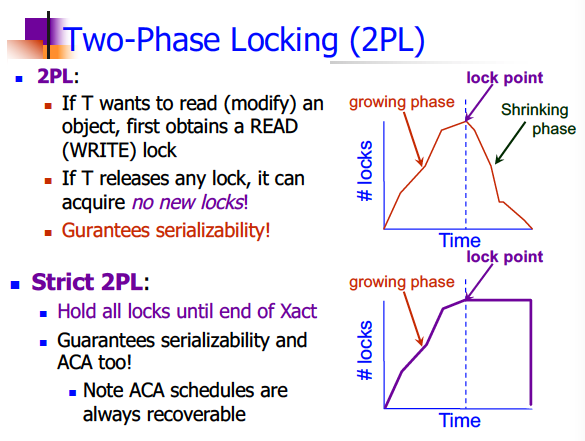
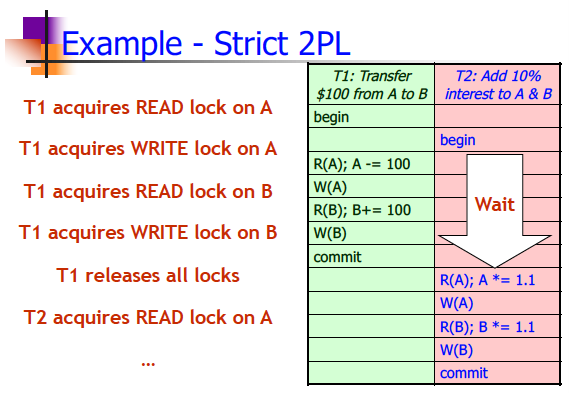
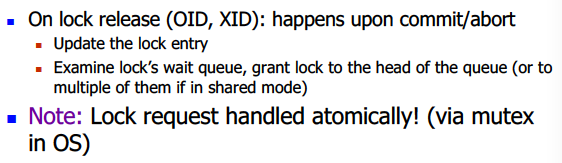
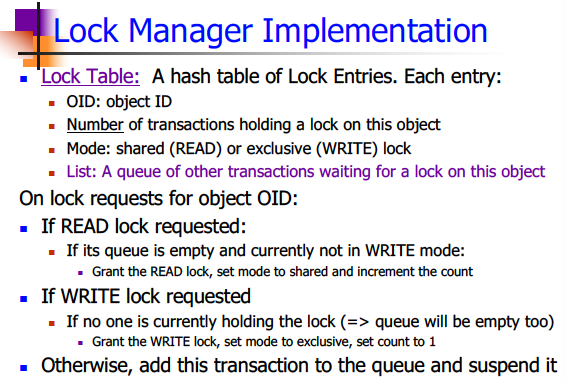
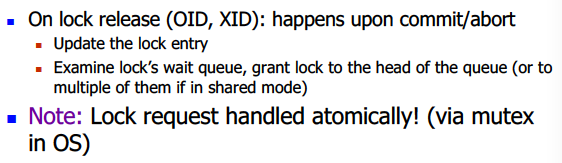
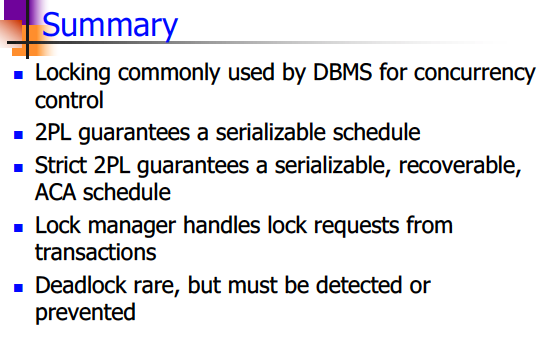
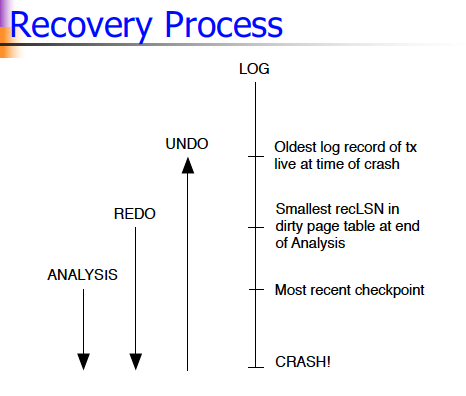
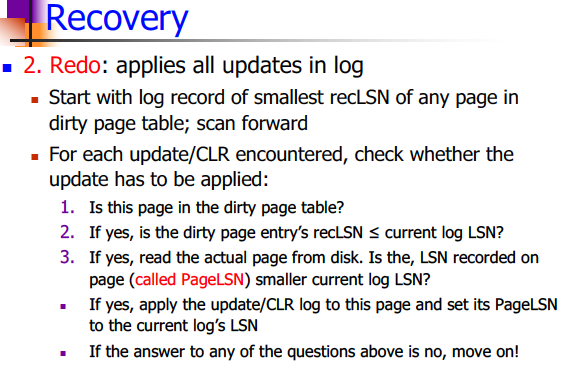
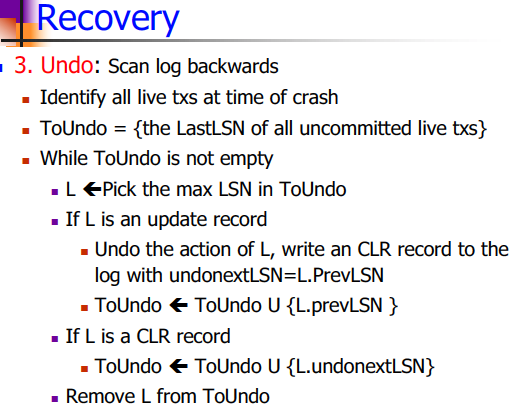
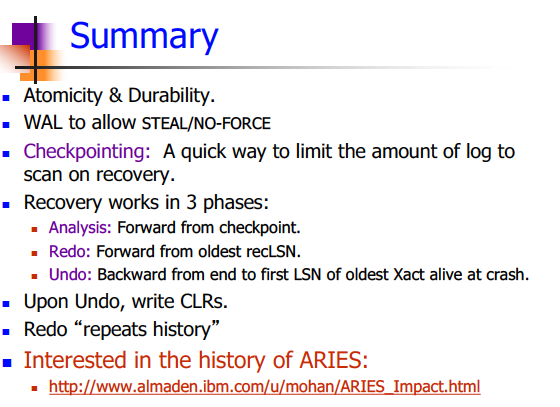
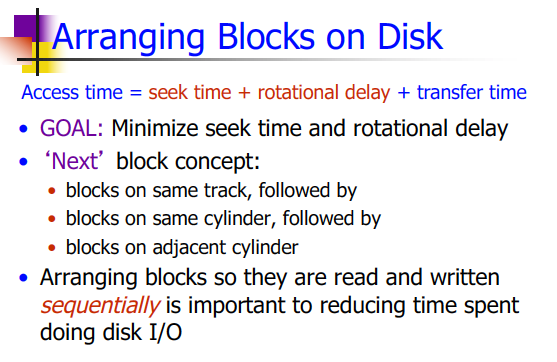
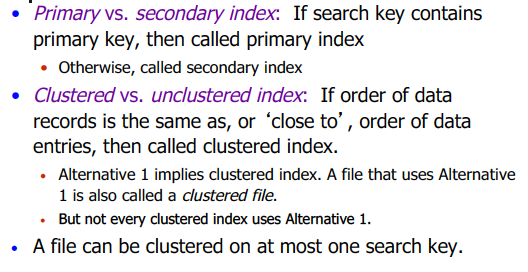
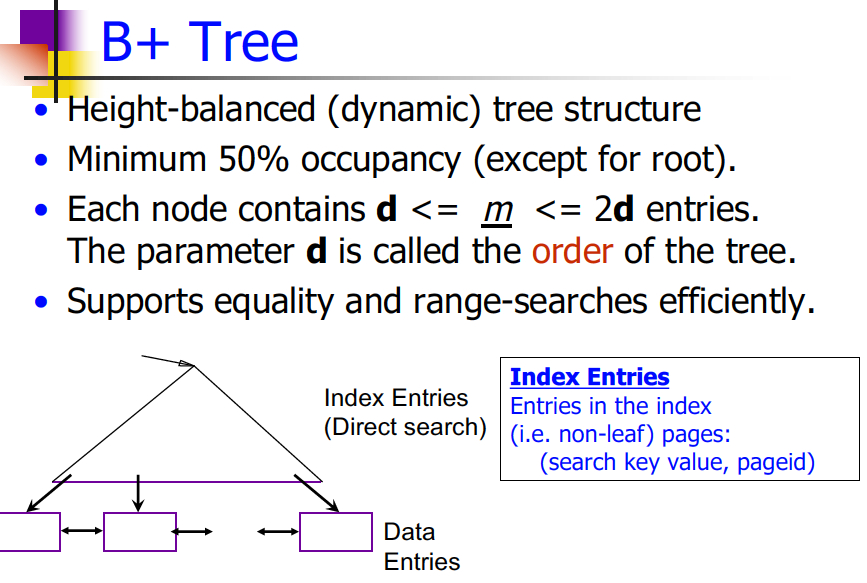
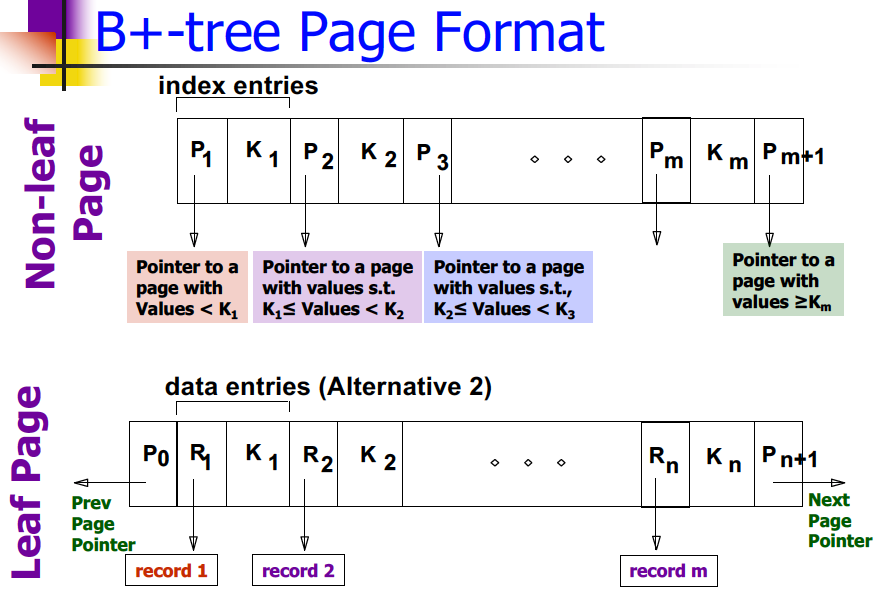
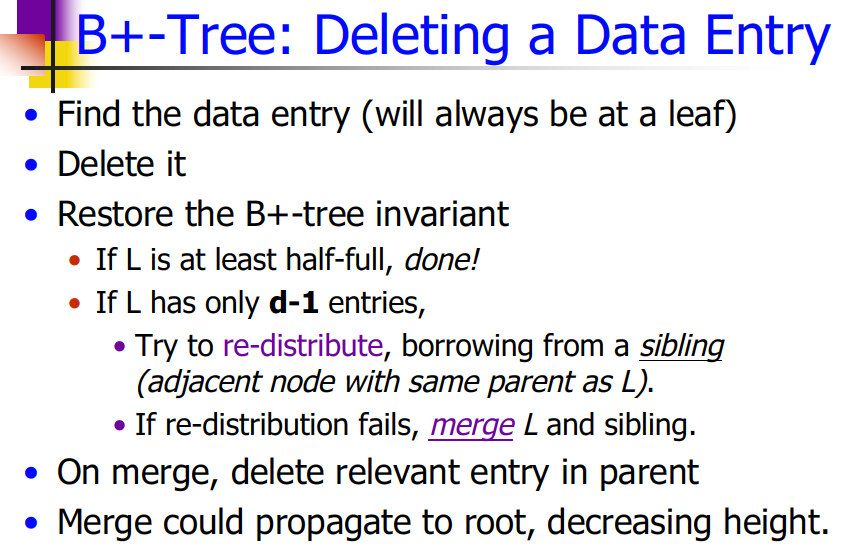
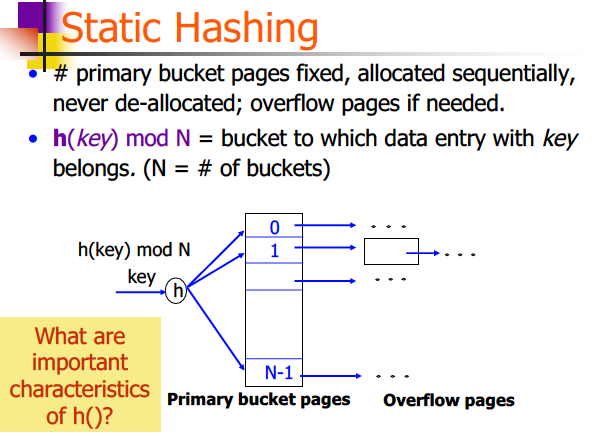
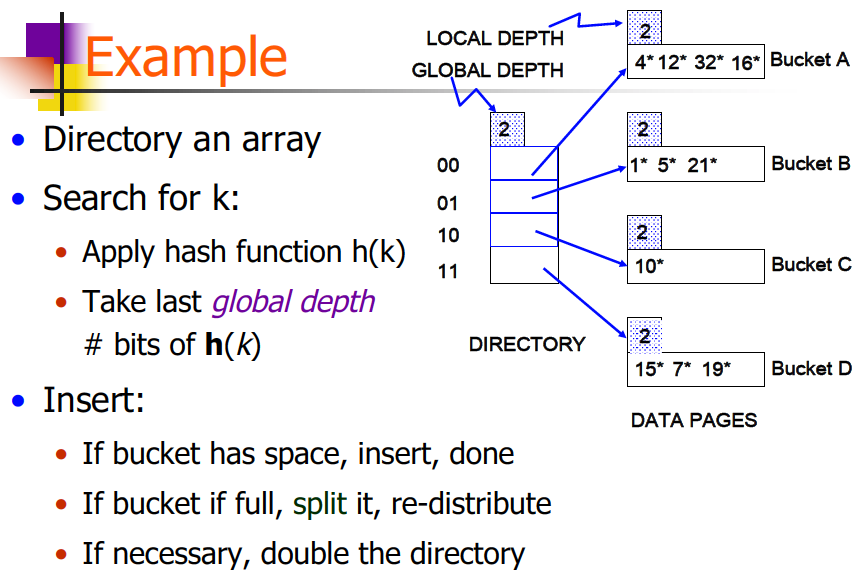
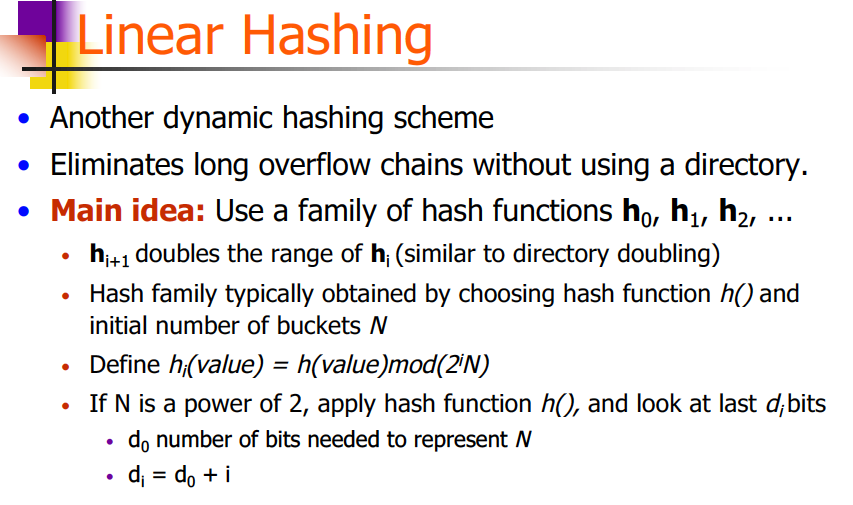
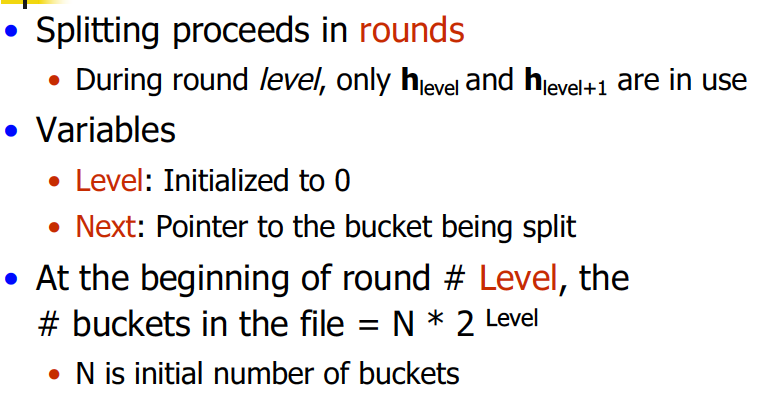
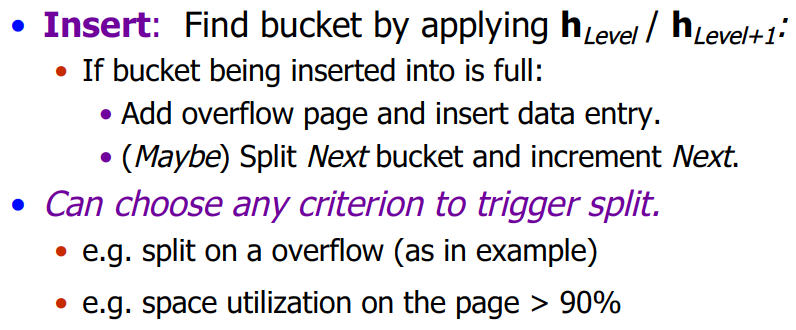
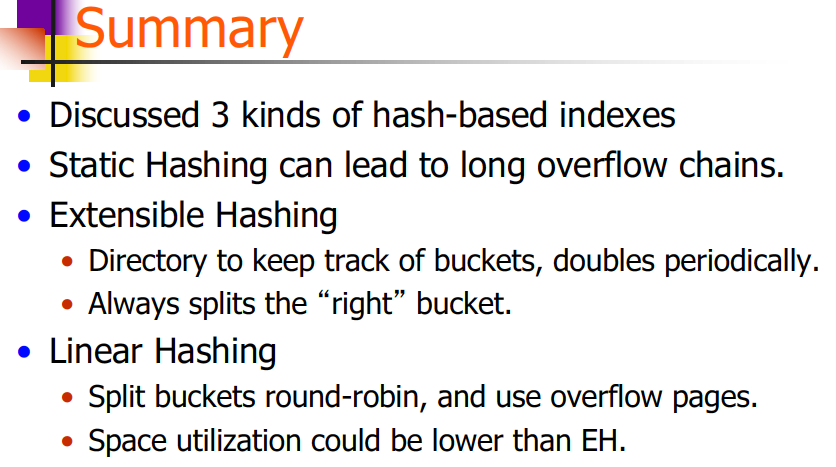
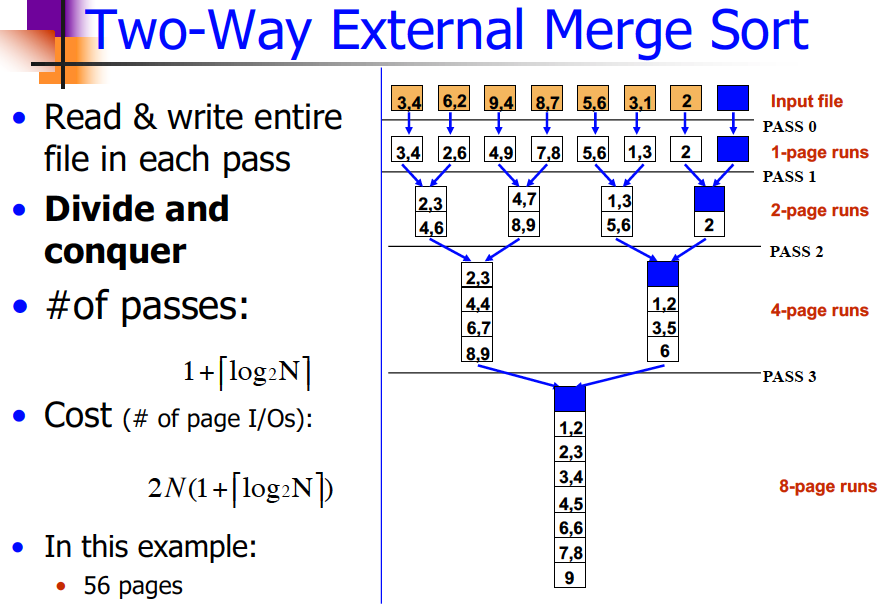
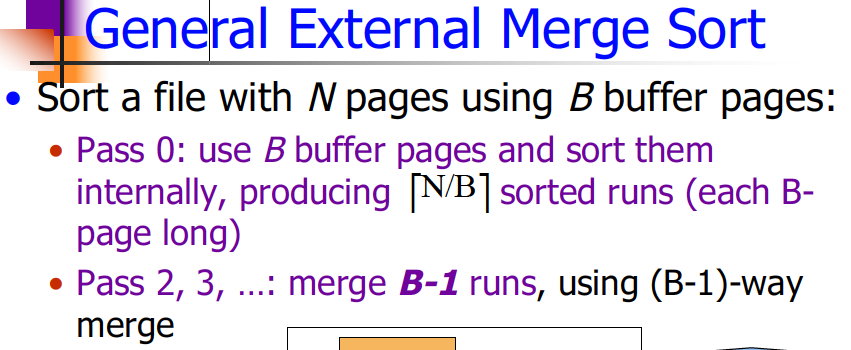
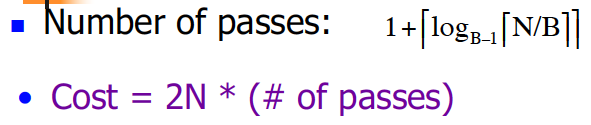
