays of up to 2GB in size.

1.3 Value Expressions

Value expressions are used in a variety of contexts, such as in the target list of the SELECT command or in search conditions. The result of a value expression is sometimes called a scalar, to distinguish it from the result of a table expression (which is a table). Value expressions are therefore also called scalar expressions (or even simply expressions). The expression syntax allows the calculation of values from primitive parts using arithmetic, logical, set, and other operations.

A value expression is one of the following:

- A constant or literal value
- A column reference
- An operator invocation
- A function call
- An aggregate expression
- A type cast
- A scalar subquery
- Another value expression in parentheses (used to group subexpressions and override precedence)

In addition to this list, there are a number of constructs that can be classified as an expression but do not follow any general syntax rules. These generally have the semantics of a function or operator and are explained in the appropriate location in Chapter 1. An example is the IS NULL clause.

1.3.1 Column References

A column can be referenced in the form:

```
qualifier.columnname
```

qualifier is the name of a table, or an alias for a table defined by means of a FROM clause. The qualifier and separating dot can be omitted if the column name is unique across all the tables being used in the current query.

1.3.2 Operator Invocation

There are three possible syntaxes for an operator invocation:

- expression operator expression (binary infix operator)
- operator expression (unary prefix operator)
- expression operator (unary postfix operator) where the operator token follows the syntax rules of Section 1.1.3, or is one of the keywords AND, OR, and NOT

1.3.3 Function Call

The syntax for a function call is the name of a function followed by its argument list enclosed in parentheses:

```
function_name ([expression [, expression ... ]] )
```

For example, the following computes maximum value of column C1:

```
max(c1)
```

The list of built-in functions is in Section 1.

1.3.4 Aggregate Expression

An aggregate expression represents the application of an aggregate function across the rows selected by a query. An aggregate function reduces multiple inputs to a single output value, such as the sum or average of the inputs. The syntax of an aggregate expression is one of the following:

```
aggregate_name (expression [ , ... ] )
aggregate_name (ALL expression [ , ... ])
aggregate_name (DISTINCT expression [ , ... ])
```

where aggregate_name is a system-defined aggregate and expression is any value expression that does not itself contain an aggregate.

The first form of aggregate expression invokes the aggregate once for each input row. The second form is the same as the first, since ALL is the default. The third form invokes the aggregate once for each distinct value of the expression (or distinct set of values, for multiple expressions) found in the input rows.

Most aggregate functions ignore null inputs, so that rows in which one or more of the expression(s) yield null are discarded.

For example, count(*) yields the total number of input rows; count(f1) yields the number of input rows in which f1 is non-null, since count ignores nulls; and count(distinct f1) yields the number of distinct non-null values of f1.

1.3.5 Type Cast

A type cast specifies a conversion from one data type to another:

```
CAST ( expression AS type )
```

When a cast is applied to a value expression of a known type, it represents a run-time type conversion. The cast will succeed only if a suitable type conversion operation has been defined. A cast applied to an unadorned string literal represents the initial assignment of a type to a literal constant value, and so it will succeed for any type (if the contents of the string literal are acceptable input syntax for the data type).

The table below shows the supported cast operations:

	Target							
Source	**INTEGER**	**DECIMAL**	**FLOAT**	**STRING**	**TIMESTAMP**	**BOOLEAN**	**BINARY**	**NULL**
INTEGER		Y	Y	Y	N	N	N	N
DECIMAL	Y	Y	Y	Y	N	N	N	N
FLOAT	Y	Y	Y	Y	N	N	N	N
STRING	Y	Y	Y	Y	Y	Y	N	N
TIMESTAMP	N	N	N	Y	Y	N	N	N
BOOLEAN	N	N	N	Y	N	Y	N	N
BINARY	N	N	N	N	N	N	Y	N
NULL	Y	Y	Y	Y	Y	Y	Y	Y

1.3.6 Decimal Expressions and Precision

Decimal expressions are mathematical expressions with a data type DECIMAL. Decimal values occur either because a column is defined as type DECIMAL or because a value is converted to DECIMAL using the CAST operator.

A decimal value has a precision, which is the total number of significant digits in the value, and a scale, which is the number of digits to the right of the decimal point.

When the RapidsDB Query Planner analyzes a decimal expression, it assumes a "canonical" precision and scale for each operator in the expression. The actual precision and scale depend on the Java class(es) involved in the calculations. The canonical precision and scale rules are designed to preserve sufficient decimal places to fully represent the possible result of the calculation. The following table summarizes the canonical precision assumptions. p1 and s1 represent the precision and scale of the first operand of a math operator; p2 and s2 represent the precision and scale of the second operand.

Operation	Canonical Precision and Scale
+ or -	Scale = max(s1, s2) Precision = max(p1 - s1, p2 - s2) + 1 + scale
*	Scale = s1 + s2 Precision = p1 + p2 + 1
/	Scale = max(4, s1 + p2 + 1) Precision = p1 - s1 + s2 + scale

Note that in all cases, the actual maximum precision of a decimal calculation depends on the underlying Java class. During execution of a query, if a calculation produces a result whose precision would exceed the maximum, the scale is typically reduced to preserve the integral part of the result.

The precision and scale of a decimal result can be specified explicitly using the CAST operator (see 1.3.5).

1.3.7 Scalar Subquery

A scalar subquery is an ordinary SELECT query in parentheses that returns exactly one row with one column. (See Chapter 6 for information about writing queries.) The SELECT query is executed and the single returned value is used as the expression result. It is an error to use a query that returns more than one row or more than one column as a scalar subquery. (But if, during a particular execution, the subquery returns no rows, there is no error; the scalar result is taken to be NULL.) The subquery can refer to variables from the surrounding query, which will act as constants during any one evaluation of the subquery.

For example, the following finds the largest city population in each state:

```
SELECT name, (SELECT max(pop) FROM cities WHERE cities.state = states.name) FROM states;
```


1.3.8 Expression Evaluation Rules

The query optimizer may significantly reorganize a query to improve performance. As a result, the order of evaluation of query expressions is not defined. Subqueries may be executed in any order or in parallel. Notably, the arguments of an operator or function are not necessarily evaluated left-to-right or in any other fixed order.

Furthermore, if the result of an expression can be determined by evaluating only some parts of it, then other subexpressions might not be evaluated at all. For instance, if one wrote:

SELECT true OR somefunc();

then somefunc() would (probably) not be called at all.

As a consequence, it is unwise to use functions with side effects as part of complex expressions. It is particularly dangerous to rely on side effects or evaluation order in WHERE and HAVING clauses, since those clauses are extensively reprocessed as part of developing an execution plan.

A common situation is trying to avoid division by zero in a WHERE clause. Attempting to check for a zero value first is not reliable:

SELECT ... WHERE x > 0 AND y/x > 1.5;

A better solution is to use the NULLIF function (see section 3.7.5).

Note that CASE statements are also not guaranteed to execute in order. For example, a CASE cannot prevent evaluation of an aggregate expression contained within it, because aggregate expressions are computed before other expressions in a SELECT list or HAVING clause are considered. For example, the following query can cause a division-by-zero error despite seemingly having protected against it:

```
SELECT CASE WHEN min(employees) > 0
            THEN avg(expenses / employees)
       END
  FROM departments;
```

The min() and avg() aggregates are computed concurrently over all the input rows, so if any row has employees equal to zero, the division-by-zero error will occur before there is any opportunity to test the result of min(). Instead, use a WHERE clause to prevent problematic input rows from reaching an aggregate function in the first place.

2 Queries

2.1 Overview

The general syntax of the SELECT command is


```
SELECT select_list FROM table_expression [sort_specification]
```

The following sections describe the details of the select list, the table expression, and the sort specification.

A simple kind of query has the form:

SELECT * FROM table1;

The select list specification * means all columns that the table expression happens to provide. A select list can also select a subset of the available columns or make calculations using the columns. For example, if table1 has columns named a, b, and c you can make the following query:

SELECT a, b + c FROM table1;

(assuming that b and c are of a numerical data type). See Section 1.3 for more details.

FROM table1 is a simple kind of table expression: it reads just one table. In general, table expressions can be complex constructs of base tables, joins, and subqueries. But you can also omit the table expression entirely and use the SELECT command as a calculator:

SELECT 3 * 4;

This is more useful if the expressions in the select list return varying results. For example, you could call a function this way:

SELECT round(123.99);

2.2 Table Expressions

A table expression computes a table. The table expression follows the FROM clause and is optionally followed by WHERE, GROUP BY, and HAVING clauses. Trivial table expressions simply refer to a table, a so-called base table, but more complex expressions can be used to modify or combine base tables in various ways.

The optional WHERE, GROUP BY, and HAVING clauses in the table expression specify a pipeline of successive transformations performed on the table derived in the FROM clause. All these transformations produce a virtual table that provides the rows that are passed to the select list to compute the output rows of the query.

2.2.1 The FROM Clause

The FROM Clause derives a table from one or more other tables given in a comma-separated table reference list.

```
FROM table_expression [, table_expression [, ...]]
```


A table expression can be a table name (optionally qualified by <catalog>.<schema> or <schema>), or a derived table such as a subquery, a JOIN construct, or complex combinations of these. If more than one table expression is listed in the FROM clause, the tables are cross-joined (that is, the Cartesian product of their rows is formed; see below). The result of the FROM list is an intermediate virtual table that can then be subject to transformations by the WHERE, GROUP BY, and HAVING clauses and is finally the result of the overall table expression.

2.2.1.1 Joined Tables

A joined table is a table derived from two other (real or derived) tables according to the rules of the particular join type. Inner, outer, and cross-joins are available. The general syntax of a joined table is

```
T1 join_type T2 [ join_condition ]
```

Joins of all types can be chained together, or nested: either or both T1 and T2 can be joined tables. Parentheses can be used around JOIN clauses to control the join order. In the absence of parentheses, JOIN clauses associate left-to-right.

The following describes the Join Types supported:

2.2.1.1.1 CROSS JOIN

T1 CROSS JOIN T2

For every possible combination of rows from T1 and T2 (i.e., a Cartesian product), the joined table will contain a row consisting of all columns in T1 followed by all columns in T2. If the tables have N and M rows respectively, the joined table will have N * M rows.

2.2.1.1.2 INNER JOIN

For each row R1 of T1, the joined table has a row for each row in T2 that satisfies the join condition with R1.

2.2.1.1.3 LEFT OUTER JOIN

First, an inner join is performed. Then, for each row in T1 that does not satisfy the join condition with any row in T2, a joined row is added with null values in columns of T2. Thus, the joined table always has at least one row for each row in T1.

2.2.1.1.4 RIGHT OUTER JOIN

First, an inner join is performed. Then, for each row in T2 that does not satisfy the join condition with any row in T1, a joined row is added with null values in columns of T1. This is the converse of a left join: the result table will always have a row for each row in T2.

2.2.1.1.5 ON Clause

The ON clause is the most general kind of join condition: it takes a Boolean value expression of the same kind as is used in a WHERE clause. A pair of rows from T1 and T2 match if the ON expression evaluates to true.

2.2.1.1.6 USING Clause

The USING clause is a shorthand that allows you to take advantage of the specific situation where both sides of the join use the same name for the joining column(s). It takes a comma-separated list of the shared column names and forms a join condition that includes an equality comparison for each one. For example, joining T1 and T2 with USING (a, b) produces the join condition ON T1.a = T2.a AND T1.b = T2.b.

To put this together, assume we have tables t1, with columns num and name:

num	name
1	a
2	b
3	c

and t2 with columns num and value:

num	value
1	xxx
3	yyy
5	zzz

then we get the following results for the various joins:

```
rapids > select * from t1 inner join t2 using(num);
  NUM NAME     NUM VALUE
  ---- ------  ---- --------
    1 a          1 xxx
    3 c          3 yyy

2 row(s) returned (0.08 sec)
rapids > select * from t1 inner join t2 on t1.num=t2.num;
  NUM NAME     NUM VALUE
  ---- ------  ---- --------
    1 a          1 xxx
    3 c          3 yyy

2 row(s) returned (0.06 sec)
rapids >  select * from t1 left join t2 on t1.num=t2.num;
  NUM NAME      NUM VALUE
  --- ----      --- -----
```



```
    1 a           1 xxx
    3 c           3 yyy
    2 b        NULL NULL

3 row(s) returned (0.08 sec)
rapids > select * from t1 right join t2 on t1.num=t2.num;
   NUM NAME     NUM VALUE
   ----- ------     ----- --------
     1 a          1 xxx
     3 c          3 yyy
  NULL NULL       5 zzz

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

The join condition specified with ON can also contain conditions that do not relate directly to the join. This can prove useful for some queries but needs to be thought out carefully. For example:

```
rapids > select * from t1 left join t2 on t1.num = t2.num and t2.value = 'xxx';
  NUM NAME      NUM VALUE
  --- ----      --- -----
    1 a           1 xxx
    2 b        NULL NULL
    3 c        NULL NULL

3 row(s) returned
```

Notice that placing the restriction in the WHERE clause produces a different result:

```
rapids > select * from t1 left join t2 on t1.num = t2.num where t2.value = 'xxx';
  NUM NAME     NUM VALUE
  --- ----     --- -----
    1 a          1 xxx

1 row(s) returned
```

This is because a restriction placed in the ON clause is processed before the join, while a restriction placed in the WHERE clause is processed after the join. That does not matter with inner joins, but it matters a lot with outer joins.

2.2.1.2 Table and Column Aliases

A temporary name can be given to tables and complex table expressions to be used for references to the derived table in the rest of the query. This is called a table alias.

To create a table alias, write

```
        FROM table_expression AS alias
Or

        FROM table_expression alias
```

The AS keyword is optional. The alias can be any valid identifier.

A typical application of table aliases is to assign short identifiers to long table names to keep the join clauses readable. For example:

SELECT * FROM some_very_long_table_name s JOIN another_fairly_long_name a ON s.id = a.num;

The alias becomes the new name of the table expression within the current query—the original name cannot be used elsewhere in the query. Thus, this is not valid:

SELECT * FROM my_table AS m WHERE my_table.a > 5; -- wrong

Table aliases are mainly for notational convenience, but it is necessary to use them when joining a table to itself, e.g.:

SELECT * FROM people AS mother JOIN people AS child ON mother.id = child.mother_id;

Parentheses are used to resolve ambiguities. In the following example, the first statement assigns the alias b to the second instance of my_table, but the second statement assigns the alias to the result of the join:

SELECT * FROM my_table AS a CROSS JOIN my_table AS b ...

SELECT * FROM (my_table AS a CROSS JOIN my_table) AS b ...

An extended form of table aliasing gives temporary names to the columns of the table, as well as the table itself:

FROM table_expression [AS] table_alias (column_alias1 [, column_alias2 [, ...]])

If fewer column aliases are specified than the number of columns in the table expression, the remaining columns are not renamed and will not participate in the query. This syntax is especially useful for self-joins or subqueries.

When an alias is applied to the output of a JOIN clause, the alias hides the original name(s) within the JOIN. For example:

SELECT a.* FROM my_table AS a JOIN your_table AS b ON ...

is valid SQL, but:

SELECT a.* FROM (my_table AS a JOIN your_table AS b ON ...) AS c

is not valid; the table alias a is not visible outside the alias c.

NOTE:

The alias name cannot be a reserved word unless it is enclosed in double quotes. For example, the following query will fail because the word “order” is a reserved word:

select cast(f_col1 as integer) as order from t1 where f_col2 > 480; To use a reserved word as an

identifier, enclose it in back-ticks:

select cast(f_col1 as integer) as `order` from t1 where f_col2 > 480;

2.2.1.3 Subqueries

Subqueries specifying a derived table must be enclosed in parentheses and may optionally be assigned a table alias name (as in Section 5.2.1.2). For example:

SELECT * FROM (SELECT * FROM table1) AS alias_name

This example is equivalent to SELECT * FROM table1 AS alias_name. More interesting cases, which cannot be reduced to a plain join, arise when the subquery involves grouping or aggregation.

For more information see Section 7.9.

2.2.2 WHERE Clause

The syntax of the WHERE Clause is

```
WHERE <search_condition>
```

where search_condition is any value expression (see Section 4.2) that returns a value of type boolean.

After the processing of the FROM clause is done, each row of the derived virtual table is checked against the search condition. If the result of the condition is true, the row is kept in the output table, otherwise (i.e., if the result is false or null) it is discarded.

Here are some examples of WHERE clauses:

SELECT ... FROM fdt WHERE c1 > 5

SELECT ... FROM fdt WHERE c1 IN (1, 2, 3)

SELECT ... FROM fdt WHERE c1 IN (SELECT c1 FROM t2)

SELECT ... FROM fdt WHERE c1 IN (SELECT c3 FROM t2 WHERE c2 = fdt.c1 + 10)

SELECT ... FROM fdt WHERE c1 BETWEEN (SELECT c3 FROM t2 WHERE c2 = fdt.c1 + 10) AND 100

SELECT ... FROM fdt WHERE EXISTS (SELECT c1 FROM t2 WHERE c2 > fdt.c1)

fdt is the table derived in the FROM clause. Rows that do not meet the search condition of the WHERE clause are eliminated from fdt. Notice the use of scalar subqueries as value expressions. Just like any other query, the subqueries can employ complex table expressions. Notice also how fdt is referenced in the subqueries. Qualifying c1 as fdt.c1 is only necessary if c1 is also the name of a column in the derived

input table of the subquery. But qualifying the column name adds clarity even when it is not required. This example shows how the column naming scope of an outer query extends into its inner queries.

2.2.3 GROUP BY and HAVING Clause

After passing the WHERE filter, the derived input table might be subject to grouping, using the GROUP BY clause, and elimination of group rows using the HAVING clause.

```
SELECT select_list
   FROM ...
   [WHERE ...]
   GROUP BY exprList
```

The GROUP BY Clause is used to group together those rows in a table that have the same values in all the columns listed. The order in which the columns are listed does not matter. The effect is to combine each set of rows having common values into one group row that represents all rows in the group. This is done to eliminate redundancy in the output and/or compute aggregates that apply to these groups.

If a table has been grouped using GROUP BY, but only certain groups are of interest, the HAVING clause can be used, much like a WHERE clause, to eliminate groups from the result. The syntax is:

```
SELECT select_list FROM ... [WHERE ...] GROUP BY ... HAVING boolean_expression
```

Expressions in the HAVING clause can refer both to grouped expressions and to ungrouped expressions (which necessarily involve an aggregate function).

Examples:

SELECT x, sum(y) FROM test1 GROUP BY x HAVING sum(y) > 3;

SELECT x, sum(y) FROM test1 GROUP BY x HAVING x < 'c';

```
SELECT product_id, p.name, (sum(s.units) * (p.price - p.cost)) AS profit
   FROM products p LEFT JOIN sales s USING (product_id)
   WHERE s.date > '2015-06-01 00:00:00' AND s.date < '2015-07-01 00:00:00'
   GROUP BY product_id, p.name, p.price, p.cost
   HAVING sum(p.price * s.units) > 5000;
```

In the example above, the WHERE clause is selecting rows by a column that is not grouped (the expression is only true for sales during the month of June), while the HAVING clause restricts the output to groups with total gross sales over 5000. Note that the aggregate expressions do not necessarily need to be the same in all parts of the query.

2.3 SELECT Lists

As shown in the previous section, the table expression in the SELECT command constructs an intermediate virtual table by possibly combining tables, views, eliminating rows, grouping, etc. This table is finally passed on to processing by the select list. The select list determines which columns of the intermediate table are include in the result.

2.3.1 SELECT List Items

The simplest kind of select list is * which emits all columns that the table expression produces. Otherwise, a select list is a comma-separated list of value expressions. For instance, it could be a list of column names:

SELECT a, b, c FROM …

The columns names a, b, and c are either the actual names of the columns of tables referenced in the FROM clause, or the aliases given to them as explained in Section 2.2.1.2. The name space available in the select list is the same as in the WHERE clause, unless grouping is used, in which case it is the same as in the HAVING clause.

If more than one table has a column of the same name, the table name must also be given, as in:

SELECT tbl1.a, tbl2.a, tbl1.b FROM ...

When working with multiple tables, it can also be useful to ask for all the columns of a particular table:

SELECT tbl1.*, tbl2.a FROM ...

(See also Section 1.2.2.)

If an arbitrary value expression is used in the select list, it conceptually adds a new virtual column to the returned table. The value expression is evaluated once for each result row, with the row's values substituted for any column references. But the expressions in the select list do not have to reference any columns in the table expression of the FROM clause; they can be constant arithmetic expressions, for instance.

2.3.2 Column Labels

The entries in the select list can be assigned names for subsequent processing, such as for use in an ORDER BY clause or for display by the client application. For example:

SELECT a AS value, b + c AS sum FROM …

If no output column name is specified using AS, the system assigns a default column name. For simple column references, this is the name of the referenced column. For complex expressions, the system will generate a generic name.


```
rapids > select * from t1;
  NUM NAME
  --- ----
    1 a
    2 b
    3 c

3 row(s) returned (0.06 sec)
rapids > select num+100 as plus_100, name from t1;
  PLUS_100 NAME
  -------- ----
       101 a
       102 b
       103 c

3 row(s) returned (0.06 sec)
```

2.3.3 DISTINCT

After the select list has been processed, the result table can optionally be subject to the elimination of duplicate rows. The DISTINCT keyword is written directly after SELECT to specify this:

```
SELECT DISTINCT select_list …
```

(Instead of DISTINCT the keyword ALL can be used to specify the default behavior of retaining all rows.)

Obviously, two rows are considered distinct if they differ in at least one column value. Null values are considered equal in this comparison.

2.4 Combining Queries (UNION, INTERSECT, EXCEPT)

2.4.1 UNION

The results of two queries can be combined using the UNION set operation. The syntax is

```
query1 UNION [ALL] query2
```

query1 and query2 are queries that can use any of the features discussed up to this point. Set operations can also be nested and chained, for example

query1 UNION query2 UNION query3

which is executed as:

(query1 UNION query2) UNION query3

UNION effectively appends the result of query2 to the result of query1 (although there is no guarantee that this is the order in which the rows are actually returned). Furthermore, it eliminates duplicate rows from its result, in the same way as DISTINCT, unless UNION ALL is used.

In order to calculate the union of two queries, the two queries must be "union compatible", which means that they return the same number of columns and the corresponding columns have compatible data types. Also, any LIMIT or ORDER BY clause can only appear at the end of statement.

```
rapids > select * from t1;
  NUM NAME
  --- ----
    1 a
    2 b
    3 c

3 row(s) returned (0.05 sec)
rapids > select * from t3;
  NUM VALUE
  --- -----
    1 xxx
    2 yyy
    3 zzz
    1 xxx

4 row(s) returned (0.05 sec)
rapids > select * from t1 union select * from t3;
  NUM NAME
  --- ----
    1 a
    2 b
    3 c
    1 xxx
    2 yyy
    3 zzz

6 row(s) returned (0.13 sec)
```

```
rapids > select * from t1 union all select * from t3;
  NUM NAME
  --- ----
    1 a
    2 b
    3 c
    1 xxx
    2 yyy
    3 zzz
    1 xxx

7 row(s) returned (0.11 sec)
```

2.4.2 INTERSECT

INTERSECT returns any distinct values that are returned by both the query on the left and right sides of the INTERSECT operator.

The syntax is

```
query1 INTERSECT query2
```


query1 and query2 are queries that can use any of the features discussed up to this point. Set operations can also be nested and chained, for example

query1 INTESECT query2 INTERSECT query3

which is executed as:

(query1 INSERSECT query2) INTERSECT query3

In order to calculate the intersect of two queries, the two queries must be "intersect compatible", which means that they return the same number of columns and the corresponding columns have compatible data types.

```
rapids > select * from t2;
  NUM VALUE
  --- -----
    1 xxx
    2 yyy
    3 zzz

3 row(s) returned (0.05 sec)
rapids > select * from t3;
  NUM VALUE
  --- -----
    1 xxx
    2 yyy
    3 zzz
    1 xxx
    4 aaa

5 row(s) returned (0.06 sec)
rapids >  select * from t3 intersect select * from t2;
  NUM VALUE
  --- -----
    1 xxx
    2 yyy
    3 zzz

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

2.4.3 EXCEPT

EXCEPT returns any distinct values from the query left of the EXCEPT operator. Those values return as long the right query doesn't return those values as well.

The syntax is

```
query1 EXCEPT query2
```

query1 and query2 are queries that can use any of the features discussed up to this point. Set operations can also be nested and chained, for example

query1 EXCEPT query2 EXCEPT query3

which is executed as:

(query1 EXCEPT query2) EXCEPT query3

In order to calculate the except of two queries, the two queries must be "except compatible", which means that they return the same number of columns and the corresponding columns have compatible data types. Also, any LIMIT or ORDER BY clause can only appear at the end of statement.

```
rapids > select * from t2;
  NUM VALUE
  --- -----
    1 xxx
    2 yyy
    3 zzz

3 row(s) returned (0.05 sec)
rapids > select * from t3;
  NUM VALUE
  --- -----
    1 xxx
    2 yyy
    3 zzz
    1 xxx
    4 aaa

5 row(s) returned (0.06 sec)
rapids >  select * from t3 except select * from t2;
  NUM VALUE
  --- -----
    4 aaa

1 row(s) returned (0.08 sec)
```

2.5 ORDER BY

After a query has produced an output table (after the select list has been processed) it can optionally be sorted. If sorting is not chosen, the rows will be returned in an unspecified order. The actual order in that case will depend on the scan and join plan types, but it must not be relied on. A particular output ordering can only be guaranteed if the sort step is explicitly chosen.

The ORDER BY clause specifies the sort order:

```
SELECT select_list
   FROM table_expression
   ORDER BY orderByList [ASC | DESC]
```

The orderByList can be any expression that would be valid in the query's select list. An example is:

SELECT a, b FROM table1 ORDER BY a + b, c;

When more than one expression is specified, the later values are used to sort rows that are equal according to the earlier values. Each expression can be followed by an optional ASC or DESC keyword to

set the sort direction to ascending or descending. ASC order is the default. Ascending order puts smaller values first, where "smaller" is defined in terms of the < operator. Similarly, descending order is determined with the > operator.

Note that the ordering options are considered independently for each sort column. For example ORDER BY x, y DESC means ORDER BY x ASC, y DESC, which is not the same as ORDER BY x DESC, y DESC.

A sort_expression can also be the column label or number of an output column, as in:

SELECT a + b AS sum, c FROM table1 ORDER BY sum;

SELECT a, max(b) FROM table1 GROUP BY a ORDER BY 1;

both of which sort by the first output column.

ORDER BY can be applied to the result of a UNION, INTERSECT, or EXCEPT combination, but in this case it is only permitted to sort by output column names or numbers, not by expressions.

2.6 LIMIT and OFFSET

LIMIT and OFFSET allow you to retrieve just a portion of the rows that are generated by the rest of the query:

```
SELECT select_list
   FROM table_expression
   [ ORDER BY ... ]
   [ LIMIT { number } ] [ OFFSET number ]
```

If a limit count is given, no more than that many rows will be returned (but possibly less, if the query itself yields less rows).

OFFSET says to skip that many rows before beginning to return rows. OFFSET 0 is the same as omitting the OFFSET clause. If both OFFSET and LIMIT appear, then OFFSET rows are skipped before starting to count the LIMIT rows that are returned.

When using LIMIT or OFFSET, it is important to use an ORDER BY clause that constrains the result rows into a unique order. Otherwise you will get an unpredictable subset of the query's rows. You might be asking for the tenth through twentieth rows, but tenth through twentieth in what ordering? The ordering is unknown, unless you specified ORDER BY.

The rows skipped by an OFFSET clause still have to be computed inside the server; therefore a large OFFSET might be inefficient.

2.7 WITH (Common Table Expressions)

WITH provides a way to write auxiliary statements for use in a larger query. These statements are often referred to as Common Table Expressions or CTEs.


```
WITH common_table_expression [, common_table_expression ...] SELECT query;

common_table_expression:: table_name [(column list)] AS (SELECT query)

column_list:: column_name [,column_name ...]
```

Key characteristics of CTEs:

- RapidsDB supports only non-recursive CTEs.
- The column_list is optional, and when specified, the columns of CTE will be known by the names specified and the column names or aliases of the underlying query are not visible when referring to the CTE. In the example below, the column names in the CTE are x, y and z whereas the column names in the underlying query are a,b and c. (Note that the original names are still used normally within the underlying query itself, e.g. in the WHERE clause in the example below):

  WITH cte_1(x, y, z) AS (SELECT a, b, c FROM t WHERE a < 5) SELECT x FROM cte_1;
- When the CTE is referenced in a SELECT statement, the CTE will be merged into the query and executed as part of the query. If there are multiple references to the same CTE, each reference to the CTE will be executed. In a future release, references to the CTE will be optimized to avoid unnecessary computation.
- If a CTE defined in the WITH clause is not referenced in the SELECT statement, it has no effect on the execution of the query.

Example:

```
WITH regional_sales AS (
            SELECT region, SUM(amount) AS total_sales
            FROM orders GROUP BY region
          ), top_regions AS (
            SELECT region
            FROM regional_sales
            WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales)
          )
SELECT region,
       product,
       SUM(quantity) AS product_units,
       SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;
```


3 Functions and Operators

3.1 Logical Operators

The usual logical operators are available:

AND
OR
NOT

SQL uses a three-valued logic system with true, false, and null, which represents "unknown". Observe the following truth tables:

a	b	a AND b	a OR b
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	NULL	NULL	NULL

a	NOT a
TRUE	FALSE
FALSE	TRUE
NULL	NULL

The operators AND and OR are commutative, that is, you can switch the left and right operand without affecting the result.

3.2 Comparison Operators and BETWEEN

The usual comparison operators are available:

Operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
=	equal

<> or !=	not equal

Comparison operators are available for all relevant data types. All comparison operators are binary operators that return values of type Boolean.

In addition to the comparison operators, the special BETWEEN construct is available:

a BETWEEN x AND y

is equivalent to

a >= x AND a <= y

Notice that BETWEEN treats the endpoint values as included in the range. NOT BETWEEN does the opposite comparison:

a NOT BETWEEN x AND y

is equivalent to

a < x OR a > y

To check whether a value is or is not null, use the constructs:

expression IS NULL

expression IS NOT NULL

Do not write expression = NULL because NULL is not "equal to" NULL. (The null value represents an unknown value, and it is not known whether two unknown values are equal.) This behavior conforms to the SQL standard.

3.3 Mathematical Operators and Functions

Mathematical Operators

Operator	Description	Example	Result
+	addition	2 + 3	5
-	subtraction	2 - 3	-1
*	multiplication	2 * 3	6
/	division (integer division truncates the result)	4 / 2	2

%	modulo (remainder)	5 % 4	1

Mathematical Functions

Function	Return Type	Description	Example	Result
abs(x)	(same as input)	absolute value	abs(-17.4)	17.4
ceil(numeric)	integer	smallest integer not less than argument	ceil(-42.8)	-42
ceiling(numeric)	integer	smallest integer not less than argument (alias for ceil)	ceiling(-95.3)	-95
floor(numeric)	integer	largest integer not greater than argument	floor(-42.8)	-43
mod(numeric, numeric)	float	returns the remainder of the first argument divided by the second argument	mod(1.25, 0.5)	0.25
power(numeric, numeric)	float	raise first argument to the power of the second argument	power(9, 2)	81.0
round(numeric)	integer	round to nearest integer	round(123.99)	123

round(numeric, int)	float	round to int number of decimal places	round((123.999,2)	123.99
sqrt(numeric)	float	square root of argument	sqrt(10)	3.1622776985168457
stddev(expression)	float	historical alias for stddev_samp		
stddev_pop(expression)	float	population standard deviation of the input values		
stddev_samp(expression)	float	sample standard deviation of the input values		
variance(expression)	float	historical alias for var_samp		
var_pop(expression)	float	population variance of the input values (square of the population standard deviation)		
var_samp(expression)	float	sample variance of the input values (square of the sample standard deviation)		

3.4 String Functions and Operators

Function	Return Type	Description	Example	Result
concat(string, string)	text	String concatentation	'Post' || 'greSQL'	'PostgreSQL'
concat(string, numeric) concat (numeric,string)	text	String concatentation	concat('Value: ', 3.1) concat(3.1, ' times')	'Value: 3.1' '3.1 times'
string || string	text	String concatenation	'Post' || 'greSQL'	'PostgreSQL'
string || numeric	text	String concatenation	'Value: ' || 3.1	'Value: 3.1'
string + string	text	String concatenation	'Post' + 'greSQL'	'PostgreSQL'
string + numeric	text	String concatenation	'Value: ' + 3.1	'Value: 3.1'
char_length(string)	int	Number of characters in string	char_length('jose')	4
lower(string)	text	Convert string to lower case	lower('TOM')	'tom'
position(substring in string)	int	Location of specified substring	position('om' in 'Thomas')	3
repeat(string, int)	text	Repeat the specified string for the specified number of times.	repeat('Post',2)	'PostPost'
substring(string from int [for int])	text	Extract substring starting at the "from" position, for	substring('Thomas' from 2 for 3)	'hom'

		the length specified by the “for” (defaults to rest of string)	substring(‘Thomas’ from 2)	'homas'
substring(string from negative int [for int])	text	Extract substring starting at the “from” position counting backwards from the right of the string for the length specified by the “for” (defaults to rest of string)	Substring(‘Thomas’ from -3 for 3)	'mas'
trim([leading | trailing | both] [character] from string)	text	Remove the longest string containing only the specified character (a space by default) from the start/end/both ends of the string	trim(both 'x' from 'xTomxx')	'Tom'
ltrim(string [,character])	text	Remove the longest string containing only the specified character (a space by default) from the start of the string	ltrim(‘ Tom’) ltrim(‘aaTom’,’a’)	'Tom' 'Tom'

rtrim(string [,character])	text	Remove the longest string containing only the specified character (a space by default) from the end of the string	rtrim('Tom ') rtrim('Tomaa','a')	'Tom' 'Tom'
upper(string)	text	Convert string to upper case	upper('tom')	'TOM'
left(str text, n int)	text	Return first n characters in the string. When n is negative an empty string will be returned.	left('Tomas',2)	'To'
right(str text, n int)	text	Return last n characters in the string. When n is negative an empty string will be returned.	right('Tomas',2)	'as'

3.5 Pattern Matching – LIKE

```
{string-expression} LIKE '{pattern}' [ESCAPE 'escape-character']
```

The LIKE expression returns true if the string-expression matches the supplied pattern. (As expected, the NOT LIKE expression returns false if LIKE returns true, and vice versa.)

If pattern does not contain percent signs or underscores, then the pattern only represents the string itself; in that case LIKE acts like the equals operator. An underscore (_) in pattern stands for (matches) any single character; a percent sign (%) matches any sequence of zero or more characters.

Some examples:

'abc' LIKE 'abc' true
'abc' LIKE 'a%' true
'abc' LIKE '_b_' true
'abc' LIKE 'c' false

LIKE pattern matching always covers the entire string. Therefore, if it's desired to match a sequence anywhere within a string, the pattern must start and end with a percent sign.

To match a literal underscore or percent sign the user must specify an escape character to be used as part of the pattern string by adding the ESCAPE clause after the pattern string. The respective character in the pattern must be preceded then by the escape character.

Some examples:

'a_b' LIKE 'a_%' ESCAPE '\' true
'abc' LIKE 'a_%' ESCAPE '\' false

3.6 Date/Time Functions

3.6.1 EXTRACT(from timestamp)

```
EXTRACT( selection-keyword FROM timestamp-expression )
```

The EXTRACT() function returns the value of the selected portion of a timestamp. The table below lists the supported keywords, the datatype of the value returned by the function, and a description of its contents.

Keyword	Datatype	Description
YEAR	INTEGER	The year as a numeric value.
QUARTER	INTEGER	The quarter of the year as a single numeric value between 1 and 4.
MONTH	INTEGER	The month of the year as a numeric value between 1 and 12.
DAY	INTEGER	The day of the month as a numeric value between 1 and 31.
WEEK	INTEGER	The week of the year as a numeric value between 1 and 52.
HOUR	INTEGER	The hour of the day as a numeric value between 0 and 23.

Keyword	Datatype	Description
MINUTE	INTEGER	The minute of the hour as a numeric value between 0 and 59.
SECOND	INTEGER	The whole part of the number of seconds within the minute as a value between 0 and 59.

```
rapids >  select * from t3;
C1
--
2021-08-19 09:01:02.12

1 row(s) returned (0.06 sec)
rapids >  select extract(second from c1) from t3;
 [1]
 ---
  2

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

3.6.2 CURRENT_TIMESTAMP

```
CURRENT_TIMESTAMP
```

Returns the current date and time as a timestamp

```
rapids > select current_timestamp;
[1]
---
2020-12-19 19:10:21.746

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

3.6.3 NOW()

```
NOW()
```

Returns the current date and time as a timestamp. Equivalent to CURRENT_TIMESTAMP.


```
rapids > select now();
[1]
---
2020-12-19 19:11:04.705

1 row(s) returned (0.06 sec)
```

3.6.4 Interval Arithmetic

3.6.4.1 Interval Types

RapidsDB provides support for INTERVAL arithmetic as defined by the SQL-99 standard. There are two types of intervals:

YEAR-MONTH INTERVAL

Examples:

- INTERVAL '1' YEAR
- INTERVAL '2' MONTH
- INTERVAL '1-2' YEAR TO MONTH

DAY_TIME INTERVAL

Examples:

- INTERVAL '5' DAY
- INTERVAL '5 10:10' DAY TO MINUTE
- INTERVAL '1 2:10:10.234' DAY TO SECOND
  and so on...

Precision:

The user can specify a leading precision for any of the intervals. The default precision for all of DAY, HOUR, MINUTE, SECOND, YEAR, MONTH is 2. The maximum precision allowed is 9. The default fractional second precision is 6, and the maximum is 9. You can specify precision for the leading field and also for the SECOND field, the remaining fields will follow the default precision.

The following are valid intervals:

- INTERVAL '1-2' YEAR TO MONTH
- INTERVAL '13-2' YEAR TO MONTH
- INTERVAL '199-2' YEAR(3) TO MONTH
- INTERVAL '199' MONTH(3)------- NOTE: we have to specify the precision of three because the value 199 is greater than the default precision
- INTERVAL '1 10:10:10.234' DAY TO SECOND
- INTERVAL '123 10:10:10.234' DAY(3) TO SECOND-------NOTE: we have to specify the precision of three because the value 123 is greater than the default precision
- INTERVAL '123 10:10:10.12345678' DAY(3) TO SECOND(8)-------NOTE: we have to specify the precision of three because the value 123 for the DAY is greater than the default precision, and the precision for SECOND is also greater than the default
- INTERVAL '123 10:10:10.12345678' DAY(3) TO SECOND : the fractional second will round off to the default 6 digits precision, and you will get back: +123 10:10:10.123457 NOTE: we have to

specify the precision of three because the value 123 for the DAY is greater than the default precision

Range:
Can be negative or positive.

