# Hive Incremental updates

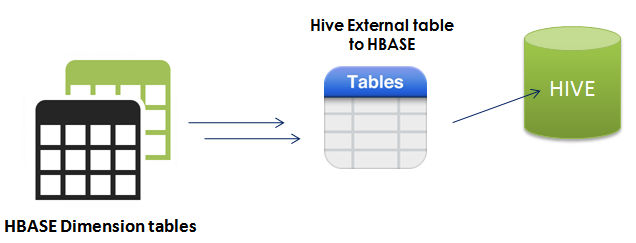
HDFS files are immutable, meaning you cannot edit a file after it has been written. This limitation makes incremental updates in hive or any other applications based on HDFS very challenging.

There are work-around and possible solutions used industry wide for implementing the incremental load strategy are discussed as below:

1. **HBASE storage**

HBase is a NoSQL database that runs on Hadoop. Since it provides native support for updates, it is often considered for dimension data maintenance.

Hive /Impala etc tools can ingest “real-time” streams of high velocity data into HBase, minimizing update latency. Also we can build Hive table which can refer to hbase table ( Hbase Storage handlers ) and will be used seamless, for queries.

**

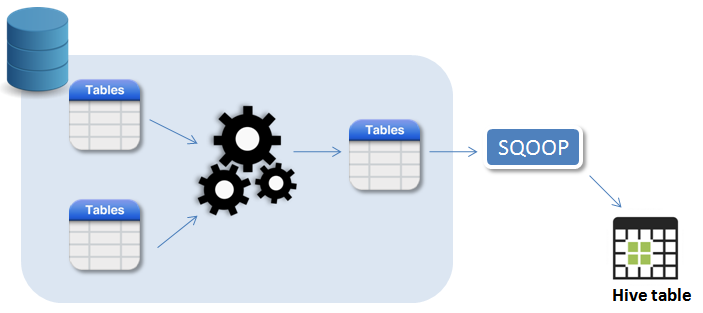
**Limitation:**

* If the query must scan the entire table (as is often the case for analytical queries, group by), HBase is quite –inefficient

1. **Hybrid ( Source ETL based ) or “truncate and Load”**

The Hybrid Update approach uses mature ETL and SQL programming to maintain dimensional data within an RDBMS then periodically copies the data, likely using Sqoop, into HDFS replacing the previous version.

All of the hard work to handle special update rules, slowly changing dimensions (i.e. versioning), surrogate key assignment, etc. can be maintained using ETL (Ab initio, SSIS etc )



**Limitations**

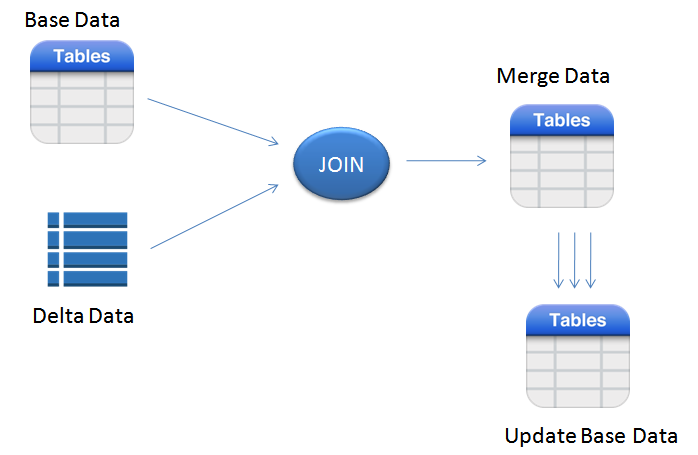
* 1. Approach is not scalable. Suppose you have a billion record dimension table. With this approach you’d extract a billion records from your RDBMS each time you need to update the dimension data in Hadoop
  2. Approach requires orchestration between processes running in different environments.

1. **Merge and Compact**

Merge and Compact approach is pure ETL strategy which can be implemented by various hadoop tools like Spark SQL, pig or hive etc. below are the steps to be followed :

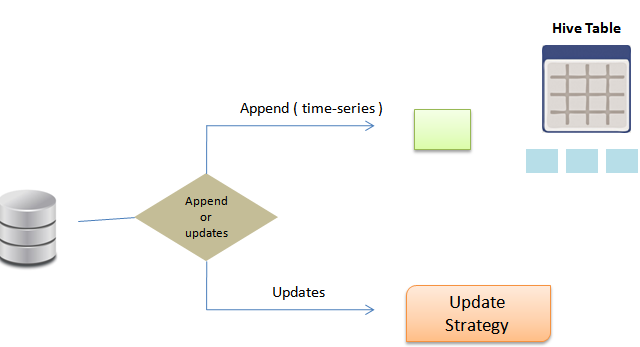
* 1. Maintain a Current Copy of the Master Data
  2. Load the Delta Data
  3. Merge the data: Join the master and delta data together on the business key field(s).
  4. Compact the data
  5. Write the Data to a Temporary Output:
  6. Overwrite the Original Master Data

