PartitionedViews

This is a followup to ViewDev for adding partition-awareness to views.

Use Cases

1. An administrator wants to create a set of views as a table/column renaming layer on top of an existing set of base tables, without breaking any existing dependencies on those tables. To read-only users, the views should behave exactly the same as the underlying tables in every way. Among other things, this means users should be able to browse available partitions.
2. A base table is partitioned on columns (ds,hr) for date and hour. Besides this fine-grained partitioning, users would also like to see a virtual table of coarse-grained (date-only) partitioning in which the partition for a given date only appears after all of the hour-level partitions of that day have been fully loaded.
3. A view is defined on a complex join+union+aggregation of a number of underlying base tables and other views, all of which are themselves partitioned. The top-level view should also be partitioned accordingly, with a new partition not appearing until corresponding partitions have been loaded for all of the underlying tables.

Approaches

1. One possible approach mentioned in HIVE-1079 is to infer view partitions automatically based on the partitions of the underlying tables. A command such as SHOW PARTITIONS could then synthesize virtual partition descriptors on the fly. This is fairly easy to do for use case #1, but potentially very difficult for use cases #2 and #3. So for now, we are punting on this approach.
2. Instead, per HIVE-1941, we will require users to explicitly declare view partitioning as part of CREATE VIEW, and explicitly manage partition metadata via ALTER VIEW ADD|DROP PARTITION. This allows all of the use cases to be satisfied (while placing more burden on the user, and taking up more metastore space). With this approach, there is no real connection between view partitions and underlying table partitions; it's even possible to create a partitioned view on an unpartitioned table, or to have data in the view which is not covered by any view partition. One downside here is that a UI will not be able to show last access time and physical information such as file size when browsing available partitions. (In theory, stats could work via an explicit ANALYZE, but analyzing a view would need some work.)

Syntax

```
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment], ...) ]
[COMMENT table_comment]
[PARTITIONED ON (col1, col2, ...)]
[TBLPROPERTIES ...]
AS SELECT ...

ALTER VIEW view_name ADD [IF NOT EXISTS] partition_spec partition_spec ...

ALTER VIEW view_name DROP [IF EXISTS] partition_spec, partition_spec, ...

partition_spec:
  : PARTITION (partition_col = partition_col_value, partition_col = partiton_col_value, ...)
```

Notes:

- Whereas CREATE TABLE uses PARTITIONED BY, CREATE VIEW uses PARTITIONED ON. This difference is intentional because in CREATE TABLE, the PARTITIONED BY clause specifies additional column definitions which are appended to the non-partitioning columns. With CREATE VIEW, the PARTITIONED ON clause references (by name) columns already produced by the view definition. Only column names appear in PARTITIONED ON; no types etc. However, to match the CREATE TABLE convention of trailing partitioning columns, the columns referenced by the PARTITIONED ON clause must be the last columns in the view definition, and their order in the PARTITIONED ON clause must match their order in the view definition.
- The ALTER VIEW ADD/DROP partition syntax is identical to ALTER TABLE, except that it is illegal to specify a LOCATION clause.
- Other ALTER TABLE commands which operate on partitions (e.g. TOUCH/ARCHIVE) are not supported. (But maybe we need to support TOUCH?)

Metastore

When storing view partition descriptors in the metastore, Hive omits the storage descriptor entirely. This is because there is no data associated with the view partition, so there is no need to keep track of partition-level column descriptors for table schema evolution, nor a partition location.

Strict Mode

Hive strict mode (enabled with hive.mapred.mode=strict) prevents execution of queries lacking a partition predicate. This only applies to base table partitions. What does this mean?

Suppose you have table T1 partitioned on C1, and view V1 which selects FROM T1 WHERE C1=5. Then a query such as SELECT * FROM V1 will succeed even in strict mode, since the predicate inside of the view constrains C1.

Likewise, suppose you have view V2 which selects from T1 (with no WHERE clause) and is partitioned on C2. Then a query such as SELECT * FROM V2 WHERE C2=3 will fail; even though the view partition column is constrained, there is no predicate on the underlying T1's partition column C1.

View Definition Changes

Currently, changing a view definition requires dropping the view and recreating it. This implies dropping and recreating all existing partitions as well, which could be very expensive.

This implies that followup support for CREATE OR REPLACE VIEW is very important, and that it needs to preserve existing partitions (after validating that they are still compatible with the new view definition).

Hook Information

Although there is currently no connection between the view partition and underlying table partitions, Hive does provide dependency information as part of the hook invocation for ALTER VIEW ADD PARTITION. It does this by compiling an internal query of the form

```
SELECT * FROM view_name
WHERE view_partition_col1 = 'val1' AND view_partition_col=2 = 'val2' ...
```

and then capturing the table/partition inputs for this query and passing them on to the ALTER VIEW ADD PARTITION hook results.

This allows applications to track the dependencies themselves. In the future, Hive will automatically populate these dependencies into the metastore as part of HIVE-1073.