3.6.4.2 YEAR-MONTH interval:

Can be negative or positive:

year - constrained by precision. Hence with a precision of 9 the maximum value can be 999999999
month - 0 to 11 (But if leading then constrained by precision).

The following example is valid:
INTERVAL '10-10' YEAR TO MONTH.

but the following is invalid:
INTERVAL '10-13' YEAR TO MONTH.

The following is also valid:
INTERVAL '13' MONTH
This is valid because MONTH is the leading number, and so it is constrained by precision, and the leading default precision is 2. So you can have a max value of 99 in month. But if you specify precision of more than 2 it can be higher.
For example, you can have:

INTERVAL '999' MONTH(3)

3.6.4.3 DAY-TIME interval:

Can be negative or positive:

day - constrained by precision. Hence with a precision of 9, the maximum value can be 999999999
hour - 0 to 23
minute - 0 to 59
second - 0 to 59.999999999

Note that if hour, minute or second are leading then we can specify a precision other than the default for them.
e.g INTERVAL '999' HOUR(3)

Also note that we can give a fractional second precision:
e.g INTERVAL '10:20.30.888' HOUR TO SECOND(3).

We can also have:
INTERVAL '10.89' SECOND(2,2)

Also note that with the fractional second, if the number does not fit the precision, it will get rounded.

e.g INTERVAL '10.23456' SECOND(2,4)
will become '+10.2346'

Support for interval comparisons:
We can compare DATE-TIME intervals with DATE-TIME intervals.
We can compare YEAR-MONTH intervals with YEAR-MONTH intervals.

3.6.4.4 Support for Interval Arithmetic:

Operand	Operator	Operand	Result Type
Timestamp	-	Timestamp	Interval
Timestamp	+	Interval	Timestamp
Timestamp	-	Interval	Timestamp
Interval	+	Timestamp	Timestamp
Interval	+	Interval	Interval
Interval	-	Interval	Interval
Interval	*	Numeric	Interval
Numeric	*	Interval	Interval
Interval	/	Numeric	Interval

Notes:

1. When you do arithmetic on intervals, the resulting interval has a precision of the maximum allowed (see examples below).
2. When doing interval arithmetic with a timestamp literal, the timestamp literal must be specified using the timestamp keyword (see examples below)

Examples:

In the following examples, the table t1 has column c2 defined as a timestamp column:

```
rapids > create table moxe.t1(c1 integer,c2 timestamp);
0 row(s) returned (0.10 sec)
rapids > insert into t1 values(1,'2018-11-12 10:12:13'),(2,'2019-12-08
09:11:30'),(3,'2017-01-01 06:12:50');
0 row(s) returned (0.09 sec)
rapids > select c2-interval '100' HOUR(3) from t1;
```



```
[1]
---
2018-11-08 06:12:13.0
2019-12-04 05:11:30.0
2016-12-28 02:12:50.0

3 row(s) returned (0.05 sec)
rapids > select c2-'2018-01-01' from t1;
   [1]
   ---
   315
   706
  -365

3 row(s) returned (0.05 sec)
rapids > select timestamp '2018-01-01 01:01:01' -interval '100'
HOUR(3) from t1 limit 1;
[1]
---
2017-12-27 21:01:01.0

1 row(s) returned (0.05 sec)
```

3.6.4.5 EXTRACT(from interval)

```
EXTRACT( selection-keyword FROM interval-expression )
```

The EXTRACT() function returns the value of the selected portion of a timestamp. The table below lists the supported keywords, the datatype of the value returned by the function, and a description of its contents.

Keyword	From Interval	Datatype	Description
YEAR	Year-month	INTEGER	The year as a numeric value.
QUARTER	Year-month	INTEGER	The quarter as a numeric value.
MONTH	Year-month	INTEGER	The month as a numeric value
DAY	Day-time	INTEGER	The day as a numeric value
HOUR	Day-time	INTEGER	The hour as a numeric value.

Keyword	From Interval	Datatype	Description
MINUTE	Day-time	INTEGER	The minute as a numeric value.
SECOND	Day-time	INTEGER	The second as a numeric value.

```
rapids > select * from t4;
C1
--
2020-12-25 09:00:00.0

1 row(s) returned (0.06 sec)
rapids > select extract(year from c1) from t4;
   [1]
   ---
  2020

1 row(s) returned (0.06 sec)
rapids > select extract(second from c1) from t4;
  [1]
  ---
    0

1 row(s) returned (0.06 sec)
rapids > select extract(hour from c1) from t4;
  [1]
  ---
    9

1 row(s) returned (0.05 sec)
```

3.6.4.6 BETWEEN Operator:

```
BETWEEN interval1 AND interval2
```

The BETWEEN operator can return the value between two day-time or two year-month intervals. For example:
SELECT …. WHERE TS_INTERVAL BETWEEN INTERVAL '100 10:00:00.000 DAY TO SECOND AND INTERVAL '299 10:00:00.000' DAY TO SECOND …

3.7 CONDITIONAL EXPRESSIONS

3.7.1 CASE

The CASE expression is a generic conditional expression, similar to if/else statements in other programming languages:

```
CASE WHEN condition THEN result
```



```
    [WHEN ...]
    [ELSE result]
END
```

CASE clauses can be used wherever an expression is valid. Each condition is an expression that returns a boolean result. If the condition's result is true, the value of the CASE expression is the result that follows the condition, and the remainder of the CASE expression is not processed. If the condition's result is not true, any subsequent WHEN clauses are examined in the same manner. If no WHEN condition yields true, the value of the CASE expression is the result of the ELSE clause. If the ELSE clause is omitted and no condition is true, the result is null.

An example:

```
rapids > select num from t1;
  NUM
  ---
    1
    2
    3

3 row(s) returned
```

```
rapids > select num,  CASE WHEN num=1 THEN 'ONE' WHEN num=2 THEN 'TWO' ELSE 'OTHER' END AS
TEXT from t1;
  NUM TEXT
  --- ----
    1 ONE
    2 TWO
    3 OTHER

3 row(s) returned
```

3.7.2 COALESCE

```
COALESCE(value [, ...])
```

The COALESCE function returns the first of its arguments that is not null. Null is returned only if all arguments are null. It is often used to substitute a default value for null values when data is retrieved for display, for example:

SELECT COALESCE(description, short_description, '(none)') …

This returns description if it is not null, otherwise short_description if it is not null, otherwise the text '(none)'.

3.7.3 IF

```
IF( boolean_expression, true_result_expression, false_result_expression )
```


IF evaluates the boolean_expression, and then evaluates one of the other two expressions to produce a result. If the boolean_expression is true, then the true_result_expression is evaluated and returned as the result; otherwise the false_result_expression is evaluated and returned as the result.

true_result_expression and false_result_expression may be of any type but the two must match or be implicitly convertible to a common type.

Example:

SELECT IF(1<2, 2, 3) …

This returns the value 3.

3.7.4 IFNULL

```
IFNULL(value1,  value2)
```

The IFNULL function returns value2 if value1 is null; otherwise it returns value1, for example.

SELECT IFNULL(description, '(none)') …

This returns the string '(none)' if the value for the description column is null, otherwise it returns the value for the column .

3.7.5 NULLIF

```
NULLIF(value1,  value2)
```

The NULLIF function returns a null value if value1 equals value2; otherwise it returns value1, for example.

SELECT NULLIF(description, '(none)') …

This returns a null value if the value for the description column equals '(none)' otherwise it returns the value for the description column.

3.8 AGGREGATE FUNCTIONS

Aggregate functions compute a single result from a set of input values.

Function	Argument Type(s)	Return Type	Description
avg(expression)	integer, decimal or float	double precision for a floating-point argument, otherwise same as the argument data type	the average (arithmetic mean) of all input values
count(*)		integer	number of input rows

count(expression)	any	integer	number of input rows for which the value of expression is not null
max(expression)	any numeric, string, or date/time types	same as argument type	maximum value of expression across all input values
min(expression)	any numeric, string, or date/time types	same as argument type	minimum value of expression across all input values
sum(expression)	Integer, decimal or float types	same as the argument data type	sum of expression across all input values

It should be noted that except for count, these functions return a null value when no rows are selected. In particular, sum of no rows returns null, not zero as one might expect. The COALESCE function can be used to substitute zero or an empty array for null when necessary.

3.9 SUB-QUERY EXPRESSIONS

3.9.1 IN

```
expression IN (subquery)
```

The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result. The result of IN is "true" if any equal subquery row is found. The result is "false" if no equal row is found (including the case where the subquery returns no rows).

Note that if the left-hand expression yields null, or if there are no equal right-hand values and at least one right-hand row yields null, the result of the IN construct will be null, not false.

```
rapids > select * from t1;
  NUM NAME
  --- ----
    1 a
    2 b
    3 c

3 row(s) returned (0.06 sec)
rapids > select * from t1 where num in (1,3);
  NUM NAME
  --- ----
    1 a
    3 c

2 row(s) returned (0.06 sec)
```


3.9.2 NOT IN

```
expression NOT IN (subquery)
```

The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result. The result of NOT IN is "true" if only unequal subquery rows are found (including the case where the subquery returns no rows). The result is "false" if any equal row is found.

Note that if the left-hand expression yields null, or if there are no equal right-hand values and at least one right-hand row yields null, the result of the NOT IN construct will be null, not true.

```
rapids > select * from t1;
  NUM NAME
  --- ----
    1 a
    2 b
    3 c

3 row(s) returned (0.06 sec)
rapids > select * from t1 where num not in (1,3);
  NUM NAME
  --- ----
    2 b

1 row(s) returned (0.05 sec)
```

3.9.3 EXISTS

```
EXISTS (subquery)
```

The argument of EXISTS is an arbitrary SELECT statement, or subquery. The subquery is evaluated to determine whether it returns any rows. If it returns at least one row, the result of EXISTS is "true"; if the subquery returns no rows, the result of EXISTS is "false".


```
rapids > select * from t1;
  NUM NAME
  --- ----
    1 a
    2 b
    3 c

3 row(s) returned (0.05 sec)
rapids > select * from t2;
  NUM VALUE
  --- -----
    1 xxx
    2 yyy
    3 zzz

3 row(s) returned (0.06 sec)
rapids > select name from t1 where exists(select value from t2 where t1.num=t2.num and
value='xxx');
NAME
----
a

1 row(s) returned (0.15 sec)
```

3.10 Session Functions

3.10.1 CURRENT_USER

The CURRENT_USER keyword can be used in a SELECT list to return the username of the current session.

```
rapids >  select current_user from rapids.system.tables limit 1;
[1]
---
RAPIDS

1 row(s) returned (0.05 sec)
```

3.10.2 CURRENT_CATALOG

The CURRENT_CATALOG keyword can be used in a SELECT list to return the name of the catalog that will be used for resolving table or object names that are not fully qualified. If a default catalog has not been set then this keyword will return NULL.

```
rapids > set catalog moxe;
rapids > select current_catalog from rapids.system.tables limit 1;
[1]
---
MOXE

1 row(s) returned (0.05 sec)
```


3.10.3 CURRENT_SCHEMA

The CURRENT_SCHEMA keyword can be used in a SELECT list to return the name of the schema that will be used for resolving table or object names that are not fully qualified. If a default schema has not been set then this keyword will return NULL.

```
rapids > set schema moxe;
rapids > select current_schema from rapids.system.tables limit 1;
[1]
---
MOXE

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

3.11 VERSION()

The VERSION function returns version information about the RapidsDB software. There are three supported variations of the VERSION function:

1. VERSION() – returns version number and build date for the RapidsDB Software, the version of Linux and the Java version:

```
rapids > select version();
[1]
---
RapidsDB 4.3.1 2022-01-20, Linux 5.4.0-74-generic, Java 1.8.0_292 (openj9-0.26.0)

1 row(s) returned (0.08 sec)
```

2. VERSION(1) – returns the version number for the RapidsD software:

```
rapids > select version(1);
[1]
---
4.3.1

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

3. VERSION(2) – returns the internal software repository commit id for the RapidsDB software:

```
rapids > select version(2);
[1]
---
bbc91e81a65297e3b945484d3b669087cf11ef77

1 row(s) returned (0.06 sec)
```


4 QUERY EXECUTION

4.1 RapidsDB SQL Statement Execution

RapidsDB will parse a SQL statement and build a query execution plan to execute the SQL statement. When optimizing the execution plan the RapidsDB Optimizer attempts to "push down" as much of the query logic as possible to the underlying Data Store using a minimum number of operations. Those parts of the query logic that cannot be pushed down will be executed by the RapidsDB Execution Engine. For example, when executing a JOIN that includes tables from two different Connectors, the join will take place in the RapidsDB Execution Engine. **EXPLAIN** (see 11.1) can be used to see which parts of the query will be executed in RapidsDB and to inspect the SQL statements that will be sent to the underlying Data Store.

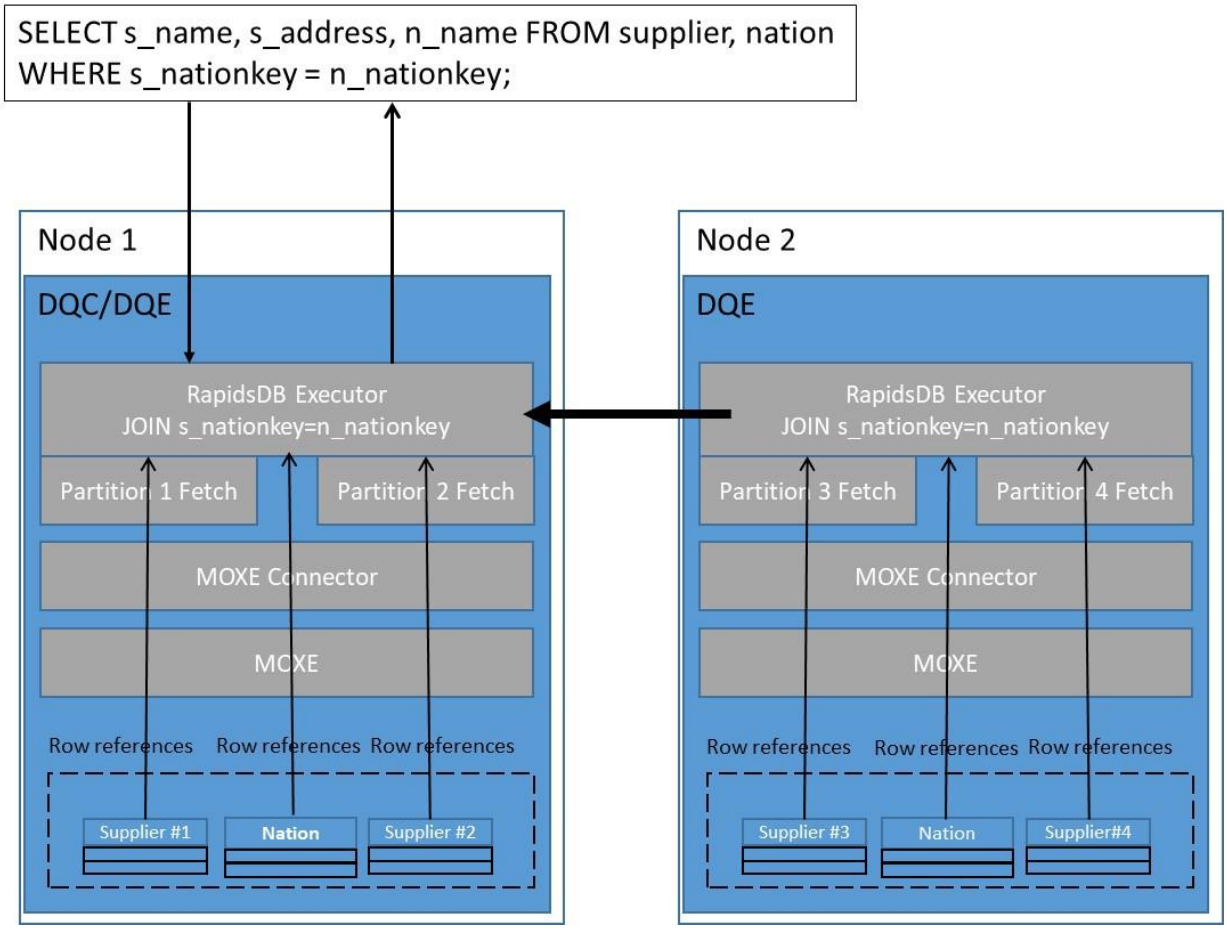
For those parts of a query that are executed by the RapidsDB Execution Engine, there are two types of query plans which can be generated:

1. Partitioned Query Plans (see 4.2)
2. Non-Partitioned Query Plans (see 4.3)

NOTE: It is important to understand that Partitioned and Non-Partitioned Query Plans only apply to those parts of the query plan that cannot be pushed down to the underlying data stores, where the RapidsDB Execution Engine will be executing that part of the query plan. For queries that can be pushed down to the underlying data store, it is the responsibility of the underlying data store to parallelize the query execution where possible.

4.2 Partitioned Query Plans

MOXE and the Hadoop Connector support partitioning of the data across nodes in the RapidsDB cluster, which allows a query to be executed in parallel against each of the partitions of the tables being queried. For MOXE and the Hadoop Connector, RapidsDB will generate a Partitioned Query Plan, where portions of the query plan will be executed in the RapidsDB Execution Engine in parallel against each partition of a table. Figure 1 below illustrates this:



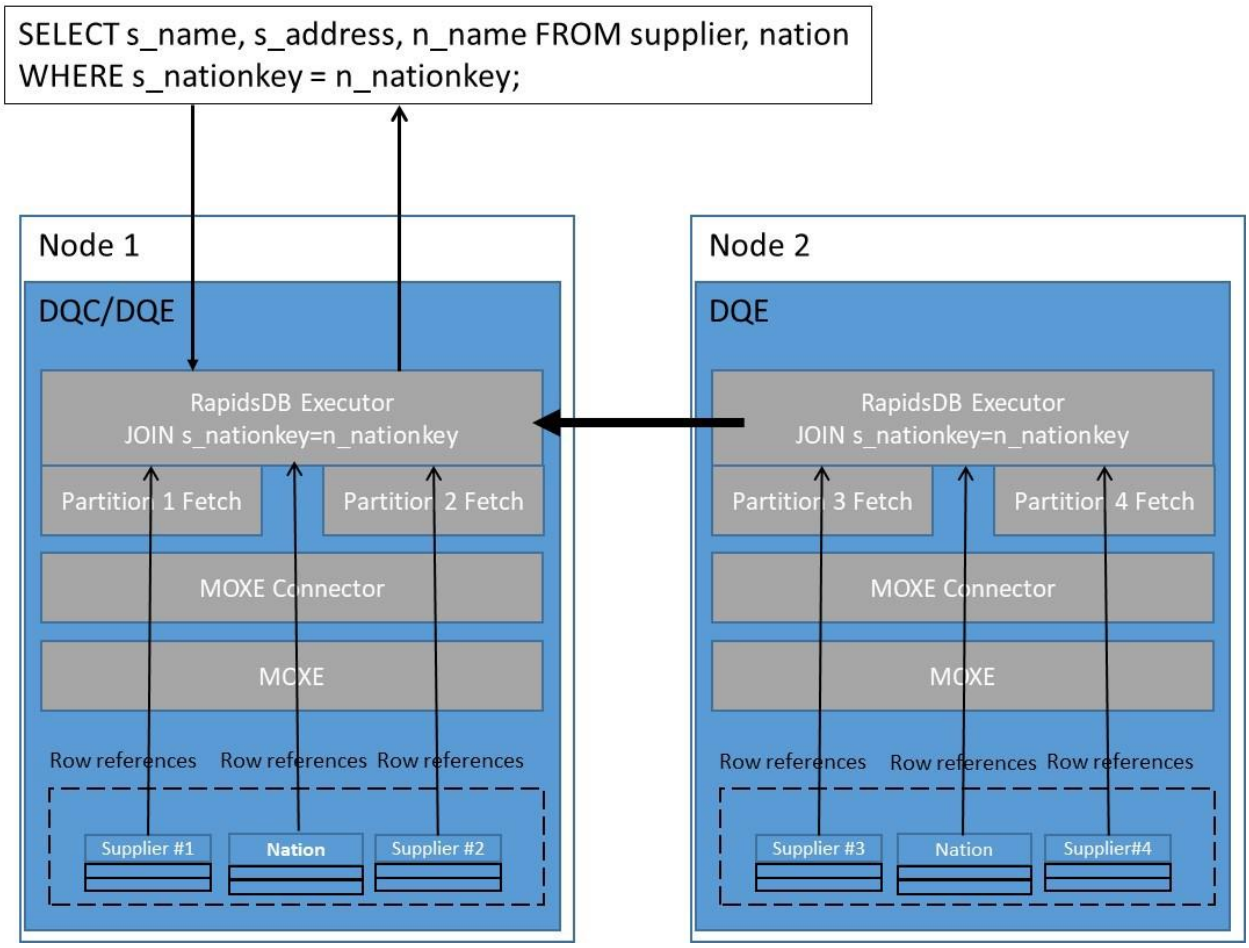


Figure 1. Partitioned Query Plan

In this example there is a join between the Supplier and Nation tables. The Supplier table is a partitioned table that is distributed across two nodes, and the Nation table is a replicated table with a copy of the table on each node. Each node will perform a parallel join between the partitions of the Supplier table and the Nation table, and then the results will be merged on the originating node for the query, which is Node1 in this example.

4.3 Non-Partitioned Query Plans

For other data stores where a single query cannot be split up and executed in parallel, RapidsDB will generate a Non-Partitioned Query plan, sending a single query to the underlying data store. It will be responsibility of the underlying data store to parallelize the execution of the query if possible. MemSQL is an example of a Data Store where the query cannot be split up by RapidsDB and executed in parallel, but MemSQL can the parallelize the execution of a SQL statement when it is pushed down to MemSQL. Figure 2 below illustrates this:

Example: select l_orderkey, l_partkey, p_mfgr from part join lineitem on p_partkey = l_partkey and p_mfgr = 'LG';

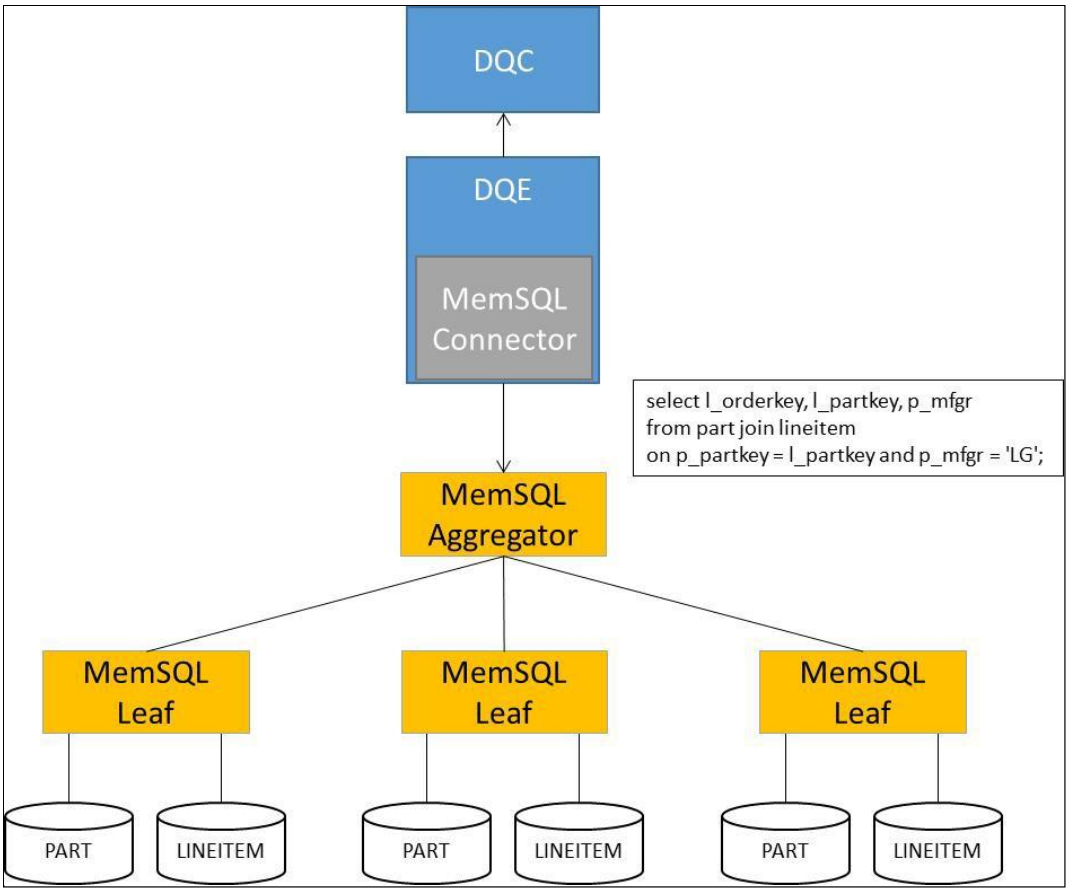


Figure 2. Non-partitioned Query Plan

Figure 2 shows the entire query getting pushed down to MemSQL (via the memSQL Aggregator) and then MemSQL parallelizing the execution across all of the MemSQL Leaf nodes in the MemSQL cluster.

Other data stores, such as Oracle, which are not distributed data stores, will execute the query on a single node. Figure 3 below shows the same query executed against Oracle:

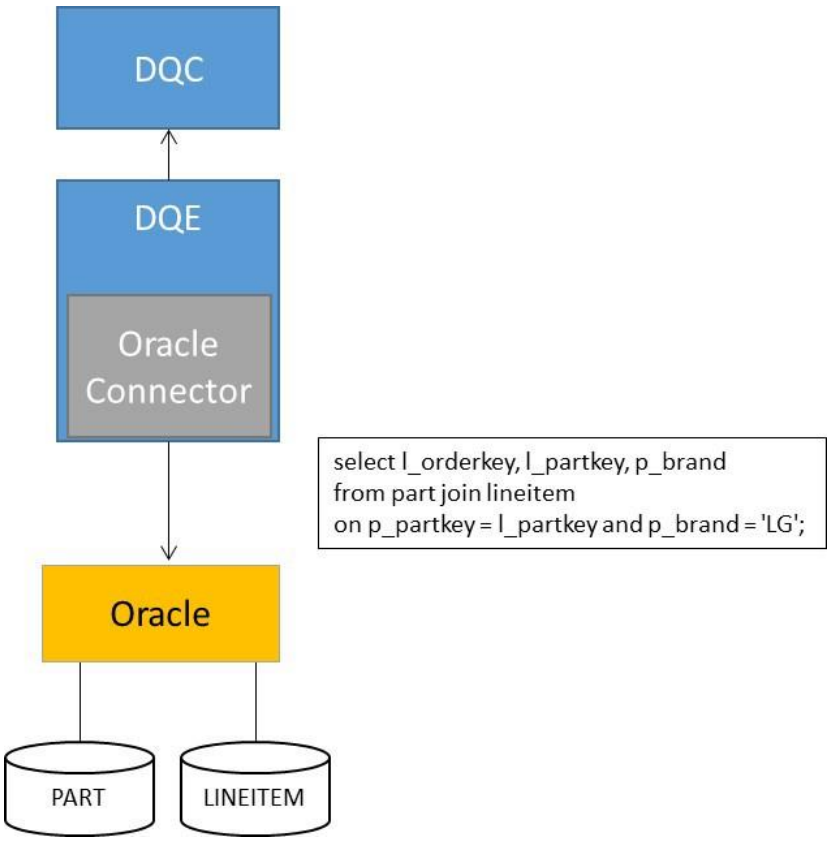


Figure 3 Non-partitioned query plan

4.4 Combination of Partitioned and Non-Partitioned Plans

When joining tables across Connectors where one Connector uses Partitioned plans and the other Connector uses Non-Partitioned plans (eg. MOXE and MemSQL), the join will be executed by RapidsDB, and RapidsDB will push down as much of the processing as possible before performing the join.

Example: select l_orderkey, l_partkey, p_mfgr from part join lineitem on p_partkey = l_partkey and p_mfgr = 'LG' and l_shipdate <= '2020-01-01';

This is the same query as the examples above, but in this example the PART table is managed by MOXE and the LINEITEM table is managed by Oracle.

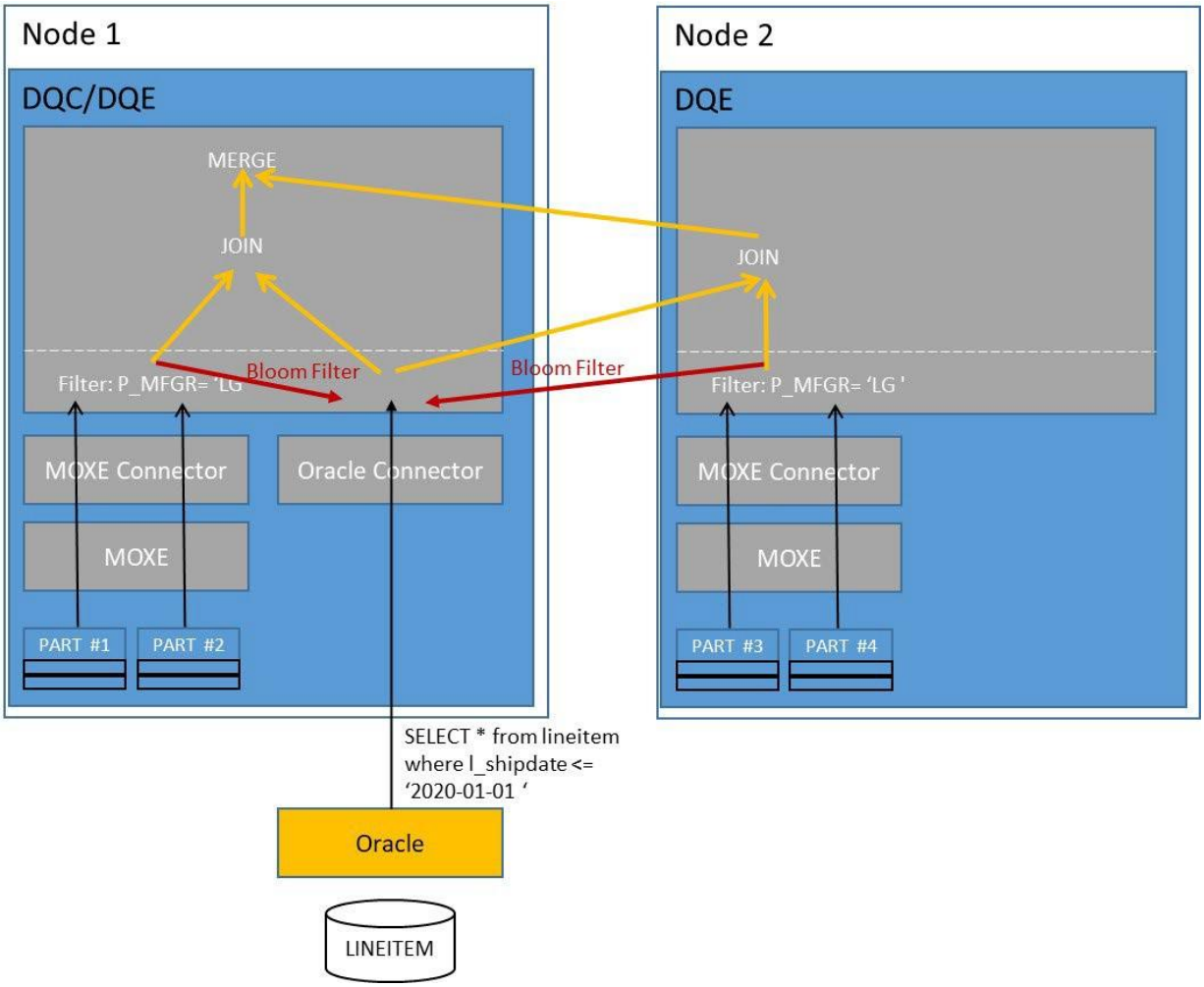


Figure 4. Combination of Partitioned and Non-Partitioned Plans

Figure 4 shows that the data from the PART table will be retrieved from MOXE in parallel using a Partitioned plan, and the data from the LINEITEM table will be retrieved from Oracle using a Non-Partitioned plan. The steps of the plan will be:

1. Execute a partitioned fetch from MOXE against the PART table, and apply a filter to the data (p_mfgr='LG')
2. Build a set of Bloom filters using the data returned from the PART table for each partition
3. Execute a non-partitioned fetch from Oracle against the LINEITEM table with the predicate l_shipdate<='2020-01-01'
4. Send the bloom filters to the node with the LINEITEM data and join the resulting rows with the rows returned from the PART table
5. The DQC will merge the results from the two nodes doing the Bloom joins and then return the results to the user

4.5 RapidsDB Join Algorithms

To reduce network data movement, RapidsDB automatically distributes join operations based on the network location(s) of the data.

The RapidsDB join optimizer analyzes available information about the size and location of the join operands (the tables or subqueries participating in the join) and plans the join operation with the goal of minimizing network cost by executing it co-locally with the data. If one or both operands are distributed across multiple network nodes, RapidsDB partitions the join operation and executes join operators on multiple nodes in parallel.

For equi-joins (i.e. where the join predicate contains at least one equality condition between the two operands) RapidsDB uses a hash join algorithm. The join operator(s) ingest the rows of one operand and build a hash index of the join keys for those rows. The operator then streams the rows of the other operand, looking up each row's join key in the index to determine whether it can satisfy the join predicate.

If the operands of an equi-join operator are not co-located, data must be streamed over the network. In this case a *Bloom filter* is dynamically created and sent to the location(s) from which data will be streamed. A Bloom filter is effectively a much smaller (but also less accurate) hash index that yields a simple yes/no answer as to whether a given row can potentially participate in the join. Candidate rows are skipped if they don't satisfy the filter, eliminating the need to send those rows over the network to be tested against the join predicate. Although a Bloom filter is not perfectly accurate (it will allow a few unqualified rows to pass), it nonetheless reduces network transmission significantly in most cases, resulting in significantly higher join performance.

For non equi-joins RapidsDB performs a distributed cross-join. To reduce network cost, the optimizer tries to plan the query such that the join operator(s) will run co-locally with the larger operand. Rows of the other operand are broadcast (if they are not co-located) and the join operator(s) apply the join predicate to every possible row combination to find combinations that satisfy the predicate. Testing all combinations can be quite time consuming, so joins of this type are not advisable if the operands have a large number of rows.

5 INSERT

The user can insert data into tables in the schema managed by the following Connectors:

- MOXE
- MySQL
- MemSQL
- Oracle
- Postgres
- Greenplum
- Hadoop

- JDBC

The syntax for the INSERT command is:

```
INSERT INTO [catalog.][schema.]<table name> [(col_name,...)]
   VALUES (expr,...),(...),...

INSERT INTO [catalog.][schema.]<table name> [(col_name,...)]
   SELECT select-query
    col_name = insert_expr
      [, col_name = insert_expr] ... ]

 select-query: any valid select query as defined by section 2
```

NOTES:

1. The catalog and schema names are used to identify which Connector the INSERT command should be sent to. The catalog name is only needed in the situation where the schema name is not unique.
2. For an INSERT … SELECT, the data types in the result set from the SELECT must be compatible with the data types for the target insert table.
3. FOR INSERT … SELECT the tables specified in the INSERT and SELECT clauses can be in different schema managed by different Connectors.

Example:

- INSERT INTO test.t1 VALUES (1,'test text', '2015-01-01 00:00:00');

  The INSERT would be sent to the Connector managing the schema named test to insert data into table t1

- INSERT INTO test.t1 VALUES (1,'test text', '2016-01-01 00:00:00'),(2,'text', '2016-02-01 12:00:00');

  The INSERT for two rows would be sent to the Connector managing the schema named test to insert data into table t1

- INSERT INTO mysql.test.t1 (c1,c2) VALUES (1,'test text');

  The INSERT would be sent to the MySQL Connector managing the schema named test to insert data into table t1 for columns c1 and c2, with default values for any other columns in table t1.

- INSERT INTO moxe.t1 SELECT t1, t2, t3 FROM memsql.test.t2;

  The INSERT would be sent to the MOXE Connector.

