Performance Enhancements in Spark using Informational Referential Integrity Constraints

We have implemented Informational Referential Integrity Constraints support in Apache Spark. This work opens up an area of query optimization techniques that rely on referential integrity constraints semantics. We have seen significant performance improvements up to 8X in some of the TPC-DS benchmark queries. In this blog, we go over some of the highlights of this work.

In the data warehousing world referential integrity (RI) constraints are used to enforce certain properties of an entity. SQL constraints specify rules that apply to data. They are used for two purposes: *data integrity* and *query optimization*.

* A *primary key constraint* is used to prevent multiple rows from having the same value in the same column or combinations of column and prevents values from being null.
* A *foreign key constraint* (also referred to as a *referential integrity constraint*) requires values in a column or combination of columns to match the values in a primary key in its parent table.

RI constraints can be enforced or informational.

* *Enforced* - ensures that all data modifications applied to a table satisfy the constraint.
* *Validated* – ensures that all the data that currently resides in a table satisfies the constraint.
* *Informational,* or *statistical* – used to improve query optimization and performance. The conditions for a given constraint are known to be true, so the constraint does not need to be validated or enforced.

We added support for *informational primary key* and *foreign key (referential integrity*) constraints in Spark to achieve enterprise level TPC-DS performance.

## Performance Improvements:

The constraints are used to optimize the query processing by the Spark optimizer.

There are two areas of improvement: query rewrite transformations, and optimizer and runtime enhancements. Some of the improvements are listed below.

1. Query rewrite transformations

* RI join elimination
* Redundant join elimination
* Group by push down through join
* Sort elimination
* Existential subquery to join

1. Optimizer and runtime improvements

* Proving maximal cardinality
* Early out joins
* Star schema detection

## Performance results using 1TB TPC-DS benchmark

The below results are with using star schema detection, existential subquery to inner join transformation rewrite and group by pushdown through join optimizations using RI.

|  |  |  |  |
| --- | --- | --- | --- |
| **TPC-DS**  **Query** | **spark-2.2**  **(secs)** | **spark-2.2-ri**  **(secs)** | **Speedup** |
| Q06 | 106 | 19 | 5x |
| Q10 | 355 | 190 | 2x |
| Q13 | 296 | 98 | 3x |
| Q15 | 147 | 17 | 8x |
| Q16 | 1394 | 706 | 2x |
| Q17 | 398 | 146 | 2x |
| Q24 | 485 | 249 | 2x |
| Q25 | 421 | 147 | 2x |
| Q29 | 380 | 126 | 3x |
| Q35 | 462 | 285 | 1.5x |
| Q45 | 93 | 17 | 5x |
| Q69 | 327 | 173 | 1.8x |
| Q74 | 237 | 119 | 2x |
| Q85 | 104 | 42 | 2x |
| Q94 | 603 | 307 | 2x |

The tests were run using the following system configuration:

**Cluster: 4-node cluster, each node having:** 12 2 TB disks,

Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 128 GB RAM, 10 Gigabit Ethernet

Number of cores: 48

**Apache Hadoop 2.7.3, Apache Spark 2.2 main**

**Database info:**

Schema: TPCDS

Scale factor: 1TB total space

Storage format: Parquet with Snappy compression

## Contributions to the community:

In order to implement the constraints, we have added the data model to define and store the constraints. We have added support for new DDLs to add a primary key and foreign key constraint using the ALTER TABLE .. ADD CONSTRAINT command.

We have submitted the following PRs to the Apache Spark community.

* [SPARK-21784][SQL] Adds support for defining informational primary key and foreign key constraints using ALTER TABLE DDL.
  + PR - https://github.com/apache/spark/pull/18994
* [SPARK-23750][SQL] Inner Join Elimination based on Informational RI constraints
  + PR - <https://github.com/apache/spark/pull/20868>
  + This PR contains the changes for the DDL’s from SPARK-21784 as well as the optimization changes for join elimination.

We have submitted the design specification that covers constraint specification, metastore storage, constraint validation, and maintenance and optimization features [here](https://docs.google.com/document/d/17r-cOqbKF7Px0xb9L7krKg2-RQB_gD2pxOmklm-ehsw/edit.).