The primary downside of this approach is that you must read and process the entire dimension table every time you perform an update -- even if only one record is change



1. **Partition based M&R**

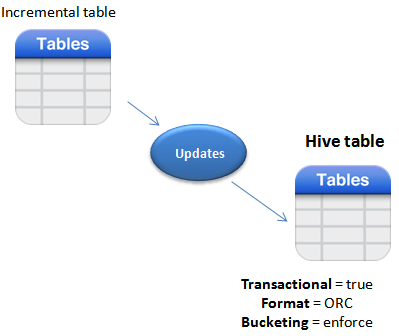
This approach is extension to the merge and compact approach explained above. This requires hive table to be partitioned in such a way that so that you isolate records which are most likely to be updated into a small subset of the partitions. This is often done using a record creation or last update date and assuming that recently inserted records are more likely to be updated than older records.



With this partitioning strategy in place, you simply perform the Merge and Compact Update on the subset of data considered most likely to change. You avoid processing the entire dataset with the risk of inserting redundant dimension records.

The redundant dimension records are created when an updated delta record cannot find its previous version in the partition subset.

We can apply Full fledged Merge and Compact ( point 3), in the weekends and partitioned based during weekdays when we can shorter window for updates to be made.

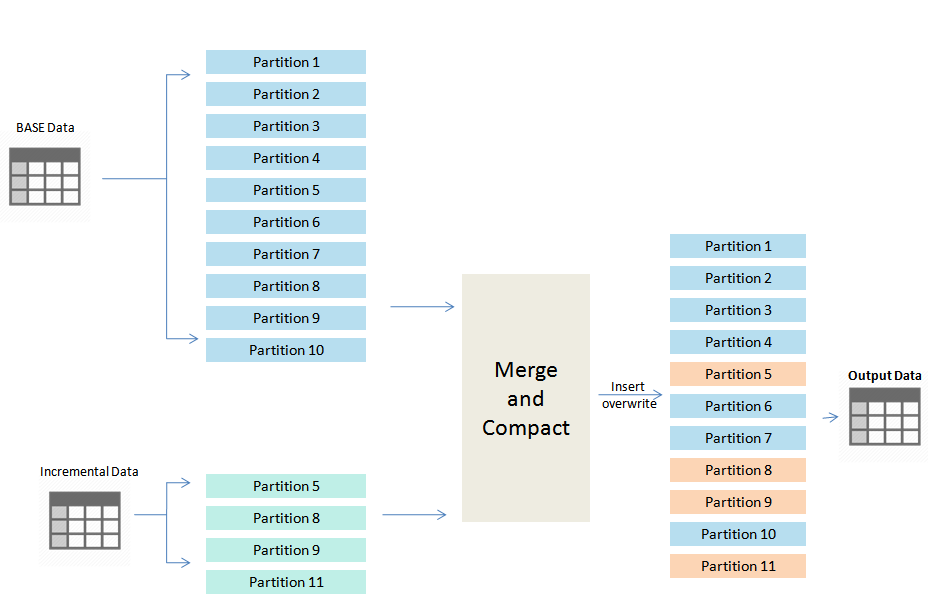


1. **Update and Merge Relevant Data**

This approach is extension to the merge and compact approach explained above and fits for the cases when hive table is enormous and records from any of the partitions can be updated ( no specific window to isolate the most frequently changing data )

Below diagram depicts the scenario where base hive table is partitioned on the specific keys.

Any incremental extracts will be maintained in the separate stage table partitioned in the same keys as base table. While processing we will try to merge the only relevant partition from the base table and created merged dataset and then we will insert overwrite the base table data to reflect the most recent changes in the source.

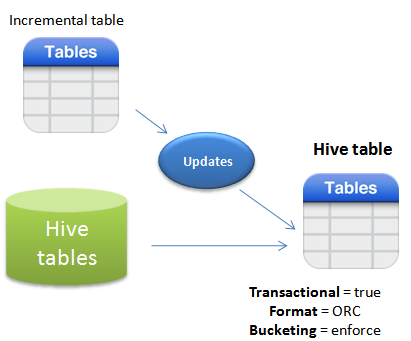
****

1. **Hive Transaction feature :**

Hive provides limited transactional support for CRUD operations after version 0.14, there are limitations on the table structure and approach is not true production ready.

For supporting Transaction feature table should be defined as below :

* Define tbl\_properties ( transactional = true )
* Storage format should be ORC
* Table should be bucketed

****