6 DDL

The user can create and drop tables in the schema managed by the following types of Connectors:

- MOXE
- MySQL
- MemSQL
- Oracle
- Postgres
- Greenplum
- JDBC
- Hadoop (when used with the Hive metastore)

6.1 CREATE TABLE

The syntax for the CREATE TABLE command is:

```
CREATE TABLE [IF NOT EXISTS] <tableReference>
(
  <columnDefinition>,  ...
 <indexDefinition>,  ...
)
[ PARTITION [BY] (<expr>, ...)] [COMMENT <string>]

where:

<tableReference> is:
  [catalog.][schema.]<table name>

<column definition> is:
  <columnName> <type> [[NOT] NULL] [COMMENT <string>]

<type> is:
            INTEGER [(precision)] |
            DECIMAL [(scale[, precision])] |
            FLOAT |
            VARCHAR [(size)] |
            BOOLEAN |
             DATE |
            TIMESTAMP

<column name> is: <SQL identifier>

<indexDefinition>  is:
```



```
INDEX <indexName> [ON]  ( <columnName>, ... )
```

NOTES:

1. The catalog and schema names are used to identify which Connector the CREATE TABLE command should be sent to. The catalog name is only needed in the situation where the schema name is not unique.
2. The column name must be a valid SQL identifier (see 1.1.1). If the column name is a reserved word then it must be enclosed in double quotes, however, the target database may still reject a quoted identifier for some reserved words. For example, Postgres will not accept “select” as a quoted identifier for a column name.
3. After creating the table, the metadata for the associated Connector will be refreshed, and there is no need to manually run the REFRESH command.
4. For the Integer, Float, Decimal, and VARCHAR data types the actual size, precision and scale of the columns will be determined by the underlying data store and can be different from the value specified by the user. The DESCRIBE TABLE command can be used to see the column information, for example:

```
rapids > describe table region;

TABLE_NAME    COLUMN_NAME    DATA_TYPE     ORDINAL    IS_PARTITION_KEY    IS_NULLABLE    PRECISION
SCALE

--------------   ---------------   ------------     ----------   ----------------------   ---------------   ------------
------

REGION        R_REGIONKEY    INTEGER              1               false          false           10
0

REGION        R_NAME         VARCHAR              2               false          false           25
NULL

REGION        R_COMMENT      VARCHAR              3               false          false          152
NULL



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

(refer to the Rapids-shell User Guide for more information on describe table)

The column information in the COLUMNS table in the RapidsDB System Metadata (see 9.9) for the created table reflects the type, size, precision and scale of columns as reported by the underlying data store and interpreted by RapidsDB. The following query can be used to see the column information for the created table:

```
SELECT * FROM rapids.system.columns
```



```
WHERE catalog_name = '<catalog name for new table>' AND
schema_name = '<schema name for new table>' AND
table_name =' <name of new table>';
```

The tables below show how the data types are handled across MOXE, MySQL, MemSQL, Postgres (includes Greenplum) and Oracle when issuing a CREATE TABLE statement from the rapids-shell:

MOXE		
Data Type	**MOXE Data Type**	**Comments**
INTEGER	INTEGER	Precision=19
INTEGER(n)	INTEGER	Precision ignored and set to 19
FLOAT	FLOAT	64-bit double precision
FLOAT(n)	FLOAT	64-bit double precision
DECIMAL	DECIMAL(19,0)	
DECIMAL(p,s)	DECIMAL(p,s)	Maximum precision of 19
BOOLEAN	BOOEAN	
DATE	DATE	
TIMESTAMP	TIMESTAMP	Maximum of 6 digits precision for seconds
VARCHAR	VARCHAR	Defaults to maximum size of 65,536 bytes
VARCHAR(n)	VARCHAR(n)	The maximum size is 65,536 bytes.

MEMSQL		
Data Type	**MemSQL Data Type**	**Comments**
INTEGER	BIGINT	Precision=19
INTEGER(n)	BIGINT	Precision=19
FLOAT	DOUBLE	Precision=22

FLOAT(n)	DOUBLE	Precision=22
DECIMAL	DECIMAL(19,0)	
DECIMAL(p,s)	DECIMAL(p,s)	Max precision is 65, max scale is 30
BOOLEAN	TINYINT(1)	
DATE	DATE	
TIMESTAMP	TIMESTAMP	
VARCHAR	VARCHAR	Defaults to 255 characters
VARCHAR(n)	VARCHAR(n)	Max of 21,845

MYSQL		
Data Type	**MemSQL Data Type**	**Comments**
INTEGER	BIGINT	Precision=19
INTEGER(n)	BIGINT	Precision=19
FLOAT	DOUBLE	Precision=22
FLOAT(n)	DOUBLE	Precision=22
DECIMAL	DECIMAL(19,0)	
DECIMAL(p,s)	DECIMAL(p,s)	Max precision is 65, max scale is 30
BOOLEAN	TINYINT(1)	
DATE	DATE	
TIMESTAMP	TIMESTAMP	
VARCHAR	VARCHAR	Defaults to 255 characters
VARCHAR(n)	VARCHAR(n)	Values in VARCHAR columns are variable-length strings. The length can be specified as a value from 0 to 65,535. The effective maximum length of a VARCHAR is subject to the maximum row size (65,535 bytes, which is shared among all columns)

Oracle		
Data Type	**Oracle Data Type**	**Comments**
INTEGER	INTEGER	Precision=19
INTEGER(n)	INTEGER	Precision ignored and set to 19
FLOAT	FLOAT	64-bit double precision
FLOAT(n)	FLOAT	64-bit double precision
DECIMAL	DECIMAL(19,0)	
DECIMAL(p,s)	DECIMAL(p,s)	Maximum precision of 19
BOOLEAN	BOOEAN	
DATE	DATE	
TIMESTAMP	TIMESTAMP	Maximum of 6 digits precision for seconds
VARCHAR	VARCHAR	Defaults to maximum size of 65,536 bytes
VARCHAR(n)	VARCHAR(n)	The maximum size is 65,536 bytes.

Postgres		
Data Type	**Postgres Data Type**	**Comments**
INTEGER	BIGINT	Precision=19
INTEGER(n)	BIGINT	Precision=19
FLOAT	FLOAT	Precision=53
FLOAT(n)	FLOAT	Precision=53
DECIMAL	DECIMAL(38,12)	

DECIMAL(p,s)	DECIMAL(38,12)	Max precision is 65, max scale is 12. Precision and scale are ignored and will always be set to 38 and 12 respectively
BOOLEAN	Not supported	
DATE	DATE	
TIMESTAMP	TIMESTAMP	Maximum of 6 digits precision for seconds
VARCHAR	VARCHAR	Defaults to maximum size of 65,536 bytes
VARCHAR(n)	VARCHAR(n)	The maximum size is 65,536 bytes.

Examples:

```
CREATE TABLE test.t1 (c1 integer not null, c2 varchar(64), c3 timestamp);
```

This command would be sent to the Connector managing the schema named “test” to create the table “t1”

```
CREATE TABLE mysql.test.t1 (c1 integer not null comment 'first column', c2 varchar(64) comment 'second column', c3 timestamp comment 'third column') comment 'test table';
```

This command would be sent to the Connector named “mysql” that is managing the schema “test” to create the table “t1”, with comments on all of the columns as well as the table.

```
CREATE TABLE test.t1 (“YEAR” integer not null, c2 varchar(64), c3 timestamp);
```

This command would be sent to the Connector managing the schema named “test” to create the table “t1” with the first column being named “YEAR”. This is an example of using a quoted identifier for a column name that is a reserved word.

6.2 Creating MOXE Tables

MOXE supports two types of tables, partitioned tables (see 6.2.1) and reference tables (see 6.2.2) as described below.

6.2.1 Partitioned Tables

A partitioned table is a table where the data is distributed across all of the nodes in the RapidsDB cluster where the associated MOXE Connector is running, and the data is partitioned using the columns specified by the “PARTITION [BY]” clause.

The following example creates a partitioned MOXE table with the column s_suppkey as the partitioning column:

```
rapids > create table moxe.SUPPLIER (
    >   s_suppkey integer NOT NULL comment 'Supplier key',
    >   s_name varchar(25),
    >   s_address varchar(40),
    >   s_nationkey integer,
    >   s_phone varchar(15),
    >   s_acctbal decimal(17,2),
    >   s_comment varchar(101)
    > ) PARTITION (s_suppkey) comment 'Supplier table';
0 row(s) returned (0.15 sec)
```

This table also has a comment on the column s_suppkey and a table level comment. The comments can be seen by querying the RapidsDB COMMENTS and TABLES tables as shown below:

```
rapids > select * from tables where table_name='SUPPLIER';
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME      IS_PARTITIONED COMMENT
PROPERTIES
-----------------   ---------------   --------------     -------------------- ----------
--------------
MOXE            MOXE           SUPPLIER                  true Supplier
table    NULL

1 row(s) returned (0.07 sec)
rapids > select * from columns where table_name='SUPPLIER';
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME    COLUMN_NAME    DATA_TYPE
ORDINAL    IS_PARTITION_KEY    IS_NULLABLE    PRECISION    PRECISION_RADIX
SCALE CHARACTER_SET    COLLATION    COMMENT         PROPERTIES
-----------------   ---------------   --------------   ---------------   -------------
----------   ----------------------   ---------------   -------------   ----------------------
-------- ------------------   -------------   ----------         --------------
MOXE            MOXE           SUPPLIER      S_SUPPKEY      INTEGER
0                true          false           64                   2
NULL NULL             NULL         Supplier key    NULL
MOXE            MOXE           SUPPLIER      S_NAME         VARCHAR
1               false           true         NULL               NULL
NULL UTF16      BINARY          NULL               NULL
MOXE            MOXE           SUPPLIER      S_ADDRESS      VARCHAR
2               false           true         NULL               NULL
NULL UTF16      BINARY          NULL               NULL
```



```
MOXE            MOXE            SUPPLIER      S_NATIONKEY    INTEGER
3               false            true           64                  2
NULL NULL             NULL          NULL            NULL
MOXE            MOXE            SUPPLIER      S_PHONE        VARCHAR
4               false            true         NULL               NULL
NULL UTF16            BINARY        NULL            NULL
MOXE            MOXE            SUPPLIER      S_ACCTBAL      DECIMAL
5               false            true           17                 10
2 NULL              NULL          NULL             NULL
MOXE            MOXE            SUPPLIER      S_COMMENT      VARCHAR
6               false            true         NULL               NULL
NULL UTF16            BINARY        NULL            NULL

7 row(s) returned (0.08 sec)
```

6.2.2 Reference Tables

Reference tables are tables that are replicated to each node in the RapidsDB cluster where the associated MOXE Connector is running. 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.

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)
```

6.3 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, CREATE TABLE [AS] SELECT is a quick and easy way to take a copy of a result set and save it in a separate table.

The column names will default to the column names from the associated columns in the SELECT query, but the names can also be specified explicitly. If the SELECT query is providing literal values for the columns, then the column names will be “col1”, “col2”, etc.

The data types for the columns in the newly created table will default to the data types from the associated columns in the SELECT query. The user can also specify the data types to be used and in this case if the data types of the columns from the SELECT query do not match those specified for the table then the columns of the SELECT query will be cast to match. If this results in an incompatible data type cast then an error will be returned.

The addition of the column data types as well as the AS clause is a RapidsDB extension to the SQL standard.

Syntax:

```
statement := CREATE TABLE [ IF NOT EXISTS ] <tableName>

[ ( <tableDefinition> ) ]

[ <partitionInformation> ]1

[ <tableProperties> ] 2

[AS] <subquery> [ WITH [NO] DATA ];

tableDefinition := <objectDefinition> [ , <objectDefinition> [, ...] ]

objectDefinition:= <columnDefinition> | <tableConstraint> | <indexDefinition>

columnDefinition := <columnName> [ <columnType> [ <columnConstraint> ] ]

columnConstraint := NOT NULL | PRIMARY KEY | UNIQUE KEY

tableConstraint := PRIMARY KEY ( <expression> )

indexDefinition := UNIQUE KEY ( <expression> ) | KEY ( <expression> )

subquery := <selectOrValuesQuery> | ( <selectOrValuesQuery> )

selectOrValuesQuery:= <selectQuery> | <valuesQuery>

selectQuery := SELECT <selectQueryExpression>

valuesQuery := VALUES ( <expression> [, <expression [, ...] ] ) [, ( ... ) ]
```

6.3.1 Examples


```
rapids > describe table t1;
TABLE_NAME   COLUMN_NAME   DATA_TYPE     ORDINAL   IS_PARTITION_KEY   IS_NULLABLE   PRECISION   SCALE
----------   -----------   ---------     -------   ----------------   -----------   ---------   -----
T1           NUM           INTEGER             0              false          true          64    NULL
T1           NAME          VARCHAR             1              false          true        NULL    NULL

2 row(s) returned (0.24 sec)
rapids > create table moxe1.t6  select * from t1;
0 row(s) returned (0.20 sec)
rapids > describe table t6;
TABLE_NAME   COLUMN_NAME   DATA_TYPE     ORDINAL   IS_PARTITION_KEY   IS_NULLABLE   PRECISION   SCALE
----------   -----------   ---------     -------   ----------------   -----------   ---------   -----
T6           NUM           INTEGER             0              false          true          64    NULL
T6           NAME          VARCHAR             1              false          true        NULL    NULL

2 row(s) returned (0.26 sec)
rapids > select * from t6;
  NUM NAME
  --- ----
    1 a
    2 b
    3 c

3 row(s) returned (0.05 sec)
```

Creates a table t with columns and data from t1. This statement is compliant with the SQL standard.

```
rapids > create table moxe1.t7  select * from t1 WITH NO DATA;
0 row(s) returned (0.09 sec)
rapids > describe table t7;
TABLE_NAME   COLUMN_NAME   DATA_TYPE     ORDINAL   IS_PARTITION_KEY   IS_NULLABLE   PRECISION   SCALE
----------   -----------   ---------     -------   ----------------   -----------   ---------   -----
T7           NUM           INTEGER             0              false          true          64    NULL
T7           NAME          VARCHAR             1              false          true        NULL    NULL

2 row(s) returned (0.23 sec)
rapids > select * from t7;
0 row(s) returned (0.05 sec)
```

Creates a table t7 which is a copy of the table t1 but without any data . This statement is compliant with the SQL standard.

```
rapids > create table moxe1.t8 as select name from t1;
0 row(s) returned (0.20 sec)
rapids > describe table t8;
TABLE_NAME   COLUMN_NAME   DATA_TYPE     ORDINAL   IS_PARTITION_KEY   IS_NULLABLE   PRECISION   SCALE
----------   -----------   ---------     -------   ----------------   -----------   ---------   -----
T8           NAME          VARCHAR             0              false          true        NULL    NULL

1 row(s) returned (0.20 sec)
rapids > select * from t8;
NAME
----
a
b
c

3 row(s) returned (0.05 sec)
```

Creates a table t8 with column name from t1.


```
rapids > create table moxe1.t1 as values (1,'abc',12.1,1.0e0);
0 row(s) returned (0.20 sec)
rapids > describe table moxe1.t1;
TABLE_NAME   COLUMN_NAME   DATA_TYPE     ORDINAL   IS_PARTITION_KEY   IS_NULLABLE   PRECISION   SCALE
----------   -----------   ---------     -------   ----------------   -----------   ---------   -----
T1           COL1          INTEGER             0              false         false          64    NULL
T1           COL2          VARCHAR             1              false         false        NULL    NULL
T1           COL3          DECIMAL             2              false         false           3       1
T1           COL4          FLOAT               3              false         false          53    NULL

4 row(s) returned (0.20 sec)
rapids > select * from t1;
  COL1 COL2     COL3   COL4
  ---- ----     ----   ----
     1 abc      12.1    1.0

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

Creates a table t1 with automatically named columns (“col1”, “col2”, “col3” and “col4”) 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 according to the SQL standard (e.g., 12.1 is a decimal while 1.0e0 is a float).

```
rapids > create table moxe1.t2(id, name, price, disc) as select * from t1;
0 row(s) returned (0.19 sec)
rapids > select * from t2;
  ID NAME     PRICE   DISC
  -- ----     -----   ----
   1 abc       12.1    1.0

1 row(s) returned (0.05 sec)
```

Creates a table t with columns named “id”, “name”, “price” and “disc” and filled with data from columns a, b and c from table u. Apart from the AS clause this statement is compliant with the SQL standard.

```
rapids > describe table t1;
TABLE_NAME   COLUMN_NAME   DATA_TYPE     ORDINAL   IS_PARTITION_KEY   IS_NULLABLE   PRECISION   SCALE
----------   -----------   ---------     -------   ----------------   -----------   ---------   -----
T1           COL1          INTEGER             0              false         false          64    NULL
T1           COL2          VARCHAR             1              false         false        NULL    NULL
T1           COL3          DECIMAL             2              false         false           3       1
T1           COL4          FLOAT               3              false         false          53    NULL

4 row(s) returned (0.25 sec)
rapids >  create table moxe1.t3(id integer, name varchar, price decimal(15,2), disc date) as select * from t1;
Error: Line 1 position 1: Type mismatch for column DISC: requires DATE using `com.rapidsdata.stdlib.FastDate`
and there is no conversion from FLOAT(53 BITS) using `com.rapidsdata.stdlib.FastFloat`. locus=BORAY01
```

Attempts to create a table t3 where the column data type for the column “disc” is not compatible with the data type for the fourth column of the source table, “col4” and so an error is returned.

6.3.2 Semantics

1. In keeping with the regular CREATE TABLE statement, the catalog and schema names of the target table name will determine which data source will ultimately hold the table and data for this query.

2. If a table with the same name as the target table already exists in RapidsDB and IF NOT EXISTS is not specified, then the query will fail and no data from the subquery will be copied into the target table.
3. If a table with the same name as the target table already exists in RapidsDB, and IF NOT EXISTS is specified, then the query will return a success indicator to the user but no data from the subquery will be copied into the target table.
4. Specifying a <tableDefinition> allows the user to rename columns from the subquery. This end result can also be achieved by applying column aliases to the subquery instead. If the <tableDefinition> is not specified then the column names from the subquery will be used instead.
5. If the <tableDefinition> contains any duplicate column names then an error will be reported.
6. If a <tableDefinition> is specified and if the number of column names in <columnList> is not equal to the number of columns in the subquery then an error will be returned.
7. If a <tableDefinition> is specified with data types and the data types are incompatible with the column types of the SELECT statement then an error will be returned. An example of this would be specifying a column with a timestamp data type when the corresponding column in the SELECT query returns a boolean. The rules surrounding what data types can be cast are determined by the RapidsDB CAST operator.
8. When the target table is created there will be no indexes created on it unless a <tableDefinition> is provided and it contains index definitions. Creating a target table based on a query on a source table will not result in indexes from the source table being copied to the target table unless they are explicitly specified in a <tableDefinition> clause.
9. If a <tableDefinition> clause is not specified then the precision and scale of each column will be set according to the table below:

Datatype	**Precision**	**Scale**
Integer	19	0
Decimal	Will be preserved from result column up to a maximum of 19. If >19 or if the precision is unknown then it will be set to 19.	Will be preserved from the result column up to a maximum of 7. If the scale is >7 or the scale is unknown then it will be set to 7.
Float	Will not be specified. All floating point columns will be created with a datatype of FLOAT.	No scale. Scale has no meaning for an approximate data type.
Varchar	Will be preserved from the result column. If the result column has no precision then it will be set to 8000.	No scale.

All other types	No precision.	No scale.

10. The nullability properties of columns in the SELECT query will be preserved in the target table being created. However no uniqueness constraints will be preserved as this implies automatic index creation.
11. If the WITH NO DATA clause is specified then the target table will be created according to the column definitions of the subquery however no data will be copied into the target table from the subquery.
12. If it is possible for RapidsDB to do so, the target table being created will be dropped if an error occurs while copying data into the table. Because RapidsDB is not transactional, there will be error scenarios where the query may fail and it is not possible for RapidsDB to drop the incomplete table automatically (e.g., if there is a problem with the connector, internal RapidsDB errors, etc).
13. By specifying table and column constraints in the CREATE TABLE AS SELECT statement (e.g., CREATE TABLE t (aa INTEGER PRIMARY KEY) AS SELECT a FROM u;), if the SELECT query retrieves a result set that does not match the column constraint (e.g., the values of column a in the above query are not unique) then the query will fail while copying the data. For a large data set, it could take a while before this constraint violation is detected and the failure status returned to the user. Examples of this would include violations of column uniqueness or nullability (e.g., CREATE TABLE t (col1 INTEGER NOT NULL) AS VALUES (NULL);)

6.3.3 Exclusions

1. The CREATE TABLE AS SELECT statement will not be executed transactionally since RapidsDB has no support for transactions. This means that it is possible for the table to be created successfully but an error occurs while copying the data such that the statement fails and the table is not able to be cleaned up (e.g., a communication problem with the underlying data source).
2. In this release CREATE TABLE AS SELECT statements will not support being pushed down directly to the underlying data source if the entire statement occurs directly in that data source. This is because the syntax of CREATE TABLE AS SELECT statements can vary significantly across data sources (and RapidsDB only supports a common subset that is defined in the standard.
3. RapidsDB will not support SELECT…INTO statements as a synonym of CREATE TABLE AS SELECT.
4. If a value from the subquery exceeds the precision or scale of the table definition then the underlying storage engine may return an error or it may silently truncate/round the data value when it is being inserted into the table.

6.3.4 Error Conditions

The following are a common set of conditions that will cause RapidsDB to generate an error:

1. Specifying a column name list where the number of column names does not match the number of columns in the subquery.

2. Specifying a column name list or table definition where a column name is not unique.
3. Specifying a full table definition but the data type of a column is not compatible with the data type of the corresponding column in the subquery and the subquery value cannot be cast.
4. Specifying a VALUES subquery where all values for a given column are NULL. In this case the data type of the column for the CREATE TABLE statement cannot be determined.
5. Specifying a VALUES subquery with multiple rows where the data type for a given column is not consistent across all rows.
6. Specifying a table definition with a column constraint (e.g., NOT NULL) where the subquery data does not conform to that constraint (e.g. contains NULL values).
7. Specifying a table definition with a table constraint (e.g., PRIMARY KEY) where the subquery data does not conform to that constraint (e.g., non-unique values across PK columns).
8. Specifying a column name and data type for some columns but not specifying a data type for all columns.

6.4 CREATE INDEX

The syntax for the CREATE INDEX command is:

```
CREATE [UNIQUE] INDEX [IF NOT EXISTS] <indexName> ON <tableReference> ( <columnName>, ... )

where:

<tableReference>  is:
  [[<catalog>.] [<schema>.]] tableName
```

NOTE:
MOXE does not support creating indexes.

Example:

```
CREATE UNIQUE INDEX idx1 on memsql.dw.t1 (c1);
```

This command would be sent to the MemSQL Connector that is managing the schema “dw” to create a unique index on table “t1”.

6.5 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.

2. After dropping the table, the metadata for the associated Connector will be refreshed, and there is no need to manually run the REFRESH command.

Examples:

```
DROP TABLE test.t1;
```

This command would be sent to the Connector managing the schema named “test” to drop the table “t1”

```
DROP TABLE memsql.test.t1;
```

This command would be sent to the MemSQL Connector that is managing the schema “test” to drop the table “t1”

6.6 TRUNCATE TABLE

The TRUNCATE TABLE deletes all of the data from a table and any associated indexes.

The syntax for the TRUNCATE TABLE command is:

```
TRUNCATE  TABLE  [[<catalog>.]<schema>.]<table  name>;
```

NOTES:

1. The catalog and schema names are only needed when the <table name> is not unique.

Examples:

```
TRUNCATE TABLE rapidsse.public.t1
```

This command would be sent to the RapidsSE Connector to delete all of the data from table t1.

```
TRUNCATE TABLE memsql.test.t1;
```

This command would be sent to the MemSQL Connector that is managing the schema “test” to delete the data from table t1.

7 IMPORT/EXPORT Using IMPEX Connector

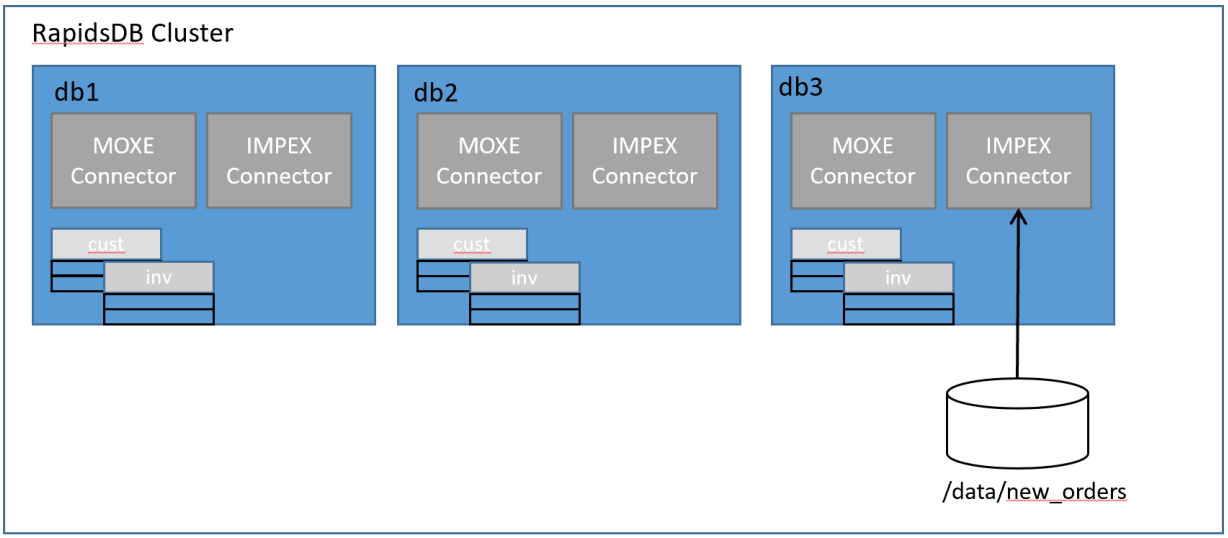
7.1 Overview

The IMPEX Connector is a new style of Connector that was introduced in Release 4.3. An IMPEX Connector supports the ability to treat disk files as regular tables which can participate in federated queries (ie. in SELECT or INSERT queries). The implication of this is that the user does not need to go through an ETL process in order to load the data from the files into regular tables, such as MOXE tables,

instead, the files can be queried directly from the disk. For Release 4.3, an IMPEX Connector can read csv (delimited) files from any node in the RapidsDB Cluster, in future releases other file systems such as Amazon S3, Google Cloud and HDFS will be supported along with other file formats such as Parquet and ORC. After any data has been written to disk (in a supported format, ie csv for Release 4.3) it is available for querying. If needed, the user can also use an IMPEX Connector to load all or a subset of the data into regular tables, such as MOXE tables or other federated data sources such as Oracle, Postgres or MySQL. When reading the data from disk, an IMPEX Connector supports both column pruning and predicate pushdown so that only the data that is needed for the query is passed to the RapidsDB Execution Engine thereby allowing very large data files to be processed by the RapidsDB Engine where the size of the data can exceed the memory of the system. When reading the disk files the user does not need to define a schema for the table, an IMPEX Connector can estimate the data type for each field in the data by reading a sample of the data and the imputing the data type based on the actual data. This means that users can do fast exploration of data files without having to first assign a schema for the table. For example, by using a LIMIT clause the user can quickly look at a subset of the data and then can use other SQL predicates to do more sophisticated analysis of the data. If the schema for a file (or set of files) is known, then the user can provide that schema to the IMPEX Connector as part of the query.

An IMPEX Connector also supports the capability to write query results to files. Finally, an IMPEX Connector supports bulk import to allow for the rapid loading of data from disk files into any federated tables, and bulk export to allow for the rapid writing of the contents of any federated tables to disk files. Bulk EXPORT provides the ability for the user to take a snapshot of the federated database, and bulk IMPORT provides the ability to reload that snapshot.

Example:



SELECT … FROM (FOLDER 'node://db3/data/new_orders'), MOXE.CUST, MOXE.INV ….;

In the example above the IMPEX Connector is reading data from the folder "/data/new_orders" on RapidsDB node "db3" and that data is then getting joined with data from two MOXE tables, "cust" and "inv".

7.2 IMPEX Connector Type

The IMPEX Connector type is used for creating Connectors that are used for doing import and export operations. The following sections provide more details on how to configure and use IMPEX Connectors.

7.3 Creating an IMPEX Connector

The user can create import and export Connectors using the IMPEX Connector type. To create an IMPEX Connector use the following command

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

where <key> is one of the supported IMPEX Connector properties as defined in the next section.

Example:

```
CREATE CONNECTOR CSV TYPE IMPEX WITH DELIMITER='|', PATH='/';
```

Would create an IMPEX Connector named "CSV" that can run on any node in the RapidsDB cluster and where the delimiter character is '|', and the base path is the root directory ('/'). All other IMPEX properties would use default values as described below (see 11.4).

```
CREATE CONNECTOR CSV TYPE IMPEX WITH DELIMITER='|', PATH='/' NODE 'db1';
```

This would create the same Connector as the previous example with the one difference being that this Connector could only run on the RapidsDB node named "db1".

7.4 IMPEX Connector Properties

The IMPEX Connector type supports the following properties which can be set either when creating the Connector using the CREATE CONNECTOR command (see examples below, also, refer to the Installation and Management Guide for more information on creating Connectors) or as part of an import reference (see 7.6) or export reference (see 7.7):

Key:	Default	Syntax	Description
FORMAT	'CSV'	'CSV' | 'RAW'	Specifies the file format: • CSV: A delimited file (see section 11.5) • RAW: will produce a table with a single VARCHAR column containing the full text of each record in the imported file. See section 11.9.2.9 for examples)
PATH	'/var/tmp/rapids'	'<fully qualified path>'	Specifies the fully qualified path name to use as the base path name for all import references (see 11.6) or export references (see 11.7).
ERROR_PATH	'/var/tmp/rapids_errors'	'<fully qualified path>'	Specifies the fully qualified path name to use as the base path for the error files generated if an import operation fails (see 11.13.1 for more information).
ERROR_LIMIT	10	Integer, -1 | 0 | >0	Specifies the maximum number of allowable errors on an import operation. Once the limit is reached the import will be terminated. The possible values are: -1 no limit 0 terminate on first error >0 terminate after specified number of errors See 11.13.2 for more information

BACKUP	false	[] | true | false	For EXPORT only. For bulk export operations (see 11.12), when the REPLACE option is specified, if BACKUP is “false”, then any existing files with a suffix of “.csv” in the specified folder or sub-folders prior to the export operation will get deleted and then new files created for the export. For bulk export operations (see 11.12), when the REPLACE option is specified, if BACKUP is “true”, then any existing files with a suffix of “.csv” in the specified folders or sub-folders prior to the export operation will be moved to a backup folder so that they can be recovered if needed and then new files created for the export. Note: if “true” or “false” are omitted and just the keyword “BACKUP” is specified, that is equivalent to “true”.

CHARSET	‘UTF-8’	‘<string>’ as defined by the Java charset class https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html	Specifies the character set to be used. Some examples: ‘GBK’ ‘GB2312’ ‘GB18030’ ‘Big5’
DELIMITER	‘,’	‘<char>’ Non-empty, single character string	Specifies the field delimiter character. This can only be a single character.

ENCLOSED_BY	‘”’ double quote	‘<char>’ Non-empty, single character string	Specifies whether a field is optionally enclosed by a specified character. This is 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 ESCAPE_CHAR below 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 delimiters and enclosed_by characters in the data .
FILTER	‘*.*’	‘<string>’ Non-empty, character string using a REGEX format	For IMPORT only. The FILTER property allows the user to control which files are imported in a wildcard import operation and, optionally, how table names are created from the names of imported files. The FILTER value is a character string containing a Java regular expression (a “regex”). When performing a wildcard import, IMPEX examines each filename available from the import source. Only files whose names satisfy the FILTER regex are imported. (For a tutorial on Java regular expressions, see

			https://www.oracle.com/technical-resources/articles/java/regex.html) A "capturing group" can be used in the regex to control how IMPEX creates a table name from the name of an imported file. The characters matched by the first group in the regex are used as the table name. If the regex contains no groups, then the table name will match the first part of the file name (before any dot suffixes). Note: for convenience, a FILTER value that starts with an asterisk is interpreted as a simple filename filter. For example FILTER='*.csv' will import all files with a ".csv" extension.
GUESS	false	[] | true | false	For IMPORT only When "false" specifies that the Connector should treat all columns as polymorphic strings, which will be automatically cast into the appropriate data types depending on the query (see 11.9.2.5 for more information). When "true" specifies that the Connector should derive the column data types for any columns whose data type has not been specified. The data types are derived by sampling the data being imported and then determining what the appropriate data type would be for each input field in the sampled data. For example, if the sampled data contained 100 records, and a given field contains alphanumeric characters for all 100 records, then it would be mapped to a VARCHAR column, if the field contained just integer characters then it would be mapped to an

			INTEGER, and so on (see 11.9.2.5 for more information). Note: if “true” or “false” are omitted and just the keyword “GUESS” is specified, that is equivalent to “true”.
HEADER	false	[] | true | false	When “true” specifies that the data file has a header record which has the column names to use on an import, or has the column names from the result set for an export. When “false” specifies that there is no header record. Note: if “true” or “false” are omitted and just the keyword “HEADER” is specified, that is equivalent to “true”.
TERMINATOR	'\n'	'\n'	Specifies how records are terminated. For this release the TERMINATOR is fixed as '\n', with an optional '\r'
TRAILING	false	[] | true | false	When “true” IMPEX will ignore a trailing field separator (i.e. where the field separator is immediately followed by the record terminator character) on each line of a file being imported and will append a trailing separator to each line of a file being exported. When “false” a trailing field separator will indicate a null value for the last column of the record being imported. For export no trailing field separator will be written out. Note: if “true” or “false” are omitted and just the keyword “TRAILING” is specified, that is equivalent to “true”.

Examples:


```
CREATE CONNECTOR CSV TYPE IMPEX WITH DELIMITER='|';
```

Would create an IMPEX Connector named “CSV” where the delimiter character is '|'. The “PATH” property was not set and so would default to “/var/tmp/rapids”.

```
CREATE CONNECTOR CSV TYPE IMPEX WITH DELIMITER='|', PATH='/';
```

Would create an IMPEX Connector named “CSV” where the delimiter character is '|' and the “PATH” property is set to the root directory (“/”).

7.5 CSV (Delimited) File Formatting

This section describes how IMPEX Connectors handle the different CSV file formatting properties described in the previous section when reading and writing delimited data.

7.5.1 Text Handling

7.5.1.1 ESCAPE SEQUENCES

There are a set of special characters that only come into effect when prefixed with the escape character (by default the escape character is set to the backslash character). In the following table, the ESCAPE_CHAR is set to the backslash character. When an escape sequence is detected in the input data it will get replaced with its associated ASCII character as shown in the table below:

Escape Sequence	ASCII Character
\b	A backspace character <x08>
\f	A form feed character <x0C>
\n	A newline (linefeed) character <x0A>
\r	A carriage return character <x0D>
\t	A tab character <x09>
\v	A vertical tab character <x0B>

On output, any ASCII escape characters will get replaced by their associated escape sequence.

Example 1:

This example shows the output for a file using the tab (\t) and newline (\n) escape characters.

Input file: /var/tmp/rapids/tab_and_newline.csv:

```
12345678901234567890123456789O
\tTabbed field\nNewline
```



```
rapids > select * from ('node://db1/text/tab_and_newline.csv');
COL1
----
12345678901234567890123456789O
        Tabbed field
Newline

2 row(s) returned (0.06 sec)
```

Example 2:

This example shows all of the possible escape sequences (using the default escape character) being read from the file "/var/tmp/rapids/text/escape_seq.csv" and then written out to the file "/var/tmp/rapids/text/escape_seq_out.csv".

Input file /var/tmp/rapids/text/escape_seq.csv:

```
Tab: \t Form Feed: \f Backspace: \b Newline: \n Vertical: \v Return: \r
```

```
rapids > select * from ('node://db1/text/escape_seq.csv') to
('node://db1/text/escape_seq_out.csv');

0 row(s) returned (0.08 sec)
```

Output file: /var/tmp/rapids/text/escape_seq_out.csv:

```
"Tab: \t Form Feed: \f Backspace: \b Newline: \n Vertical: \v Return: \r"
```

7.5.1.2 Handling of Leading and Trailing Blanks

Leading and trailing space characters are considered part of a VARCHAR column.

When the ENCLOSED_BY (see 7.5.6) is used to enclose the string, the leading and trailing space characters are ONLY those characters contained within the enclosed string (see example below for more on this), any space characters outside of the enclosing characters are ignored. When the string is not enclosed by the ENCLOSED_BY character, then all characters in the field are included, including all leading and trailing blanks.

Example:

In this example the first record has two fields that are not enclosed by the ENCLOSED_BY character, and so all of the data between the field delimiters for those fields is included. In the third record, the second and third fields are enclosed, and so only the characters between the ENCLOSED_BY characters are included.

File: /var/tmp/rapids/text/lead_trail_blanks.csv:


```
1,  4 leading blanks,3 trailing blanks  ,1
2,A2345678901234567890,A1234567890123456789,2
3,"   4 leading blanks","3 trailing blanks   ",3
```

```
rapids > select * from ('node://db1/text/lead_trail_blanks.csv');
  COL1 COL2                  COL3                      COL4
  ---- ----                  ----                      ----
     1     4 leading blanks   3 trailing blanks           1
     2 A2345678901234567890   A1234567890123456789        2
     3     4 leading blanks   3 trailing blanks           3

3 row(s) returned (0.07 sec)
rapids > select char_length(col3) from
('node://db1/text/lead_trail_blanks.csv');
  [1]
  ---
   20
   20
   20

3 row(s) returned (0.06 sec)
```

7.5.1.3 Empty Strings

An empty (zero-length) string is defined as a field with two adjacent ENCLOSED_BY characters (see 7.5.6) for more information on ENCLOSED_BY character). 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 double quote character:

Example:

File: /var/tmp/rapids/text/empty_string.csv:

```
1,"",1
```

```
rapids > select * from ('node://db1/text/empty_string.csv');
  COL1 COL2     COL3
  ---- ----     ----
     1             1

1 row(s) returned (0.05 sec)
rapids > select char_length(COL2) from ('node://db1/text/empty_string.csv');
  [1]
  ---
    0

1 row(s) returned (0.05 sec)
```


NOTE – this is different from an empty field, where there are two adjacent field delimiter characters, which is interpreted as a NULL value (see 7.5.4) for more information on nulls) as shown in the example below:

File: /var/tmp/rapids/text/null_string.csv:

```
1,,,,
```

```
rapids > select * from ('node://db1/text/null_string.csv');
  COL1 COL2   COL3   COL4
  ---- ----   ----   ----
     1 NULL   NULL   NULL

1 row(s) returned (0.06 sec)
```

In this example, fields two through four are all interpreted as null values.

7.5.2 Dates and Timestamps

The format for dates is YYYY-MM-DD, and the format for the time portion of a timestamp is HH.MM.SS.nnnnnn

Example:

File: /var/tmp/rapids/text/date_and_timestamp.csv

```
1,2021-09-01,2021-09-01  11:17:23.123456
2,"2021-09-01","2021-09-01  11:17:23.123456"
```

```
rapids > select * from (FILE 'node://db1/text/date_and_timestamp.csv') AS
t(c1 integer, c2 date, c3 timestamp);
  C1 C2           C3
  -- --           --
   1 2021-09-01   2021-09-01 11:17:23.123456
   2 2021-09-01   2021-09-01 11:17:23.123456

2 row(s) returned (0.59 sec)
```

7.5.3 Booleans

The table below specifies the valid input values for booleans:

Column value	Possible Inputs
FALSE	0 any string start with one of the following characters: f, F, n, N
TRUE	>0 any string start with one of the following characters: t, T, y, Y

Example:

File: /var/tmp/rapids/text/booleans.csv:

```
0
false
FALSE
n
no
1
true
TRUE
y
YES
```

```
rapids > select * from ('node://db1/text/booleans.csv') AS t(c1 boolean);
     C1
     --
  false
  false
  false
  false
  false
   true
   true
   true
   true
   true

10 row(s) returned (0.06 sec)
```

7.5.4 NULL Values

A null value is represented by an empty field, where an empty field is defined as two adjacent delimiters with no intervening spaces, or a delimiter followed immediately by the record terminator.

NOTE

This is not the same as an empty string (see 11.5.1.3) which is defined as two adjacent “ENCLOSED_BY” characters.

Example:

File: /var/tmp/rapids/text/null_string.csv:

```
1,,,,
```

```
rapids > select * from ('node://db1/text/null_string.csv') AS t(c1 integer,
c2 integer, c3 decimal, c4 varchar);
  C1      C2      C3 C4
  --      --      -- --
   1    NULL    NULL NULL

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

In this example, fields two through four are all interpreted as null values.

7.5.5 DELIMITER='<char> | \t'

The field delimiter can be a single character or the tab character ('\t').

