Versioning Data – Second Proposed Design

Single System Design

Versioning

In this proposed system, there will be no compaction (LSM-Tree compaction)

Compaction: throwing away older timestamped duplicate keys and deletion marker keys in the log segment, and keeping only the most recent update for each key. Merging them in one segment and discarding older ones in the process.

- We are not throwing away duplicate keys in the log, we are versioning them.
- Keys that have a deletion marker will not be discarded as well, it will just spell the end of that object's growth, but it will be kept in the database.
- Each key (data object) will have additional metadata; such as version number and predecessor.
  - Version number will be used to track and indicate the state that data is in after changes (updates, deletion).
  - Predecessor/Parent will be used to indicate which branch of the data object's version that specific version was derived from (the parent version).

Proposed Data Object

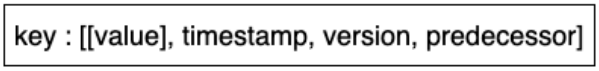

Figure 1: Data object design

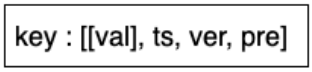

Figure 2: Shorthand of data object design

- Each key will have a value, timestamp, version, and predecessor/parent.
  - The value might be a collection containing multiple values, as seen below:

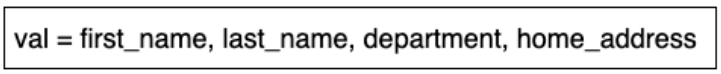

Figure 3: Example of value collection for a key

Examples

New object

- Assign latest/unused key
- Assign version number 1 to key; since it's the first entry for the key
- Accept and assign input to key: [[value], timestamp, 0/null (since it's the first version, thus no predecessor/parent)]

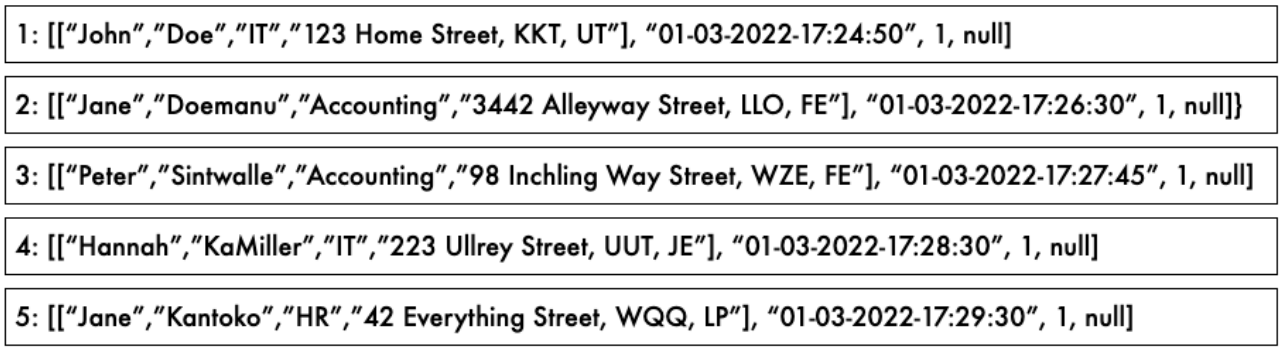

Figure 4: New key inserted

Update object

- Search for key first and display all versions of object once found
- User should indicate which version to view/focus on
- Then the user can choose to edit that value, thus creating a new branch from that specific version