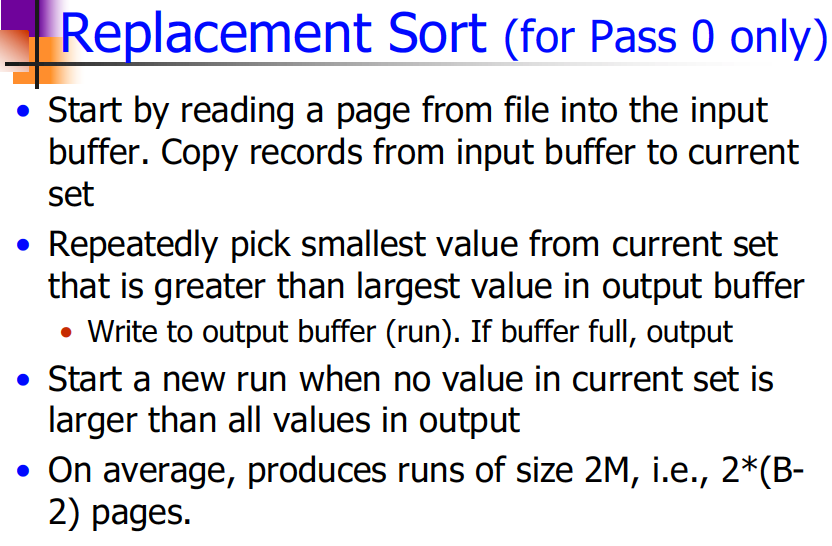
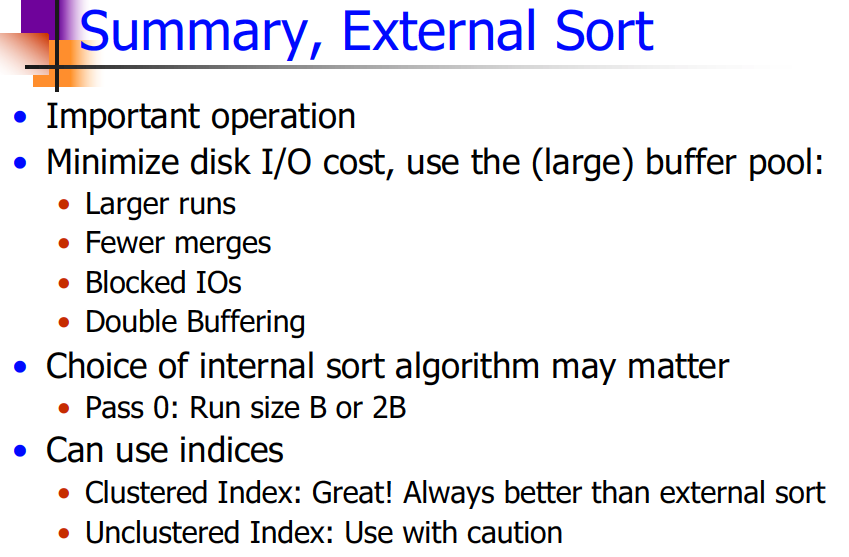
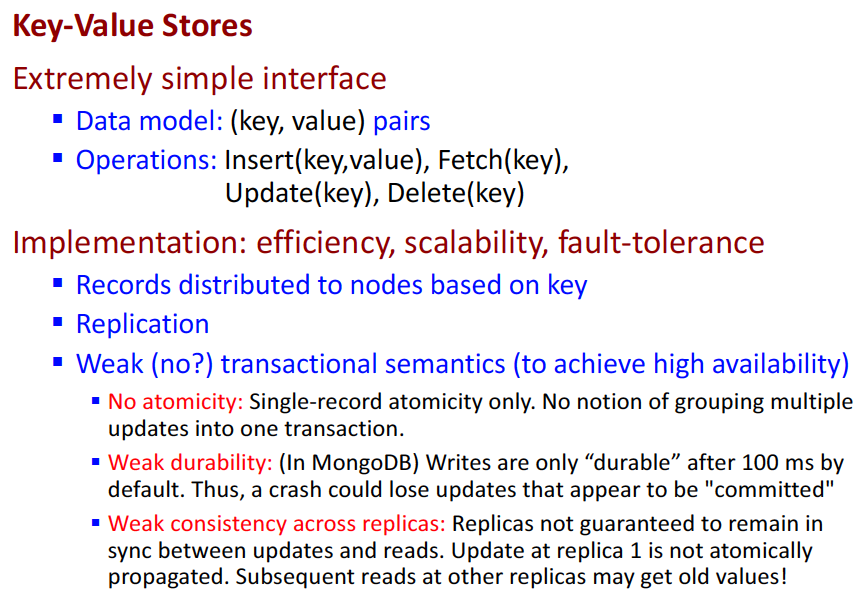
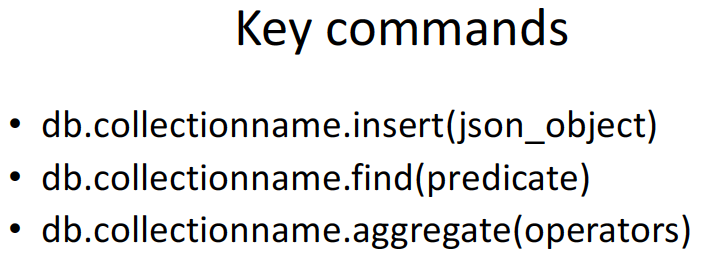
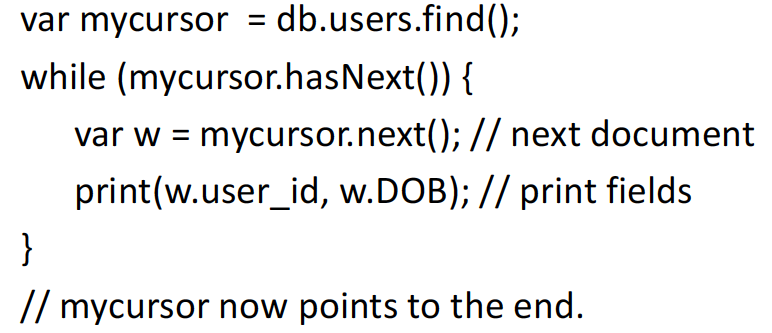
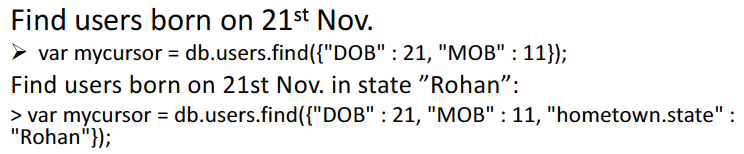
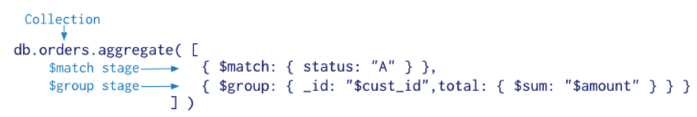
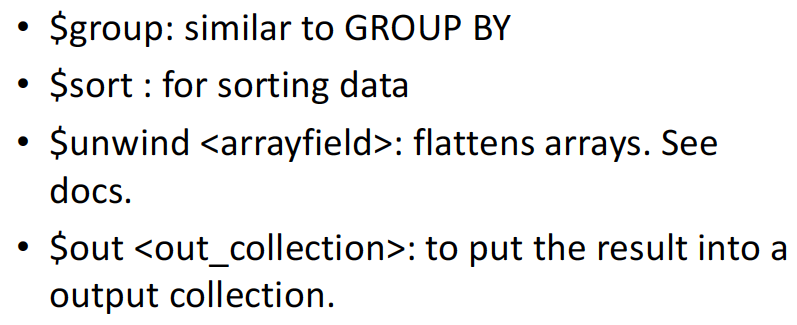
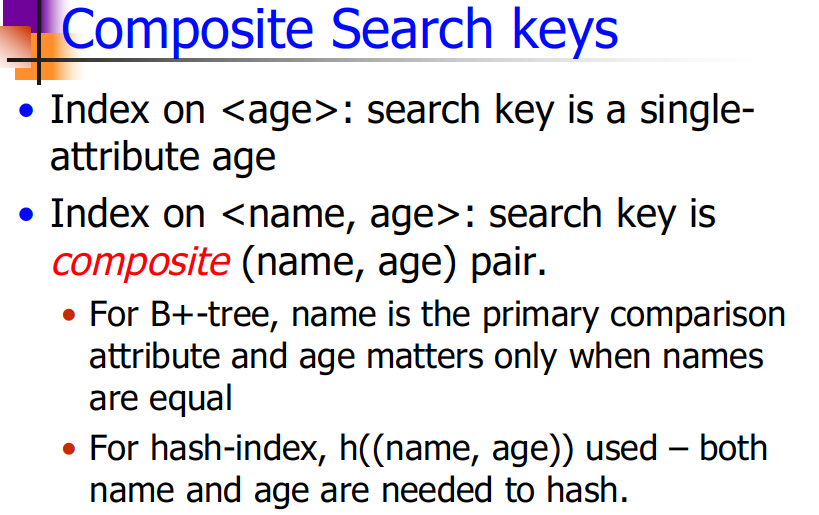
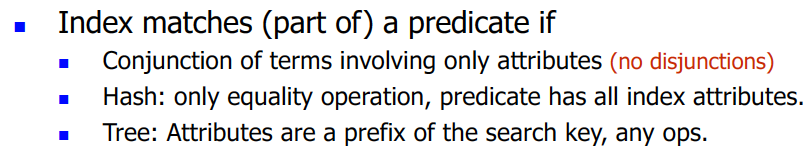
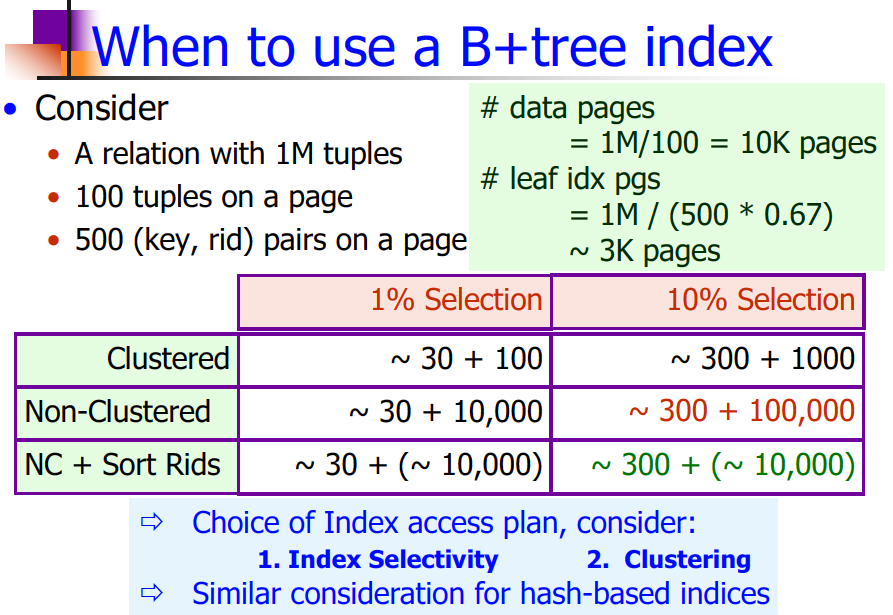
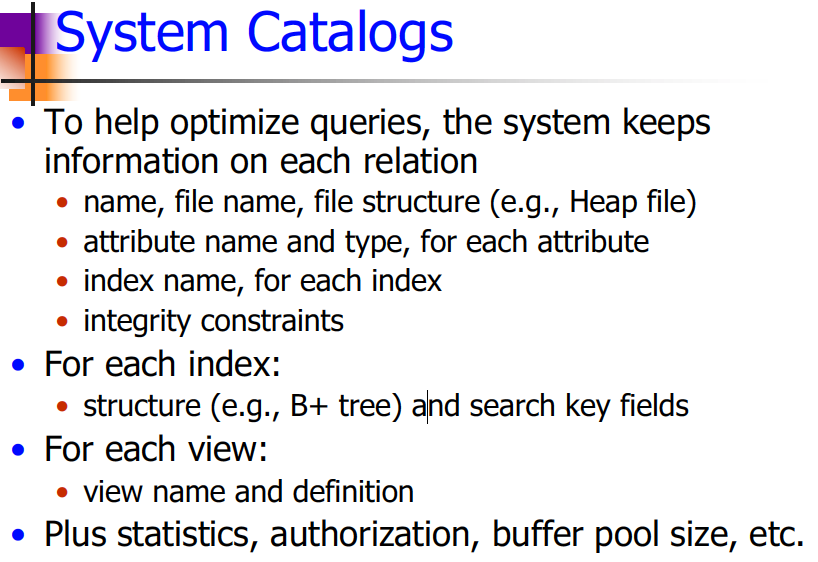
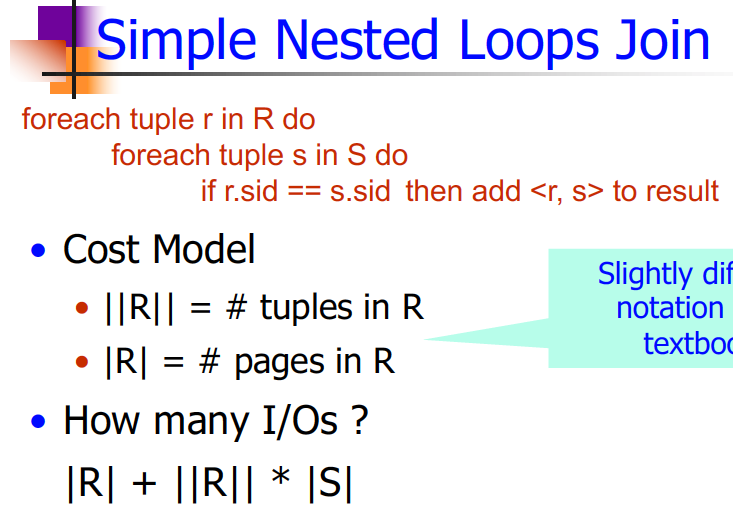
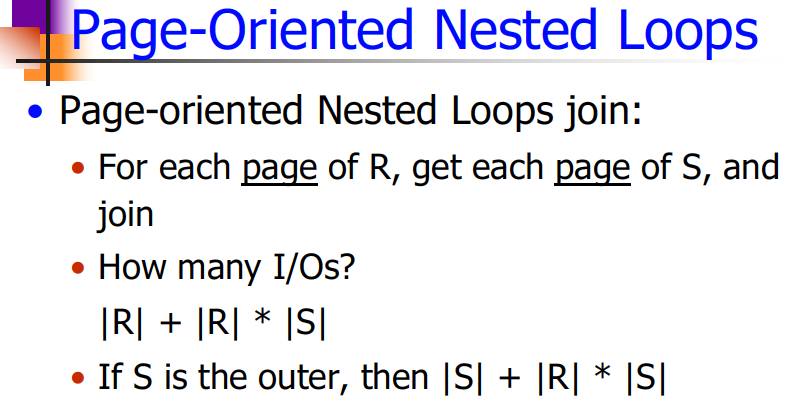
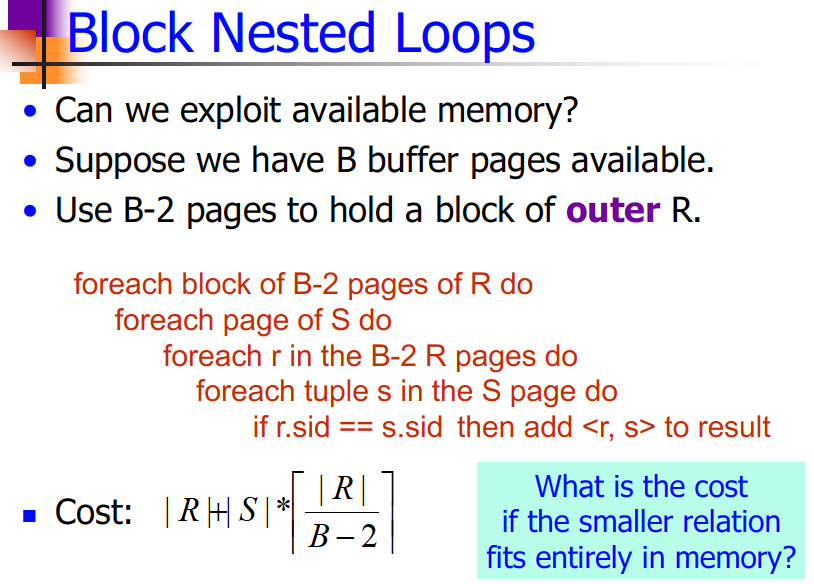
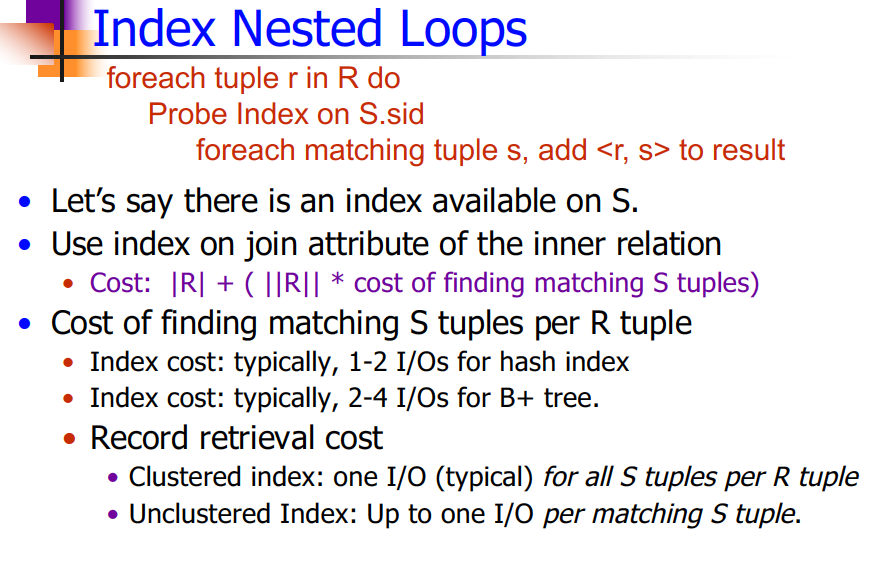
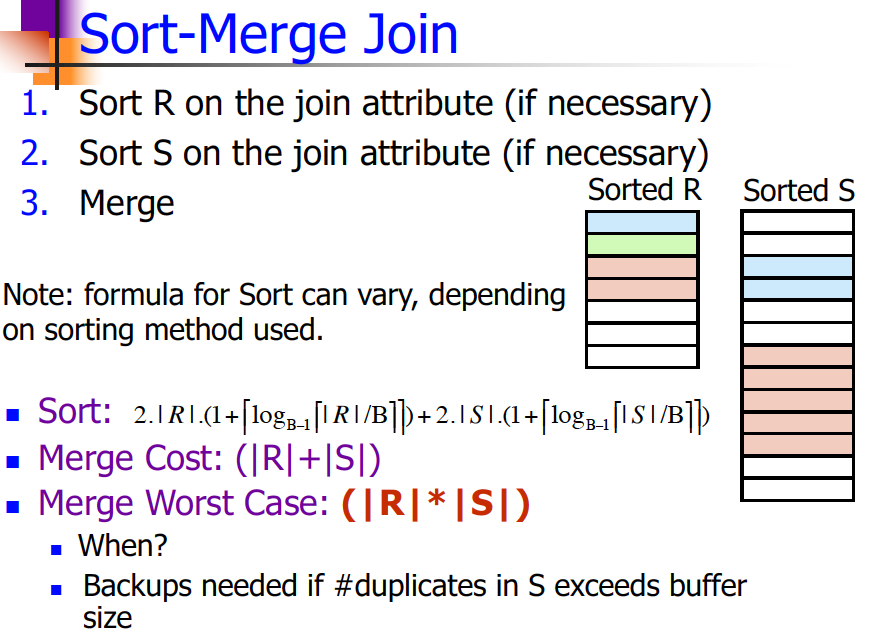
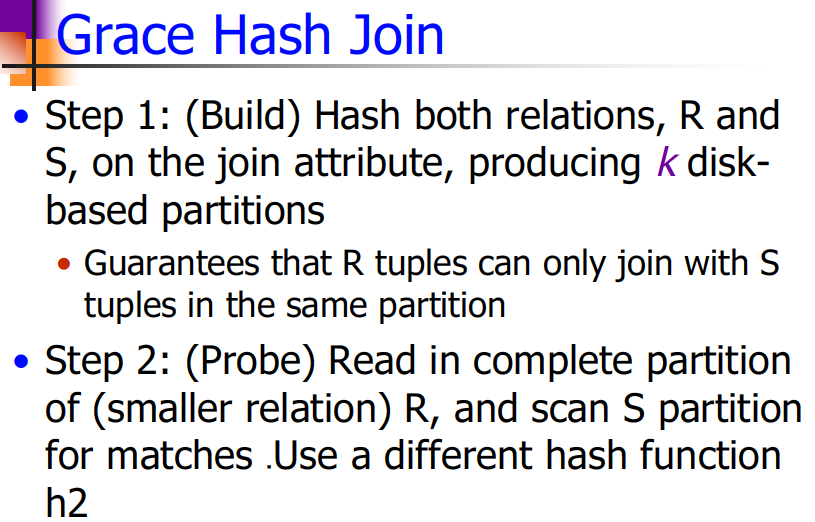
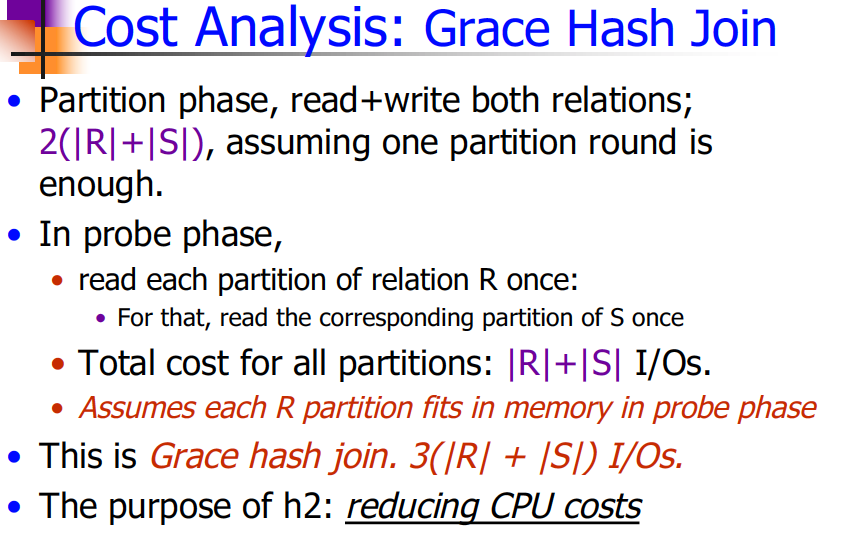
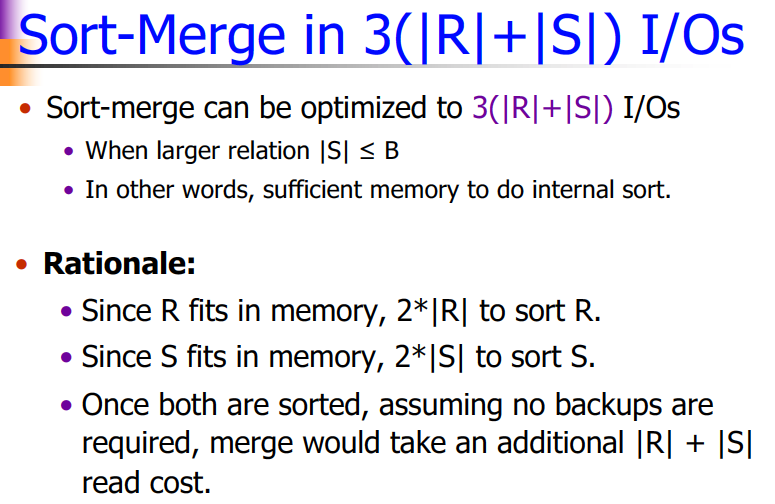
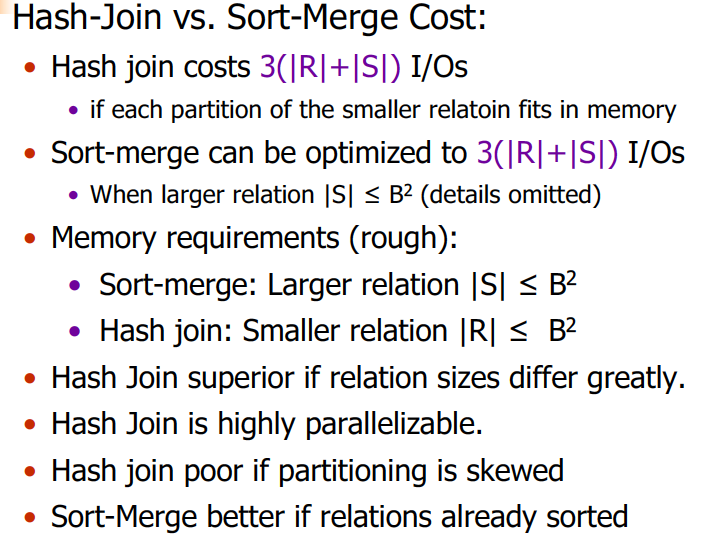
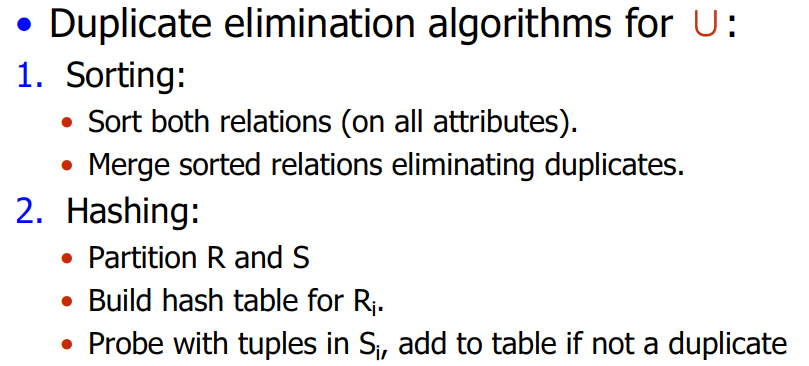
1. Lecture 10: Transaction Management
   1. ACID Properties
      1. Atomicity
