Splice Machine Documentation Test

Last generated: February 27, 2018

Table of Contents


splice
MACHINE

Tuning and Debugging Query Performance

This section contains the following topics to help you optimize your Splice Machine database queries:

Topic	Describes
Optimizing Queries	Gets you started with optimizing your queries for Splice Machine.
Using Statistics	Using our statistics gathering facility to help optimize queries.
Explain Plan	The Explain Plan feature, which allows you to examine the execution plan for a query without executing the query.
Logging	The Splice Machine logging facility.
Debugging	Describes the parameter values to use when using debugger software with Splice Machine.

For access to the source code for the Community Edition of Splice Machine, visit our open source GitHub repository.

Optimizing Splice Machine Queries

This topic introduces you to Splice Machine query optimization techniques, including information about executing SQL expressions without a table context, and using optimization hints.

Introduction to Query Optimization

Here are a few mechanisms you can use to optimize your Splice Machine queries:

Optimization Mechanism	Description
Use Explain Plan	You can use the Splice Machine Explain Plan facility to display the execution plan for a statement without actually executing the statement. You can use Explain Plan to help determine options such as which join strategy to use or which index to select. See the About Explain Plan topic.
Use Statistics	Your database administrator can refine which statistics are collected on your database, which in turn enhances the operation of the query optimizer. See the Using Statistics topic.
Use `WHERE` clauses	Use a `WHERE` clause in your queries to restrict how much data is scanned.
Use indexes	You can speed up a query by having an index on the criteria used in the `WHERE` clause, or if the `WHERE` clause is using a primary key. Composite indexes work well for optimizing queries.
Use Splice Machine *query hints*	The Splice Machine query optimizer allows you to provide hints to help in the optimization process.

Using Select Without a Table

Sometimes you want to execute SQL scalar expressions without having a table context. For example, you might want to create a query that evaluates an expression and returns a table with a single row and one column. Or you might want to evaluate a list of comma-separated expressions and return a table with a single row and multiple columns one for each expression.

In Splice Machine, you can execute queries without a table by using the `sysibm.sysdummy1` dummy table. Here's the syntax:

```
select expression FROM sysibm.sysdummy1
```

And here's an example:

```
splice> select 1+ 1 from sysibm.sysdummy1;
```

Using Splice Machine Query Hints

You can use *hints* to help the Splice Machine query interface optimize your database queries.

> **NOTE:** The Splice Machine optimizer is constantly being improved, and new hint types sometimes get added. One recent addition is the ability to specify that a query should be run on (or not on) Spark, if possible.

Types of Hints

There are different kinds of hints you can supply, each of which is described in a section below; here's a summary:

Hint Type	Examples	Used to Specify
Index	`--splice-properties index=my_index`	Which index to use or not use
Join Order	`--splice-properties joinOrder=fixed`	Which join order to use for two tables
Join Strategy	`--splice-properties joinStrategy=sortmerge`	How a join is processed (in conjunction with the Join Order hint)
Pinned Table	`--splice-properties pin=true`	That you want the pinned (cached in memory) version of a table used in a query
Spark	`--splice-properties useSpark=true`	That you want a query to run (or not run) on Spark
Delete	`--splice-properties bulkDeleteDirectory='/path'`	That you are deleting a large amount of data and want to bypass the normal write pipeline to speed up the deletion.

Including Hints in Your Queries

Hints MUST ALWAYS be at the end of a line, meaning that you must always terminate hints with a newline character.

You cannot add the semicolon that terminates the command immediately after a hint; the semicolon must go on the next line, as shown in the examples in this topic.

Many of the examples in this section show usage of hints on the `splice>` command line. Follow the same rules when using hints programmatically.

Hints can be used in two locations: after a table identifier or after a `FROM` clause. Some hint types can be use after a table identifier, and some can be used after a `FROM` clause:

Hint after a:	Hint types	Example

Table identifier	`index`
	`joinStrategy`
	`pin`
	`useSpark`
	`bulkDeleteDirectory`
		SELECT * FROM
		member_info m, rewards r,
		points p --SPLICE-PROPERTIES index=ie_point WHERE...

A `FROM` clause	`joinOrder`	SELECT * FROM --SPLICE-PROPERTIES joinOrder=fixed mytable1 e, mytable2 t WHERE e.id = t.parent_id;

This example shows proper placement of the hint and semicolon when the hint is at the end of statement:

```
SELECT * FROM my_table --splice-properties index=my_index;
```

If your command is broken into multiple lines, you still must add the hints at the end of the line, and you can add hints at the ends of multiple lines; for example:

```
SELECT * FROM my_table_1 --splice-properties index=my_index
, my_table_2 --splice-properties index=my_index_2
WHERE my_table_1.id = my_table_2.parent_id;
```

NOTE: In the above query, the first command line ends with the first index hint, because hints must always be the last thing on a command line. That's why the comma separating the table specifications appears at the beginning of the next line.

Index Hints

Use *index hints* to tell the query interface how to use certain indexes for an operation.

To force the use of a particular index, you can specify the index name; for example:

```
splice> SELECT * FROM my_table --splice-properties index=my_index
> ;
```

To tell the query interface to not use an index for an operation, specify the null index. For example:

```
splice> SELECT * FROM my_table --splice-properties index=null
> ;
```

And to tell the query interface to use specific indexes on specific tables for an operation, you can add multiple hints. For example:

```
splice> SELECT * FROM my_table_1   --splice-properties index=my_index
> , my_table_2                --splice-properties index=my_index_2
> WHERE my_table_1.id = my_table_2.parent_id;
```

Important Note About Placement of Index Hints

Each `index` hint in a query **MUST** be specified alongside the table containing the index, or an error will occur.