Example 1: using the pipe character as the delimiter

File: /var/tmp/rapids/text/delimiter.csv

```
1|   4 leading blanks|3 trailing blanks   |1
2|12345678901234567890|12345678901234567890|2
3|"   4 leading blanks"|"3 trailing blanks   "|3
```

```
rapids > select * from ('node://db1/text/delimiter.csv' WI TH DELIMITER='|');
COL1    COL2                    COL3                    COL4
----    ----                    ----                    ----
1           4 leading blanks    3 trailing blanks       1
2       12345678901234567890    12345678901234567890    2
3           4 leading blanks    3 trailing blanks       3

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

Example 2: using the tab character as the delimiter:

File: /var/tmp/rapids/text/tab_delimiter.csv

```
1       4 leading blanks    3 trailing blanks       1
2    12345678901234567890   12345678901234567890   2
3    "   4 leading blanks"  "3 trailing blanks   "  3
```

```
rapids > select * from ('node://db1/text/tab_delimiter.csv' WITH
DELIMITER='\t');
COL1    COL2                    COL3                    COL4
```



```
----   ----                    ----                    ------
1           4 leading blanks   3 trailing blanks       1
2      12345678901234567890    12345678901234567890    2
3           4 leading blanks   3 trailing blanks       3

3 row(s) returned (0.06 sec)
```

7.5.6 ENCLOSED_BY='<char> ' | """

Specifies whether an input field is optionally enclosed by the specified character. 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 is also included as part of the field data, then the character must be escaped (see ESCAPE_CHAR 7.5.7).
4. When exporting data, character fields will only be enclosed using the ENCLOSED_BY character when the data for that field includes one or more DELIMITER (see 7.5.5) characters.
5. Numercial and Boolean values cannot be enclosed

The default enclosed_by character is a double quote.

Examples:

ENCLOSED_BY	DATA TYPE	INPUT	DATA TO BE STORED
	VARCHAR	'DAVE's DATA'	'DAVE's DATA'
	VARCHAR	"'DAVE's DATA'"	'DAVE's DATA'
	VARCHAR	'DAVE\'s DATA'	'DAVE's DATA'
	VARCHAR	'"DAVE"s DATA"'	INVALID
	INTEGER	'9'	INVALID
	DECIMAL	'9.0'	INVALID
	FLOAT	'9.0'	INVALID
	TIMESTAMP	'2020-09-01 09:00:00'	2020-09-01 09:00:00
	BOOLEAN	'T'	FALSE
	BOOLEAN	"T"	TRUE
	INTEGER	"9"	9
	DECIMAL	"9.0"	9.0
	FLOAT	"9.0"	9.0

	TIMESTAMP	"2020-09-01 09:00:00"	2020-09-01 09:00:00
ENCLOSED_BY="'"	VARCHAR	'DAVE's DATA'	INVALID
	VARCHAR	"'DAVE's DATA'"	INVALID
	VARCHAR	'DAVE\'s DATA'	DAVE's DATA
	VARCHAR	'"DAVE"s DATA"'	"DAVE"s DATA"
	INTEGER	'9'	9
	DECIMAL	'9.0'	9.0
	FLOAT	'9.0'	9.0
	TIMESTAMP	'2020-09-01 09:00:00'	2020-09-01 09:00:00
	BOOLEAN	'T'	TRUE
	BOOLEAN	"T"	FALSE
	INTEGER	"9"	INVALID
	DECIMAL	"9.0"	INVALID
	FLOAT	"9.0"	INVALID
	TIMESTAMP	"2020-09-01 09:00:00"	2020-09-01 09:00:00

Example 1: This example uses the default ENCLOSED_BY double quote character.

File: /var/tmp/rapids/text/default_enclosed_by.csv

```
1,'DAVE's  DATA',"DAVE's  DATA",T,9.0,9.0,"2020-09-01  09:00:00"
```

```
rapids > select * from ('node://db1/text/default_enclosed_by.csv') AS t(c1
integer, c2 varchar, c3 varchar, c4 boolean, c5 decimal, c6 float, c7
timestamp);
  C1 C2              C3                C4    C5    C6 C7
  -- --              --                --    --    -- --
   1 'DAVE's DATA'   DAVE's DATA     true   9.0   9.0 2020-09-01 09:00:00.0

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

Example 2: This example sets the ENCLOSED_BY character to a single quote:

File: /var/tmp/rapids/text/single_quote_enclosed_by.csv

```
1,'DAVE\'s  DATA','"DAVE"s  DATA"',T,9.0,9.0,'2020-09-01  09:00:00'
```

```
rapids > select * from ('node://db1/text/single_quote_enclosed_by.csv' WITH
ENCLOSED_BY="'") AS t(c1 integer, c2 varchar, c3 varchar, c4 boolean, c5
decimal, c6 float, c7 timestamp);
  C1 C2            C3                  C4    C5    C6 C7
  -- --            --                  --    --    -- --
   1 DAVE's DATA   "DAVE"s DATA"     true   9.0   9.0 2020-09-01 09:00:00.0

1 row(s) returned (0.59 sec)
```


7.5.7 ESCAPE_CHAR='<char>'

Specifies the character to be used as the escape character. This allows the user to include embedded field delimiters and enclosed_by characters in the data.

Default: '\' (backslash)

Example 1:

Shows the escaping of the ENCLOSED_BY and DELIMITER characters when those characters are the defaults. The escaped characters are hilited:

File:/var/tmp/rapids/text/escape_char.csv

```
1,"Escaped ENCLOSED_BY\"",Escaped DELIMITER \,,End of row
```

```
rapids > select * from ('node://db1/text/escape_char.csv');
  COL1 COL2                    COL3                    COL4
  ---- ----                    ----                    ----
     1 Escaped ENCLOSED_BY"    Escaped DELIMITER ,     End of row

1 row(s) returned (0.56 sec)
```

Example 2:

Shows the same example as before except in this example the ESCAPE_CHAR is set to the dollar character. The escaped characters are hilited:

File: /var/tmp/rapids/text/escape_char_dollar.csv

```
1,"Escaped ENCLOSED_BY$"",Escaped DELIMITER $,,End of row
```

```
rapids > select * from ('node://db1/text/escape_char_dollar.csv' WITH
ESCAPE_CHAR='$') ;
  COL1 COL2                    COL3                    COL4
  ---- ----                    ----                    ----
     1 Escaped ENCLOSED_BY"    Escaped DELIMITER ,     End of row

1 row(s) returned (0.56 sec)
```

7.5.8 HEADER

Specifies whether the data file has a header record which has the column names to use.

File: /var/tmp/rapids/text/header.csv

```
id,name,dob
1,Jim  Smith,2004-04-01
```



```
rapids > select * from ('node://db1/text/header.csv' WITH HEADER);
  id name         dob
  -- ----         ---
   1 Jim Smith    2004-04-01

1 row(s) returned (0.06 sec)
```

7.5.9 CHARSET

Specifies the character set to be used. Some examples:

- 'GBK'
- 'GB2312'
- 'GB18030'
- 'Big5'

Refer to https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html for a list of the possible character set names.

Example:

This example shows the character set being set to the “GBK” character set:

File: /var/tmp/rapids/text/charset_gbk.csv

```
1, 今天天气很暖和上周六下午温度升至9度。太阳很明亮
```

```
rapids > select * from ('node://db1/text/charset_gbk.csv' WITH
CHARSET='GBK');

COL1    COL2

----    ----

1       浠婂ぉ澶╂皵寰堟殩鍜屼笂鍛ㄥ叚涓嬪崍娓┿害鍗囧崈9搴﹀伌お闐冲緝鏄庝寒


1 row(s) returned (0.14 sec)
```

7.5.10 TRAILING

When “true” IMPEX will ignore a trailing field separator (i.e. where the field separator is immediately followed by the record terminator character) on each line of a file being imported and will append a trailing separator to each line of a file being exported.

When “false” a trailing field separator will indicate a null value for the last column of the record being imported. For export no trailing field separator will be written out.

Example 1:

This example shows how the trailing delimiter will be ignored:

File: /var/tmp/rapids/text/trailing.csv

1,field 2,field 3,

```
rapids > select * from ('node://db1/text/trailing.csv' WITH TRAILING);
COL1    COL2       COL3
----    ----       ----
1       field 2    field 3

1 row(s) returned (0.05 sec)
```

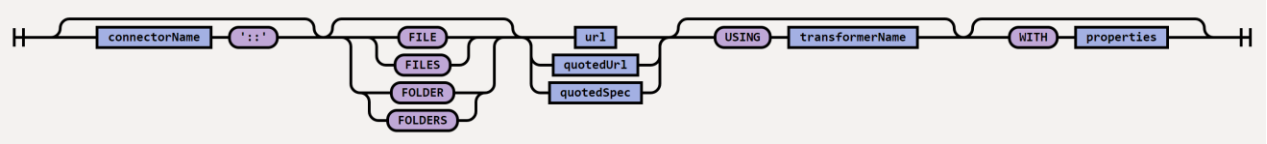
Example 2:

This example uses the same data file as example 1, but this time the “TRAILING” property is not set and so the trailing delimiter will be included in the import operation resulting in a null value for the final field:

```
rapids > select * from ('node://db1/text/trailing.csv');
COL1    COL2       COL3       COL4
----    ----       ----       ----
1       field 2    field 3    NULL

1 row(s) returned (0.05 sec)
```

7.6 IMPORT References

The user specifies the location and formatting information for the data to be imported using an Import Reference:



The table below describes each option:

Option	Required?	Default	Description
connectorName	No	IMPORT	The name of an existing IMPEX Connector.
FILE	No	FILE	Indicates that the name specified by the quotedUrl (see below) refers to a file to be imported. This is the default for non-bulk IMPORT/EXPORT operations
FILES	No	FILES	Used for bulk import operations (see 7.10). Indicates that name specified by the quotedUrl (see below) refers to a folder which contains a set of files to be imported. The name of each file (minus any dot suffixes) is the name of the table where the data from the file will be written. This is the default for bulk IMPORT/EXPORT operations
FOLDER	No		Indicates that the name specified by the quotedUrl (see below) refers to a folder which contains a set of files to be imported.
FOLDERS	No		Used for bulk import operations (see 7.10). Indicates that name specified by the quotedUrl (see below) refers to a folder, which contains a set of sub-folders to be imported. The name of each sub-folder is the name of the table where the imported data from the files in the sub-folder will be written.
url			see quotedUrl

quotedUrl	Yes		Specifies the location of the data to be imported using the following format: 'node://<RDP node>/< path name>' where <RDP node> is the RDP node name where the data to be imported is located <path name> is the relative path name to the location of the data (relative to the setting for the PATH property, see 7.4) Examples: With PATH set to the default "/var/tmp/rapids": 'node://db1/data' specifies that the data is to be imported from the directory "/var/tmp/rapids/data" on RapidsDB node "db1" For a custom IMPEX Connector with PATH set to "/": 'node://db1/data/log1.csv' specifies that the data is to be imported from the file "/data/log1.csv" on RapidsDB node "db1"
quotedSpec	No		See quotedUrl
transformerName	No		**Not supported for this release**
properties	No		Connector-defined properties (as key = value pairs) for the operation. By convention, explicitly specified properties override properties of the same name in the Connector (see 7.4 for the list of Properties).

Below are some examples of import references:

- 'node://db1/data/table1.csv'

  As no Connector name is specified, this import reference is for the default import Connector named "IMPORT" (see 3.9). The file name specified, "data/table1.csv" is relative to the PATH Property for the Connector, which for the "IMPORT" Connector is "/var/tmp/rapids/"(unless the PATH Property for the "IMPORT" Connector is changed – see 3.9.3), and so the fully qualified path name is "/var/tmp/rapids/data/table1.csv" on RapidsDB cluster node "db1".

- CSV:: 'node://db1/data/table1.csv' WITH DELIMITER='|'

Specifies that the IMPEX Connector named “CSV” should be used and so the file name “data/table1.csv” will be relative to the PATH Property for the “CSV” Connector. For example if the “CSV” has Connector has the PATH Property set to '/' (root directory), then the specified file name would get resolved as “/data/table1.csv”. The field delimiter is set to the pipe character '|'.

- FOLDER 'node://db1/data/table1' WITH DELIMITER='|', HEADER

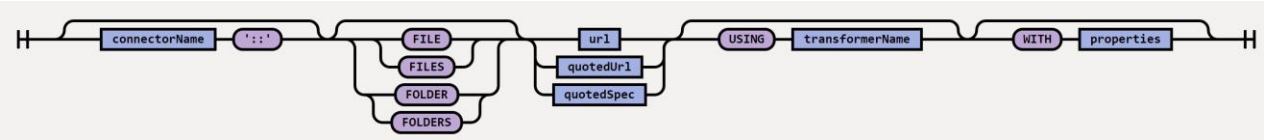
  Using the default “IMPORT” Connector, the path for the folder “data/table1” would get resolved to “/var/tmp/rapids/data/table1” on RapidsDB cluster node “db1”. The field delimiter would be set to the pipe character and the HEADER option is set to indicate that the data file has a header record

- FOLDERS CSV:: 'node://db1/data/tpch'

  For a bulk import, specifies that the IMPEX Connector named “CSV” should be used and so the folder name “data/tpch” will be relative to the PATH Property for the “CSV” Connector. For example if the “CSV” Connector has the PATH Property set to '/' (root directory), then the specified folder name would get resolved as “/data/tpch”.

7.7 EXPORT References

The user specifies the location and formatting information for the data to be exported using an Export Reference:



The table below describes each option:

Option	Required?	Default	Description
connectorName	No	EXPORT	The name of an existing IMPEX Connector
FILE	No	FILE	Indicates that the name specified by the quotedUrl (see below) refers to a file where the data for the table to be exported will be written.
FILES	No	FILES	Used for bulk export operations (see 3.12). Indicates that name specified by the quotedUrl (see below) refers to a folder which will contain the files for the set of tables being exported.
FOLDER	No		Indicates that the name specified by the quotedUrl (see below) refers to a folder where a file will be created that will store the data from the table being exported.
FOLDERS	No		Used for bulk export operations (see 3.12). Indicates that name specified by the quotedUrl (see below) refers to a folder where the sub-folders for the exported tables will be created (if needed) using the table name for the sub-folder name. Each sub-folder will then have a file created in it that will store the data from the table being exported
url	No		See quotedUrl below
quotedUrl	Yes		Specifies the location for the exported data using the following format: 'node://<RDP node>/< path name>' where <RDP node> is the RDP node name where the data to be imported is located <path name> is the relative path name to the location of the data (relative to the setting for the PATH property, see 7.4) With PATH set to the default "/var/tmp/rapids":

			'node://db1/data' specifies that the data is to be exported to the directory "/var/tmp/rapids/data" on RapidsDB node "db1" For a custom IMPEX Connector with PATH set to "/": 'node://db1/data/log1.csv' specifies that the data is to be exported to the file "/data/log1.csv" on RapidsDB node "db1"
quotedSpec	No		See quotedUrl above
transformerName	No		**Not supported for this release**
properties	No		Connector-defined properties (as key = value pairs) for the operation. By convention, explicitly specified properties override properties of the same name in the Connector (see 7.4 for the list of Properties).

Below are some examples of export references:

- 'node://db1/data/table1.csv'

  As no Connector name is specified, this import reference is for the default export Connector named "EXPORT" (see 7.8). The file name specified, "data/table1.csv" is relative to the PATH Property for the Connector, which for the "EXPORT" Connector is "/var/tmp/rapids/"(unless the PATH Property for the "IMPORT" Connector is changed – see 7.8.3), and so the fully qualified path name is "/var/tmp/rapids/data/table1.csv" on RapidsDB cluster node "db1".

- CSV:: 'node://db1/data/table1.csv'

  Specifies that the IMPEX Connector named "CSV" should be used and so the file name "data/table1.csv" will be relative to the PATH Property for the "CSV" Connector. For example, if the "CSV" has Connector has the PATH Property set to '/' (root directory), then the specified file name would get resolved as "/data/table1.csv".

- FOLDER 'node://db1/data/table1'

  Using the default "EXPORT" Connector, the path for the folder "data/table1" would get resolved to "/var/tmp/rapids/data/table1" on RapidsDB cluster node "db1".

- FOLDERS CSV:: 'node://db1/data/tpch_files'

For a bulk export, specifies that the IMPEX Connector named “CSV” should be used and so the folder name “data/tpch” will be relative to the PATH Property for the “CSV” Connector. For example, if the “CSV” has Connector has the PATH Property set to '/' (root directory), then the specified folder name would get resolved as “/data/tpch”.

7.8 Default IMPORT and EXPORT Connectors

7.8.1 Usage

RapidsDB comes with two built-in Connectors named “IMPORT” and “EXPORT”, that are used when an import reference (see 7.6) or an export reference (see 7.7) does not specify a Connector name. For example, 'node://db1/data/table1.csv'.

7.8.2 Default Properties

By default, the “IMPORT” and “EXPORT” Connectors have the following IMPEX Connector Properties (see 7.4):

Key:	Value
FORMAT	'CSV'
PATH	'/var/tmp/rapids'
ERROR_PATH	'/var/tmp/rapids_errors'
ERROR_LIMIT	10
BACKUP	false
CHARSET	'UTF-8'
DELIMITER	','
ENCLOSED_BY	'"'
ESCAPE_CHAR	'\'
FILTER	'*.*'
GUESS	false
HEADER	false
TERMINATOR	'\n'
TRAILING	false

7.8.3 Changing the IMPEX Properties for the “IMPORT” and “EXPORT” Connectors

The user can change any of the properties for the “IMPORT” or “EXPORT” Connectors by dropping the Connector and then recreating the Connector with the same name.

NOTE: If the “IMPORT” or “EXPORT” Connector is dropped, it must be recreated with the same name because when doing an import or export operation, RapidsDB will attempt to use a Connector named “IMPORT” when an import reference (see 11.6) does not specify a Connector name, and similarly, the system will attempt to use a Connector named “EXPORT” when an export reference (see 11.7) does not specify a Connector name. If the system cannot find the relevant Connector then the import or export operation will fail.

One of the most common Properties to change would be the “PATH” property, to set an alternate default root path name. For example, if all data files will come from the folder “/data” then the default IMPORT Connector could be changed as shown below:

```
rapids > drop connector import;
0 row(s) returned (0.13 sec)
rapids > create connector import type impex with PATH='/data';
0 row(s) returned (2.37 sec)
```

The following import examples assume that the default “IMPORT” Connector was not reconfigured, and that an IMPEX Connector named “CSV” was created with the “PATH” Property set to the root directory (“/”):

```
rapids > create connector csv type impex with PATH='/data';

0 row(s) returned (2.09 sec)

SELECT * FROM ('node://db1/data/table1.csv');
```

This command would result in data being read from the file “table1.csv” in the directory “/var/tmp/rapids/data” on RapidsDB node “db1”using the default “IMPORT” Connector.

```
SELECT id, name FROM (CSV:: 'node://db1/data/table1.csv') AS t(id integer, name varchar)  WHERE t.name='BorayData';
```

This command would select the data for the first two fields from the file “table1.csv” using the Connector named “CSV” where the file “table1.csv” resides on the RapidsDB cluster node “db1” in the directory “/data” (because PATH='/'), where the “name” field is the string 'BorayData'.

The following export examples assume that the default “EXPORT” Connector was not reconfigured:

```
SELECT * FROM table1 TO 'node://db1/data/table1.csv';
```

This command would append the contents of the table named “table1” to the file “table1.csv” using the default EXPORT Connector where the file “table1.csv” resides in the directory “/var/tmp/rapids/data” on the RapidsDB cluster node “db1” and where the delimiter is ','.

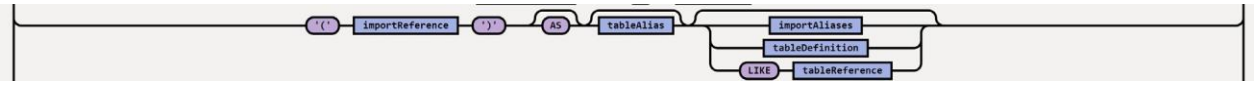
```
SELECT * FROM table1 TO CSV:: 'node://db1/data/table1.csv';
```

In this example the EXPORT reference specifies the Connector name “CSV”, and so the default “EXPORT” Connector would not be used.

7.9 IMPORT using SELECT and INSERT

7.9.1 IMPORT Table Expressions

A table expression has now been extended to also include an import reference:



This means that an import reference (see 7.6) can now appear anywhere in a SELECT or INSERT statement where a regular table reference can appear, and it also means that the user can provide an alias name for the import reference, and can also specify a subset of the columns (fields) to be imported. For example:

```
SELECT * FROM ('node://db1/data/table1.csv');
```

The hilited text above is an import reference which replaces the usual table reference.

Section 7.9.2 provides examples of using an import reference with SELECT and INSERT statements.

7.9.2 IMPORT using a SELECT statement

7.9.2.1 Overview

The user can import the data as part of a regular SELECT statement, where the usual table reference is replaced with an import reference. In the following examples the hilited text is the import reference.

Example 1:

```
SELECT * FROM (FILE 'node://db1/data/table1.csv');
```

This command would result in data being read from the file “table1.csv” in directory “var/tmp/rapids/data” (the default PATH) on RapidsDB node “db1”using the default “IMPORT” IMPEX Connector (see 7.8).

Example 2:

```
SELECT id, name FROM (FILE 'node://db1/data/table1.csv' WITH PATH='/') AS t(id integer, name varchar)  WHERE t.name='Jones';
```

This command would select the data for the first two fields from the file “table1.csv” using the default IMPEX Connector (see 7.8) where the file “table1.csv” resides on the RapidsDB cluster node “db1” in the directory “/data” (because PATH='/'), where the field delimiter is the character '|', with the column names “id” and “name” being used for the first two fields and where the “name” field is the string 'Jones'.

Example 3:

```
SELECT * FROM (CSV:: FOLDER 'node://db1/data/table1');
```

This command would select all of the data from the files in the folder “table1” using the IMPEX Connector named “CSV” where the folder “table1” resides on the RapidsDB cluster node “db1” in the directory “/data” (because PATH='/' for the “CSV” IMPEX Connector), and where the field

delimiter is the character '|'. The column headings would be the usual default column headings for a result set, which are “COL1”, “COL2” … etc.

See sections 7.9.2.10 and 7.9.2.11 for more detailed examples using SELECT statements.

7.9.2.2 Column Naming Using Default Column Names

If there are no column names associated with the input data (see the following sections for assigning column names), then the IMPEX Connector will use the default column names used by the RapidsDB Execution Engine which are “COL1”, “COL2”, etc.

Example:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/region.csv') LIMIT 1;
COL1    COL2             COL3
----    ----             ----
1       UNITED STATES    adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg

1 row(s) returned (0.06 sec)
```

7.9.2.3 Column Naming Using AS clause

The user can also specify the column names to be associated with each of the input fields using the “AS <tableAlias><(<columnAliases>)” clause as shown in the example below:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/region.csv') AS (r_regionkey ,r_name
,r_comment) LIMIT  1;

R_REGIONKEY    R_NAME           R_COMMENT
-----------    ------           ---------
1              UNITED STATES    adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg

1 row(s) returned (0.05 sec)
```

7.9.2.4 Column Naming Using HEADER option

It is common with csv files for the first record of the file to be a header record that contains a list of the column names. The IMPEX Connector allows this by setting the “HEADER” option to true. For example:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/regionPipe.csv' WITH DELIMITER='|',
HEADER) LIMIT 1;
R_REGIONKEY    R_NAME           R_COMMENT
-----------    ------           ---------
1              UNITED STATES    adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg

1 row(s) returned (0.05 sec)
```


7.9.2.5 Column Data Typing Using GUESS Property

If no column data types are assigned to the columns (see sections 7.9.2.5, 7.9.2.6 and 7.9.2.8) then the IMPEX Connector will use the "GUESS" Property (see 7.4) to determine the data types as follows:

1. GUESS=FALSE (default) – the IMPEX Connector will treat all columns where the data type is not explicitly specified (see 7.9.2.6 and 7.9.2.8 for specifying data types) as polymorphic strings, which are strings that will be automatically cast into the appropriate data type based on the expression where the column is referenced in a query. For example,

   SELECT * FROM ('node://db1/SFSMALL/region.csv') WHERE COL1>10;

   In this example, the first column will be automatically cast to an integer

   In the following example, the same field will be left as a string and not cast:

   SELECT * FROM ('node://db1/SFSMALL/region.csv') WHERE COL1='10A';

   If the column contains values that cannot be cast to the required data type, then an error will be returned.

2. GUESS=TRUE – for any columns where the data type is not explicitly specified (see 7.9.2.6 and 7.9.2.8 for specifying data types) the IMPEX Connector will examine a sample of the data from the file and use that sample to determine what is the best data type that fits each column in the data. If the data is uniform across the entire file, then the IMPEX Connector will generally assign the correct data type, but if the data is not uniform then the IMPEX Connector could assign the wrong data type which could result in a data type conversion error when querying the data. For example, if a field had mostly integer values, but there were a few values that were alphanumeric, then the IMPEX Connector could assign a data type of INTEGER to the column because in the data sample that was read to determine the data types, all of the values for that field were integers. This could result in an error when querying the column associated with that field. For example, in the query below, the IMPEX Connector assigned a data type of INTEGER to the column "r_regionkey", but the query failed when looking for an alphanumeric value:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/regionPipe.csv' WITH
HEADER,delimiter='|',GUESS) where r_regionkey='4A';
Unexpected Exception:
Line 1 position 90: Unresolved operator or function name: =(FastInteger, LiteralString)
```

   The CAST function can be used to address this issue:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/regionPipe.csv' WITH
HEADER,delimiter='|',GUESS) where cast(r_regionkey as varchar)='4A';
0 row(s) returned (0.04 sec)
```

   Example 1:

File: /var/tmp/SFSMALL/partsupp.csv:

```
1,1,42,562.15,unsure
1,2,640,974.09,reliable
1,3,720,550.17,dishonest
1,4,644,461.39,weak
…
```

```
rapids > SELECT * FROM ('node://db1/SFSMALL/partsupp.csv' WITH GUESS)
LIMIT 1;
  COL1   COL2   COL3     COL4 COL5
  ----   ----   ----     ---- ----
     1      1     42   562.15 unsure

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

The following data types will be assigned:

Column	Data type
COL1	integer
COL2	integer
COL3	integer
COL4	decimal
COL5	varchar

Example 2:

In this example the data type for the fourth column was specified using the "AS" clause (see 7.9.2.6), and all other data types will be imputed by the Connector:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/partsupp.csv' WITH GUESS)
AS (COL1,COL2,COL3,COL4 float,COL5) LIMIT 1;
COL1   COL2   COL3       COL4 COL5
----   ----   ----       ---- ----
1      1      42       562.15 unsure

1 row(s) returned (0.05 sec)
```

The following data types will be assigned:

Column	Data type
COL1	integer
COL2	integer
COL3	integer
COL4	float
COL5	varchar

7.9.2.6 Column Data Typing Using AS clause

In addition to using the AS clause to name columns, the user can also use the AS clause to specify the data types to be used for the columns. For example:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/region.csv') AS (r_regionkey integer,
r_name varchar, r_comment varchar) LIMIT  1;
  R_REGIONKEY R_NAME          R_COMMENT
  ----------- ------          ---------
            1 UNITED STATES   adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg

1 row(s) returned (0.05 sec)
```

The following data types will be assigned:

Column	Data type
r_regionkey	integer
r_name	varchar
r_comment	varchar

It is also possible to just specify the data types for some of the columns and let the IMPEX Connector assign the other data types by setting the "GUESS" Property to TRUE. In the example below, the data types for two of the columns were specified and the "GUESS" Property was set TRUE so that the Impex Connector would assign the data types for the other columns:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/customer.csv' WITH GUESS) AS
(c_custkey integer, c_name, c_address, c_nationkey, c_phone, c_acctbal
decimal, c_mktsegment, c_comment) LIMIT 1;
  C_CUSTKEY C_NAME        C_ADDRESS    C_NATIONKEY    C_PHONE      C_ACCTBAL
C_MKTSEGMENT    C_COMMENT
  --------- ------        ---------    -----------    -------      --------- ----
--------   ---------
          0 Richardson    Market       3              111            7994.73
FURNITURE       negative

1 row(s) returned (0.05 sec)
```

The following data types will be assigned:

Column	Data type
c_custkey	integer
c_name	varchar
c_address	varchar
c_nationkey	integer
c_phone	integer
c_acctbal	decimal
c_mktsegment	varchar
c_comment	varchar

7.9.2.7 Column Skipping/Pruning Using AS Clause

The user can skip columns of the input data using the AS clause by simply omitting the column name from the list of columns in the input data. For example, the following "AS" clause would include columns one and two of the input, skip columns three and four, include column five, and then ignore any remaining columns: "AS(L_ORDERKEY integer, L_PARTKEY integer,,,L_QUANTITY integer)"

Example:

Below is an entire record of data:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/lineitem.csv')  LIKE
moxe.lineitem limit 1;

  L_ORDERKEY    L_PARTKEY    L_SUPPKEY    L_LINENUMBER    L_QUANTITY
L_EXTENDEDPRICE L_DISCOUNT   L_TAX L_RETURNFLAG         L_LINESTATUS
L_SHIPDATE    L_COMMITDATE    L_RECEIPTDATE    L_SHIPINSTRUCT    L_SHIPMODE
L_COMMENT


--------
--------

          462           27            2               0           117
3908.88        419.61    411.43 R               P               2010-09-23
2015-05-17      2011-01-03       COLLECT COD       RAIL          lousy

1 row(s) returned (0.05 sec)
```

The following statement uses the "AS" clause from above to only select a subset of the input columns:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/lineitem.csv') AS(L_ORDERKEY
integer, L_PARTKEY integer,,,L_QUANTITY integer) limit 1;

  L_ORDERKEY    L_PARTKEY    L_QUANTITY

  ----------   ---------   ----------

          462           27          117

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

7.9.2.8 Column Naming and Data Typing Using LIKE clause

An alternative way to assign a schema to a file is to use the "LIKE" clause where the name of an existing table can be specified which will result in the schema for that table being used for the file being imported. For example:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/region.csv') LIKE moxe.region LIMIT 1;
```



```
  R_REGIONKEY R_NAME           R_COMMENT
  ----------- ------           ---------
            1 UNITED STATES    adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg

1 row(s) returned (0.05 sec)
```

7.9.2.9 RAW Data Format

In situations where a data file contains data that is completely unknown, such as where the field delimiter is not known, the user can use the "RAW" format option to import the data as a single VARCHAR column. The data can then be viewed and further processing can then be done based on the actual data. This is format is also useful for fetching miscellaneous text files from locations within the RapidsDB cluster.

Example:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/region.csv' WITH FORMAT='RAW');
RAW
---
1,UNITED STATES,adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2,NORTH AMERICA,aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3,EUROPE,dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4,SOUTH AMERICA,csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5,ASIA,i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt

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

```
rapids > SELECT * FROM ('node://db1/var/log/dmesg' WITH
PATH='/',FORMAT='RAW') LIMIT 2;
RAW
---
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu

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

7.9.2.10 SELECT FROM FILE

The following examples illustrate the use of an IMPEX Connector for importing data directly from a file to be used in a SELECT statement. As "FILE" is the default option it is not necessary to specify that option when referencing a file.

Example 1:

Select from the file "text/lead_trail_blanks.csv" which is in the folder "/var/tmp/rapids".

Below is the contents of the file "lead_trail_blanks.csv":

```
[rapids@db1 text]$ cat /var/tmp/rapids/text/lead_trail_blanks.csv
```



```
1,    4 leading blanks,3 trailing blanks    ,1
2,A2345678901234567890,A1234567890123456789,2
3,"    4 leading blanks","3 trailing blanks    ",3
```

The SELECT command below selected all of the data from the file “lead_trail_blanks.csv” using the default “IMPORT” Connector where the file “lead_trail_blanks.csv” resides on the RapidsDB cluster node “db1” in the folder “/var/tmp/rapids/text”, and where the field delimiter is the character ',' (this is the default for “IMPORT” Connector). The column headings would be the usual default column headings for a result set, which are “COL1”, and “COL2”.

```
rapids > select * from ('node://db1/text/lead_trail_blanks.csv');
  COL1 COL2                  COL3                      COL4
  ---- ----                  ----                      ----
     1     4 leading blanks   3 trailing blanks           1
     2 A2345678901234567890   A1234567890123456789        2
     3     4 leading blanks   3 trailing blanks           3

3 row(s) returned (0.07 sec)
rapids > select char_length(col3) from
('node://db1/text/lead_trail_blanks.csv');
  [1]
  ---
   20
   20
   20

3 row(s) returned (0.06 sec)
```

Example 2:

Selecting from a file where the delimiter is the pipe character ('|') and the file includes a header record with the column names to use.

Below is the content of the file:

```
[rapids@db1 rapids]$ cat /var/tmp/rapids/SFSMALL/regionPipe.csv
R_REGIONKEY|R_NAME|R_COMMENT
1|UNITED STATES|adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2|NORTH AMERICA|aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3|EUROPE|dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4|SOUTH AMERICA|csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5|ASIA|i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt
```

This command below selected all of the data from the file “regionPipe.csv” using the default “IMPORT” Connector, where the file “regionPipe.csv” resides on the RapidsDB cluster node “db1” in the folder

"/var/tmp/rapids/SFMALL", and where the field delimiter is the character '|'. The file includes a header record which has the column names to use.

```
rapids > SELECT * FROM ('node://db1/SFSMALL/regionPipe.csv' WITH
HEADER,delimiter='|');

R_REGIONKEY   R_NAME          R_COMMENT

-----------   ------          ---------

1             UNITED STATES   adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg

2             NORTH AMERICA   aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd

3             EUROPE          dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds

4             SOUTH AMERICA   csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda

5             ASIA            i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt

5 row(s) returned (0.05 sec)
```

Example 3:
This example shows the use of a custom IMPEX Connector named "CSV_HEADER" which has the default delimiter set to the pipe character, and with HEADER set true:

```
rapids > CREATE CONNECTOR CSV_HEADER TYPE IMPEX WITH DELIMITER='|', HEADER;
0 row(s) returned (2.21 sec)
```

```
rapids > SELECT * FROM (csv_header:: 'node://db1/SFSMALL/regionPipe.csv');

R_REGIONKEY   R_NAME          R_COMMENT

-----------   ------          ---------

1             UNITED STATES   adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg

2             NORTH AMERICA   aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd

3             EUROPE          dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds

4             SOUTH AMERICA   csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda

5             ASIA            i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt

5 row(s) returned (0.05 sec)
```

Example 4:

This example shows data filtering by using a predicate ("r_regionkey=4") on the input data, and illustrates the use of the "AS" clause for defining the column names.

```
rapids > SELECT * FROM ('node://db1/SFSMALL/region.csv') AS
region(r_regionkey integer, r_name varchar) WHERE r_regionkey=4;
  R_REGIONKEY R_NAME
  ----------- ------
            4 SOUTH AMERICA

1 row(s) returned (0.10 sec)
```

Example 5:

This example illustrates the use of a complex query to filter the data, where the query includes both predicates on the input data along with a join to another MOXE table:

```
rapids > SELECT * FROM (FILE 'node://db1/SFSMALL/customer.csv') AS IMPORTED
(c_custkey INTEGER, c_name VARCHAR, c_address VARCHAR, c_nationkey  INTEGER,
c_phone VARCHAR, c_acctbal  DECIMAL, c_mktsegment  VARCHAR, c_comment
VARCHAR) WHERE c_acctbal > 3000 and c_nationkey=22 AND EXISTS (SELECT * FROM
vip_customer WHERE vip_customer.c_custkey = IMPORTED.c_custkey) ;
  C_CUSTKEY C_NAME     C_ADDRESSC_NATIONKEY C_PHONE          C_ACCTBAL
C_MKTSEGMENT    C_COMMENT         [9]
  --------- ------     ---------     ----------- -------     --------- ------
------   ---------        ---
          20 Egerton    Main                    22 111           3815.45
AUTOMOBILE      satisfied          1
          28 Stringer   Mission                 22 111           8516.37
FURNITURE       dissatisfied       1
          61 Riley      Turk                    22 111           6320.39
MACHINERY       angry              1

3 row(s) returned (0.18 sec)
```

Example 6:

This example illustrates the use of the LIKE clause to provide the column names and column data types for the input data. In this example, the table definition for the table "moxe.customer" is being used:

```
rapids > SELECT * FROM ('node://db1/SFSMALL/customer.csv') AS IMPORTED LIKE
moxe.customer WHERE IMPORTED.c_acctbal > 3000 and c_nationkey=22 AND EXISTS
(SELECT * FROM vip_customer WHERE vip_customer.c_custkey =
IMPORTED.c_custkey) ;
  C_CUSTKEY C_NAME     C_ADDRESS     C_NATIONKEY C_PHONE     C_ACCTBAL
C_MKTSEGMENT    C_COMMENT         [9]
  --------- ------     ---------     ----------- -------     --------- ------
------   ---------        ---
          20 Egerton    Main                    22 111           3815.45
AUTOMOBILE      satisfied          1
```



```
         28 Stringer    Mission                 22 111            8516.37
FURNITURE       dissatisfied        1
         61 Riley       Turk                    22 111            6320.39
MACHINERY       angry               1

3 row(s) returned (0.18 sec)
```

NOTE:

When filtering data using predicates, it is highly recommended that the AS clause includes the data types for all columns as shown in examples 3, and 4 above or the LIKE clause is used to provide the column definitions from an existing table as shown in example 5 above.

7.9.2.11 SELECT FROM FOLDER

The following examples illustrate the use of an IMPEX Connector for importing data directly from a folder to be used in a SELECT statement. The files to be accessed from the folder are controlled by the "FILTER" property, which by default is set to '*.*' (to read all files).

