# 05 – Multi-Version Concurrency Control

1. MVCC Deletes
    A. Deleted flag
        i. Maintain flag to indicate logical tuple has been deleted
        ii. Separate column / tuple header
    B. Tombstone tuple
        i. Create an empty version to indicate deletion
        ii. Use separate pool for it

2. Garbage Collection
    A. Reclaimable versions
        i. No active txn can see
        ii. Created by an aborted txn
    B. Traditional garbage collection methods cannot handle HTAP well if there are some long running queries
    C. Old versions issue
        i. Memory usage increase
        ii. Memory allocator overhead
        iii. Slow read because of longer version chains
        iv. Slow garbage collection because of lots of garbage versions
        v. Poor time-based version locality

D. Design Decisions

i. Index clean up
Index should clean up key that is not visible anymore

ii. Version tracking level

1. Tuple level – background vacuuming / cooperative cleaning

2. Txn level – txns keep track of their old versions

3. Epochs – multiple txns are managed in epochs

iii. Frequency
tradeoffs : more frequent GC
-> efficient memory management, slow down txns.

1. Periodically
run GC at some intervals (fixed vs. adjust on load)

2. Continuously
run GC with txn execution

iv. Granularity
tradeoffs : more fine granularity
-> more computational overhead for find garbage

1. Single version
high overhead, find-grained control

2. Group version
less overhead, delay reclamation

3. Tables (If no txns access table, then run GC)
special case

v. Comparison unit

1. Timestamp – use global minimum timestamp

2. Interval – use timestamp range that are not visible

3. Block Compaction (need to re-study)
    A. Deleted tuples
        i. Reuse slots – destroy temporal locality
        ii. Leave slot unoccupied – need some method to fill holes
    B. Block compaction
        i. Time since last update
        ii. Time since last access
        iii. Application level semantics