The umbrella JIRA is [SPARK-19842]. Please feel free to provide any feedback or comments on this JIRA or the PR’s.

Join Elimination Rewrite:

This transformation detects RI joins and eliminates the parent/Primary Key(PK) table if none of its columns, other than the PK columns, are referenced in the query.

### Get the code:

In order to use and try out the Join elimination optimization, you can fetch the code from PR: <https://github.com/apache/spark/pull/20868>

### Build:

Once you have fetched the code from this PR, you can build the source. The assumption is you are already setup to build spark. If so, the command to build this branch is also similar to spark.

For e.g.

mvn -Pyarn -Phadoop-2.7 -Dhadoop.version=2.7.0 -Phive -Phive-thriftserver -DskipTests clean package

### Enable the RI optimization join elimination rewrite rule

spark.conf.set("spark.sql.riJoinElimination", "true")

### Test:

Run the following sql commands in the spark-shell.

Under the spark home directory, run ./bin/spark-shell

and copy paste the commands below in the shell.

sql("create table dim1 (dim1c1 int, dim1c2 int, dim1c3 int, dim1c4 int)")

sql("create table dim2 (dim2c1 int, dim2c2 int, dim2c3 int, dim2c4 int)")

sql("create table fact (factc1 int, factc2 int, factc3 int, factc4 int)")

sql("alter table dim1 add constraint pk1 primary key(dim1c1)")

sql("alter table dim2 add constraint pk1 primary key(dim2c1)")

sql("alter table fact add constraint fk1 foreign key (factc1) references dim1(dim1c1)")

sql("alter table fact add constraint fk2 foreign key (factc2) references dim2(dim2c1)")

sql("insert into dim1 values (1,1,1,1)")

sql("insert into dim1 values (2,2,2,2)")

sql("insert into dim2 values (10,1,1,1)")

sql("insert into dim2 values (20,2,2,2)")

sql("insert into fact values (1,10,1,1)")

sql("insert into fact values (1,20,1,1)")

sql("insert into fact values (2,20,2,2)")

sql("insert into fact values (2,20,2,2)")

spark.sql("select factc1 from fact, dim1, dim2 where factc1 = dim1c1 and factc2 = dim2c1 and dim1c1= 1 and dim2c1=20").explain(true)

spark.conf.set("spark.sql.riJoinElimination", "true")

spark.sql("select factc1 from fact, dim1, dim2 where factc1 = dim1c1 and factc2 = dim2c1 and dim1c1= 1 and dim2c1=20").explain(true)

After enabling the spark.conf.set("spark.sql.riJoinElimination", "true"), the physical plan and optimized plan look like this:

== Optimized Logical Plan ==

Project [factc1#200]

+- Filter ((((factc1#200 = 1) && (factc2#201 = 2)) && isnotnull(factc1#200)) && isnotnull(factc2#201))

+- HiveTableRelation `default`.`fact`, org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, [factc1#200, factc2#201, factc3#202, factc4#203]

== Physical Plan ==

\*(1) Project [factc1#200]

+- \*(1) Filter ((((factc1#200 = 1) && (factc2#201 = 2)) && isnotnull(factc1#200)) && isnotnull(factc2#201))

+- HiveTableScan [factc1#200, factc2#201], HiveTableRelation `default`.`fact`, org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, [factc1#200, factc2#201, factc3#202, factc4#203]

You should see that the join between fact, dim1, dim2 has been rewritten and optimized to just a query on the fact table.

## Summary:

We added support for informational primary key and foreign key (referential integrity) constraints in Spark to achieve enterprise level TPC-DS performance. This work opens up an area of query optimization techniques that rely on referential integrity constraints semantics.  We have showcased the 8X improvements in this [Apache Summit talk](https://www.youtube.com/watch?v=zrtdN05wQPs). There is a related blog that explains the star schema enhancements here: [here](https://developer.ibm.com/code/2018/04/16/star-schema-enhancements-in-apache-spark/).

We have the initial PRs submitted to the community. We would need the DDL and data model PR to be merged before submitting PR’s for the other DDL’s and optimizations. Please feel free to provide community feedback on the umbrella JIRA or the PRs.