Example 1:
This example shows reading all of the files from a folder:

Folder /var/tmp/rapids/tpch_small/region:

```
[rapids@db1 region]$ ls /var/tmp/rapids/tpch_small/region
region.csv
```

```
rapids > SELECT * FROM (FOLDER 'node://db1/tpch_small/region');

COL1    COL2             COL3
----    ----             ----
1       UNITED STATES    adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2       NORTH AMERICA    aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3       EUROPE           dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4       SOUTH AMERICA    csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5       ASIA             i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt

5 row(s) returned (0.49 sec)
```

Example 2:
In this example a "FILTER" option is used to only import files ending in ".csv". The folder for the example below includes two ".csv" files which will get loaded and one file named "junk" that will be ignored:

Folder /var/tmp/rapids/SFSMALL/region:

```
[rapids@db1 region]$ ls /var/tmp/rapids/SFSMALL/region
junk  region1.csv  region2.csv
```



```
rapids > SELECT * FROM (FOLDER 'node://db1/SFSMALL/region' WITH
FILTER='*.csv');
COL1    COL2             COL3
----    ----             ----
1       UNITED STATES    adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2       NORTH AMERICA    aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3       EUROPE           dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4       SOUTH AMERICA    csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5       ASIA             i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt

5 row(s) returned (0.05 sec)
```

Example 3:

This example shows the use of a custom Connector named “CSV_HEADER” which has the “HEADER” Property set true and the delimiter character set to the pipe character:

```
rapids > SELECT * FROM (csv_header::  FOLDER 'node://db1/SFSMALL/regionPipe');
R_REGIONKEY    R_NAME           R_COMMENT
-----------    ------           ---------
1              UNITED STATES    adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2              NORTH AMERICA    aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3              EUROPE           dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4              SOUTH AMERICA    csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5              ASIA             i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt

5 row(s) returned (0.08 sec)
```

Example 4:

This example illustrates the use of a predicate to filter the data, and also includes the “AS” clause to specify the column names and data types:

```
rapids > SELECT * FROM (FOLDER 'node://db1/SFSMALL/region' WITH
FILTER='*.csv') AS region(r_regionkey integer, r_name varchar) WHERE
r_regionkey=4;
  R_REGIONKEY R_NAME
  ----------- ------
            4 SOUTH AMERICA

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

Example 5:

This example is the same as the previous example except a “LIKE” clause is used in place of the “AS” clause to specify the column names and data types:


```
rapids > SELECT * FROM (FOLDER 'node://db1/SFSMALL/region' WITH
FILTER='*.csv') AS region LIKE moxe.region WHERE r_regionkey=4;
  R_REGIONKEY R_NAME          R_COMMENT
  ----------- ------          ---------
            4 SOUTH AMERICA   csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja  shc
asbvda

1 row(s) returned (0.09 sec)
```

NOTE:

When filtering data using predicates, it is highly recommended that the AS clause includes the data types for all columns in the AS clause as shown in example 2 above or includes the LIKE clause as shown in example 4 above.

7.9.2.12 INSERT … SELECT

When doing an INSERT … SELECT, the data types the IMPEX Connector will use when reading the data associated with any of the files specified in the SELECT statement will be controlled by the column data types assigned to that file using either the "AS" clause (see 7.10.2.6) or the "LIKE" clause (see 7.10.2.8). If the data types are not specified then, by default, the IMPEX Connector will

Example 1:

This example shows an insert into the MOXE table named "region" of the data from all of the files ending in ".csv" in the folder "region" using the default "IMPORT" Connector where the folder "region" resides on the RapidsDB cluster node "db1" in the folder "/var/tmp/rapids/SFSMALL".

```
rapids > create table moxe.REGION (
       >    r_regionkey integer NOT NULL,
       >    r_name varchar(25),
       >    r_comment varchar(152)
       > );
0 row(s) returned (0.09 sec)
rapids > INSERT INTO moxe.region SELECT * FROM (FOLDER 'node://db1/SFSMALL/region'
WITH FILTER='*.csv');
0 row(s) returned (0.08 sec)
rapids > select * from moxe.region;
  R_REGIONKEY R_NAME          R_COMMENT
  ----------- ------          ---------
            1 UNITED STATES   adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
            2 NORTH AMERICA   aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
            3 EUROPE          dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
            4 SOUTH AMERICA   csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
            5 ASIA            i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt

5 row(s) returned (0.05 sec)
```

Example 2:

This example demonstrates selecting a subset of the fields from the input file by using the "AS" clause to name the fields from the input file that are to be imported, and also the use of a predicate to filter the

data being inserted:

```
rapids > create table moxe.REGION2 (
       >    r_regionkey integer NOT NULL,
       >    r_name varchar(25)
       > );
0 row(s) returned (0.10 sec)
rapids > INSERT into moxe.region2 SELECT r_regionkey, r_name FROM
('node://db1/SFSMALL/region.csv') AS r(r_regionkey integer, r_name varchar)
WHERE r.r_regionkey<3;
0 row(s) returned (0.08 sec)
rapids > select * from moxe.region2;
  R_REGIONKEY R_NAME
  ----------- ------
            1 UNITED STATES
            2 NORTH AMERICA

2 row(s) returned (0.05 sec)
```

Example 3:

This command also selects a subset of the fields to be inserted and uses a header record in the input file to name the fields in the input file which can then be used to specify which fields are to be imported:

```
rapids > truncate region2;
0 row(s) returned (0.05 sec)
rapids > INSERT into moxe.region2 SELECT r_regionkey, r_name FROM
('node://db1/SFSMALL/regionPipe.csv' WITH DELIMITER='|', HEADER);
0 row(s) returned (0.08 sec)
rapids > select * from region2;
  R_REGIONKEY R_NAME
  ----------- ------
            1 UNITED STATES
            2 NORTH AMERICA
            3 EUROPE
            4 SOUTH AMERICA
            5 ASIA

5 row(s) returned (0.05 sec)
```

Example 4:

This is the same as the previous example except this example uses a custom Connector named "CSV_HEADER" which has the "HEADER" Property set "true" and the "DELIMITER" set to the pipe character:

```
rapids >  truncate region2;
0 row(s) returned (0.05 sec)
rapids > INSERT into moxe.region2 SELECT r_regionkey, r_name FROM
(csv_header::'node://db1/SFSMALL/regionPipe.csv' );
0 row(s) returned (0.08 sec)
```



```
rapids > select * from region2;
  R_REGIONKEY R_NAME
  ----------- ------
            1 UNITED STATES
            2 NORTH AMERICA
            3 EUROPE
            4 SOUTH AMERICA
            5 ASIA

5 row(s) returned (0.05 sec)
```

7.9.2.13 CREATE AS SELECT

The user can create a new table us the CREATE <table> as SELECT … clause, where the SELECT can include an import reference. The column names and data types will follow the rules specified in sections 7.10.2.2 to 7.10.2.8.

NOTE:

1. When creating a MOXE table, if the "PARITITION BY" clause is not specified then the table will get created as a reference (replicated) table, where there will be a full copy of the data on every node in the RapidsDB cluster where the associated MOXE Connector is running. If the "PARTITION BY" clause is specified, then there will be one copy of the data distributed across all of the nodes in the RapidsDB cluster where the associated MOXE Connector is running, and the data will be partitioned using the column(s) specified in the "PARTITION BY" clause. Refer to sections 6.2.1 and 6.2.2 for more information on partitioned and reference tables. Examples 3 and 4 below shows the use of the "PARTITION BY" clause.
2. When creating a table managed by any of the Connectors except for MOXE, the "PARTITION BY" clause is not currently supported, and so the "CREATE TABLE" command will either fail with an error or the "PARTITION BY" clause will get ignored and the table will get created as a non-partitioned table.

Example 1:

This command creates a replicated MOXE table named "customer1" with the data from the file "customer.csv". The column names are the default column names, "COL1", "COL2" etc and the data types would all be set to VARCHAR because the "GUESS" property (see 11.4) is set false by default.

```
rapids > create table moxe.customer1 as select * from
('node://db1/SFSMALL/customer.csv');

0 row(s) returned (0.26 sec)

rapids > describe table customer1;

TABLE_NAME    COLUMN_NAME    DATA_TYPE      ORDINAL    IS_PARTITION_KEY
IS_NULLABLE    PRECISION    SCALE COMMENT    PROPERTIES
```



```
----------   -----------   ---------     -------   ------------------------   -------
----   ---------   -------- -------   --------------

CUSTOMER1     COL1           VARCHAR              0                false
true          NULL     NULL NULL       NULL

CUSTOMER1     COL2           VARCHAR              1                false
true          NULL     NULL NULL       NULL

CUSTOMER1     COL3           VARCHAR              2                false
true          NULL     NULL NULL       NULL

CUSTOMER1     COL4           VARCHAR              3                false
true          NULL     NULL NULL       NULL

CUSTOMER1     COL5           VARCHAR              4                false
true          NULL     NULL NULL       NULL

CUSTOMER1     COL6           VARCHAR              5                false
true          NULL     NULL NULL       NULL

CUSTOMER1     COL7           VARCHAR              6                false
true          NULL     NULL NULL       NULL

CUSTOMER1     COL8           VARCHAR              7                false
true          NULL     NULL NULL       NULL



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

Example 2:

This example creates a replicated MOXE table named “customer1” with the data from the file “customer.csv”. The column names are the default column names, “COL1”, “COL2” etc and the data types are derived by looking at a sample of the data because the “GUESS” property (see 7.4) is set to true.

```
rapids > create table moxe.customer1 as select * from
('node://db1/SFSMALL/customer.csv' WITH GUESS);
0 row(s) returned (0.27 sec)
rapids > describe table customer1;
TABLE_NAME    COLUMN_NAME    DATA_TYPE     ORDINAL    IS_PARTITION_KEY
IS_NULLABLE    PRECISION    SCALE COMMENT    PROPERTIES
-----------
-------    ----------
CUSTOMER1     COL1           INTEGER              0                false
true           64     NULL NULL       NULL
CUSTOMER1     COL2           VARCHAR              1                false
true          NULL     NULL NULL       NULL
```



```
CUSTOMER1     COL3           VARCHAR              2               false
true          NULL     NULL NULL       NULL
CUSTOMER1     COL4           INTEGER              3               false
true           64      NULL NULL       NULL
CUSTOMER1     COL5           INTEGER              4               false
true           64      NULL NULL       NULL
CUSTOMER1     COL6           DECIMAL              5               false
true           17         2 NULL       NULL
CUSTOMER1     COL7           VARCHAR              6               false
true          NULL     NULL NULL       NULL
CUSTOMER1     COL8           VARCHAR              7               false
true          NULL     NULL NULL       NULL

8 row(s) returned (0.31 sec)
```

Example 3:

This example creates a partitioned MOXE table using the "PARTITION BY" clause and also shows the use of the "AS" clause to specify the column names and data types to be used in the new table:

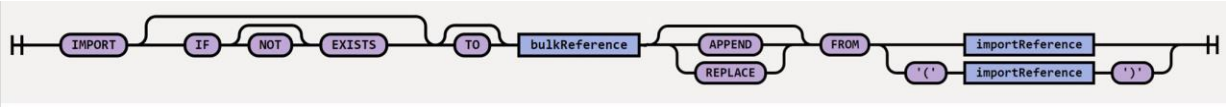
```
rapids > CREATE TABLE moxe.special_customer AS SELECT * FROM
('node://db1/SFSMALL/customer.csv') AS IMPORTED (c_custkey INTEGER, c_name
VARCHAR, c_address VARCHAR, c_nationkey  INTEGER, c_phone VARCHAR, c_acctbal
DECIMAL, c_mktsegment  VARCHAR, c_comment  VARCHAR) PARTITION BY HASH ON
(c_custkey) WHERE c_acctbal > 0 AND EXISTS (SELECT * FROM vip_customer WHERE
vip_customer.c_custkey = IMPORTED.c_custkey) ;
0 row(s) returned (0.41 sec)
rapids > describe table special_customer;
TABLE_NAME          COLUMN_NAME     DATA_TYPE      ORDINAL    IS_PARTITION_KEY
IS_NULLABLE    PRECISION    SCALE COMMENT    PROPERTIES
---------------
---------------
SPECIAL_CUSTOMER    C_CUSTKEY       INTEGER              0               false
true            64     NULL NULL       NULL
SPECIAL_CUSTOMER    C_NAME          VARCHAR              1               false
true          NULL     NULL NULL       NULL
SPECIAL_CUSTOMER    C_ADDRESS       VARCHAR              2               false
true          NULL     NULL NULL       NULL
SPECIAL_CUSTOMER    C_NATIONKEY     INTEGER              3               false
true            64     NULL NULL       NULL
SPECIAL_CUSTOMER    C_PHONE         VARCHAR              4               false
true          NULL     NULL NULL       NULL
SPECIAL_CUSTOMER    C_ACCTBAL       DECIMAL              5               false
true            17         2 NULL       NULL
SPECIAL_CUSTOMER    C_MKTSEGMENT    VARCHAR              6               false
true          NULL     NULL NULL       NULL
SPECIAL_CUSTOMER    C_COMMENT       VARCHAR              7               false
true          NULL     NULL NULL       NULL
```



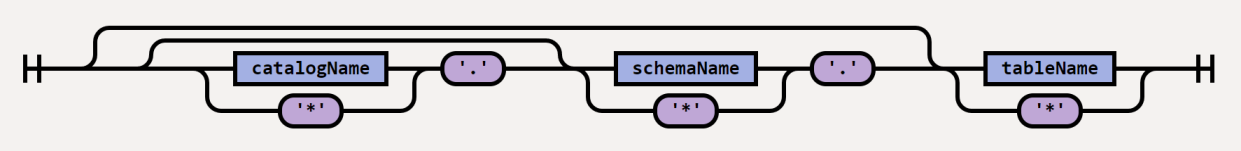
```
8 row(s) returned (0.31 sec)
rapids > select count(*) from special_customer;
  [1]
  ---
   75

1 row(s) returned (0.08 sec)
```

7.10 Bulk IMPORT

A new IMPORT statement is now supported for directly importing multiple tables with a single request:



bulkReference:



The table below describes each option:

Option	Required?	Default?	Description
IF EXISTS	No	No	Import only items whose name matches an existing table in the catalog and/or schema specified by the bulkReference (see below)
IF NOT EXISTS	No	No	Import only items whose name does *not* match an existing table in the catalog and/or schema specified by the bulkReference (see below)
APPEND	No	Yes	Append the imported data to an existing table, if any.
REPLACE	No	No	Truncate an existing table before new data is imported
bulkReference	Yes	No	Specifies the three-level (catalog, schema, table) naming for the target table(s) into which data will be imported. Wildcards may be specified (using an asterisk '*') for any of the name components. If the catalog name and/or schema name are omitted, the *CURRENT_CATALOG* and *CURRENT_SCHEMA* session settings are used, if set. The Connector may support properties (see below) which impact how the bulk reference is interpreted. **NOTE:** All of the tables to be imported into must be managed by the same Connector, if that is not the case then an error will be returned (see Example 4 in section 7.10.2 below for more details). To ensure

			that this does not happen it is recommended that the schema name is always specified and if this is not unique, then the catalog name should also be included.
importReference	Yes	No	An Import Reference (see 7.7) identifying the data to be imported

NOTE:

1. When doing a bulk import, if the target table does not exist then the table will get created by the Connector associated with the bulkReference (see above) using the following rules:
   a. The column names will be the default column names described in section 7.10.2.2
   b. The column data types will depend on the setting of the "GUESS" property as described in section 7.10.2.5. If the "GUESS" Property is set to FALSE, then all of the columns will be created as VARCHAR columns.
   c. The table will get created with defaults for any primary keys or partitioning keys, which could result in unexpected performance or memory limits issues. For example, a MOXE Connector will create the table as a Reference table (see 6.2.2) because there is no partitioning information, and this means that each node in the RapidsDB cluster where the associated MOXE Connector is running will have a full copy of the imported data, and this could result in MOXE running out of memory. To avoid any such issues, it is highly recommended that all target tables in a bulk import are first created with the appropriate primary and partitioning keys, and then the bulk import executed against the existing tables. Alternatively, the tables can be imported individually using a CREATE … AS SELECT statement where the primary and partitioning keys can be specified (see 7.10.2.12).

2. When doing a bulk import, if the target tables already exist, then it is recommended that the "GUESS" property is set to FALSE so that all data fields are read in as polymorphic strings (see 7.10.2.5) and then automatically cast to the data type for the column in the target table.

7.10.1 Bulk IMPORT Using FILES Option

When doing a bulk import operation the "FILES" option indicates that the path name specified in the import reference (see 7.6) refers to a folder which contains a set of files to be imported. The name of each file (minus any dot suffixes) is the name of the table where the data from the file will be written. The "FILES" option is the default and so it does not need to be specified.

Example 1:

Bulk import a set of files from the folder "/var/tmp/rapids/tpch_small_files" from RapidsDB Cluster node "db1" using the default "IMPORT" Connector (see 7.8).

Folder /var/tmp/rapids/tpch_small_files:


```
[rapids@db1 rapids]$ ls tpch_small_files
customer.csv  lineitem.csv  nation.csv  orders.csv  part.csv  partsupp.csv
region.csv  supplier.csv
```

```
rapids > IMPORT MOXE.* FROM 'node://db1/tpch_small_files' WITH GUESS;
0 row(s) returned (12.17 sec)
```

This command imported all the files from folder "/var/tmp/rapids/tpch_small_files" ("customer.csv", "lineitem.csv", "nation.csv", etc) creating new tables (see "show table;" output below) of the same names (minus the ".csv" suffix) in the schema MOXE if they did not already exist and where the column names for any newly created table were set to "COL1", "COL2", etc, and where the data types were imputed from the data in the files (based on the "GUESS" property being set TRUE (see 7.9.2.5) (see "describe table" output below). If the tables already existed, the new data would be inserted alongside the existing data.

```
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
…

rapids > describe table region;
TABLE_NAME    COLUMN_NAME    DATA_TYPE      ORDINAL    IS_PARTITION_KEY
IS_NULLABLE    PRECISION    SCALE
-------
-------
REGION        COL1           INTEGER              0               false
true           64     NULL
REGION        COL2           VARCHAR              1               false
true         NULL     NULL
REGION        COL3           VARCHAR              2               false
true         NULL     NULL

3 row(s) returned (0.28 sec)
```

Finally, the query below shows that the correct number of records were loaded into the table:

```
[rapids@db1 rapids]$ cat tpch_small_files/lineitem.csv | wc -l
3000
```



```
rapids > select count(*) from lineitem;
     [1]
     ---
    3000

1 row(s) returned (0.22 sec)
```

Example 2:

This example shows the use of the “IF NOT EXISTS” clause to restrict a bulk import to only those tables that do not exist.

Bulk import a set of files from the folder “/var/tmp/rapids/tpch_small_files” on RapidsDB Cluster node “db1”.

Folder /var/tmp/rapids/tpch_small_files:

```
[rapids@db1 rapids]$ ls tpch_small_files
customer.csv  lineitem.csv  nation.csv  orders.csv  part.csv  partsupp.csv
region.csv  supplier.csv
```

Drop the existing tables “customer” and “region”, show the existing row count for the “nation” table to show that the row count does not change after the import, and that only the “customer” and “region” tables are imported:

```
rapids > drop table customer;
0 row(s) returned (0.10 sec)
rapids > drop table region;
0 row(s) returned (0.10 sec)
rapids > show tables;
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME
------------    -----------    ----------
MOXE            MOXE           LINEITEM
MOXE            MOXE           NATION
MOXE            MOXE           ORDERS
MOXE            MOXE           PART
MOXE            MOXE           PARTSUPP
MOXE            MOXE           SUPPLIER
…
24 row(s) returned (0.28 sec)
rapids > select count(*) from nation;
  [1]
  ---
   25
```


IMPORT command with the “IF NOT EXISTS” clause

```
1 row(s) returned (0.09 sec)
rapids > IMPORT IF NOT EXISTS MOXE.* FROM 'node://db1/tpch_small_files' WITH
GUESS;
0 row(s) returned (0.70 sec)
```

The “SHOW TABLES” command shows that the “customer” and “region” tables were imported, and the row count for the “nation” table has not changed, showing that no data was imported into that table:

```
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
…
26 row(s) returned (0.27 sec)
rapids > select count(*) from nation;
  [1]
  ---
   25

1 row(s) returned (0.09 sec)
```

Example 3:

This example shows importing a set of files which were created using the EXPORT command (see 7.12.2):

```
rapids > EXPORT MOXE.* TO 'node://db1/tpch_small_file_with_headers' WITH
HEADER;

0 row(s) returned (8.33 sec)
```

Now drop the current tables and then import the files specifying the HEADER and GUESS options:

```
rapids > drop table moxe.lineitem;
0 row(s) returned (0.19 sec)
rapids > drop table moxe.orders;
0 row(s) returned (0.10 sec)
rapids > drop table moxe.customer;
```



```
0 row(s) returned (0.09 sec)
rapids > drop table moxe.supplier;
0 row(s) returned (0.09 sec)
rapids > drop table moxe.part;
0 row(s) returned (0.10 sec)
rapids > drop table moxe.partsupp;
0 row(s) returned (0.11 sec)
rapids > drop table moxe.nation;
0 row(s) returned (0.09 sec)
rapids > drop table moxe.region;
0 row(s) returned (0.09 sec)
rapids > IMPORT MOXE.* FROM 'node://db1/tpch_small_file_with_headers' WITH
HEADER, GUESS;
0 row(s) returned (16.26 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
…
31 row(s) returned (0.27 sec)
rapids > describe table region;
TABLE_NAME    COLUMN_NAME      DATA_TYPE      ORDINAL    IS_PARTITION_KEY
IS_NULLABLE    PRECISION    SCALE
----------
----------
REGION        'R_REGIONKEY'    INTEGER              0               false
true           64     NULL
REGION        'R_NAME'         VARCHAR              1               false
true         NULL     NULL
REGION        'R_COMMENT'      VARCHAR              2               false
true         NULL     NULL

3 row(s) returned (0.27 sec)
```

Example 4:

This example shows that when the tables already exist a bulk import operation will append to the existing tables.

Current row counts:

```
rapids > select count(*) from customer;
```



```
  [1]
  ---
   75

1 row(s) returned (0.06 sec)
rapids > select count(*) from lineitem;
   [1]
   ---
  3000

1 row(s) returned (0.06 sec)
rapids > select count(*) from nation;
  [1]
  ---
   25

1 row(s) returned (0.06 sec)
rapids > select count(*) from orders;
  [1]
  ---
  750

1 row(s) returned (0.06 sec)
rapids > select count(*) from part;
  [1]
  ---
  100

1 row(s) returned (0.07 sec)
rapids > select count(*) from partsupp;
  [1]
  ---
  400

1 row(s) returned (0.06 sec)
rapids > select count(*) from region;
  [1]
  ---
    5

1 row(s) returned (0.06 sec)
rapids > select count(*) from supplier;
  [1]
  ---
    5

1 row(s) returned (0.05 sec)
```


In this example the “IF EXISTS” clause is used to only import into those tables that already exist. The “GUESS” Property is set to FALSE (by default) so that all data will be read as polymorphic strings and then cast to the data type for each column. The “FILES” option is explicitly stated (although as mentioned earlier it is not needed because it is the default), and the FILTER option is specified to only include those files with the suffix “.csv”. The new row counts show that the number of rows in each table doubled as a result of the bulk import inserting the new data:

```
rapids > IMPORT IF EXISTS MOXE.* FROM FILES 'node://db1/tpch_small_files'
WITH FILTER='*.csv';
0 row(s) returned (9.94 sec)
rapids > select count(*) from customer;
  [1]
  ---
  150

1 row(s) returned (0.06 sec)
rapids > select count(*) from lineitem;
   [1]
   ---
  6000

1 row(s) returned (0.06 sec)
rapids > select count(*) from nation;
  [1]
  ---
   50

1 row(s) returned (0.06 sec)
rapids > select count(*) from orders;
   [1]
   ---
  1500

1 row(s) returned (0.06 sec)
rapids > select count(*) from part;
  [1]
  ---
  200

1 row(s) returned (0.05 sec)
rapids > select count(*) from partsupp;
  [1]
  ---
  800

1 row(s) returned (0.06 sec)
rapids > select count(*) from region;
  [1]
```



```
  ---
   10

1 row(s) returned (0.06 sec)
rapids > select count(*) from supplier;
  [1]
  ---
   10

1 row(s) returned (0.06 sec)
```

Example 5:

This command demonstrates the use of the “REPLACE” option to replace the existing data in the tables (by doing a “TRUNCATE” operation before importing the data) with new data.

Below are the current row counts for the tables:

```
rapids > select count(*) from customer;
  [1]
  ---
  150

1 row(s) returned (0.06 sec)
rapids > select count(*) from lineitem;
   [1]
   ---
  6000

1 row(s) returned (0.06 sec)
rapids > select count(*) from nation;
  [1]
  ---
   50

1 row(s) returned (0.06 sec)
rapids > select count(*) from orders;
   [1]
   ---
  1500

1 row(s) returned (0.06 sec)
rapids > select count(*) from part;
  [1]
  ---
  200

1 row(s) returned (0.05 sec)
rapids > select count(*) from partsupp;
  [1]
```



```
  ---
  800

1 row(s) returned (0.06 sec)
rapids > select count(*) from region;
  [1]
  ---
   10

1 row(s) returned (0.06 sec)
rapids > select count(*) from supplier;
  [1]
  ---
   10

1 row(s) returned (0.06 sec)
rapids > IMPORT MOXE.* REPLACE FROM 'node://db1/tpch_small_files';
0 row(s) returned (10.13 sec)
```

Note that the table counts below now reflect a single copy of the data:

```
rapids > select count(*) from customer;
  [1]
  ---
   75

1 row(s) returned (0.06 sec)
rapids > select count(*) from lineitem;
   [1]
   ---
  3000

1 row(s) returned (0.06 sec)
rapids > select count(*) from nation;
  [1]
  ---
   25

1 row(s) returned (0.06 sec)
rapids > select count(*) from orders;
  [1]
  ---
  750

1 row(s) returned (0.06 sec)
rapids > select count(*) from part;
  [1]
  ---
```



```
  100

1 row(s) returned (0.06 sec)
rapids > select count(*) from partsupp;
  [1]
  ---
  400

1 row(s) returned (0.06 sec)
rapids > select count(*) from region;
  [1]
  ---
    5

1 row(s) returned (0.06 sec)
rapids > select count(*) from supplier;
  [1]
  ---
    5

1 row(s) returned (0.06 sec)
```

7.10.2 Bulk IMPORT Using FILES Option With FILTER

The FILTER property allows the user to control which files are imported in a wildcard import operation and, optionally, how table names are created from the names of imported files. The FILTER value is a character string containing a Java regular expression (a "regex").

When performing a wildcard import, an IMPEX Connector examines each filename available from the import source. Only files whose names satisfy the FILTER regex are imported. (For a tutorial on Java regular expressions, see https://www.oracle.com/technical-resources/articles/java/regex.html)

Example 1:

Bulk import a set of files from the folder "/var/tmp/rapids/tpch_small_files" on RapidsDB Cluster node "db1".

Folder /var/tmp/rapids/tpch_small_files:

```
[rapids@db1 rapids]$ ls tpch_small_files
customer.csv  lineitem.csv  nation.csv  orders.csv  part.csv  partsupp.csv
region.csv  supplier.csv
```

```
rapids > show  tables;
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME
------------------
RAPIDS          SYSTEM         AUTHENTICATORS
RAPIDS          SYSTEM         AUTHENTICATOR_CONFIG
```



```
RAPIDS          SYSTEM          CATALOGS
RAPIDS          SYSTEM          COLUMNS
RAPIDS          SYSTEM          CONNECTORS
RAPIDS          SYSTEM          FEDERATIONS
RAPIDS          SYSTEM          INDEXES
RAPIDS          SYSTEM          NODES
RAPIDS          SYSTEM          PATTERN_MAPS
RAPIDS          SYSTEM          QUERIES
RAPIDS          SYSTEM          QUERY_STATS
RAPIDS          SYSTEM          SCHEMAS
RAPIDS          SYSTEM          SESSIONS
RAPIDS          SYSTEM          TABLES
RAPIDS          SYSTEM          TABLE_PROVIDERS
RAPIDS          SYSTEM          USERNAME_MAPS
RAPIDS          SYSTEM          USERS
RAPIDS          SYSTEM          USER_CONFIG

18 row(s) returned (0.22 sec)
rapids > IMPORT MOXE.* FROM 'node://db1/tpch_small_files' WITH
FILTER='*.csv',GUESS;
0 row(s) returned (12.17 sec)
```

This command imported all the files from folder "/var/tmp/rapids/tpch_small_files" with a file suffix of ".csv" creating new tables (see "show tables;" output below) of the same names (minus the ".csv" suffix) in the schema MOXE if they did not already exist and where the column names for any newly created table were set to "COL1", "COL2", etc, and where the data types were imputed from the data in the files (based on the "GUESS" property being set TRUE (see 7.9.2.5) (see "describe table" output below). If the tables already existed, the new data would be inserted alongside the existing data.

```
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
RAPIDS          SYSTEM         AUTHENTICATORS
RAPIDS          SYSTEM         AUTHENTICATOR_CONFIG
RAPIDS          SYSTEM         CATALOGS
RAPIDS          SYSTEM         COLUMNS
RAPIDS          SYSTEM         CONNECTORS
RAPIDS          SYSTEM         FEDERATIONS
RAPIDS          SYSTEM         INDEXES
RAPIDS          SYSTEM         NODES
```



```
RAPIDS          SYSTEM          PATTERN_MAPS
RAPIDS          SYSTEM          QUERIES
RAPIDS          SYSTEM          QUERY_STATS
RAPIDS          SYSTEM          SCHEMAS
RAPIDS          SYSTEM          SESSIONS
RAPIDS          SYSTEM          TABLES
RAPIDS          SYSTEM          TABLE_PROVIDERS
RAPIDS          SYSTEM          USERNAME_MAPS
RAPIDS          SYSTEM          USERS

CATALOG_NAME    SCHEMA_NAME     TABLE_NAME
------------------
RAPIDS          SYSTEM          USER_CONFIG

26 row(s) returned (0.22 sec)
rapids > describe table region;
TABLE_NAME    COLUMN_NAME    DATA_TYPE      ORDINAL    IS_PARTITION_KEY
IS_NULLABLE    PRECISION    SCALE COMMENT    PROPERTIES
-------
-------
REGION        COL1            INTEGER              0               false
true           64     NULL NULL       NULL
REGION        COL2            VARCHAR              1               false
true          NULL    NULL NULL       NULL
REGION        COL3            VARCHAR              2               false
true          NULL    NULL NULL       NULL

3 row(s) returned (0.20 sec)
```

Example 2:

Import the file whose name starts with the string “region” (this is the capturing group, see hilited text below), and create a table of that name. This example uses the same folder as the previous example, but this time only the “region” table will get imported:

```
rapids > drop table region;
0 row(s) returned (0.09 sec)
rapids > IMPORT MOXE.* FROM 'node://db1/tpch_small_files' WITH
FILTER='(region).*',GUESS;
0 row(s) returned (0.12 sec)
rapids > select count(*) from region;
  [1]
  ---
    5

1 row(s) returned (0.06 sec)
```


7.10.3 Bulk IMPORT Using FOLDERS Option

When doing a bulk import operation the "FOLDERS" option indicates that the path name specified in the import reference (see 7.6) refers to a folder, which contains a set of sub-folders to be imported. The name of each sub-folder is the name of the table where the imported data from the files in the sub-folder will be written.

Example 1:

This example shows the use of the FOLDERS option to import the data from a set of sub-folders where each sub-folder corresponds to a table of the same name.

Folder structure under parent folder /var/tmp/rapids/small_tpch:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/tpch_small/*
/var/tmp/rapids/tpch_small/customer:
customer.csv

/var/tmp/rapids/tpch_small/lineitem:
lineitem.csv

/var/tmp/rapids/tpch_small/nation:
nation.csv

/var/tmp/rapids/tpch_small/orders:
orders.csv

/var/tmp/rapids/tpch_small/part:
part.csv

/var/tmp/rapids/tpch_small/partsupp:
partsupp.csv

/var/tmp/rapids/tpch_small/region:
region.csv

/var/tmp/rapids/tpch_small/supplier:
supplier.csv:
```

The bulk import command below is using the "IF NOT EXISTS" clause to only import those tables that do not currently exist along with the "FOLDERS" option. As the tables are being created, the "GUESS" Property is set TRUE so that the IMPEX Connector will assign the data types. In this example only the "lineitem" table will be imported because all of the other tables already exist:

```
rapids > show  tables;
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME
------------------

MOXE            MOXE           CUSTOMER
```



```
MOXE            MOXE          NATION
MOXE            MOXE          ORDERS
MOXE            MOXE          PART
MOXE            MOXE          PARTSUPP
MOXE            MOXE          REGION
MOXE            MOXE          SUPPLIER
…

25 row(s) returned (0.22 sec)
rapids > select count(*) from customer;
  [1]
  ---
   75

1 row(s) returned (0.06 sec)

rapids > IMPORT IF NOT EXISTS MOXE.* FROM FOLDERS 'node://db1/tpch_small'
WITH GUESS;
0 row(s) returned (10.63 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
…
25 row(s) returned (0.23 sec)

rapids > select count(*) from lineitem;
   [1]
   ---
  3000

1 row(s) returned (0.06 sec)
rapids > select count(*) from customer;
  [1]
  ---
   75

1 row(s) returned (0.06 sec)
```

Example 2:

This example shows the use of the “REPLACE” option, where the tables being imported will first be truncated and then the data inserted.

Current row count for “lineitem” table:

```
rapids > select count(*) from lineitem;
   [1]
   ---
  3000

1 row(s) returned (0.06 sec)
```

Import command with “REPLACE” option specified. The “GUESS” Property is set to FALSE (by default) so that all data will be read as polymorphic strings and then cast to the data type for each column:

```
rapids > IMPORT MOXE.* REPLACE FROM FOLDERS 'node://db1/tpch_small' WITH
GUESS=FALSE;
0 row(s) returned (10.03 sec)
```

Current row count showing that only one copy of the data is in the table:

```
rapids > select count(*) from lineitem;
   [1]
   ---
  3000

1 row(s) returned (0.06 sec)
```

Example 3:

This example shows a bulk import where the input files include header records with the column names.

```
rapids@db1:/var/tmp/rapids$ ls tpch_small_folders_with_headers/*/*
tpch_small_folders_with_headers/CUSTOMER/customer.csv
tpch_small_folders_with_headers/PART/part.csv
tpch_small_folders_with_headers/LINEITEM/lineitem.csv
tpch_small_folders_with_headers/PARTSUPP/partsupp.csv
tpch_small_folders_with_headers/NATION/nation.csv
tpch_small_folders_with_headers/REGION/region.csv
tpch_small_folders_with_headers/ORDERS/orders.csv
tpch_small_folders_with_headers/SUPPLIER/supplier.csv
```

Below is the file “region.csv” showing the header row:

```
[rapids@db1 tpch_small_folders_with_headers]$ cat region/REGION.csv
R_REGIONKEY,R_NAME,R_COMMENT
1,UNITED STATES,adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
```



```
2,NORTH AMERICA,aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3,EUROPE,dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4,SOUTH AMERICA,csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5,ASIA,i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt
```

Below is the bulk import command with the "FOLDERS" option set, and the "HEADER" option set to indicate that each file in a sub-folder includes a header record with the column names:

```
rapids > drop table customer;
0 row(s) returned (0.10 sec)
rapids > drop table lineitem;
0 row(s) returned (0.10 sec)
rapids > drop table nation;
0 row(s) returned (0.10 sec)
rapids > drop table orders;
0 row(s) returned (0.10 sec)
rapids >  drop table part;
0 row(s) returned (0.09 sec)
rapids > drop table partsupp;
0 row(s) returned (0.09 sec)
rapids > drop table region;
0 row(s) returned (0.10 sec)
rapids > drop table supplier;
0 row(s) returned (0.09 sec)
rapids > IMPORT MOXE.* FROM FOLDERS 'node://db1/
tpch_small_folders_with_headers WITH HEADER;
0 row(s) returned (0.32 sec)
```

The output below shows the table definitions for two of the tables imported, "nation" and "region", where you can see that the column names in the header record were used:

```
rapids > describe table nation;
TABLE_NAME    COLUMN_NAME    DATA_TYPE      ORDINAL    IS_PARTITION_KEY
IS_NULLABLE    PRECISION    SCALE COMMENT    PROPERTIES
-------
-------
NATION        N_NATIONKEY    VARCHAR              0              false
true         NULL     NULL NULL       NULL
NATION        N_NAME         VARCHAR              1              false
true         NULL     NULL NULL       NULL
NATION        N_REGIONKEY    VARCHAR              2              false
true         NULL     NULL NULL       NULL
NATION        N_COMMENT      VARCHAR              3              false
true         NULL     NULL NULL       NULL

4 row(s) returned (0.17 sec)
rapids > describe table region;
```



```
TABLE_NAME    COLUMN_NAME    DATA_TYPE      ORDINAL    IS_PARTITION_KEY
IS_NULLABLE    PRECISION    SCALE COMMENT    PROPERTIES
-------
-------
REGION        R_REGIONKEY    VARCHAR              0               false
true         NULL     NULL NULL       NULL
REGION        R_NAME         VARCHAR              1               false
true         NULL     NULL NULL       NULL
REGION        R_COMMENT      VARCHAR              2               false
true         NULL     NULL NULL       NULL

3 row(s) returned (0.19 sec)
```

Example 4:
This example shows an example where an attempt is made to do a bulk import operation where the target tables are managed by different Connectors, which is not allowed:

Below are the current tables that are managed by the “MOXE” and “MOXE2” Connectors:

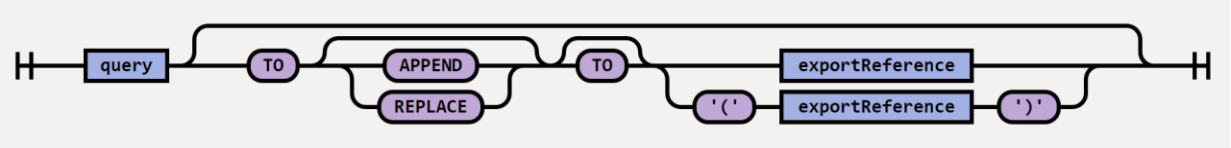
```
rapids > show tables;
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME
------------    -----------    ----------
MOXE            MOXE           CUSTOMER
MOXE            MOXE           LINEITEM
MOXE            MOXE           NATION
MOXE            MOXE           ORDERS
MOXE            MOXE           REGION
MOXE            MOXE           SUPPLIER
MOXE2           MOXE2          PART
MOXE2           MOXE2          PARTSUPP
…
29 row(s) returned (0.23 sec)
```

Below is the bulk import command that will attempt to do imports against all of the tables shown above, which is not allowed because the tables are managed by two different Connectors. The error message shows one table from each Connector which would have been imported into, which in this example are the “PART” table managed by the “MOXE2” Connector and “NATION” table managed by the “MOXE” Connector.

