DynamicPartitions

Dynamic Partitions

Documentation

This is the design document for dynamic partitions in Hive. Usage information is also available:

- Tutorial: Dynamic-Partition Insert
- Hive DML: Dynamic Partition Inserts
- HCatalog Dynamic Partitioning
  - Usage with Pig
  - Usage from MapReduce

References:

- Original design doc
- HIVE-936

Terminology

- Static Partition (SP) columns: in DML/DDL involving multiple partitioning columns, the columns whose values are known at COMPILE TIME (given by user).
- Dynamic Partition (DP) columns: columns whose values are only known at EXECUTION TIME.

Syntax

DP columns are specified the same way as it is for SP columns – in the partition clause. The only difference is that DP columns do not have values, while SP columns do. In the partition clause, we need to specify all partitioning columns, even if all of them are DP columns.

In INSERT ... SELECT ... queries, the dynamic partition columns must be **specified last** among the columns in the SELECT statement and **in the same order** in which they appear in the PARTITION() clause.

- all DP columns – only allowed in nonstrict mode. In strict mode, we should throw an error. e.g.,

```
INSERT OVERWRITE TABLE T PARTITION (ds, hr)
SELECT key, value, ds, hr FROM srcpart WHERE ds is not null and hr>10;
```

- mixed SP & DP columns. e.g.,

```
INSERT OVERWRITE TABLE T PARTITION (ds='2010-03-03', hr)
SELECT key, value, /*ds,*/ hr FROM srcpart WHERE ds is not null and hr>10;
```

- SP is a subpartition of a DP: should throw an error because partition column order determins directory hierarchy. We cannot change the hierarchy in DML. e.g.,

```
-- throw an exception
INSERT OVERWRITE TABLE T PARTITION (ds, hr = 11)
SELECT key, value, ds/*, hr*/ FROM srcpart WHERE ds is not null and hr=11;
```

- multi-table insert. e.g.,

```
FROM S
INSERT OVERWRITE TABLE T PARTITION (ds='2010-03-03', hr)
SELECT key, value, ds, hr FROM srcpart WHERE ds is not null and hr>10
INSERT OVERWRITE TABLE R PARTITION (ds='2010-03-03, hr=12)
SELECT key, value, ds, hr from srcpart where ds is not null and hr = 12;
```

- CTAS – syntax is a little bit different from CTAS on non-partitioned tables, since the schema of the target table is not totally derived from the select-clause. We need to specify the schema including partitioning columns in the create-clause. e.g.,

```
CREATE TABLE T (key int, value string) PARTITIONED BY (ds string, hr int) AS
SELECT key, value, ds, hr+1 hr1 FROM srcpart WHERE ds is not null and hr>10;
```

The above example shows the case of all DP columns in CTAS. If you want put some constant for some partitioning column, you can specify it in the select-clause. e.g,

```
CREATE TABLE T (key int, value string) PARTITIONED BY (ds string, hr int) AS
SELECT key, value, "2010-03-03", hr+1 hr1 FROM srcpart WHERE ds is not null and hr>10;
```

Design

- In SemanticAnalyser.genFileSinkPlan(), parse the input and generate a list of SP and DP columns. We also generate a mapping from the input ExpressionNode to the output DP columns in FileSinkDesc.
- We also need to keep a HashFunction class in FileSinkDesc to evaluate partition directory names from the input expression value.
- In FileSinkOperator, setup the input -> DP mapping and Hash in initOp(). and determine the output path in processOp() from the mapping.
- ObjectInspector setup?
- MoveTask: since DP columns represent a subdirectory tree, it is possible to use one MoveTask at the end to move the results from the temporary directory to the final directory.
- post exec hook for replication: remove all existing data in DP before creating new partitions. We should make sure replication hook recognize all the modified partitions.
- metastore support: since we are creating multiple parititions in a DML, metastore should be able to create all these partitions. Need to investigate.

Design issues

1) Data type of the dynamic partitioning column:
A dynamic partitioning column could be the result of an expression. For example:

```
-- part_col is partitioning column
create table T as select a, concat("part_", part_col) from S where part_col is not null;
```

Although currently there is not restriction on the data type of the partitioning column, allowing non-primitive columns to be partitioning column probably doesn't make sense. The dynamic partitioning column's type should be derived from the expression. The data type has to be able to be converted to a string in order to be saved as a directory name in HDFS.

2) Partitioning column value to directory name conversion:
After converting column value to string, we still need to convert the string value to a valid directory name. Some reasons are:

- string length is unlimited in theory, but HDFS/local FS directory name length is limited.
- string value could contains special characters that is reserved in FS path names (such as '/' or '..').
- what should we do for partition column ObjectInspector?

We need to define a UDF (say hive_qname_partition(T.part_col)) to take a primitive typed value and convert it to a qualified partition name.

3) Due to 2), this dynamic partitioning scheme qualifies as a hash-based partitioning scheme, except that we define the hash function to be as close as the input value. We should allow users to plugin their own UDF for the partition hash function. Will file a follow up JIRA if there is sufficient interests.

4) If there are multiple partitioning columns, their order is significant since that translates to the directory structure in HDFS: partitioned by (ds string, dept int) implies a directory structure of ds=2009-02-26/dept=2. In a DML or DDL involving partitioned table, So if a subset of partitioning columns are specified (static), we should throw an error if a dynamic partitioning column is lower. Example:

```
create table nzhang_part(a string) partitioned by (ds string, dept int);
insert overwrite nzhang_part (dept=1) select a, ds, dept from T where dept=1 and ds is not null;
```