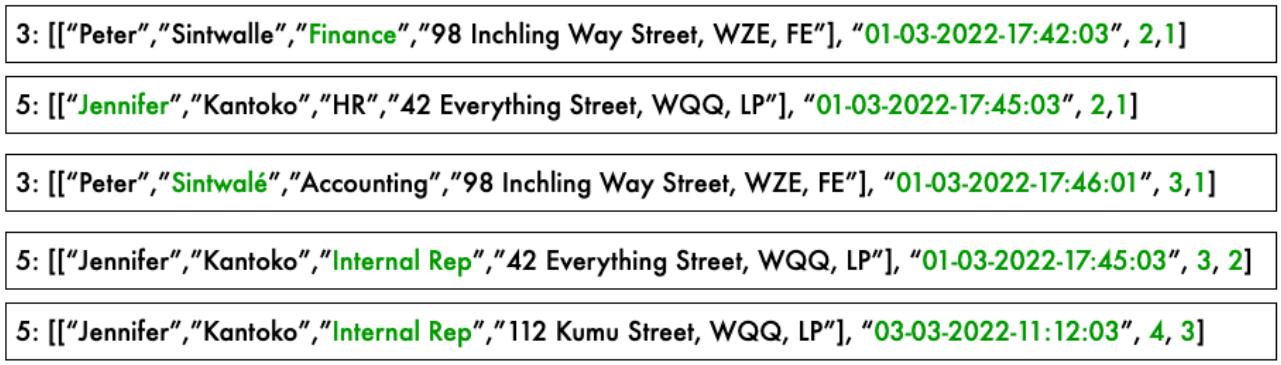

Figure 5: Update existing objects – green highlights indicate changes to objects

Delete/Discontinue/Terminate objects

- Search for key first and display all versions of object once found
- Then the user can choose to terminate the entire object, thus halting the growth of that object
- No new branches/versions can be derived from this object
- The user can still view the object and all its versions/branches, but cannot update

1: [[DELETE (X)], "02-03-2022-12:44:50", 2, 1(?)]

5: [[DELETE (X)], "04-03-2022-09:02:03", 5, 4(?)]

Figure 6: Discontinue objects – green highlights indicate changes to objects, red indicate deletion marker

Merging/Reconciliation

- In essence, the design will allow for duplication of keys, this will ensure that each key (data object) has a history of all the changes that took place. The best way to distinguish the duplicate keys will be the metadata (timestamp, version) in addition to the value.
- Since this design will not employ the compaction process – merging the segments and discarding older keys and deletion marker keys – the duplicate keys will be scattered all over multiple segments, increasing the difficulty for range scans.
- Thus, the design will undergo a merging/reconciliation process to sort the keys. This will reclaim some storage capacity. This process will use a Tree Data Structure to sort and store keys from old segments into new segments in a two-way sorting sequence:
  - first by key
  - then by timestamp
- A sorting mechanism similar to memtable's Tree DS to sort incoming writes
- NOTE: Possibly perform the merge/reconciliation in-memory if possible.