```
rapids > IMPORT * FROM FOLDERS 'node://db1/tpch_small';
Unexpected Exception:
Wildcard import to multiple Connectors: PART, NATION
```

7.11 EXPORT Using SELECT

This section covers the exporting of the results of a query to a file or folder.



7.11.1 EXPORT Using SELECT TO a File

This section provides examples for writing the results of a query to a specified file using the "TO" clause as shown below. In the examples below the hilited text is the export reference (see 7.7):

Example 1:

This example shows exporting the contents of a table, "moxe.region", to a file "region.csv", where the file is located in the folder "/var/tmp/rapids/tpch_small_file_backups" on RapidsDB Cluster node "db1".

Folder: /var/tmp/rapids/tpch_small_file_backups:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/tpch_small_file_backups
[rapids@db1 rapids]$
```

Query to export the data using the default "EXPORT" IMPEX Connector (see 7.8):

```
rapids > SELECT * FROM moxe.region TO
'node://db1/tpch_small_file_backups/region.csv';
0 row(s) returned (0.10 sec)
```

Folder "/var/tmp/rapids/tpch_small_file_backups" after the export, showing that the file has 5 records:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/tpch_small_file_backups
region.csv
[rapids@db1 rapids]$ cat /var/tmp/rapids/tpch_small_file_backups/region.csv |
wc -l
5
```

Example 2:

This example shows that the data being exported will by default be appended to the target.

Current contents of file

```
[rapids@db1 rapids]$ cat /var/tmp/rapids/tpch_small_file_backups/region.csv |
wc -l
5
```

```
rapids > SELECT * FROM moxe.region TO
'node://db1/tpch_small_file_backups/region.csv';
0 row(s) returned (0.10 sec)
```


New count:

```
[rapids@db1 rapids]$ cat /var/tmp/rapids/tpch_small_file_backups/region.csv |
wc -l
10
```

Example 3:

In this example, the “REPLACE” option is used, which will result in the target file getting deleted prior to writing the results of the query, and then the results of the query will get written to the file. In addition, the “HEADER” option is used which will result in the first record of the file being a header record which contains the names of the columns from the result set.

Current contents of file “region.csv”:

```
[rapids@db1 rapids]$ cat /var/tmp/rapids/tpch_small_file_backups/region.csv
1,UNITED STATES,adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2,NORTH AMERICA,aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3,EUROPE,dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4,SOUTH AMERICA,csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5,ASIA,i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt
1,UNITED STATES,adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2,NORTH AMERICA,aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3,EUROPE,dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4,SOUTH AMERICA,csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5,ASIA,i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt
```

Export command specifying the “REPLACE” and “HEADER” options:

```
rapids > SELECT * FROM moxe.region TO REPLACE
'node://db1/tpch_small_file_backups/region.csv' WITH HEADER;
0 row(s) returned (0.08 sec)
```

Contents of exported file, with a header record with the column names from the result set:

```
[rapids@db1 rapids]$ cat /var/tmp/rapids/tpch_small_file_backups/region.csv
R_REGIONKEY,R_NAME,R_COMMENT
1,UNITED STATES,adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2,NORTH AMERICA,aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3,EUROPE,dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4,SOUTH AMERICA,csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5,ASIA,i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt
```

Example 4:

This is the same as the previous example, except this time the BACKUP Property is set “true” which results in the “region.csv” file getting moved to a backup folder named “_.backup.<internal_timestamp>” and then the export of the query results is done.

where <internal_timestamp> is a numerical value for the timestamp when the query results were generated.

```
rapids > SELECT * FROM moxe.region TO REPLACE
'node://db1/tpch_small_file_backups/region.csv' WITH HEADER, BACKUP;

0 row(s) returned (0.08 sec)
```

Target folder with exported file along with backup folder “/_.backup.777214467718270673”, has the original “region.csv” file before the export was done.

```
[rapids@db1 tpch_small_file_backups]$ ls
/var/tmp/rapids/tpch_small_file_backups/*
/var/tmp/rapids/tpch_small_file_backups/region.csv

/var/tmp/rapids/tpch_small_file_backups/_.backup.777214467718270673:
region.csv
```

7.11.2 EXPORT Using SELECT TO a Folder

This section provides examples for writing the results of a query to a specified folder using the “TO FOLDER” clause as shown below. The query results are written to a file named:

- query_results_<internal_timestamp>.csv

where,

<internal_timestamp> is the timestamp when the query results were generated.

Example 1:

This example uses the “FOLDER” option to write the results of the specified query to the specified folder, “/var/tmp/rapids/query_results”. The file with the query results will be named “query.<timestamp>”, where <timestamp> is the timestamp when the query was executed.

Folder: /var/tmp/rapids/query_results

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/query_results
[rapids@db1 rapids]$
```

```
rapids > SELECT * FROM moxe.region TO FOLDER  'node://db1/query_results';
0 row(s) returned (0.08 sec)
```


Target folder, "/var/tmp/rapids/query_results", after export:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/query_results
query_results.3959867675829689450.csv

[rapids@db1 rapids]$ cat
/var/tmp/rapids/query_results/query_results.3959867675829689450.csv
1,UNITED STATES,adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2,NORTH AMERICA,aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3,EUROPE,dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4,SOUTH AMERICA,csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5,ASIA,i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt
```

Example 2:

This example shows a second query results file being written to the same target folder:

Current contents of target folder:

```
[rapids@db1 query_results]$ ls /var/tmp/rapids/query_results
query_results.2320079842552510970.csv
```

Export command:

```
rapids > SELECT * FROM moxe.customer WHERE customer.c_acctbal >0 AND EXISTS
(SELECT * FROM vip_customer WHERE vip_customer.c_custkey =
customer.c_custkey) TO FOLDER 'node://db1/query_results';

0 row(s) returned (0.41 sec)
```

Contents of target folder with new query results file:

```
[rapids@db1 query_results]$ ls /var/tmp/rapids/query_results
query_results.1910530767001465758.csv  query_results.2320079842552510970.csv
```

Example 3:

This example shows the use of the "REPLACE" and "BACKUP" options where the existing files (with a ".csv" suffix) from the specified folder are first moved to a backup folder and then the new query results files are written:

Current target folder:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/query_results
query_results.1500591431418200400.csv  query_results.3449950617860697261.csv
```


Export command with “REPLACE” option set:

```
rapids > SELECT * FROM moxe.customer WHERE customer.c_acctbal > 0 AND EXISTS
(SELECT * FROM vip_customer WHERE vip_customer.c_custkey =
customer.c_custkey) TO REPLACE FOLDER 'node://db1/query_results' WITH BACKUP;

0 row(s) returned (0.76 sec)
```

The backup folder will get created in the parent directory of the folder specified in the export reference, which in this case would be “/var/tmp/rapids”:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids
_.backup.4636680165345356631  query_results  SFSMALL  text
tpch_small_backup         tpch_small_files
formats                        SF100          single   tpch_small
tpch_small_file_backups   tpch_small_with_headers
```

The contents of the backup folder are the original files prior to the export:

```
[rapids@db1 rapids]$ ls
/var/tmp/rapids/_.backup.4636680165345356631/query_results

query_results.1500591431418200400.csv  query_results.3449950617860697261.csv
```

Example 4:

This example shows the use of the “REPLACE” option with “BACKUP=false” to first delete all existing files from the specified folder before writing the new query results file.

Contents of target folder with new query results file:

```
[rapids@db1 query_results]$ ls /var/tmp/rapids/query_results
query_results.1910530767001465758.csv  query_results.2320079842552510970.csv
```

Contents of the parent directory:

```
[rapids@db1 /var/tmp/rapids]$ ls /var/tmp/rapids
_.backup.4107715935291356825   SFSMALL  tpch_small
tpch_small_files               tpch_small_folders_with_headers
query_results                  text     tpch_small_file_backups
tpch_small_file_with_headers
```

Export command using “REPLACE” option:

```
rapids > SELECT * FROM moxe.customer WHERE customer.c_acctbal > 0 AND EXISTS
(SELECT * FROM vip_customer WHERE vip_customer.c_custkey =
customer.c_custkey) TO REPLACE FOLDER 'node://db1/query_results';
```



```
0 row(s) returned (0.37 sec)
```

Contents of target folder after export showing that two previous files were deleted:

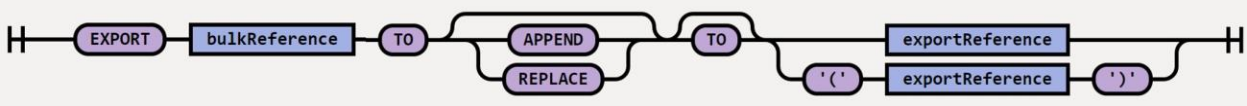
```
[rapids@db1 query_results]$ ls /var/tmp/rapids/query_results
query_results.9088214351952178143.csv
```

No new backup folder was created:

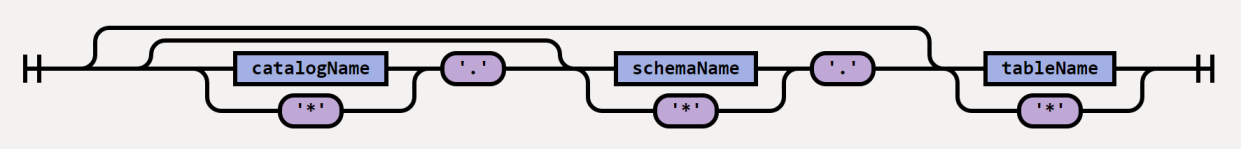
```
[rapids@db1 /var/tmp/rapids]$ ls /var/tmp/rapids
_.backup.4107715935291356825   SFSMALL  tpch_small
tpch_small_files               tpch_small_folders_with_headers
query_results                  text     tpch_small_file_backups
tpch_small_file_with_headers
```

7.12 Bulk EXPORT

The *EXPORT* statement is used for directly exporting multiple tables in a single request:



bulkReference:



Option	Required?	Default?	Description
bulkReference	Yes	N/A	Specifies the three-level (catalog/schema/table) names for the table(s) to be exported. Wildcards may be specified (using asterisk '*') for any of the name components. If catalog name and/or schema name are omitted, CURRENT_CATALOG and CURRENT_SCHEMA are used (if set).
APPEND	No	Yes	Append the exported data to any existing file or folder at the destination (as specified by the exportReference – see below). (Note: may not be supported for some Connectors and/or destinations.) This is the default behavior
REPLACE	No	No	Delete any existing files with a suffix of ".csv" (when the "FILES" option is specified in the export reference (see 11.8)) or all the files with a suffix of ".csv" in a sub-folder

			(when the "FOLDERS" option is specified in the export reference (see folder 7.7)) . If the "BACKUP" Property for the Connector (see 7.4) is "true" then instead of deleting the files in the folder (or sub-folder), the files will be moved to a backup folder.
exportReference	Yes	N/A	An Export Reference (see 7.7) identifying the destination for the exported data.

7.12.1 Backing Up Files/Sub-Folders When Doing a REPLACE

When doing a bulk export operation with the "REPLACE" option (see above), the user is requesting that the target files for the export (when the export reference is using the "FILES" option, which is the default - see 7.12.2 for examples), or the files in the target sub-folders when the export reference is using the "FOLDERS" option (see 7.12.3 for examples), are to be replaced with new copies. In order to allow the user to recover any replaced files, the IMPEX Connector supports the "BACKUP" Property (see 7.12.2 for examples), which when set to "true" results in the system moving the files to be replaced to a backup folder so that they can be recovered after the export operation if needed. See 7.12.1.1 for more information on backup with the "FILES"option, and 7.12.1.2 for more information on backup with the "FOLDERS" option. By default, all IMPEX Connectors have the "BACKUP" Property set "false" which means that all bulk export operations where the "REPLACE" option is specified no backup of the existing files will be done.

7.12.1.1 Backup for FILES option

When doing a backup for the FILES option (BACKUP=true), a backup folder will get created in the parent directory of the folder specified in the export reference, with the name of the backup folder being:

_.backup.<epoch timestamp>/<export folder>

where,

<epoch timestamp> is the Unix Epoch timestamp when the export command was executed

<export folder> is the name of the folder specified in the export reference in the bulk export command

All files with a suffix of ".csv" from the folder specified in the export reference in the bulk export command will be moved to the backup folder.

NOTE:
Since the "BACKUP" Property is set "false" by default, no backup of existing files will be performed when the "REPLACE" option is specified. If needed, the user can change the default setting for the "BACKUP" Property to "true" to ensure that a backup copy is made:

```
CREATE CONNECTOR EXPORT_NOBACKUP TYPE IMPEX WITH BACKUP,…;
```

or, by setting the "BACKUP" Property as part of the bulk export command:


```
EXPORT MOXE.* TO REPLACE 'node://db1/tpch_small_file_backups' WITH BACKUP, HEADER,
DELIMITER='|', ENCLOSED_BY='"';
```

See Examples 2 and 3 in section 7.13.2 below for more information. Example:

This example is using the default “EXPORT” Connector, which for this example is using the default setting for the “PATH” Property which is “/var/tmp/rapids”.

```
EXPORT MOXE.* TO REPLACE 'node://db1/tpch_small_file_backups' WITH HEADER, DELIMITER='|',
ENCLOSED_BY='"';
```

In this example any files with a suffix of “.csv” that are present in the folder specified in the export reference, which in this example would be “/var/tmp/rapids/tpch_small_backups”, would be moved to a folder in the parent directory, which in this example would be “/var/tmp/rapids”, where the backup folder would be named similarly to the following:

“/var/tmp/rapids/tpch_small_file_backups/_.backup.777214467718270673/ tpch_small_file_backups”

If needed, the user can then recover any needed files from the backup folder.

7.12.1.2 Backup for FOLDERS option

When doing a backup for the FOLDERS option (in the export reference), a backup folder will get created in each subfolder which holds the export files for each table being exported. The name of the backup folder will be:

_.backup.<epoch timestamp>

where,

<epoch timestamp> is the Unix Epoch timestamp when the export command was executed

All files with a suffix of “.csv” from the sub-folder moved to the backup folder. See example below for more details.

NOTE:
Since the “BACKUP” Property is set “false” by default, no backup of existing files will be performed when the “REPLACE” option is specified. If needed, the user can change the default setting for the “BACKUP” Property to “true” to ensure that a backup copy is made:

```
CREATE CONNECTOR EXPORT_NOBACKUP TYPE IMPEX WITH BACKUP,…;
```

or, by setting the “BACKUP” Property as part of the bulk export command:

```
rapids > EXPORT MOXE.* TO FOLDERS 'node://db1/tpch_small_backup' WITH HEADER, BACKUP;
```

See Examples 2 and 3 in section 7.12.3 below for more information.

Example:

This example is using the default “EXPORT” Connector, which for this example is using the default setting for the “PATH” Property which is “/var/tmp/rapids”. In this example, assume that one of the tables being exported in named “NATION”.

```
rapids > EXPORT MOXE.* TO FOLDERS 'node://db1/tpch_small_backup' WITH BACKUP, HEADER;
```

In this example any files with a suffix of “.csv” that are present in the sub-folder associated with each table being exported, such as “/var/tmp/rapids/tpch_small_backup/NATION” would be moved to a backup folder in the sub-folder directory for that table, which in this example would be named similarly to the following:

“/var/tmp/rapids/ tpch_small_backup/NATION/_.backup.777214467718270673/”

If needed, the user can then recover any needed files from any of the backup folders.

7.12.2 Bulk EXPORT Using FILES Option

The “FILES” option for a bulk export indicates that each table should be written out to the specified folder with name <table name>.csv.

Example 1:

This is an example of a simple bulk export where all of the tables from the schema “MOXE” are to be exported to the folder /var/tmp/rapids/tpch_small_file_backups, with each file having a header record with the column names.

Current contents of folder /var/tmp/rapids/tpch_small_file_backups:

```
[rapids@db1 tpch_small_file_backups]$ ls
/var/tmp/rapids/tpch_small_file_backups
AAREADme.txt
```

Tables to be exported:

```
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
MOXE            MOXE           VIP_CUSTOMER
```


Export command. The “FILES” option is not required because it is the default. The “HEADER” property is set to indicate that a header record is to be written for each exported table, and the “DELIMITER” property is set to the semicolon character and the “ENCLOSED_BY” property is set to the single quote character:

```
rapids > EXPORT MOXE.* TO 'node://db1/tpch_small_file_backups' WITH HEADER,
DELIMITER=';', ENCLOSED_BY="'";
0 row(s) returned (9.43 sec)
```

Folder after the export showing the exported tables:

```
[rapids@db1 tpch_small_file_backups]$ ls
AAREADME.txt  CUSTOMER.csv  LINEITEM.csv  NATION.csv  ORDERS.csv  PART.csv
PARTSUPP.csv  REGION.csv  SUPPLIER.csv  VIP_CUSTOMER.csv
```

Example of header record in one of the files, along with the field delimiter being set to the semicolon character and the enclosed_by character being set to single quote. Note that some of the varchar fields are enclosed in single quotes, and this is because those fields include the delimiter character which is a semicolon. The only time that character fields are enclosed is when they contain the delimiter character.:

```
[rapids@db1 tpch_small_file_backups]$ cat NATION.csv
N_NATIONKEY;N_NAME;N_REGIONKEY;N_COMMENT
1;UNITED STATES;1;adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2;CANADA;2;aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3;MEXICO;2;94iuakdjvoakdjvoadjogadhgkjagkjazdgkjzgd
4;GERMANY;3;zoidhgdjhtehgkhgnkjsdzdkjg
5;FRANCE;3;odijzoietjoizejtoizejgioazgkjan
6;ENGLAND;3;oaidjfozdjfoizjeofjelkjtkljlkjg
7;NORWAY;3;oaieurlajelgtkjalkgjlkajsgl
8;DENMARK;3;zokjkvlzkjdlgfjzdlkgjlakdjglkdjglkajglkjglkjdglkjfdlhk
9;SWEDEN;3;zkjdhfkjdvmn mv bmz lkfnvkzdfng
10;ITALY;3;'dkjlz;f;lsfkh;lakhr;lajlfkgakjdba'
11;SWITZERLAND;3;'ozkdjvdjtkjaldg;lg;lkfh;lks;flhadfbjfhd'
12;POLAND;3;kdjlkzdjtljelktjlzjlkrjlkdhkbafjebe
13;RUSSIA;3;'zflkzdjglzjglzdkg;lrdkh;ldfkh;dflkh'
14;CHILE;4;'akdjflkjd;ljfal;djgsdjglkdsfjgkjasjfbjha'
15;PERU;4;'ldkjflaje;ajlgdjzlkdgkzjdhfzbsjdhfb'
16;ARGENTINA;4;dkjzkldjlkgjzldgjkhghfjgfsg
17;BOLIVIA;4;alkjdflkajdljgaljglkajgljadfg
18;BRAZIL;4;jadalkjakdjlkjakgjalkfjglkajfgjalfg
19;GUYANA;4;zlkjdzkjdlkgjaldkgjlakjdglkajdglkjdfkgjlsdfkjhlksjfh
20;CHINA;5;lkdjaljdlkajdgkjafghskfdjghksjffhkjslkh
21;JAPAN;5;'skfjg;lskjfhksjkhjsjhakjbgbgbs'
22;INDIA;5;'kadjflkajdlkjgs;lkfjglskfjglksfjhks'
```



```
23;PHILLIPINES;5;'aldkjflkadjflkjadvjz;lvlkdlgjakjdg'
24;THAILAND;5;aldkjflakdjglkajgflkfjgkjsflkjgalkjfg
25;SINGAPORE;5;dlkjalkdjflkajdlkznvknzkdfglkzjdfhglkjflkh
```

Example 2:

This example shows the use of the “REPLACE” option to replace the existing files in the specified folder, (with the default “BACKUP” option set “false”), which will result in the existing files in the target folder getting deleted prior to the export of the tables.

Current contents of target folder:

```
[rapids@db1 tpch_small_file_backups]$ ls -l
/var/tmp/rapids/tpch_small_file_backups
total 145032
-rw-rw-r--. 1 rapids rapids        20 Sep 20 20:21 AAREADME.txt
-rw-rw-r--. 1 rapids rapids   4516901 Sep 20 20:52 CUSTOMER.csv
-rw-rw-r--. 1 rapids rapids 125657939 Sep 20 20:52 LINEITEM.csv
-rw-rw-r--. 1 rapids rapids        85 Sep 20 20:52 MYTABLE.csv
-rw-rw-r--. 1 rapids rapids      2348 Sep 20 20:52 NATION.csv
-rw-rw-r--. 1 rapids rapids   7550516 Sep 20 20:52 ORDERS.csv
-rw-rw-r--. 1 rapids rapids   4526668 Sep 20 20:52 PART.csv
-rw-rw-r--. 1 rapids rapids   4739265 Sep 20 20:52 PARTSUPP.csv
-rw-rw-r--. 1 rapids rapids        82 Sep 20 20:52 REGION2.csv
-rw-rw-r--. 1 rapids rapids       439 Sep 20 20:52 REGION.csv
-rw-rw-r--. 1 rapids rapids      3344 Sep 20 20:52 SPECIAL_CUSTOMER.csv
-rw-rw-r--. 1 rapids rapids   1478162 Sep 20 20:52 SUPPLIER.csv
-rw-rw-r--. 1 rapids rapids      4601 Sep 20 20:52 VIP_CUSTOMER.csv
```

Export command with “REPLACE” option using the default “BACKUP” Property (=”false”):

```
rapids > EXPORT MOXE.* TO REPLACE 'node://db1/tpch_small_file_backups' WITH
HEADER, DELIMITER='|', ENCLOSED_BY="'";

0 row(s) returned (9.06 sec)
```

Contents of target folder after the export showing new copies of the exported files and no backups:

```
[rapids@db1 tpch_small_file_backups]$ ls -l
/var/tmp/rapids/tpch_small_file_backups
total 145032
-rw-rw-r--. 1 rapids rapids        20 Sep 20 20:21 AAREADME.txt
-rw-rw-r--. 1 rapids rapids   4516901 Sep 20 21:12 CUSTOMER.csv
-rw-rw-r--. 1 rapids rapids 125657939 Sep 20 21:12 LINEITEM.csv
-rw-rw-r--. 1 rapids rapids        85 Sep 20 21:12 MYTABLE.csv
-rw-rw-r--. 1 rapids rapids      2348 Sep 20 21:12 NATION.csv
-rw-rw-r--. 1 rapids rapids   7550516 Sep 20 21:12 ORDERS.csv
-rw-rw-r--. 1 rapids rapids   4526668 Sep 20 21:12 PART.csv
```



```
-rw-rw-r--. 1 rapids rapids   4739265 Sep 20 21:12 PARTSUPP.csv
-rw-rw-r--. 1 rapids rapids        82 Sep 20 21:12 REGION2.csv
-rw-rw-r--. 1 rapids rapids       439 Sep 20 21:12 REGION.csv
-rw-rw-r--. 1 rapids rapids      3344 Sep 20 21:12 SPECIAL_CUSTOMER.csv
-rw-rw-r--. 1 rapids rapids   1478162 Sep 20 21:12 SUPPLIER.csv
-rw-rw-r--. 1 rapids rapids      4601 Sep 20 21:12 VIP_CUSTOMER.csv
```

Example 3:

This example shows the use of the “REPLACE” option to replace the existing files (ending in “.csv”) in the specified folder, with the “BACKUP” option set to “true”, which will result in the existing files being moved to a backup folder so that they can be recovered in the future if needed (see 7.12.1) for more information).

Current backup folder:

```
[rapids@db1 tpch_small_file_backups]$ ls
/var/tmp/rapids/tpch_small_file_backups
total 145032
-rw-rw-r--. 1 rapids rapids        20 Sep 20 20:21 AAREADME.txt
-rw-rw-r--. 1 rapids rapids   4516901 Sep 20 20:46 CUSTOMER.csv
-rw-rw-r--. 1 rapids rapids 125657939 Sep 20 20:46 LINEITEM.csv
-rw-rw-r--. 1 rapids rapids        85 Sep 20 20:46 MYTABLE.csv
-rw-rw-r--. 1 rapids rapids      2348 Sep 20 20:46 NATION.csv
-rw-rw-r--. 1 rapids rapids   7550516 Sep 20 20:46 ORDERS.csv
-rw-rw-r--. 1 rapids rapids   4526668 Sep 20 20:46 PART.csv
-rw-rw-r--. 1 rapids rapids   4739265 Sep 20 20:46 PARTSUPP.csv
-rw-rw-r--. 1 rapids rapids        82 Sep 20 20:46 REGION2.csv
-rw-rw-r--. 1 rapids rapids       439 Sep 20 20:46 REGION.csv
-rw-rw-r--. 1 rapids rapids      3344 Sep 20 20:46 SPECIAL_CUSTOMER.csv
-rw-rw-r--. 1 rapids rapids   1478162 Sep 20 20:46 SUPPLIER.csv
-rw-rw-r--. 1 rapids rapids      4601 Sep 20 20:46 VIP_CUSTOMER.csv
```

Bulk export command with the “REPLACE” option specified, and the “BACKUP” Property set:

```
rapids > EXPORT MOXE.* TO REPLACE 'node://db1/tpch_small_file_backups' WITH
BACKUP, HEADER, DELIMITER='|', ENCLOSED_BY="'";
0 row(s) returned (9.29 sec)
```

Contents of target folder after the export. Note that there are new copies of the files for the exported tables, and that there is a backup folder, “_.backup.8186816724347336840” that contains the original files:

```
[rapids@db1 tpch_small_file_backups]$ ls -l
/var/tmp/rapids/tpch_small_file_backups
total 145032
```



```
drwx------. 2 rapids rapids       246 Sep 20 20:52
_.backup.8186816724347336840
-rw-rw-r--. 1 rapids rapids        20 Sep 20 20:21 AAREADME.txt
-rw-rw-r--. 1 rapids rapids   4516901 Sep 20 20:52 CUSTOMER.csv
-rw-rw-r--. 1 rapids rapids 125657939 Sep 20 20:52 LINEITEM.csv
-rw-rw-r--. 1 rapids rapids        85 Sep 20 20:52 MYTABLE.csv
-rw-rw-r--. 1 rapids rapids      2348 Sep 20 20:52 NATION.csv
-rw-rw-r--. 1 rapids rapids   7550516 Sep 20 20:52 ORDERS.csv
-rw-rw-r--. 1 rapids rapids   4526668 Sep 20 20:52 PART.csv
-rw-rw-r--. 1 rapids rapids   4739265 Sep 20 20:52 PARTSUPP.csv
-rw-rw-r--. 1 rapids rapids       439 Sep 20 20:52 REGION.csv
-rw-rw-r--. 1 rapids rapids   1478162 Sep 20 20:52 SUPPLIER.csv
-rw-rw-r--. 1 rapids rapids      4601 Sep 20 20:52 VIP_CUSTOMER.csv
```

Here is the backup folder with the original files:

```
[rapids@db1 tpch_small_file_backups]$ ls -l
/var/tmp/rapids/tpch_small_file_backups/_.backup.8186816724347336840
total 145032
-rw-rw-r--. 1 rapids rapids   4516901 Sep 20 20:46 CUSTOMER.csv
-rw-rw-r--. 1 rapids rapids 125657939 Sep 20 20:46 LINEITEM.csv
-rw-rw-r--. 1 rapids rapids      2348 Sep 20 20:46 NATION.csv
-rw-rw-r--. 1 rapids rapids   7550516 Sep 20 20:46 ORDERS.csv
-rw-rw-r--. 1 rapids rapids   4526668 Sep 20 20:46 PART.csv
-rw-rw-r--. 1 rapids rapids   4739265 Sep 20 20:46 PARTSUPP.csv
-rw-rw-r--. 1 rapids rapids       439 Sep 20 20:46 REGION.csv
-rw-rw-r--. 1 rapids rapids   1478162 Sep 20 20:46 SUPPLIER.csv
-rw-rw-r--. 1 rapids rapids      4601 Sep 20 20:46 VIP_CUSTOMER.csv
```

7.12.3 Bulk EXPORT Using FOLDERS Option

The “FOLDERS” option for a bulk export indicates that each table should be written out in a file in a separate sub-folder (under the folder name specified in the export reference) of the same name. The name of the file for the exported table will be: <table name>>internal timestamp>.csv, for example “SUPPLIER.5674361502309579557.csv.”

The following examples all assume that the schema “MOXE” has the following tables: CUSTOMER, LINEITEM, NATION, ORDERS, PART, PARTSUPP, REGION, SUPPLIER, and VIP_CUSTOMER

Example 1:

This is an example of a bulk export where all of the tables from the schema “MOXE” are to be exported to sub-folders under the folder “/var/tmp/rapids/tpch_small_backup”, with each export file having a header record with the column names.

Current contents of target folder:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/tpch_small_folder_backup
```



```
[rapids@db1 rapids]$
```

Export command with the “FOLDERS” option

```
rapids > EXPORT MOXE.* TO FOLDERS 'node://db1/tpch_small_folder_backup' WITH
HEADER;
0 row(s) returned (9.43 sec)
```

Contents of target folder after export:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/tpch_small_folder_backup/*
/var/tmp/rapids/tpch_small_folder_backup/CUSTOMER:
customer.7842865417157613111.csv

/var/tmp/rapids/tpch_small_folder_backup/LINEITEM:
lineitem.5733617788919186767.csv

/var/tmp/rapids/tpch_small_folder_backup/NATION:
nation.1554588814455789912.csv

/var/tmp/rapids/tpch_small_folder_backup/ORDERS:
orders.6236921272679811628.csv

/var/tmp/rapids/tpch_small_folder_backup/PART:
part.5359586359844433055.csv

/var/tmp/rapids/tpch_small_folder_backup/PARTSUPP:
partsupp.9185786343941211758.csv

/var/tmp/rapids/tpch_small_folder_backup/REGION:
region.7738184263372301381.csv

/var/tmp/rapids/tpch_small_folder_backup/SUPPLIER:
supplier.4865294074583220638.csv

/var/tmp/rapids/tpch_small_folder_backup/VIP_CUSTOMER:
vip_customer.6098121580374720698.csv
```

Example 2:
This example shows the use of the “REPLACE” option which will result in the existing files (ending in “.csv”) in the sub-folder for each table being deleted prior to the export being executed.

Below is the current contents of one of the sub-folders for one of the tables (MOXE.LINEITEM)being exported:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/tpch_small_folder_backup/LINEITEM
lineitem.6482498489887946522.csv
```


Export command with the “REPLACE” option :

```
rapids > EXPORT MOXE.* TO REPLACE FOLDERS
'node://db1/tpch_small_folder_backup' WITH HEADER;

0 row(s) returned (9.06 sec)
```

Contents of the target folder showing the sub-folders for each table, and then the contents of the sub-folder for the “LINEITEM” table showing that a new copy of the exported file has been created, with no backup folder getting created:

```
[rapids@db1 tpch_small_folder_backup]$ ls
CUSTOMER  LINEITEM  NATION  ORDERS  PART  PARTSUPP  REGION  SUPPLIER
VIP_CUSTOMER
[rapids@db1 tpch_small_backup]$ ls LINEITEM
lineitem.7547050142483739703.csv
```

Example 3:

This example again shows the use of the “REPLACE” option to replace the existing files (ending in “.csv”) in the sub-folder for each table, but in this example the “BACKUP” option is set which results in the existing files being moved to a backup folder so that they can be recovered in the future if needed (see 11.13.1.2 for more information). The backup folder will be created in the folder specified in the export reference.

Below is the current contents of one of the sub-folders for one of the tables (MOXE.LINEITEM)being exported:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/tpch_small_folder_backup/LINEITEM
lineitem.6260590681045515755.csv
```

Export command specifying the “REPLACE” option:

```
rapids > EXPORT MOXE.* TO REPLACE FOLDERS
'node://db1/tpch_small_folder_backup' WITH HEADER, BACKUP;

0 row(s) returned (9.06 sec)
```

Contents of sub-folder after the export showing the new export file:

```
[rapids@db1 rapids]$ ls /var/tmp/rapids/tpch_small_backup/LINEITEM
lineitem.6482498489887946522.csv
```

Contents of folder, “/var/tmp/rapids/tpch_small_backup”, specified in the export reference, showing the backup folder that was created, “_.backup.6582410041113988538”:


```
[rapids@db1 rapids]$ ls /var/tmp/rapids/tpch_small_folder_backup
_.backup.6582410041113988538 CUSTOMER LINEITEM NATION ORDERS PART
PARTSUPP REGION SUPPLIER VIP_CUSTOMER
```

Contents of backup folder showing all of the original sub-folders:

```
[rapids@db1 rapids]$ ls
/var/tmp/rapids/tpch_small_folder_backup/_.backup.6582410041113988538
CUSTOMER LINEITEM NATION ORDERS PART PARTSUPP REGION SUPPLIER
VIP_CUSTOMER
```

Sample contents of the backup LINEITEM sub-folder from the backup showing that it contains the original export file for the "LINEITEM" table:

```
[rapids@db1 rapids]$ ls
/var/tmp/rapids/tpch_small_folder_backup/_.backup.6582410041113988538/LINEITE
M
lineitem.6260590681045515755.csv
```

7.13 Error Handling

7.13.1 ERROR_PATH

The "ERROR_PATH" Property (see 7.4) specifies the fully qualified path name to use as the base path for the error files generated if an import operation fails. By default, the ERROR_PATH is set to "/var/tmp/rapids_errors". For each failed import, a sub-folder will be created in the folder specified by the "ERROR_PATH":

The sub-folder name is of the form:

SSN_<session number>_<node name>_<query number>

where,

<session number> is the session number for the query that failed

<node name> is the name of the RapidsDB cluster node where the query was submitted

<query number> is the query number for that session

For example, "SSN_2_DB1_67" indicates that this error occurred on session #2 on the RapidsDB Cluster node "DB1" and it was query #67.

The sub-folder will include two files:

1 A log file containing details on the conversion errors, one line per error. The format for the log file name is:

<source>-messages.log

where,

<source> is used to identify the source file or folder with the errors, and has the format:

node___<node name>_<path name>-messages.log

where,

<node name> is the RapidsDB Cluster node name where the input file or folder resides

<path name> is the path name to the source file or folder

Example:

```
rapids > insert into region_b SELECT * FROM (csv_header::
'node://db1/SFSMALL/regionPipe.csv' );

Error: import errors (5) in /var/tmp/rapids_errors/SSN_1_DB1_234
```

In this example, the errors occurred in the input file "/var/tmp/rapids/SFSMALL/regionPipe.csv" on RapidsDB cluster node "db1", and the details for the errors will be in the folder "/var/tmp/rapids_errors/SSN_1_DB1_234" on the same node as the input file, which is RapidsDB cluster node "db1". The log file will be the file /var/tmp/rapids_errors/SSN_1_DB1_234/node____db1_var_tmp_rapids_SFSMALL_regionPipe_csv-messages.log:

```
[rapids@db1 tpch_small_backup]$ cat
/var/tmp/rapids_errors/SSN_1_DB1_234/node__db1_var_tmp_rapids_SFSM
ALL_regionPipe_csv-messages.log
segment 0 line 1: java.lang.NumberFormatException: For input
string: "UNITED STATES"
segment 0 line 2: java.lang.NumberFormatException: For input
string: "NORTH AMERICA,aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd"
segment 0 line 3: java.lang.NumberFormatException: For input
string: "EUROPE"
segment 0 line 4: java.lang.NumberFormatException: For input
string: "SOUTH AMERICA"
segment 0 line 5: java.lang.NumberFormatException: For input
string: "ASIA"
```


2 A data file containing the records with the conversion errors, where the records in the data file match up with the error line in the log file described in the previous section. The format for the data file name follows the same format as for the log file:

<source>-records.csv

where,

<source> is used to identify the source file or folder with the errors, and has the format:

node___<node name>_<path name>-records.csv

where,

<node name> is the RapidsDB Cluster node name where the file or folder resides

<path name> is the path name to the source file or folder

Example:

From the previous example, the data file with the error records will be the file: "/var/tmp/rapids_errors/SSN_1_DB1_234/node____db1_var_tmp_rapids_SFSMALL_regionPipe_csv-records.csv":

```
[rapids@db1 tpch_small_backup]$ cat
/var/tmp/rapids_errors/SSN_1_DB1_234/node__db1_var_tmp_rapids_SFSM
ALL_regionPipe_csv-records.csv
1|UNITED
STATES|adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2|NORTH AMERICA,aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3|EUROPE|dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4|SOUTH AMERICA|csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5|ASIA|i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt
```

See 11.13.3 for more examples

The default for the "ERROR_PATH" is "/var/tmp/rapids_errors", but it can be changed either at the Connector level or as part of the import command:

```
rapids > create connector csv_error type impex with
error_path='/data/errors';
0 row(s) returned (2.31 sec)
```

```
rapids > insert into bad2 select col1, col2 from
('node://db1/text/lead_trail_blanks.csv' WITH
ERROR_PATH='/var/tmp/dcerrors');
Unexpected Exception:
import errors (6) in /var/tmp/dcerrors/SSN_2_DB1_76
```


7.13.2 ERROR_LIMIT

The "ERROR_LIMIT" Property specifies the maximum number of allowable errors for an import. Once the limit is reached the import operation will terminate. By default, the limit is set to ten. The possible values for ERROR_LIMIT are:

- -1 No limit, the import will continue regardless of the number of errors
- 0 The import will terminate on the first error
- >0 The import will terminate after the specified number

Example 1:

This example shows an import hitting the default limit:

```
rapids > create table moxe.nation_a(c1 integer, c2 integer, c3 integer, c4
integer);
0 row(s) returned (0.12 sec)
rapids > insert into moxe.nation_a SELECT * FROM
('node://db1/SFSMALL/nation.csv' );
Unexpected Exception:
Error limit (10) reached:
/var/tmp/rapids_errors/SSN_1_DB1_241/node___db1_var_tmp_rapids_SFSMALL_nation
_csv-messages.log
```

Example 2:

This example shows the effect of increasing the limit:

```
rapids > insert into moxe.nation_a SELECT * FROM
('node://db1/SFSMALL/nation.csv' WITH ERROR_LIMIT=100);

Error: import errors (25) in /var/tmp/rapids_errors/SSN_1_DB1_242
```

7.13.3 Data Conversion Errors

As described in the section 7.13.1, when data conversion errors happen, a sub-folder will be created in the folder referenced by the "ERROR_PATH" property, with two files created in that sub-folder, a "*.log" log file with details of the errors and a "*.csv" file with the errant data records.

The following examples show how the errors are reported for an INSERT … SELECT, bulk import using the "FILES" option and a bulk insert using the "FOLDERS" option.

Example 1 INSERT … SELECT:

In this example, an attempt was made to insert a text field into an integer column:

```
rapids > create table moxe.REGION_B (
       > r_regionkey integer NOT NULL,
       > r_id integer,
       > r_comment varchar(152)
       > ) PARTITION(r_regionkey);
```



```
0 row(s) returned (0.10 sec)

rapids > insert into region_b SELECT * FROM (csv_header::
'node://db1/SFSMALL/regionPipe.csv' );
Error: import errors (5) in /var/tmp/rapids_errors/SSN_1_DB1_243
```

The error message above indicates that the log file containing a description of the errors will be in the log file:
"/var/tmp/rapids_errors/SSN_1_DB1_243/ node___db1_var_tmp_rapids_SFSMALL_regionPipe_csv-messages.log"

```
[rapids@db1 SSN_1_DB1_243]$ cat
/var/tmp/rapids_errors/SSN_1_DB1_243/node__db1_var_tmp_rapids_SFSMALL_region
Pipe_csv-messages.log
segment 0 line 1: java.lang.NumberFormatException: For input string: "UNITED
STATES"
segment 0 line 2: java.lang.NumberFormatException: For input string: "NORTH
AMERICA,aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd"
segment 0 line 3: java.lang.NumberFormatException: For input string: "EUROPE"
segment 0 line 4: java.lang.NumberFormatException: For input string: "SOUTH
AMERICA"
segment 0 line 5: java.lang.NumberFormatException: For input string: "ASIA"
```

The data associated with the error messages will be in the file "/var/tmp/rapids_errors/SSN_1_DB1_243/node____db1_var_tmp_rapids_SFSMALL_regionPipe_csv-records.csv"