For example, if we have a table named `points` with an index named `ie_point` and another table named `rewards` with an index named `ie_rewards`, then this hint works as expected:

```
SELECT * FROM   member_info m,
   rewards r,
   points p    --SPLICE-PROPERTIES index=ie_point
WHERE...
```

But the following hint will generate an error because `ie_rewards` is not an index on the points table.

```
SELECT * FROM
   member_info m,
   rewards r,
   points p    --SPLICE-PROPERTIES index=ie_rewards
WHERE...
```

JoinOrder Hints

Use `JoinOrder` hints to tell the query interface in which order to join two tables. You can specify these values for a `JoinOrder` hint:

- Use `joinOrder=FIXED` to tell the query optimizer to order the table join according to how where they are named in the `FROM` clause.
- Use `joinOrder=UNFIXED` to specify that the query optimizer can rearrange the table order.

> **NOTE:** `joinOrder=UNFIXED` is the default, which means that you don't need to specify this hint to allow the optimizer to rearrange the table order.

Here are examples:

Hint	Example
joinOrder=FIXED	splice> SELECT * FROM --SPLICE-PROPERTIES joinOrder=fixed> mytable1 e, mytable2 t> WHERE e.id = t.parent_id;
joinOrder=UNFIXED	splice> SELECT * from --SPLICE-PROPERTIES joinOrder=unfixed> mytable1 e, mytable2 t WHERE e.id = t.parent_id;

JoinStrategy Hints

You can use a `JoinStrategy` hint in conjunction with a `joinOrder` hint to tell the query interface how to process a join. For example, this query specifies that the `SORTMERGE` join strategy should be used:

```
SELECT * FROM      --SPLICE-PROPERTIES joinOrder=fixed
   mytable1 e, mytable2 t --SPLICE-PROPERTIES joinStrategy=SORTMERGE
   WHERE e.id = t.parent_id;
```

And this uses a `joinOrder` hint along with two `joinStrategy` hints:

```
SELECT *
  FROM --SPLICE-PROPERTIES joinOrder=fixed
  keyword k
  JOIN campaign c  --SPLICE-PROPERTIES joinStrategy=NESTEDLOOP
    ON k.campaignid = c.campaignid
  JOIN adgroup g  --SPLICE-PROPERTIES joinStrategy=NESTEDLOOP
    ON k.adgroupid = g.adgroupid
  WHERE adid LIKE '%us_gse%'
```

You can specify these join strategies:

JoinStrategy Value	Strategy Description
BROADCAST	Read the results of the Right Result Set (*RHS*) into memory, then for each row in the left result set (*LHS*), perform a local lookup to determine the right side of the join. `BROADCAST` will only work on equijoin (=) predicates that do not include a function call.

`MERGE`	Read the Right and Left result sets simultaneously in order and join them together as they are read. `MERGE` joins require that both the left and right result sets be sorted according to the join keys. `MERGE` requires an equijoin predicate that does not include a function call.
`NESTEDLOOP`	For each row on the left, fetch the values on the right that match the join. `NESTEDLOOP` is the only join that can work with any join predicate of any type; however this type of join is generally very slow.
`SORTMERGE`	Re-sort both the left and right sides according to the join keys, then perform a `MERGE` join on the results. `SORTMERGE` requires an equijoin predicate with no function calls.

Pinned Table Hint

You can use the `pin` hint to specify to specify that you want a query to run against a pinned version of a table.

```
splice> PIN TABLE myTable;splice> SELECT COUNT(*) FROM my_table --splice-properties
pin=true> ;
```

You can read more about pinning tables in the `PIN TABLE` statement topic.

Spark Hints

You can use the `useSpark` hint to specify to the optimizer that you want a query to run on (or not on) Spark. The Splice Machine query optimizer automatically determines whether to run a query through our Spark engine or our HBase engine, based on the type of query; you can override this by using a hint:

NOTE: The Splice Machine optimizer uses its estimated cost for a query to decide whether to use spark. If your statistics are out of date, the optimizer may end up choosing the wrong engine for the query.

```
splice> SELECT COUNT(*) FROM my_table --splice-properties useSpark=true> ;
```

You can also specify that you want the query to run on HBase and not on Spark. For example:

```
splice> SELECT COUNT(*) FROM your_table --splice-properties useSpark=false> ;
```

Delete Hints

You can use the `bulkDeleteDirectory` hint to specify that you want to use our bulk delete feature to optimize the deletion of a large amount of data. Similar to our bulk import feature, bulk delete generates HFiles, which allows us to bypass the Splice Machine write pipeline and HBase write path when performing the deletion. This can significantly speed up the deletion process.

You need to specify the directory to which you want the temporary HFiles written; you must have write permissions on this directory to use this feature. If you're specifying an S3 bucket on AWS, please review our Configuring an S3 Bucket for Splice Machine Access tutorial before proceeding.

```
splice> DELETE FROM my_table --splice-properties bulkDeleteDirectory='/bulkFilesPat
h'
;
```

NOTE: We recommend performing a major compaction on your database after deleting a large amount of data; you should also be aware of our new `SYSCS_UTIL.SET_PURGE_DELETED_ROWS` system procedure, which you can call before a compaction to specify that you want the data physically (not just logically) deleted during compaction.