         1. All or nothing property (transactions don’t execute partially)
      2. Consistency
         1. Assume that each submitted transaction, if executed, satisfies any integrity constraints
      3. Isolation
         1. Each transaction appears to execute without concurrent transactions (serially)
      4. Durability
         1. If a transaction commits, its effects persist
   2. Schedules
      1. Interleaving actions from a set or transactions
   3. Serial Schedule
      1. No interleaving actions among transactions
   4. Serializable Schedule
      1. Equivalent to a serial schedule. Can be transformed into a serial schedule by re-ordering the non-serial actions.
      2. Essential!!
      3. Different serial schedules can have different final states
   5. Conflicts
      1. Write-read (WR) conflict
         1. “Dirty Read”
         2. Read a page after it has been written to
      2. Read-write (RW) conflict
         1. “Unrepeatable read”
         2. T1 reads the value of object A, then T2 updates A (before T1 has committed)
      3. Write-write (WW) conflict
         1. Overwriting uncommitted data
         2. T2 overwrites a page that T1 wrote to before T1 was committed
   6. Aborts
      1. Makes the transaction “disappear”
   7. Cascading Abort
   8. Recoverable Schedule
      1. Transactions commit only after all transactions whose changes they have read commit.
      2. Avoid cascading aborts (ACA)
         1. Transactions read only the changes of committed transactions
   9. Locking
      1. 
      2. 
   10. Precedence (Serializability) Graph
       1. A node for each committed transaction in L
       2. An arc from T­i to Tj if some action in Ti precedes and conflicts with some action in Tj
   11. Conflict Equivalent
       1. If they involve the same transactions
       2. And order each pair of conflicting transactions in the same way
   12. Conflict Serializable
       1. If it is conflict equivalent to a serial schedule