```
[rapids@db1 SSN_1_DB1_243]$ cat
/var/tmp/rapids_errors/SSN_1_DB1_243/node__db1_var_tmp_rapids_SFSMALL_region
Pipe_csv-records.csv
1|UNITED STATES|adknladnganfbmanlgnalkfnglkajglkafjglksjfglkajfgkjadg
2|NORTH AMERICA|aldkjlakngkjangkjnfkngakldflkadjlkajdlkajd
3|EUROPE|dxldgkzcsoisjoicnkjebfjhqwgrygwuihvokjdgojdvpds
4|SOUTH AMERICA|csvbdcavbscdbvacdhvjhxdkgnlkglfglksdnja shc asbvda
5|ASIA|i4y5qiuyrqghrushfxhghirohtiuehtaytaiuerytaiurt
```

The log file is indicating that there were conversion errors in the second data fields, where the data was a text string whereas the table was expecting an integer.

Example 2 Bulk IMPORT with "FILES" option:

This example shows how errors are reported when doing a bulk import using the "FILES" option:

```
rapids > IMPORT MOXE.* REPLACE FROM 'node://db1/tpch_small_files';
Unexpected Exception:
Import partially succeeded: succeeded: 7, failed: 1
```



```
    Error limit (10) reached:
/var/tmp/rapids_errors/SSN_3_DB1_68/node__db1_var_tmp_rapids_tpch_small_file
s_nation_csv-messages.log
```

The error message above indicates that the log file containing a description of the errors will be in the file:
"/var/tmp/rapids_errors/SSN_3_DB1_68/node____db1_var_tmp_rapids_tpch_small_files_nation_csv-messages.log"

Below is the content of the log file:

```
[rapids@db1 rapids]$ cat
/var/tmp/rapids_errors/SSN_3_DB1_68/node__db1_var_tmp_rapids_tpch_small_file
s_nation_csv-messages.log
segment 0 line 1: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
segment 0 line 2: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
segment 0 line 3: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
segment 0 line 4: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
segment 0 line 5: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
segment 0 line 6: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
segment 0 line 7: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
segment 0 line 8: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
segment 0 line 9: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
segment 0 line 10: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
```

NOTE:

The column number shown is zero-based, and so the error refers to the second field in the input data. In the above example, the input data with the reported error will be the second data field of the input record (column 1).

The data associated with the error messages will be in the file:
/var/tmp/rapids_errors/SSN_3_DB1_68/node____db1_var_tmp_rapids_tpch_small_files_nation_csv-records.csv

Contents of the above file, with the data fields in error hilited:


```
[rapids@db1 rapids]$ cat
/var/tmp/rapids_errors/SSN_3_DB1_68/node__db1_var_tmp_rapids_tpch_small_file
s_nation_csv-records.csv
0,ALGERIA,0, haggle. carefully final deposits detect slyly agai
7,GERMANY,3,l platelets. regular accounts x-ray: unusual\, regular acco
24,UNITED STATES,1,y final packages. slow foxes cajole quickly. quickly
silent platelets breach ironic accounts. unusual pinto be
3,CANADA,1,eas hang ironic\, silent packages. slyly regular packages are
furiously over the tithes. fluffily bold
14,KENYA,0, pending excuses haggle furiously deposits. pending\, express
pinto beans wake fluffily past t
20,SAUDI ARABIA,4,ts. silent requests haggle. closely express packages sleep
across the blithely
2,BRAZIL,1,y alongside of the pending deposits. carefully special packages
are about the ironic forges. slyly special
22,RUSSIA,3, requests against the platelets use never according to the
quickly regular pint
5,ETHIOPIA,0,ven packages wake quickly. regu
12,JAPAN,2,ously. final\, express gifts cajole a
```

Example 3 Bulk IMPORT with "FOLDERS" option:
This example shows how errors are reported when doing a bulk import using the "FILES" option:

```
rapids > import MOXE.* FROM FOLDERS 'node://db1/tpch_small';
Unexpected Exception:
Import partially succeeded: succeeded: 7, failed: 1
    Error limit (10) reached:
/var/tmp/rapids_errors/SSN_3_DB1_69/node__db1_var_tmp_rapids_tpch_small_nati
on-messages.log
```

The error message above indicates that the log file containing a description of the errors will be in the file:
"/var/tmp/rapids_errors/SSN_3_DB1_69/node___db1_var_tmp_rapids_tpch_small_nation-messages.log"
and the associated data file will be in the file:
"/var/tmp/rapids_errors/SSN_3_DB1_69/node___db1_var_tmp_rapids_tpch_small_nation-records.csv"

As for the "FILES" option, the column number reported in the log file will be zero-based, for example:

```
segment 0 line 1: com.rapidsdata.impex.ImpexParseException: Invalid
characters between delimiter ( , ) and enclosed_by ( " ) column: 1, integer
```

Would refer to the second data field in the input data.

7.13.4 Mismatched Number of Fields and Columns on INSERT

The following error will be returned when the number of fields in the data file being imported does not match the number of columns in the target table:

Unexpected Exception:
Line 1 position 1: Column lists differ, x column(s) vs y column(s).

where, x is the number of columns in the import file and y is the number of columns in the target table

Example:
Target table with 3 columns:

```
rapids > create table moxe.bad1(c1 integer, c2 integer, c3 integer);
0 row(s) returned (0.09 sec)
rapids > insert into bad1 select * from
('node://db1/text/lead_trail_blanks.csv');
Unexpected Exception:
Line 1 position 1: Column lists differ, 4 column(s) vs 3 column(s).
```

Import file, with 4 fields:

```
[rapids@db1 text]$ cat lead_trail_blanks.csv
1,    4 leading blanks,3 trailing blanks   ,1
2,A2345678901234567890,A1234567890123456789,2
3,"    4 leading blanks","3 trailing blanks   ",3
```

7.13.5 Wildcard import to multiple connectors

This error occurs when an attempt is made to do a bulk import operation where the target tables are managed by different Connectors, which is not allowed:

Example:
Below are the current tables that are managed by the “MOXE” and “MOXE2” Connectors:

```
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
MOXE2           MOXE2          PART
MOXE2           MOXE2          PARTSUPP
…
29 row(s) returned (0.23 sec)
```

Below is the bulk import command that will attempt to do imports against all of the tables shown above, which is not allowed because the tables are managed by two different Connectors. The error message

shows one table from each Connector which would have been imported into, which in this example are the “PART” table managed by the “MOXE2” Connector and “NATION” table managed by the “MOXE” Connector.

```
rapids > IMPORT * FROM FOLDERS 'node://db1/tpch_small';
Unexpected Exception:
Wildcard import to multiple Connectors: PART, NATION
```

8 REFRESH Command

The REFRESH command must be used when the underlying table metadata in a Data Store has been changed, where the change was not the result of a CREATE or DROP table request that was sent from RapidsDB (see section 6) and the user wishes to access the metadata information from RapidsDB. An example would be when a new table is created in Postgres using the native Postgres psql command interface.

The syntax for the refresh command is:

```
refresh [<Connector>];
```

If the Connector name is specified then the refresh command will only be applied to that Connector, if the Connector name is omitted then the refresh command will be applied to all enabled Connectors.

9 SYSTEM METADATA TABLES

9.1 OVERVIEW

RapidsDB provides a set of system metadata tables which provide metadata information about the RapidsDB system which is similar to the information provided by the ANSI Information Schema. The system metadata tables reside in the RAPIDS.SYSTEM catalog and schema. Each Federation has a METADATA Connector that maintains the system metadata tables for that Federation. The table below lists the system metadata tables:

Table Name	Description
NODES	A list of all of the nodes in the RapidsDB Cluster
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 and views that can be accessed from the current Federation.
INDEXES	Metadata for any indexes defined on tables 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, including any duplicates.
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 all the active queries
QUERY_STATS	Query statistics for all the active queries. This table is not fully operational as of this release and should be ignored

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;

9.2 NODES Table

The NODES table contains a list of the nodes in the RapidsDB Cluster. The table below shows the columns in the NODES table:

Column Name	Description
NODE_NAME	The name assigned by the user to this node
IS_DQC	Set to 'true' if this node is the DQC node, otherwise set to false
HOSTNAME	The host name or ip address for this node
CLIENT_PORT	The port number that the wireline protocol is listening on, which will be used by the RapidsDB Unified JDBC Driver for connecting to the RapidsDB cluster
CLUSTER_PORT	The port number that this node will be listening on
INSTALLATION_DIR	The installation directory for the RapidsDB Cluster software
WORKING_DIR	The working directory used for the RapidsDB Cluster software

Example:

```
rapids > select * from nodes;
NODE_NAME     IS_DQC HOSTNAME          CLIENT_PORT   CLUSTER_PORT INSTALLATION_DIR   WORKING_DIR
---------     ------ --------          -----------   ------------ ----------------   -----------
DB1             true 192.168.1.98             4333           4334 /opt/rdp           /opt/rdp/current
DB2            false 192.168.1.76             4333           4334 /opt/rdp           /opt/rdp/current
DB3            false 192.168.1.56             4333           4334 /opt/rdp           /opt/rdp/current
DB4            false 192.168.1.126            4333           4334 /opt/rdp           /opt/rdp/current

4 row(s) returned (0.10 sec)
```

This example shows a 4 node RapidsDB Cluster with the node having ip address 192.168.1.98 being assigned the node name "DB1" and acting as the DQC node. The other node is a DQE node.

9.3 FEDERATIONS Table

The FEDERATIONS table contains a list of the Federations in the RapidsDB Cluster. The table below shows the columns in the FEDERATIONS table:

Column Name	Description
FEDERATION_NAME	The name of the Federation. By default there will always be one Federation named DEFAULTFED.
IS_DEFAULT	Set to 'true' if this is the default Federation, otherwise set to false

Example:


```
rapids > select * from federations;
FEDERATION_NAME     IS_DEFAULT
---------------     ----------
DEFAULTFED                true

1 row(s) returned
```

9.4 CONNECTORS Table

The CONNECTORS table contains a list of the Connectors in the RapidsDB Cluster. The table below shows the columns in the CONNECTORS table:

Column Name	Description
FEDERATION_NAME	The name that of the Federation that this Connector belongs to
CONNECTOR_NAME	The name of the Connector
CONNECTOR_TYPE	The type of Connector, such as IMPEX, METADATA, MOXE or MYSQL
CONNECTOR_DDL	The CREATE CONNECTOR command that was used to create this Connector
IS_ENABLED	Set to 'true' if the Connector is enabled, otherwise it is set to false

The query below shows the sample output for querying this table:

```
rapids > select * from connectors;
FEDERATION_NAME   CONNECTOR_NAME   CONNECTOR_TYPE     IS_ENABLED CONNECTOR_DDL
---------------   --------------   --------------     ---------- -------------
DEFAULTFED        METADATA         METADATA                 true CREATE CONNECTOR METADATA TYPE METADATA NODE *
CATALOG * SCHEMA * TABLE *
DEFAULTFED        PARQSF10         HADOOP                   true CREATE CONNECTOR PARQSF10 TYPE HADOOP WITH
HDFS='hdfs://192.168.10.15:8020', FORMAT='parquet' NODE * CATALOG * SCHEMA * TABLE ORDERS (O_ORDERKEY INTEGER,
O_CUSTKEY INTEGER, O_ORDERSTATUS VARCHAR(1 CHARS), O_TOTALPRICE DECIMAL(15, 2), O_ORDERDATE DATE,
O_ORDERPRIORITY VARCHAR(15 CHARS), O_CLERK VARCHAR(15 CHARS), O_SHIPPRIORITY INTEGER, O_COMMENT VARCHAR(79
CHARS)) WITH USER='rapids', PATH='/user/rapids/warehouse/tpch/sf100/parquet/orders/' TABLE LINEITEM (L_ORDERKEY
INTEGER, L_PARTKEY INTEGER, L_SUPPKEY INTEGER, L_LINENUMBER INTEGER, L_QUANTITY DECIMAL(15, 2), L_EXTENDEDPRICE
DECIMAL(15, 2), L_DISCOUNT DECIMAL(15, 2), L_TAX DECIMAL(15, 2), L_RETURNFLAG VARCHAR(1 CHARS), L_LINESTATUS
VARCHAR(1 CHARS), L_SHIPDATE DATE, L_COMMITDATE DATE, L_RECEIPTDATE DATE, L_SHIPINSTRUCT VARCHAR(25 CHARS),
L_SHIPMODE VARCHAR(10 CHARS), L_COMMENT VARCHAR(44 CHARS)) WITH USER='rapids',
PATH='/user/rapids/warehouse/tpch/sf100/parquet/lineitem/'
DEFAULTFED        MOXE1            MOXE                     true CREATE CONNECTOR MOXE1 TYPE MOXE NODE * CATALOG
* SCHEMA * TABLE *

3 row(s) returned (0.06 sec)
```

In this example there are 3 Connectors in the DEFAULTFED Federation:

1. METADATA – manages the metadata for the DEFAULTFED Federation.
2. PARQSF10 – a Hadoop Connector
3. MOXE1 – a MOXE Connector

9.5 CATALOGS Table

The CATALOGS table contains a list of the catalogs in the current Federation. The table below shows the columns in the CATALOGS table:

Column Name	Description
CATALOG_NAME	The name of the catalog

The query below shows the sample output for querying this table:

```
rapids > select * from catalogs;
CATALOG_NAME
------------
MOXE_1
MYSQL_A
RAPIDS

3 row(s) returned
rapids >
```

In this example there are 3 catalogs:

1. MOXE_1 – this is the catalog for MOXE Connector named “MOXE_1”
2. MYSQL_A – this is the catalog for the MemSQL Connector named “MEM1”
3. RAPIDS – this is the catalog for the METADATA Connector

9.6 SCHEMAS Table

The SCHEMAS table contains a list of the schemas in the current Federation. The table below shows the columns in the SCHEMAS table:

Column Name	Description
CATALOG_NAME	The name of the catalog
SCHEMA_NAME	The name of the schema

The query below shows the sample output for querying this table:

```
rapids > select * from schemas;
CATALOG_NAME  SCHEMA_NAME
------------  -----------
MOXE_1        MOXE_1
MYSQL_A       CUSTOMER
RAPIDS        SYSTEM

3 row(s) returned
rapids >
```

In this example there are 3 schemas:

1. MOXE_1 – this is the schema for the MOXE Connector, “MOXE_1”
2. CUSTOMER – this is the schema for the MySQL Connector, “MYSQL_A”
3. SYSTEM – this is the schema for the Metadata Connector

9.7 TABLES Table

The TABLES table contains a list of the tables that can be accessed from the current Federation. The table below shows the columns in the TABLES table:

Column Name	Description
CATALOG_NAME	The name of the catalog for this table
SCHEMA_NAME	The name of the schema for this table
TABLE_NAME	The name of the table
IS_PARTITIONED	Set to 'true' if this table is partitioned, otherwise it is set to false
COMMENT	Comment for the table, if any
PROPERTIES	Indicates any properties associated with the table, such as the HDFS path name for tables managed by a Hadoop Connector

Example:

The following example shows tables from a MySQL database where there are comments on the columns and tables:

```
rapids > select * from tables where schema_name='test';
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME      IS_PARTITIONED COMMENT
PROPERTIES
---------------
---------------
test            test           COMMENTS                false Test table for
comments              NULL
test            test           TESTING01               false NULL
NULL
test            test           dctest                  false This is a test
table for comments   NULL

3 row(s) returned (0.05 sec)
```

9.8 INDEXES Table

The INDEXES table contains a list of the tables that can be accessed from the current Federation. The table below shows the columns in the INDEXES table:

Column Name	Description
CATALOG_NAME	The name of the catalog for this table

SCHEMA_NAME	The name of the schema for this table
TABLE_NAME	The name of the table
INDEX_NAME	The name of the index
IS_UNIQUE	true if the index is unique
IS_PRIMARY	true if the index is the primary key
INDEX_TYPE	The type of index
ORDINAL	The position of the column in the index
COLUMN_NAME	The column name

The example output below shows the index metadata for indexes defined on tables in MemSQL:

```
rapids > select * from indexes;
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME    INDEX_NAME          IS_UNIQUE    IS_PRIMARY INDEX_TYPE      ORDINAL COLUMN_NAME
--------------- -------------- ------------- ------------------- ------------ ---------- --------------- ------- ---------------
MEMSQL          TPCH           LINEITEM      PRIMARY                   true       true TREE                    1 L_ORDERKEY
MEMSQL          TPCH           LINEITEM      PRIMARY                   true       true TREE                    2 L_LINENUMBER
MEMSQL          TPCH           LINEITEM      li_com_dt_idx            false      false TREE                    1 L_COMMITDATE
MEMSQL          TPCH           LINEITEM      li_rcpt_dt_idx           false      false TREE                    1 L_RECEIPTDATE
MEMSQL          TPCH           LINEITEM      li_shp_dt_idx            false      false TREE                    1 L_SHIPDATE
MEMSQL          TPCH           LINEITEM      lineitem_fk1             false      false TREE                    1 L_ORDERKEY
MEMSQL          TPCH           LINEITEM      lineitem_fk2             false      false TREE                    1 L_SUPPKEY
MEMSQL          TPCH           LINEITEM      lineitem_fk3             false      false TREE                    1 L_PARTKEY
MEMSQL          TPCH           LINEITEM      lineitem_fk3             false      false TREE                    2 L_SUPPKEY
MEMSQL          TPCH           LINEITEM      lineitem_fk4             false      false TREE                    1 L_PARTKEY
```

9.9 COLUMNS Table

The COLUMNS table contains a list of the columns for all of the tables that can be accessed in the current Federation. The table below shows the columns in the COLUMNS table:

Column Name	Description
CATALOG_NAME	The name of the catalog for this table
SCHEMA_NAME	The name of the schema for this table
TABLE_NAME	The name of the table
COLUMN_NAME	The name of the column
DATA_TYPE	The data type for the column
ORDINAL	The column number (one-based)

IS_PARTITION_KEY	Set to 'true' if this column part of the partition (shard) key
IS_NULLABLE	True if the column is nullable
PRECISION	Precision for numerical columns
PRECISION_RADIX	If data_type identifies a numeric type, this column indicates in which base the values in the columns numeric_precision and numeric_scale are expressed. The value is either 2 or 10 as follows: INTEGER, FLOAT: 2 DECIMAL: 10
PRECISION_SCALE	Scale for decimal and float columns
CHARACTER_SET	Character set for column
COLLATION	Not used
COMMENT	Column comment
PROPERTIES	Properties associated with column

Example 1:

This example shows the column definition for the TPC-H "ORDERS" table:

rapids > select * from columns where table_name='ORDERS';

```
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME    COLUMN_NAME        DATA_TYPE
ORDINAL    IS_PARTITION_KEY    IS_NULLABLE    PRECISION    PRECISION_RADIX
SCALE CHARACTER_SET    COLLATION    COMMENT    PROPERTIES
------------  -----------  ----------  -----------  ---------     ---
----   ----------------  -----------  ---------  ---------------  ----- -
------------  ---------  -------  ----------
MOXE            MOXE           ORDERS        O_ORDERKEY         INTEGER
0                true           false           64                 2    NULL NULL
NULL        NULL      serial:Kind=integer64
MOXE            MOXE           ORDERS        O_CUSTKEY          INTEGER
1               false           false           64                 2    NULL NULL
NULL        NULL      serial:Kind=integer64
MOXE            MOXE           ORDERS        O_ORDERSTATUS      VARCHAR
2               false           true         NULL               NULL    NULL
UTF16           BINARY       NULL      serial:Kind=stringRA2
MOXE            MOXE           ORDERS        O_TOTALPRICE       DECIMAL
3               false           true           17                10        2 NULL
NULL        NULL      serial:Kind=decimal64
```



```
MOXE            MOXE           ORDERS        O_ORDERDATE        DATE
4               false           true         NULL               NULL     NULL NULL
NULL         NULL       serial:Kind=timestamp
MOXE            MOXE           ORDERS        O_ORDERPRIORITY    VARCHAR
5               false           true         NULL               NULL     NULL
UTF16            BINARY       NULL       serial:Kind=stringRA2
MOXE            MOXE           ORDERS        O_CLERK            VARCHAR
6               false           true         NULL               NULL     NULL
UTF16            BINARY       NULL       serial:Kind=stringRA2
MOXE            MOXE           ORDERS        O_SHIPPRIORITY     INTEGER
7               false           true           64                  2     NULL NULL
NULL         NULL       serial:Kind=integer64
MOXE            MOXE           ORDERS        O_COMMENT          VARCHAR
8               false           true         NULL               NULL     NULL
UTF16            BINARY       NULL       serial:Kind=stringRA2

9 row(s) returned (0.05 sec)
```

Example 2

This example shows a table with column comments:

```
rapids > select * from columns where table_name='COMMENTS';
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME    COLUMN_NAME    DATA_TYPE
ORDINAL    IS_PARTITION_KEY    IS_NULLABLE    PRECISION    PRECISION_RADIX
SCALE CHARACTER_SET    COLLATION    COMMENT           PROPERTIES
------------
------------
--------
test            test           COMMENTS      C1             INTEGER
0               false           true           64                  2
NULL NULL             NULL         Integer column    NULL
test            test           COMMENTS      C2             DECIMAL
1               false           true           15                 10
2 NULL             NULL         decimal column    NULL
test            test           COMMENTS      C3             FLOAT
2               false           true           53                  2
NULL NULL             NULL         float column      NULL

3 row(s) returned (0.05 sec)
```

9.10 TABLE_PROVIDERS Table

The table below shows the columns in the TABLE_PROVIDERS table:

Column Name	Description

CATALOG_NAME	The name of the catalog for this table
SCHEMA_NAME	The name of the schema for this table
TABLE_NAME	The name of the table
CONNECTOR_NAME	The name of the Connector that is managing access to this table

```
rapids > select * from table_providers;
CATALOG_NAME   SCHEMA_NAME   TABLE_NAME            CONNECTOR_NAME
------------   -----------   ----------            --------------
RAPIDS         SYSTEM        FEDERATIONS           METADATA
RAPIDS         SYSTEM        CONNECTORS            METADATA
RAPIDS         SYSTEM        NODES                 METADATA
RAPIDS         SYSTEM        CATALOGS              METADATA
RAPIDS         SYSTEM        SCHEMAS               METADATA
RAPIDS         SYSTEM        TABLES                METADATA
RAPIDS         SYSTEM        COLUMNS               METADATA
RAPIDS         SYSTEM        TABLE_PROVIDERS       METADATA
RAPIDS         SYSTEM        INDEXES               METADATA
RAPIDS         SYSTEM        QUERY_STATS           METADATA
RAPIDS         SYSTEM        QUERIES               METADATA
RAPIDS         SYSTEM        SESSIONS              METADATA
RAPIDS         SYSTEM        USERS                 METADATA
RAPIDS         SYSTEM        USER_CONFIG           METADATA
RAPIDS         SYSTEM        AUTHENTICATORS        METADATA
RAPIDS         SYSTEM        AUTHENTICATOR_CONFIG  METADATA
RAPIDS         SYSTEM        USERNAME_MAPS         METADATA
RAPIDS         SYSTEM        PATTERN_MAPS          METADATA
PARQSF10       PUBLIC        ORDERS                PARQSF10
PARQSF10       PUBLIC        LINEITEM              PARQSF10
MOXE1          MOXE1         T1                    MOXE1
MOXE1          MOXE1         T2                    MOXE1

22 row(s) returned (0.11 sec)
```

9.11 AUTHENTICATORS Table

The table below shows the columns in the AUTHENTICATORS table:

Column Name	Description
AUTHNAME	The name of the authenticator instance.
TYPE	The name of the type of authenticator.
ENABLED	Whether the authenticator is enabled or disabled.
DDL	The DDL string to recreate this authenticator.

```
rapids > select * from authenticators;
AUTHNAME    TYPE          ENABLED DDL
--------    ----          ------- ---
RDPAUTH     RDP              true CREATE AUTHENTICATOR RDPAUTH TYPE RDP ;
```



```
KRB         KERBEROS        true CREATE AUTHENTICATOR KRB TYPE KERBEROS WITH REALM = 'HOME';
```

9.12 AUTHENTICATOR_CONFIG Table

The table below shows the columns in the AUTHENTICATOR_CONFIG table:

Column Name	Description
AUTHNAME	The name of the authenticator instance.
KEY	The name of the custom property for this authenticator instance.
VALUE	The value of the custom property for this authenticator instance.

```
rapids > select * from authenticator_config;
AUTHNAME    KEY                       VALUE
--------    ---                       -----
RDPAUTH     rdp.authenticator.name    RDPAUTH
RDPAUTH     rdp.authenticator.type    RDP
KRB         REALM                     HOME
KRB         rdp.authenticator.name    KRB
KRB         rdp.authenticator.type    KERBEROS
```

9.13 USERS Table

The table below shows the columns in the USERS table:

Column Name	Description
USERNAME	The unique name of the user.
ENABLED	Whether the user is enabled or disabled.
AUTHNAME	The name of the authenticator instance this user is associated with.

```
rapids > select * from users;
USERNAME     ENABLED AUTHNAME
--------     ------- ----------
CRAIG           true KRB
RAPIDS          true RDPAUTH
john            true RDPAUTH
```

9.14 USER_CONFIG Table

The table below shows the columns in the USER_CONFIG table:

Column Name	Description
USERNAME	The username.
KEY	The name of the custom property for this authenticator instance.
VALUE	The value of the custom property for this authenticator instance.

```
rapids > select * from user_config;
USERNAME    KEY          VALUE
--------    ---          -----
CRAIG       PRINCIPAL    craig@HOME
```

9.15 SESSIONS Table

The table below shows the columns in the SESSIONS table:

Column Name	Description
SESSION_ID	The unique name of the session across the cluster.
USERNAME	The username that the client has authenticated as, or null.
NODE	The node that the client connected to.
CLIENT_IP	The IP address of the client.
CLIENT_PORT	The port address that the client is connecting from.
SERVER_PORT	The port address that the client is connected to.
ESTABLISHED	The timestamp when the client first connected.

```
rapids > select * from sessions;
SESSION_ID    USERNAME    NODE     CLIENT_IP      CLIENT_PORT    SERVER_PORT ESTABLISHED
----------    --------    ------   ----------     -----------    ----------- -----------
S1@NODE1      RAPIDS      NODE1    127.0.0.1            50547           4333 2019-04-18 07:00:22.376
```

9.16 USERNAME_MAPS Table

The table below shows the columns in the USERNAME_MAPS table:

Column Name	Description
ID	The external identifier to map from.
USERNAME	The RapidsDB username to map this external identifier to.


```
ID           USERNAME
--           --------
craig@HOME   CRAIG
```

9.17 PATTERN_MAPS Table

The table below shows the columns in the PATTERN_MAPS table:

Column Name	Description
PRIORITY	The order in which the pattern mapping is tried (highest first).
SEARCH	The pattern to test against the external user ID.
REPLACE	The replacement pattern to be applied against the external user ID in conjunction with the search pattern.

```
rapids > select * from pattern_maps;
  PRIORITY SEARCH                          REPLACE
  -------- ------                          -------
       100 ^(.+)/admin@COMPANY.COM$         ADMIN
        90 ^(.+?)(/[^@]*)?@COMPANY.COM$     $1
        80 ^(.+?)(/[^@]*)?@EXAMPLE.COM$     $1_EXAMPLE
```

9.18 QUERIES Table

The table below shows the columns in the QUERIES table:

Column Name	Description
QUERY_ID	The query id
SESSION_ID	The session id where this query is running
NODE	The RapidsDB node where the query was started
USERNAME	The name of the user running this query
START_TIME	The timestamp when the query was started
QUERY_TEXT	The SQL query

When querying the QUERIES table, only the queries submitted by the current user will be displayed unless the userid is “RAPIDS” in which case the queries for all users will be displayed.

Example:

Below is a query submitted from the rapids-shell:

```
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;
```

Below is the content of the QUERIES table while this query is running:

```
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)
```

10 Cancelling a Query

There are three ways that an active query can be cancelled as explained in the following sections.

10.1 rapids-shell

If the query was started from the rapids-shell then the user can enter Ctrl-k from the rapids-shell window where the query is running. This will result in a message being sent to RapidsDB to cancel the query currently running on that connection. This requires rapids-shell version 4 and the RapidsDB JDBC Driver version 4, both of which are included with this release.

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;
```

Checking the QUERIES table from another window shows this query running:


```
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)
```

Press Ctrl-K:

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

Checking the QUERIES table shows this query no longer running:

```
rapids > select * from queries;
QUERY_ID          SESSION_ID      NODE      USERNAME   START_TIME                QUERY_TEXT
--------          ----------      ----      --------   ----------                ----------
SSN_2@BORAY01#1   SSN_2@BORAY01   BORAY01   RAPIDS     2020-07-30 21:18:11.365   select * from queries;

1 row(s) returned (0.14 sec)
```

10.2 JDBC

Programmatically via the RapidsDB JDBC Driver using the Statement.cancel() interface:

- When a Statement instance is being executed, a second thread can call the cancel() method on it. This will cause the JDBC driver to create a temporary connection to the server, authenticate and issue a query cancellation. When the query is cancelled, the execution of the Statement object will return with a JDBC SqlException indicating that the query was cancelled.
- The call to Statement.cancel() will return once the cancellation request has been submitted to RapidsDB. This is not necessarily the same time that the cancelled query actually exits early.
- If Statement.cancel() is called on a statement that has already completed or hasn't been executed in RapidsDB yet then an error is returned to the caller.
- Requires RapidsDB JDBC driver version 4.

10.3 CANCEL QUERY command

Using the SQL command CANCEL QUERY.

The syntax for the CANCEL QUERY command is:

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

- The <queryId> can be found from the QUERIES metadata table (see 9.18).

- Queries can be cancelled on remote nodes as well as the local node.
- Only the user that initiated the query can cancel the query, unless the user is the RAPIDS user in which case that user can cancel any query

Example:

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 – from another session, list the currently active queries and then cancel the query started from session 1:

```
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 on session 1 after the query is cancelled:

```
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;
[CANCELED]
rapids>
```


11 Performance Tuning

11.1 EXPLAIN

EXPLAIN instructs the rapids-shell to output a schematic representation of the query plan for the associated statement, including the SQL queries that will be sent to the underlying Data Store and the internal operators and routing for operations that will be performed by the RapidsDB Execution Engine.

Syntax:

EXPLAIN <SQL statement>

Example 1 – simple select

```
rapids > explain select sum(o_totalprice) from orders group by o_orderkey;
QUERY_PLAN
----------
Execute "select sum(o_totalprice) from orders group by o_orderkey;"
P2:Project O1
A1:Aggregate {O_ORDERKEY}  SUM(O_TOTALPRICE) `O1`
P1:Project ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE
T1:Stream ORDERS


1 row(s) returned (0.83 sec)
```

Example 2 – 1-way join

```
rapids > explain select o_orderkey from orders, lineitem where o_orderdate=l_commitdate and
o_orderdate < '2019-10-20 09:00:00';
QUERY_PLAN
----------
Execute "select o_orderkey from orders, lineitem where o_orderdate=l_commitdate and
o_orderdate < '2019-10-20 09:00:00';"
P4:Project O_ORDERKEY
P3:Project O_ORDERKEY

J1:LpJoin O_ORDERDATE = L_COMMITDATE
  F1:Filter O12
  P1:Project ORDERS.O_ORDERDATE, ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE < 2019-10-20 `O12`
  T1:Stream ORDERS

  F2:Filter L18
  P2:Project LINEITEM.L_COMMITDATE, LINEITEM.L_COMMITDATE < 2019-10-20 `L18`
  T2:Stream LINEITEM


1 row(s) returned (1.94 sec)
```

11.2 JOIN Order

For SELECT statements with JOINs the user should order the tables in the query from left to right so that the table with the smallest number of rows, given the supplied WHERE clause predicates, is to the left and the table with the largest number of rows is the last table. For example, with the following query:


```
SELECT l_orderkey,
    SUM(l_extendedprice) AS revenue,
    o_orderdate,
    o_shippriority
FROM  lineitem
    join orders
    ON l_orderkey = o_orderkey
    join customer
     ON c_custkey = o_custkey
WHERE  c_mktsegment = 'FURNITURE'
    AND o_orderdate <  '2014-05-01 12:00:00'
    AND l_shipdate >  '2014-04-01 12:00:00'
GROUP  BY l_orderkey,
      o_orderdate,
      o_shippriority;
```

If the rows returned from the customer table are the smallest and the rows returned from the lineitem table are the largest, then the query should be changed to the following in order to achieve the best query performance:

```
SELECT l_orderkey,
    SUM(l_extendedprice) AS revenue,
    o_orderdate,
    o_shippriority
FROM   customer
    join orders
     ON c_custkey = o_custkey
    join lineitem
     ON l_orderkey = o_orderkey
WHERE  c_mktsegment = 'FURNITURE'
    AND o_orderdate <  '2014-05-01 12:00:00'
    AND l_shipdate >  '2014-04-01 12:00:00'
GROUP  BY l_orderkey,
      o_orderdate,
      o_shippriority;
```

11.3 Restrict Amount of Data

Care should be taken to restrict the amount of data in JOINs by using the appropriate predicates in the WHERE clause. Failure to do this could result in too much data being requested from the Data Store

which could cause one or more of the DQE nodes to fail due to exhausting their heaps. In the previous example, the timestamps in the WHERE clause should be further restricted, for example:

```
WHERE c_mktsegment = 'FURNITURE'
    AND o_orderdate < '2014-05-01 12:00:00'
    AND o_orderdate > '2014-04-01 12:00:00'
    AND l_shipdate < '2014-05-01 12:00:00'
    AND l_shipdate > '2014-04-01 12:00:00'
```

12 Managing RapidsDB

12.1 Command-line Interface: rapids-shell

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

12.2 Running the rapids-shell

12.2.1 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@db01:/opt/rdp/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)
```

12.2.2 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.

12.2.3 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.

12.2.4 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@RDP1#33 requested.
java.sql.SQLException: System exception:
com.rapidsdata.common.exceptions.RdpRemoteException:
com.rapidsdata.plan.exceptions.ExecCanceledException:  Canceled  locus=RDP1
```

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@RDP1#7    SSN_3@RDP1   RDP1   RAPIDS     2020-07-30 21:17:11.608   select * from
queries;
SSN_1@RDP1#32   SSN_1@RDP1   RDP1   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@RDP1#32 ;
0 row(s) returned (0.70 sec)
```

Session 1 – indicates that the query is cancelled:

```
[CANCELED]
rapids>
```

12.3 Adding Authenticators – CREATE AUTHENTICATOR

12.3.1 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)
```

12.3.2 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 RapidsDB Cluster. How this information is specified is detailed below.

12.3.2.1 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 installation 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.

12.3.2.2 Specifying the Keytab file

By default, the Kerberos authenticator will look for a keytab file in the path `../rdp.keytab` (relative to the RapidsDB installation directory) on each node in the RapidsDB Cluster

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.

12.4 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)
```

12.5 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 the rules for SQL identifiers.

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)
```

12.6 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 SQL 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)
```

12.6.1 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. 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)
```

12.7 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)
```

12.8 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 the rules for SQL identifiers.

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)
```

12.9 User ID Mapping

12.9.1 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 12.6.1 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)
```

12.9.2 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)
```

12.9.3 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)
```


12.9.4 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 be copied to every node in the RapidsDB Cluster.

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@db01:/opt/rdp/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)
```

12.9.5 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)
```

12.10 Adding Connectors

The examples in this section use the following 3-node RapidsDB Cluster:

Linux node name	RapidsDB node name	RapidsDB node type	ip address
db01	RDP1	DQC	192.168.1.1
db02	RDP2	DQE	192.168.1.2
db03	RDP3	DQE	192.168.1.3

12.10.1 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>]]
[<Include Clause> [<Include 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 USER='rapids', PASSWORD='rdpuser',
HOST='10.1.1.20', PORT='1521', SID='XEPDB1' NODE RDP1;
```

This command would create an Oracle Connector named ORA1 running on RapidsDB Cluster node RDP1.

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

12.10.2 Include Clause

When configuring a Connector the user can specify the tables to be included from the back-end data store for querying and can also specify the names under which the included catalog, schema and/or tables will appear in RapidsDB. The following describes how to specify this information, which will be part of the Connector configuration command described later.

The <Include Clause> is used for the following:

1. To specify the tables that should be brought into the name space managed by the Connector.
2. To optionally specify alternative names under which the included catalogs, schemas and/or tables will appear in RapidsDB.

An <Include 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 will appear within RapidsDB under the specified name, and that is the name that the user will use when submitting queries to RapidsDB.

The USING clause applies to the Hadoop Connector.

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

CATALOG * SCHEMA * TABLE *

which will bring in the metadata for all of the tables that the user is authorized to access available for querying.

NOTE:
It is strongly recommended that the user be as selective as possible when specifying the tables to be included for querying

The catalog, schema and table names are case-insensitive and do not have to be enclosed in back-ticks or 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 back-ticks or double quotes.

Below are some sample Include Clauses for an Oracle Connector named "ORA1":

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

This clause would include 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 *
SCHEMA EAST
TABLE CUSTOMERS AS EASTCUSTOMERS, ORDERS AS EASTORDERS
```


This clause would include 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 *
SCHEMA EAST
TABLE CUSTOMERS AS EASTCUSTOMERS, ORDERS AS EASTORDERS
SCHEMA WEST
TABLE CUSTOMERS AS WESTCUSTOMERS, ORDERS AS WESTORDERS
```

This clause would include the tables CUSTOMERS and ORDERS from schemas named EAST, and WEST 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:

ORA1.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 include 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.

12.10.3 Handling of Decimal Datatypes

The RapidsDB Execution Engine supports a high performance internal data type for decimal values, 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 an arithmetic overflow. 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.

12.10.4 Metadata Handling

Starting with the R4.2.2 (GA) release, the Connectors no longer cache their metadata information in Zookeeper, and the implication of this is that if the RapidsDB Cluster is stopped and restarted, any metadata information from the previously running RapidsDB cluster will be lost and the Connectors will have to reacquire their metadata when the RapidsDB cluster is restarted. This behavior will be changed in an upcoming release where the Connector metadata will be cached.

12.10.5 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 on every node in the RapidsDB Cluster 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 RDP2 NODE RDP3;
```

This command would create a MOXE Connector to run on two RapidsDB Cluster nodes (“RDP2” and “RDP3”) 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 13 for details on managing MOXE tables

12.10.6 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> [,NODE <node name>]]
[<Include Clause> [<Include Clause>]]
```

For the MemSQL Connector, best performance is usually obtained by configuring the Connector to operate on the same node as the MemSQL Aggregator node. By default, the Connector will be configured to run from any node in the RapidsDB Cluster, and in this case RapidsDB will use the node which minimizes the movement of data across the network.

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. When the database is specified this is equivalent to specifying the Include Clause (see 12.10.2): CATALOG * SCHEMA database TABLE * **NOTE:** If the database is not specified the metadata for all of

			the databases that the user has access to will be brought in.
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 | 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 12.10.3 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 MemSQL database should be batched. Each batch will contain the number of rows specified by the fetch_size.
ignore_case	TRUE	[] | TRUE | 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	Specifies a key value pair that will be set as part of the Connection

		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 (`)	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 12.10.2 for details on how to specify the <Include Clause>.

Example commands:

```
1. CREATE CONNECTOR MEMSQL1
   TYPE MEMSQL WITH host='192.168.1.1', user='user1', database='tpch'
    NODE RDP2;
```

This command would create a MemSQL Connector on RapidsDB Cluster node RDP2 and it would connect via JDBC to MemSQL using the default port number (3306) using ip address

192.168.1.1. The Connnector would incldue 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';
```

This command is similar to the pervious command with the one difference being that the Connector can run on any node in the RapidsDB Cluster.

```
3. CREATE CONNECTOR MEMSQL1
   TYPE MEMSQL WITH host='192.168.1.1', user='user1'
    NODE RDP2 CATALOG * SCHEMA tpch;
```

This example is equivalent to the first example, but in this example the “database” option is omitted and instead an Include Clause (see 12.10.2) is used where the SCHEMA is set to “tpch” to indicate that the metadata for the tables from the “tpch” database should be brought in.

```
4. CREATE CONNECTOR MEMSQL1
   TYPE MEMSQL WITH host='192.168.1.1', user='user1'
   NODE RDP2
   CATALOG *
   SCHEMA tpch
   TABLE customers, orders;
```

This command would bring in the metadata for the customers and orders tables from the database named “tpch”.

```
5. CREATE CONNECTOR MEMSQL1
   TYPE MEMSQL WITH host='192.168.1.1', user='user1', ignore_case=false
   NODE RDP2
   CATALOG *
   SCHEMA tpch
   TABLE “customers” ,“CUSTOMERS”;
```

This command uses the option “ignore_case” which has been set to FALSE because there are two tables both named the same but with different cases. In this situation 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”;

```
6. CREATE CONNECTOR MEMSQL1
   TYPE MEMSQL WITH host='192.168.1.1', port='3306', user='user1', ignore_case=false
   NODE RDP2
   CATALOG *
   SCHEMA tpch
   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;

```
7. CREATE CONNECTOR MEMSQL1
   TYPE MEMSQL WITH host='192.168.1.1', port='3306', user='user1', database='tpch',
   _`useCompression`=true, bigdecimal
   NODE RDP2;
```

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.

12.10.7 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> [NODE <node name>]]
[<Include Clause> [<Include Clause>]]
```

By default, the Connector will be configured to run from any node in the RapidsDB Cluster, and in this case RapidsDB will use the node which minimizes the movement of data across the network.

