Use Case

While working with providing solutions for various applications, we see that there is at times, a need to provide multi version concurrency support for certain datasets. The requirement of multi versioned concurrency is mainly due to two reasons –

- Simultaneous querying and loading from tables or datasets, which requires maintaining versions for reading and writing (Locking is not the right option here)
- Maintaining historical load of tables/datasets upto some extent

Both of these requirements are seen in data management systems (warehouses etc).

What happens without MVCC in Hive?

In cases, where MVCC had to be done, design similar to this - https://dzone.com/articles/zookeeper-a-real-world-example-of-how-to-use-it was followed to make it work. Zookeeper was used to maintain versions and provide MVCC support. However, this design poses a limitation if a normal user would like to query a hive table because he will not be aware of the current version to be queried. The additional layer to match versions in zookeeper with the dataset to be queried introduces a bit of an overhead for normal users and hence, the request to make this feature available in Hive.

Hive Design for Support of MVCC

The hive design for MVCC support can be as described below (It would somewhat follow the article mentioned in the previous section) –

1. The first thing should be the ability for the user to specify that this is a MVCC table. So, a DDL something like this –

   ```
   create table <table_name>  ( <column_specs>) MULTI_VERSIONED ON
   [sequence, time]
   ```

   Internally this **DDL** can be translated to a partitioned table either on a sequence number (auto-generated by Hive) or a timestamp. The metastore would keep this information.
2. **DMLs related to inserting or loading data** to the table would remain the same for an end user. However, internally Hive would automatically detect that a table is a multi-versioned table and write the new data to a new partition with a new version of the dataset. The Hive Metastore would also be updated with the current version.

3. **DMLs related to querying data** from the table would remain the same for a user. However, internally Hive would use the latest version for queries. Latest version is always stored in the metastore.

As shown in the article mentioned above, this would require incrementing version count whenever a version is queried and decrement it once the query is done. The management of obsolete versions can be done by –

1. Either a setting which simply says delete the version which is older than a threshold and is not active, OR
2. By tracking the count of queries running on older versions and deleting the ones which are not the latest and are not being used by any query. This would require some sort of a background thread monitoring the table for obsolete versions.