       2. All CS schedules are also serializable.
       3. **A schedule is conflict serializable if and only if its precedence graph is acyclic**
   13. 2PL
       1. Guarantees an acyclic precedence graph
   14. Strict 2PL
       1. Guarantees ACA (read only committed values) by holding write locks until the end of a transaction
   15. Lock Manager Implementation
       1. 
   16. Deadlocks
       1. A cycle in the “Waits-for” Graph
       2. Prevention
          1. Assign Priorities to transactions
          2. 
   17. Non-Locking CC protocols
       1. Optimistic CC
       2. Multiversion CC
   18. 
2. Lecture 11: Database Recovery
   1. Insert
      1. Read the page from disk
      2. Update the page in memory to add a new record
      3. Write the page back to disk
   2. ARIES
      1. No Force and steal
      2. Every update written to two places
         1. Database page
         2. Transaction log
      3. Transaction Log used to:
         1. Undo stolen pages with uncommitted writes
         2. Redo committed writes
      4. Possible Log Record Types:
         1. Update
         2. Commit
         3. Abort
         4. End (end of commit or abort)
         5. Compensation Log Rec. (CLRs)
            1. For UNDO actions
   3. Write-Ahead Logging (WAL)
      1. Force the log record for an update before the corresponding data page gets to disk.
      2. Must write all log records for a Xact before commit.
      3. #1 guarantees Atomicity
         1. Allows undo of stolen pages
      4. #2 guarantees Durability
         1. Allows redo of unforced but committed pages
   4. Compensating Log Records
      1. Describes updates about to be undone
      2. Add a CLR entry to the log for every write that is about to be undone due to abort
         1. For UNDO actions.
         2. Contains undoNextLSN
            1. Reverse chain of update logs
   5. Checkpoint: (Logical) Snapshot of the database
      1. Minimize recovery time by limiting log we need to examine