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		Non-empty, case-sensitive, non-whitespace string	The MySQL database to be used. The case must match the case used by MySQL. When the database is specified this is equivalent to specifying the Include Clause (see 12.10.2): CATALOG * SCHEMA database TABLE * **NOTE:** If the database is not specified the metadata for all of the databases that the user has access to will be brought in.
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 | 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 12.10.3 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 | 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

usessl	FALSE	[] | TRUE | 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 12.10.2 for details on how to specify the <Include clause>.

Example commands:

```
CREATE CONNECTOR MYSQL1
TYPE MYSQL WITH host='192.168.1.1', user='user1', database='tpch'
NODE RDP2;
```

This command would create a MySQL Connector on RapidsDB Cluster node RDP2 and it would connect to MySQL using the default port number,3306, and ip address 192.168.1.1. The Connnector would bring in the metadata for 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', user='user1', database='tpch';
```

This command would create a MySQL Connector the could run on any node in the RapidsDB Cluster and it would connect to MySQL using the default port number,3306, and ip address 192.168.1.1. The Connnector would bring in the metadata for 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', user='user1'
CATALOG * SCHEMA tpch;
```

This command is equivalent to the previous example. In this example the “database” option is omitted and instead an Include Clause (see 12.10.2) is used where the SCHEMA is set to “tpch” to indicate that the metadata for the tables from the “tpch” database should be brought in.

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

This command would create a MySQL Connector that can run on any node in the RapidsDB Cluster and it would connect to MySQL using port number 3307 and ip address 192.168.1.1,

and it would also pass the “useSSL=true” option to the MySQL Connector-J . The Connector would also use Java bigdecimal for all decimal values. The Connnector would include 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', ignore_case=false
CATALOG *
SCHEMA tpch
TABLE “customers” ,“CUSTOMERS”;
```

In this command the option “ignore_case” has been set to FALSE because there are two tables both named the same but with different cases. In this situation 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', ignore_case=false
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 12.10.6)

12.10.8 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> [NODE <node name>]]
[<Include Clause> [<Include Clause>]]
```

By default, the Connector will be configured to run from any node in the RapidsDB Cluster, and in this case RapidsDB will use the node which minimizes the movement of data across the network.

NOTE:

The RapidsDB system includes the following jar file for the Oracle JDBC Driver: ojdbc8- 21.1.0.0.jar. If the user wishes to use a different version of the Oracle JDBC Driver then the user must delete the ojdbc8-21.1.0.0.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 **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 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 | 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 12.10.3 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 | 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 12.10.2 for details on how to specify the <Include clause>.

Examples:

1. The following is a sample Connector to an Oracle database with a SID of “dev1” that will include all of the schemas and tables accessible by the specified user. The Connector is configured to run on any node in the RapidsDB Cluster:

```
CREATE CONNECTOR ORA1 TYPE ORACLE WITH HOST='10.1.1.20', SID= 'dev1', USER='rapids',
PASSWORD='rdpuser';
```

2. In this example the Oracle Connector is configured to only run on the node RDP1 and the Connector will use Java bigdecimal for all decimal values:

```
CREATE CONNECTOR ORA1 TYPE ORACLE WITH HOST='10.1.1.20', SID= 'dev1', USER='rapids',
PASSWORD='rdpuser', bigdecimal NODE RDP1;
```

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

```
CREATE CONNECTOR ORA1 TYPE ORACLE WITH HOST='10.1.1.20', SID= 'dev1', USER='rapids',
PASSWORD='rdpuser' NODE RDP1 CATALOG * SCHEMA “orders” SCHEMA “sales” TABLE *;
```

12.10.9 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> [NODE <node name>]]
[<Include Clause> [<Include Clause>]]
```

By default, the Connector will be configured to run from any node in the RapidsDB Cluster, and in this case RapidsDB will use the node which minimizes the movement of data across the network.

NOTE:

The RapidsDB system includes the following jar file for the Postgres JDBC Driver: postgres- 4.3.1.jar. If the user wishes to use a different version of the Postgres JDBC Driver then the user must delete the postgres-4.3.1.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 | 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 | 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 12.10.2 for details on how to specify the <Include clause>.

When specifying schema or table names using the SCHEMA or TABLE clauses, the names must be enclosed in back-ticks or 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” using port 6432 that will include all of the schemas and tables accessible by the specified user, and the Connector will use Java big decimal datatypes for all numeric values. The Connector is configured to run from any node in the RapidsDB Cluster:

```
CREATE CONNECTOR PG1 TYPE POSTGRES WITH host= '10.10.1.1', port='6432', database='dw1',
USER='adm', PASSWORD='admpsw', bigdecimal;
```

2. The following is a sample Connector to a Postgres database “dw1” which will include only the schemas named “orders” and “sales”, and the Connector is configured to only run on the node RDP2 in the RapidsDB Cluster:

```
CREATE CONNECTOR PG1 TYPE POSTGRES WITH host= '10.10.1.1',  database='dw1', USER='adm',
PASSWORD='admpsw' NODE RDP2 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 RapidsDB 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', PASSWORD='rapids',
DATABASE='rdp4', HOST='192.168.10.12';
```

```
CREATE CONNECTOR PG2 TYPE POSTGRES WITH USER='rapids', PASSWORD='rapids',
DATABASE='rdp4', HOST='192.168.10.12' CATALOG "rdp4" AS "pg2";
```

   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;

12.10.10 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> [NODE <name>]]
[<Include Clause> [<Include Clause>]]
```

By default, the Connector will be configured to run from any node in the RapidsDB Cluster, and in this case RapidsDB will use the node which minimizes the movement of data across the network.

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 | 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 12.10.3 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 | 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 12.10.2 for details on how to specify the <Include 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 include all of the schemas and tables accessible by the specified user. The Connector is configured to run on any node in the RapidsDB Cluster:

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

2. The following is a sample Connector to a Greenplum database “dwctr” which will include only the schemas named “orders” and “sales”, and the Connector will use Java bigdecimal for all decimal values and the Connector is configured to only run from the node RDP2 in the RapidsDB Cluster:

```
CREATE CONNECTOR GP1 TYPE POSTGRES WITH
CONNECTIONSTRING= 'jdbc: pivotal:greenplum://10.10.1.1:5432;DatabaseName=dwctr',
USER='adm', PASSWORD='admpsw', bigdecimal NODE RDP2 CATALOG * SCHEMA “orders”
SCHEMA “sales”;
```

12.10.11 Adding a Generic JDBC Connector

To add a Generic JDBC Connector use the following command:


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

By default, the Connector will be configured to run from any node in the RapidsDB Cluster, and in this case RapidsDB will use the node which minimizes the movement of data across the network.

NOTES

1. For the JDBC Connector, the user must copy the JDBC Driver for the associated data source to the "lib" directory of the RapidsDB installation directory on all nodes where the JDBC Connector is going to run.

2. NOTE: With Release 4.3, the JDBC Connector no longer caches it's metadata in Zookeeper, which means that when the RapidsDB Cluster is restarted a JDBC Connector will have to reacquire its metadata from the back-end data source at startup time. In the case where there a large number (thousands) of tables in the metadata, this can take some time and delay the availability of the JDBC Connector. The previous version of the JDBC Connector can still be used by specifying the "TYPE" as "OLDJDBC", in which case the metadata will continue to be cached in Zookeeper and will be automatically restored when a RapidsDB Cluster is restarted. The disadvantage of using an "OLDJDBC" Connector is that the Connector will not use the higher performance and more stable infrastructure that the latest "JDBC" Connector uses.

Example:

```
CREATE CONNECTOR SQL_SERVER TYPE OLDJDBC WITH URL ='
jdbc:sqlserver://10.0.0.1:1433;databaseName=Sales;  loginTimeout=30',  USER='sales';
```

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

Key:	Default Value	Value syntax	Description
url		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 | 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 12.10.3 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 | 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.

useSSL	FALSE	[] | TRUE | 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: _cuseCompression`=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 12.10.2 for details on how to specify the <Include 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',
```

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 URL=' jdbc:sqlserver://10.0.0.1:1433;databaseName=Sales; loginTimeout=30', USER='sales';
```

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 URL =' jdbc:sqlserver://10.0.0.1:1433;databaseName=Sales', USER='sales',
_loginTimeout=30;
```

   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 URL =' jdbc:sqlserver://10.0.0.1:1433;databaseName=Sales', USER='sales', bigdecimal;
```

12.10.12 Adding a Hadoop Connector

12.10.12.1 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 ( c1 integer, c2 timestamp) WITH path='/data/sample/t1'
TABLE T2 ( 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 RDP2 NODE RDP3
CATALOG *
SCHEMA *
```



```
TABLE T1 ( c1 integer, c2 timestamp) WITH path='/data/sample/t1'
TABLE T2 ( 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 12.10.12.8 for more details
delimiter	','	'<char>' Non-empty, single character string	Specifies the field delimiter. See 12.10.12.5.7 for more details
enclosed_by		'<char>[<char>]' or '"'	Specifies whether a field is optionally enclosed by a

		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	specified character. This is 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 12.10.12.5.3.1 for more details). See 12.10.12.5.8 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 12.10.12.5.9 for more details
ignore_header	'0'	Integer value >= 0	Specifies the number of header records to be skipped See 12.10.12.5.11 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, 'db01:8020, db02:8020' 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 7.10.12.5.10 for more details
use_datanode_hostname	FALSE	[] | TRUE | 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 12.10.12.3.1 for more details.

NOTES:

1. For a Hadoop Connector, the catalog name will be the name of the Connector, unless overridden by the “AS” clause
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

12.10.12.2 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='db01:8020, db02:8020',
FORMAT='delimited', ENCLOSED_BY="'", USER='hdfs' NODE RDP2 NODE RDP3
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 RDP2 NODE RDP3
TABLE …
```

Example 3, using the Hive Metastore (see 12.10.12.10.1):


```
CREATE CONNECTOR HDFS_HIVEM TYPE HADOOP WITH METASTORE='192.168.10.14:9083',
NAMESERVICE='boray', NAMENODES='db01:8020, db02:8020',
NODE * CATALOG * SCHEMA TPCH_SF1 TABLE * ;
```

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

12.10.12.3.1 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 RDP2 NODE RDP3
TABLE …
```

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

12.10.12.3.2 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.

12.10.12.3.3 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.

12.10.12.3.4 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.

12.10.12.3.5 CREATE/DROP TABLE

In order to be able to create and drop tables from the Hive Metastore (see 12.10.12.10), the userid being used by the Hadoop Connector must be given write access to the underlying HDFS files associated with the tables.

12.10.12.4 Kerberos Authentication

12.10.12.4.1 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 12.10.12.4.4) instruct the Hadoop Connector to use Kerberos authentication.

12.10.12.4.2 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 start --jvm_settings "-Djava.security.krb5.conf=<path to krb5.conf file>"
    For example:
    ./bootstrapper.sh 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)

12.10.12.4.3 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

12.10.12.4.4 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 12.10.12.4.2).

   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'

12.10.12.4.5 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 RDP2 NODE RDP3
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 RapidsDB Cluster nodes RDP2 and RDP3).

```
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 RDP2 NODE RDP3
TABLE …
```

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

12.10.12.5 Delimited File Formatting

12.10.12.5.1 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';
```

12.10.12.5.2 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 12.10.12.5.7 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 12.10.12.5.3.1 for more details). See 12.10.12.5.8 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 12.10.12.5.9 for more details
ignore_header	'0'	Integer value >= 0	Specifies the number of header records to be skipped See 12for 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 12.10.12.5.10 for more details

12.10.12.5.3 Text Handling

12.10.12.5.3.1. 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 <x0C>
\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}

12.10.12.5.3.2. 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

12.10.12.5.3.3. EMPTY STRINGS

An empty (zero-length) string is defined as a field with two adjacent enclosed_by characters (see 12.10.11.5.8 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 12.10.12.5.6 for more information on nulls) as shown in the example below:

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

12.10.12.5.4 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) (see 12.10.12.5.8).

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

12.10.12.5.5 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.

12.10.12.5.6 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 12.10.12.5.8) 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 12.10.12.5.3.2), 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 12.10.12.5.3.2), 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

12.10.12.5.7 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
```

12.10.12.5.8 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 12.10.12.5.9).

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 12.10.12.5.8). 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:  {}
```

12.10.12.5.9 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
```

12.10.12.5.10 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
```

12.10.12.5.11 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
```

12.10.12.5.12 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 issue 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.

12.10.12.6 ORC Format

12.10.12.6.1 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';
```

12.10.12.6.2 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 7.10.12.7.3 for more details
stripe_size	'262144000'	Integer value >= 0	Specifies the number of bytes in each stripe The default is 250MB

12.10.12.6.3 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.

12.10.12.7 Parquet Format

12.10.12.7.1 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';
```

12.10.12.7.2 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 7.10.12.7.3 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

12.10.12.7.3 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.

12.10.12.8 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:

4. 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 * CATALOG * SCHEMA *
TABLE …
```

5. 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 * 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';
```

12.10.12.9 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 of the HDFS directory names. 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".

12.10.12.10 Hive Metatstore Integration

The Hadoop Connector supports accessing the Hive Metastore to get the schema information for any of the supported table types (see 12.10.12.10.2). The user can specify the Hive databases and tables to be accessed (see 12.10.12.10.1) and then issue SELECT, INSERT, TRUNCATE or DROP TABLE commands against those tables. In addition the user can create new tables (see 12.10.12.10.4) 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.

12.10.12.10.1 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.

12.10.12.10.1.1. 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.

12.10.12.10.1.1.1. 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 INCLUDES='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.

12.10.12.10.1.1.2. 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 12.10.12.10.1.1.3)

```
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”.

12.10.12.10.1.1.3. 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”.

12.10.12.10.2 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 12.10.12.10.3).

12.10.12.10.3 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

12.10.12.10.4 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 12.10.12.10.2.

12.10.12.10.4.1. 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 12.10.12.5.7 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 12.10.12.5.9 for more details

12.10.12.10.4.2. 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 12.10.12.10.5.1) 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 RDP1 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 ','

12.10.12.10.4.3. 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 12.10.12.10.4.1). For tables created as external tables, the DROP table command (see 12.10.12.10.5.2) 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)  |        |
```

12.10.12.10.5 DROP TABLE

Syntax:

```
DROP TABLE [<catalog>.][<schema>.]<table>;
```

The Hadoop Connector supports DROP TABLE for any table currently being accessed from the Hive Metastore.

12.10.12.10.5.1. Dropping Hive-managed Tables

For Hive-managed tables (see 12.10.12.10.4.2) 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@db01:/opt/rdp/current$ 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@db01:/opt/rdp/current$ 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
```

12.10.12.10.5.2. Dropping External Tables

For External tables (see 12.10.12.10.4.3) 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=RDP1.
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)
```

12.10.12.11 Writing to HDFS

12.10.12.11.1 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.

12.10.12.11.1.1 FORMAT='DELIMITED'

The format of the data written to HDFS will follow the delimited file options specified for the Connector (see 7.10.12.5). 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

12.10.12.11.1.2 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

12.10.12.11.1.3 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

12.10.12.11.2 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)
```

12.10.13 Adding an IMPEX Connector

12.10.13.1 Creating an IMPEX Connector

The user can create import and export Connectors using the IMPEX Connector type. To create an IMPEX Connector use the following command

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

where <key> is one of the supported IMPEX Connector properties as defined in the next section.

Example:

```
CREATE CONNECTOR CSV TYPE IMPEX WITH DELIMITER='|', PATH='/';
```

Would create an IMPEX Connector named “CSV” that can run on any node in the RapidsDB cluster and where the delimiter character is '|', and the base path is the root directory ('/'). All other IMPEX properties would use default values as described below.

```
CREATE CONNECTOR CSV TYPE IMPEX WITH DELIMITER='|', PATH='/' NODE RDP1;
```

This would create the same Connector as the previous example with the one difference being that this Connector could only operate on the RapidsDB Cluster node “RDP1”. This restricts the Connector to only the specified node when reading or writing data using a “node://“ URL.

12.10.13.2 IMPEX Connector Properties

The IMPEX Connector type supports the following properties which can be set either when creating the Connector using the CREATE CONNECTOR command (see examples below) or as part of an import reference or export reference (refer to the RapidsDB User Guide for more information on import and export references and creating IMPEX Connectors)

Key:	Default	Syntax	Description
FORMAT	'CSV'	'CSV' | 'RAW'	Specifies the file format: • CSV: A delimited file • RAW: will produce a table with a single VARCHAR column containing the full text of each record in the imported file.

PATH	'/var/tmp/rapids'	'<fully qualified path>'	Specifies the fully qualified path name to use as the base path name for all import references or export references.
ERROR_PATH	'/var/tmp/rapids_errors'	'<fully qualified path>'	Specifies the fully qualified path name to use as the base path for the error files generated if an import operation fails.
ERROR_LIMIT	10	Integer, -1 | 0 | >0	Specifies the maximum number of allowable errors on an import operation. Once the limit is reached the import will be terminated. The possible values are: -1 no limit 0 terminate on first error >0 terminate after specified number of errors
BACKUP	false	[] | true | false	For EXPORT only. For bulk export operations, when the REPLACE option is specified, if BACKUP is “false”, then any existing files with a suffix of “.csv” in the specified folder or sub-folders prior to the export operation will get deleted and then new files created for the export. For bulk export operations, when the REPLACE option is specified, if BACKUP is “true”, then any existing files with a suffix of “.csv” in the specified folders or sub-folders prior to the export operation will be moved to a backup folder so that they can be recovered if needed and then new files created for the export. Note: if “true” or “false” are omitted and just the keyword “BACKUP” is specified, that is equivalent to “true”.
CHARSET	‘UTF-8’	‘<string>’ as defined by the Java charset class	Specifies the character set to be used. Some examples: ‘GBK’ ‘GB2312’

		https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html	'GB18030' 'Big5'
DELIMITER	','	'<char>' Non-empty, single character string	Specifies the field delimiter character. This can only be a single character.
ENCLOSED_BY	"" double quote	'<char>' Non-empty, single character string	Specifies whether a field is optionally enclosed by a specified character. This is 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 ESCAPE_CHAR below 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 delimiters and enclosed_by characters in the data .
FILTER	'*.*'	'<string>' Non-empty, character string using a REGEX format	For IMPORT only. The FILTER property allows the user to control which files are imported in a wildcard import operation and, optionally, how table names are created from the names of imported files. The FILTER value is a character string containing a Java regular expression (a "regex"). When performing a wildcard import, IMPEX examines each filename available from the import source. Only files whose names satisfy the FILTER regex are imported. (For a tutorial on Java regular expressions, see https://www.oracle.com/technical-resources/articles/java/regex.html)

			A "capturing group" can be used in the regex to control how IMPEX creates a table name from the name of an imported file. The characters matched by the first group in the regex are used as the table name. If the regex contains no groups, then the table name will match the first part of the file name (before any dot suffixes). Note: for convenience, a FILTER value that starts with an asterisk is interpreted as a simple filename filter. For example FILTER='*.csv' will import all files with a ".csv" extension.
GUESS	false	[] | true | false	For IMPORT only When "false" specifies that the Connector should treat all columns as polymorphic strings, which will be automatically cast into the appropriate data types depending on the query. When "true" specifies that the Connector should derive the column data types for any columns whose data type has not been specified. The data types are derived by sampling the data being imported and then determining what the appropriate data type would be for each input field in the sampled data. For example, if the sampled data contained 100 records, and a given field contains alphanumeric characters for all 100 records, then it would be mapped to a VARCHAR column, if the field contained just integer characters then it would be mapped to an INTEGER, and so on. Note: if "true" or "false" are omitted and just the keyword

			"GUESS" is specified, that is equivalent to "true".
HEADER	false	[] | true | false	When "true" specifies that the data file has a header record which has the column names to use on an import, or has the column names from the result set for an export. When "false" specifies that there is no header record. Note: if "true" or "false" are omitted and just the keyword "HEADER" is specified, that is equivalent to "true".
TERMINATOR	'\n'	'\n'	Specifies how records are terminated. For this release the TERMINATOR is fixed as '\n', with an optional '\r'
TRAILING	false	[] | true | false	When "true" IMPEX will ignore a trailing field separator (i.e. where the field separator is immediately followed by the record terminator character) on each line of a file being imported and will append a trailing separator to each line of a file being exported. When "false" a trailing field separator will indicate a null value for the last column of the record being imported. For export no trailing field separator will be written out. Note: if "true" or "false" are omitted and just the keyword "TRAILING" is specified, that is equivalent to "true".

Examples:

```
CREATE CONNECTOR CSV TYPE IMPEX WITH DELIMITER='|';
```

Would create an IMPEX Connector named "CSV" where the delimiter character is '|'. The "PATH" property was not set and so would default to "/var/tmp/rapids".


```
CREATE CONNECTOR CSV TYPE IMPEX WITH DELIMITER='|', PATH='/';
```

Would create an IMPEX Connector named “CSV” where the delimiter character is '|' and the “PATH” property is set to the root directory (“/”).

12.11 Refreshing Connector Metadata Information

12.11.1 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 HOST= '10.1.1.20', USER='rapids', PASSWORD='rdpuser'
NODE RDP1 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)
```

12.12 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 RapidsDB System Metadata table rapids.system.connectors (refer to the User Guide for more information on system metadata tables). NOTE: this query must be executed before dropping the Connector.

12.13 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
```

12.14 Disabling and Enabling Connectors

12.14.1 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 (refer to the User Guide for more information on system metadata tables) includes the column “IS_ENABLED” which indicates whether the Connector is enabled or disabled.

Example – the following example queries the CONNECTORS table for a Connector named “ORA1”, which indicates that it is enabled, and then the “ORA1” 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='ORA1';
 IS_ENABLED
 -------
     true

1 row(s) returned
rapids > disable connector ORA1;
0 row(s) returned
rapids > select IS_ENABLED from connectors where connector_name='ORA1';
 IS_ENABLED
 -------
```



```
    false

1 row(s) returned
```

12.14.2 ENABLE Connector

If a Connector has been disabled (see 12.14.1), it can be enabled again using the ENABLE CONNECTOR command:

```
ENABLE CONNECTOR <Connector name>;
```

The CONNECTORS table in the RapidsDB System Metadata tables (refer to the User Guide for more information on system metadata tables) includes the column “IS_ENABLED” which indicates whether the Connector is enabled or disabled.

Example – the following example queries the CONNECTORS table for the “ORA1” Connector, which indicates that it is disabled, and then the “ORA1” 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='ORA1';
 IS_ENABLED
 -------
    false

1 row(s) returned
rapids > enable connector ORA1;
0 row(s) returned
rapids > select IS_ENABLED from connectors where connector_name='ORA1';
 IS_ENABLED
 -------
     true

1 row(s) returned
```

13 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 12.10.5. Having created a MOXE Connector the following sections describe how to create, drop, backup and restore MOXE tables.

13.1 CREATE TABLE

The user can create MOXE tables from the rapids-shell or from JDBC using the CREATE TABLE command:

```
CREATE [REFERENCE] TABLE [IF NOT EXISTS] <tableReference>
(
  <columnDefinition>, …
)
[ PARTITION [BY] (<expr>, ...)] [COMMENT <string>]

where:

<tableReference> is:
  [<connector name>.]MOXE. tableName

<column definition> is:
  <columnName> <type> [[NOT] NULL] [COMMENT <string>]

<type> is:
            INTEGER [(precision)] |
            DECIMAL [(scale[, precision])] |
            FLOAT |
            VARCHAR [(size)] |
            BOOELAN |
            DATE |
            TIMESTAMP
```

13.1.1 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 13.1.2 for more information on reference tables).

Example:

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 comment 'Supplier key',
    >   s_name varchar(25),
    >   s_address varchar(40),
```



```
    >   s_nationkey integer,
    >   s_phone varchar(15),
    >   s_acctbal decimal(17,2),
    >   s_comment varchar(101)
    > ) PARTITION (s_suppkey) comment 'Supplier table';
0 row(s) returned (0.15 sec)
```

This table also has a comment on the column s_suppkey and a table level comment. The comments can be seen by querying the RapidsDB COMMENTS and TABLES tables as shown below:

```
rapids > select * from tables where table_name='SUPPLIER';
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME      IS_PARTITIONED COMMENT
PROPERTIES
-----------------   ---------------   --------------     -------------------  ----------
--------------
MOXE            MOXE           SUPPLIER                  true Supplier
table    NULL

1 row(s) returned (0.07 sec)
rapids > select * from columns where table_name='SUPPLIER';
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME    COLUMN_NAME    DATA_TYPE
ORDINAL    IS_PARTITION_KEY    IS_NULLABLE    PRECISION    PRECISION_RADIX
SCALE CHARACTER_SET    COLLATION    COMMENT         PROPERTIES
-----------------   ---------------   --------------   ---------------   ------------
----------   ----------------------   ---------------   -------------   ----------------------
-------  ------------------   ------------   ----------         --------------
MOXE            MOXE           SUPPLIER      S_SUPPKEY      INTEGER
0                true          false           64                  2
NULL NULL             NULL         Supplier key    NULL
MOXE            MOXE           SUPPLIER      S_NAME         VARCHAR
1               false           true         NULL               NULL
NULL UTF16            BINARY       NULL            NULL
MOXE            MOXE           SUPPLIER      S_ADDRESS      VARCHAR
2               false           true         NULL               NULL
NULL UTF16            BINARY       NULL            NULL
MOXE            MOXE           SUPPLIER      S_NATIONKEY    INTEGER
3               false           true           64                  2
NULL NULL             NULL         NULL            NULL
MOXE            MOXE           SUPPLIER      S_PHONE        VARCHAR
4               false           true         NULL               NULL
NULL UTF16            BINARY       NULL            NULL
MOXE            MOXE           SUPPLIER      S_ACCTBAL      DECIMAL
5               false           true           17                 10
2 NULL             NULL         NULL            NULL
```



```
MOXE            MOXE            SUPPLIER      S_COMMENT      VARCHAR
6               false            true         NULL                NULL
NULL UTF16            BINARY       NULL            NULL

7 row(s) returned (0.08 sec)
```

13.1.2 Reference(replicated) 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 13.2 for details on how to create a reference table.

The following example creates a replicated table that will be replicated to every RapidsDB node in the cluster:

```
rapids > create reference 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)
```

13.1.3 Data Types

13.1.3.1 INTEGER

By default the precision is 64 bits.

13.1.3.2 DECIMAL

By default the scale is 17 and precision is zero. The maximum precision is 17.

13.1.3.3 FLOAT

MOXE uses 64-bit double precision for floats.

13.1.3.4 VARCHAR

MOXE treats all CHAR and VARCHAR values as variable-length strings with a maximum of 32766 characters.

13.1.3.5 TIMESTAMP

The maximum precision for the fractional component of a timestamp is 6 digits, for example:

2018-01-01 09:00:00.123456

13.1.3.6 DATE

The format for a date is YYYY-MM-DD.

13.2 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.

13.3 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”.

13.4 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;
C1C2 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 >
```

13.5 Backing up and Restoring MOXE Tables

MOXE provides support for backing up either the entire database or individual tables using the UNLOAD command (see 13.5.1 below), which can subsequently be restored using the RELOAD command (see 13.5.2). 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 RapidsDB Cluster, then any MOXE "unload" files (see 13.5.1) that were created to backup the data in MOXE tables will no longer be loadable. Important data saved in "unload" files should be persisted (e.g. using the EXPORT statement or an alternate Connector) so it can be recreated after the cluster is reconfigured.

13.5.1 UNLOAD

13.5.1.1 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 RapidsDB node RDP1:


```
rapids@db01:/opt/rdp/current $ ls /home/rapids/dave/moxe/moxe.RDP1/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
```

13.5.1.2 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. Using the example from that section we would have the directory /home/rapids/dave/moxe.RDP1 on RapidsDB Cluster node "RDP1" and the directory /home/rapids/dave/moxe.RDP2 on RapidsDB Cluster node RDP2 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 RapidsDB Cluster node "RDP1":

- /user/rapids/dave/moxe/moxe.RDP1/db_2019_10_31
- /user/rapids/dave/moxe/moxe.RDP1/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 the supplier table on RapidsDB Cluster node "RDP1" would have the following files:

- MOXE.MOXE.SUPPLIER.json
- MOXE.MOXE.SUPPLER$0.casks
- MOXE.MOXE.SUPPLER$1.casks

13.5.2 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 13.6.1), 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)
```

13.6 Checking the Distribution of Data in a Table (Beta)

13.6.1 Partitioned Tables

The following feature is a beta version, and will undergo change in a future release.

The user can display the distribution of data across the partitions of a MOXE table by querying the MOXE system table “rapids.moxe.space”. For this beta version, this table is a special internal table that is not currently integrated into the RapidsDB System Metadata, and it does not share the usual properties of other tables, such as the ability to use DESCRIBE to show the schema for the table. This table is also not intended to be joined with other tables. The user can use a WHERE clause to restrict the information being returned from a query. The schema for this table is shown below:

Column	Data Type	Description
CATALOG_NAME	VARCHAR	The name of the MOXE catalog
SCHEMA_NAME	VARCHAR	The name of the MOXE schema
TABLE_NAME	VARCHAR	The name of the MOXE table

size	INTEGER	The memory allocated to the table measured in MB. Memory is allocated in 4MB chunks, so the minimum size for any table is 4MB.
rows	INTEGER	The number of rows in this partition of the table
node	VARCHAR	The name of the RapidsDB Cluster node for this partition
part	INTEGER	The partition number

When displaying the information for a partitioned table, a row will be returned for each partition of the table which will show the maximum size for any row in that partition and the number of rows in that partition. The example below shows the information for the MOXE table "LINEITEM" which is partitioned across the three nodes in the RapidsDB Cluster:

```
rapids > select * from rapids.moxe.space where table_name='LINEITEM';
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME      size       rows node    part
----------------   --------------   -------------               ------
MOXE            MOXE           LINEITEM         204    1000188 RDP1    0
MOXE            MOXE           LINEITEM         204     998978 RDP1    1
MOXE            MOXE           LINEITEM         204     999590 RDP3    4
MOXE            MOXE           LINEITEM         204     999672 RDP3    5
MOXE            MOXE           LINEITEM         204    1000176 RDP2    2
MOXE            MOXE           LINEITEM         204    1002611 RDP2    3


rapids > select count(*) from lineitem;
      [1]
      ---
  6001215

1 row(s) returned (0.41 sec)
```

As can be seen from the above example, the data is evenly distributed across the partitions. In the event that the data is not evenly distributed then the partitioning column(s) can be changed (see 13.1.1) to try and achieve a more even distribution. Note: it is not possible to change the partitioning columns for an existing table, in order to change the partitioning columns the table must be dropped and recreated with a different partitioning column(s).

The total memory allocated to the table is 1.24GB (204MB * 6), and the average row size is 216 bytes (1.24GB/6001215).

13.6.2 Monitoring IMPORT into a Partitioned Table

This feature is very useful when importing data into a MOXE table to monitor the progress of an import operation. The example below shows how the progress of an import operation into a table named "LINEITEM":

From one rapids-shell session an import is performed using an IMPEX Connector named "IMPORT_TPCH":

```
rapids > import lineitem from
import_tpch::node://rdp1/software/data/data/tpch/sf1/lineitem.tbl;
0 row(s) returned (26.42 sec)
```

From a second rapids-shell session the progress of the import can be monitored:

```
rapids > select * from rapids.moxe.space;
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME      size      rows node    part
-----------------   ---------------   --------------     ----------     ------ ------   ------
MOXE            MOXE           LINEITEM          64    310272 RDP2    2
MOXE            MOXE           LINEITEM          64    309760 RDP2    3
MOXE            MOXE           LINEITEM          64    310528 RDP1    0
MOXE            MOXE           LINEITEM          64    310272 RDP1    1
MOXE            MOXE           LINEITEM          48    226166 RDP3    4
MOXE            MOXE           LINEITEM          48    229121 RDP3    5

6 row(s) returned (0.27 sec)
rapids > select * from rapids.moxe.space;
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME      size      rows node    part
-----------------   ---------------   --------------     ----------     ------ ------   ------
MOXE            MOXE           LINEITEM         112    542720 RDP2    2
MOXE            MOXE           LINEITEM         112    542208 RDP2    3
MOXE            MOXE           LINEITEM         112    542976 RDP1    0
MOXE            MOXE           LINEITEM         112    542976 RDP1    1
MOXE            MOXE           LINEITEM          96    472752 RDP3    4
MOXE            MOXE           LINEITEM          96    471501 RDP3    5

6 row(s) returned (0.25 sec)
rapids > select * from rapids.moxe.space;
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME      size       rows node    part
-----------------   ---------------   --------------     ----------     ------ ------   ------
MOXE            MOXE           LINEITEM         204     999590 RDP3    4
MOXE            MOXE           LINEITEM         204     999672 RDP3    5
MOXE            MOXE           LINEITEM         204    1000176 RDP2    2
MOXE            MOXE           LINEITEM         204    1002611 RDP2    3
MOXE            MOXE           LINEITEM         204    1000188 RDP1    0
MOXE            MOXE           LINEITEM         204     998978 RDP1    1

6 row(s) returned (0.11 sec)
rapids > select * from rapids.moxe.space;
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME      size       rows node    part
-----------------   ---------------   --------------     ------       ------ ------   ------
```



```
MOXE            MOXE           LINEITEM         204    1000188 RDP1    0
MOXE            MOXE           LINEITEM         204     998978 RDP1    1
MOXE            MOXE           LINEITEM         204    1000176 RDP2    2
MOXE            MOXE           LINEITEM         204    1002611 RDP2    3
MOXE            MOXE           LINEITEM         204     999590 RDP3    4
MOXE            MOXE           LINEITEM         204     999672 RDP3    5

6 row(s) returned (0.08 sec)
rapids > select count(*) from lineitem;
      [1]
      ---
  6001215

1 row(s) returned (0.41 sec)
```

13.6.3 Replicated Tables

For replicated tables, the information returned will be shown for each copy of table on each node in the RapidsDB Cluster. The example below shows the information returned for a replicated table named "NATION" on a 3-node RapidsDB Cluster:

```
rapids > select * from rapids.moxe.space where table_nam ='N TI N';
CATALOG_NAME    SCHEMA_NAME    TABLE_NAME      size    rows  node    part
-----------------   ---------------   ----------------------      ---------   ---------   ---------   ------
MOXE            MOXE           NATION             4      25 RDP3    0
MOXE            MOXE           NATION             4      25 RDP2    0
MOXE            MOXE           NATION             4      25 RDP1    0

3 row(s) returned (0.09 sec)
```

Each copy of the table is 4MB in size, which is the minimum size for a table.

13.7 Changing the MOXE Configuration

In order to change the MOXE configuration dynamically any existing database must be dropped using the RESET command (see 13.7.1). Before dropping the database the user should consider whether they want to take a backup of the existing database (see 13.5). 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 13.7.1)
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 13.7.1)
   b. Drop the existing Connector
   c. Create the new MOXE Connector

13.7.1 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)
```


14 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.

15 Audit Logging

15.1 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).

15.2 Audit Logging Commands

15.2.1 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.

15.2.2 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)
```

15.2.3 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 15.2.4)

15.2.4 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 15.2.2)

15.3 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 RDP1 ;
0 row(s) returned (0.64 sec)
```

audit.log:

```
[INFO ] 2020-08-05 13:44:15 audit-log - RAPIDS,RDP1,SSN_1@RDP1,"CREATE USER DAVE "
[INFO ] 2020-08-05 13:45:38 audit-log - RAPIDS,RDP1,SSN_1@RDP1,"ADD AUDIT ON USER DAVE;"
[INFO ] 2020-08-05 14:23:37 audit-log - RAPIDS,RDP1,SSN_1@RDP1,"select * from tables;"
[INFO ] 2020-08-05 14:25:23 audit-log - RAPIDS,RDP1,SSN_1@RDP1,"CREATE CONNECTOR PGP6 TYPE
POSTGRES WITH PASSWORD='XXXX', URL='jdbc:postgresql://localhost/tpch', USER='XXXX' NODE RDP1;"
```

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.

15.4 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.

