Example Memcache
ID = Now
(row,type,fam,col,ts,value)

(RowA, Put, Fam, ColA, 23, A)
(RowA, Put, Fam, ColA, 22, B)
(RowA, Put, Fam, ColB, 24, A)
*(RowA, DeleteFamily, Fam, *, 9, *)*

Examples

GetColumns(RowA, Fam, ColA)
- Read first key of Memcache
- compareTo returns 0
- Read next key of Memcache
- compareTo returns 4

Delete(RowA, Fam, ColA, ts=22)
- Memcache.containsKey(above)
- Memcache.remove(above)
- This is a single version that was in Memcache, so we do not need to add it to the deletes

GetColumns(RowA, Fam, [ColA, ColB], ts=(20,0)
- Read first key of Memcache
- compareTo returns 3 (newest stamp we want is older than oldest stamp in current SF/MC)
- Seek/scan of deletes in the Memcache
- DeleteFamily(Fam,9) is added to DeleteSet
- Read first key of StoreFile1
- compareTo returns 0, key added to ResultSet
- Read second, third, forth keys of StoreFile1
- compareTo returns 0 in each case
- Read fifth key of StoreFile1
- compareTo returns 1
- Seek down to seventh key of StoreFile1
- DeleteFamily(Fam,8) gets merged with existing DeleteFamily(Fam,9) so no change
- Delete(Fam,ColA,10) is added to DeleteSet
- DeleteCol(Fam,ColA,9) is merged with DF(Fam,9)
- Delete(Fam,ColB,5) is next column, next StoreFile
- Read first key of StoreFile2
- compareTo returns -1 (DeleteSet returns -1)
- Delete(Fam,ColA,10) is removed from DeleteSet
- Read second key of StoreFile2
- compareTo returns 4 (DeleteSet returns 1)

It's possible DeleteSet could return 1 at the first key, but it seems the logic would be a little too complex inside of DeleteSet to support that kind of look-ahead

Example StoreFile1
ID = 21
(row,type,fam,col,ts,value)

(RowA, Put, Fam, ColA, 20, A)
(RowA, Put, Fam, ColA, 19, B)
(RowA, Put, Fam, ColA, 18, C)
(RowA, Put, Fam, ColA, 17, D)
(RowA, Put, Fam, ColB, 20, A)
(RowA, Put, Fam, ColB, 18, B)
*(RowA, DeleteFamily, Fam, *, 8, *)*
(RowA, Delete, Fam, ColA, 10, X)
(RowA, DeleteCol, Fam, ColA, 9, X)
(RowA, Delete, Fam, ColB, 5, X)
(RowB, Put, Fam, ColA, 10, X)
(RowB, Put, Fam, ColA, 9, Y)

Example StoreFile2
ID = 11
(row,type,fam,col,ts,value)

(RowA, Put, Fam, ColA, 10, W)
(RowA, Put, Fam, ColA, 9, X)
(RowA, Put, Fam, ColA, 8, Y)
(RowA, Put, Fam, ColA, 7, Z)
(RowA, Put, Fam, ColB, 10, X)
(RowA, Put, Fam, ColB, 8, Y)
*(RowA, DeleteFamily, Fam, *, 3, *)*
*(RowA, Delete, Fam, ColA, 5, *)*
*(RowA, DeleteCol, Fam, ColA, 4, *)*
*(RowA, Delete, Fam, ColB, 5, *)*
(RowB, Put, Fam, ColA, 10, X)
(RowB, Put, Fam, ColA, 9, Y)

Column Border

Put/Del Border

Row Border

Delete Notes

Deletes are immediately applied directly to what is stored in the Memcache.

Deletes (always scoped to a row and family) are stored at the end of the row in StoreFiles.

The deletes in a particular StoreFile will never be relevant for the puts in that same StoreFile, because of above properties.

We read them in after we've scanned everything in this StoreFile, for this row, and if we must go to the next StoreFile, then we read in the deletes and add them to our DeleteSet.

When adding to the DeleteSet, we do an intelligent merge of Deletes if any are overlapped.

Note: in reality, the oldest StoreFile in a Store will NEVER have deletes in it, so StoreFile2 is not accurate in that respect (we are considering it as the last StoreFile of the Store in examples)

Another Example

GetFamily(RowA, Fam, num=1)
- Instantiate CountMap<Column,Count>
- Read first key of Memcache
- compareTo returns 0, key added to ResultSet CountMap.increment(ColA,1)
- Read second key of Memcache
- compareTo returns -1 (CountMap.get(ColA) = num)
- Read third key of Memcache
- compareTo returns 0, key added to ResultSet CountMap.increment(ColB, 1)
- Read fourth key of Memcache
- compareTo returns 1
- Add DeleteFamily(Fam,9) to DeleteSet
- Read first 6 keys of StoreFile1
- compareTo returns -1 in all cases
- Read seventh key of StoreFile1
- compareTo returns 1
- Merge deletes into DeleteSet as before
- Read first 6 keys of StoreFile2
- compareTo returns -1 in all cases
- Read seventh key of StoreFile1
- compareTo returns 4
  We know we are in the last StoreFile and have reached the last columns we're interested in

Another Example

GetTop(RowA, Fam, num=2)
- Instantiate ColumnSet<Column> and count = 0
- Read first key of Memcache
- compareTo returns 0, key added to ResultSet CountSet.add(ColA), count++
- Read second key of Memcache
- compareTo returns -1 (CountSet.contains(RowA))
- Read third key of Memcache
- compareTo returns 0, key added to ResultSet CountSet.add(ColB), count++
- Read fourth key of Memcache
- compareTo returns 4, count == num