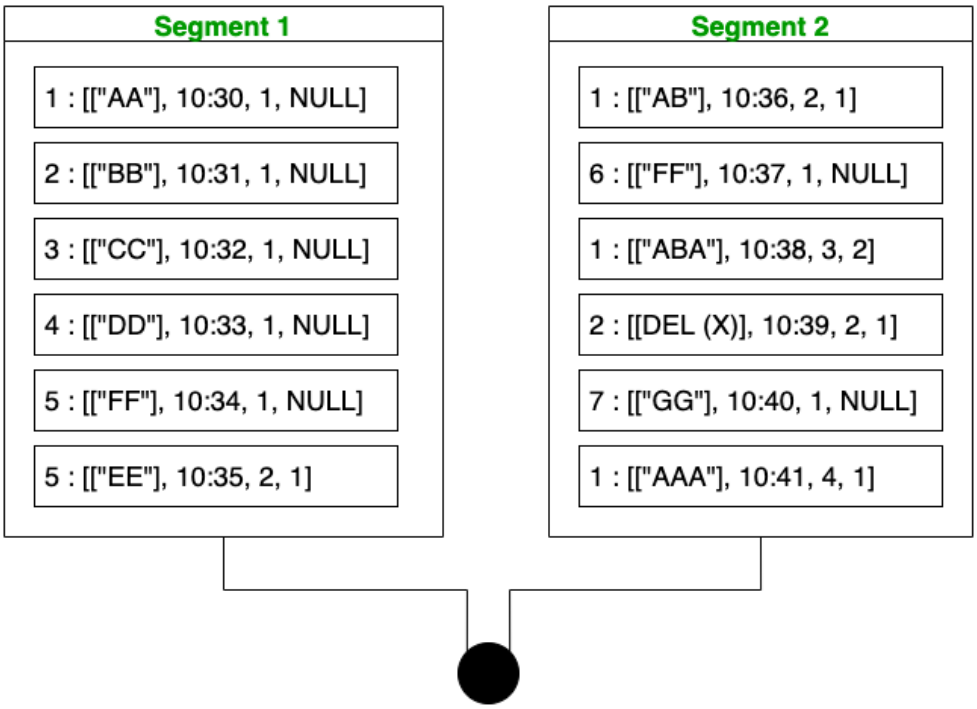

Figure 7: Merging/Reconciliation process

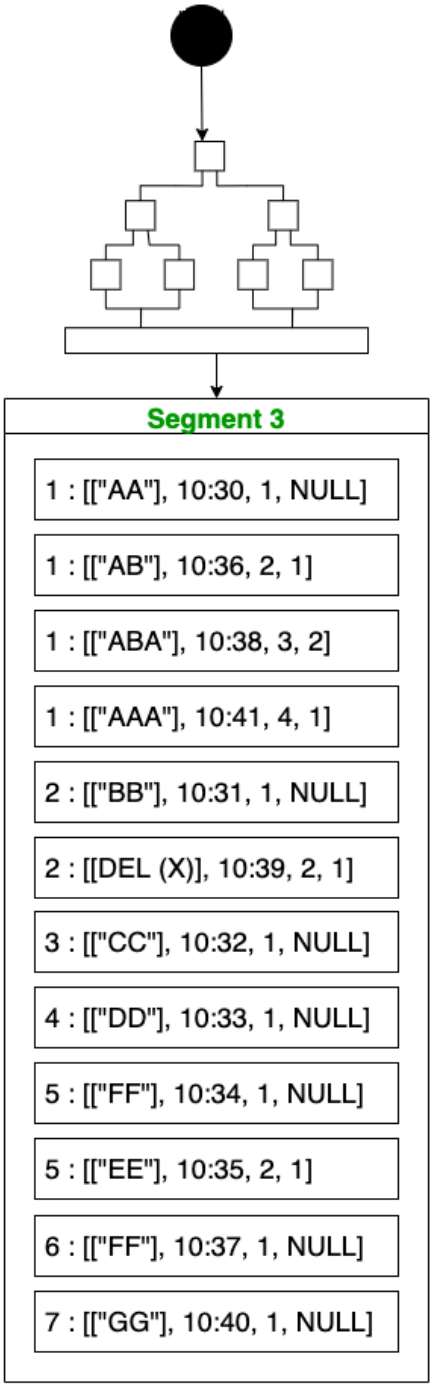

Figure 8: A Tree DS will be used to perform two-way sort sequence; by key, then timestamp

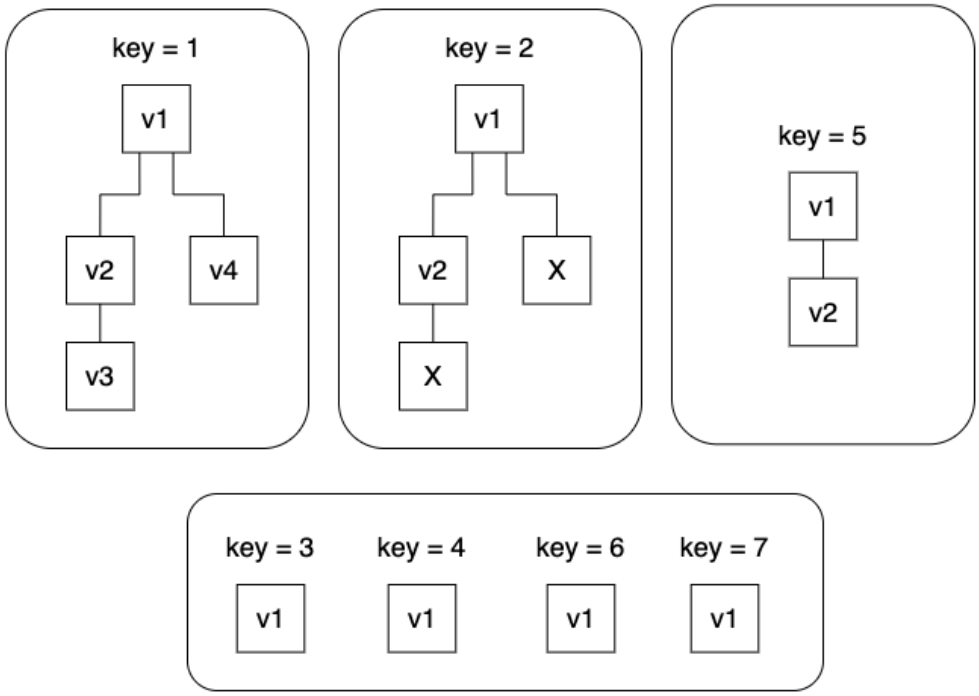

Figure 9: Data object acyclic graph