      2. After crash, system locates most recent begin\_checkpoint
   6. Transaction table
      1. Points to most recent LSN for each live tx.
      2. Tells us “latest possible undo point” for tx
   7. Dirty page table
      1. Points to first log record that dirtied the page
      2. Tells us “earliest possible redo point” for page
   8. 
   9. 
   10. 
   11. 
   12. 
3. Lecture 12: Storage and Indexing



1. Indexes
2. A data structure that organizes data records on disk to optimize certain operations
3. Speed up selections on the search key field of the index (denoted k)
4. 
5. B+ Tree Indexes
6. Leaf pages contain data entries, and are chained (prev & next)
7. Non-leaf pages contain index entries and direct searches
8. Hash Based Indexes
9. Index is a collection of buckets. Bucket = primary page plus zero or more overflow pages.
10. Lecture 13: B+ Trees
    1. 
    2. 
       1. Fanout: F
       2. Total leaf pages: N
       3. Height =
       4. Typical Capacity = Average F (67%) ^ height
       5. Order (d) > 50%
    3. Splitting overfull non-leaf nodes
    4. Inserting into B+ tree
       1. Re-Distribution with siblings
          1. Improves page occupancy, possibly reduces height
          2. Use only for leaf level entries
    5. 
       1. Try redistribution with all siblings first, then merge.
    6. 
11. Lecture 14: Hash-Based Indexes
    1. 
       1. Number of buckets (N) is fixed ahead of time
       2. Not efficient for many insertions (entire file has to be read and written when rehashing after doubling N)
    2. 
       1. 
    3. 
       1. 
       2. 
    4. 
12. Lecture 15: External Sorting
    1. Two-way external merge sort
       1. Three total buffer pages used
       2. Pass 1: Read a page, sort it, write it (a run)
          1. One buffer page used
       3. Pass 2, 3, …, etc.
          1. Three buffer pages used
       4. 
    2. 
       1. 
    3. 
    4. Clustered B+ Tree Sorting
       1. Go to the left-most leaf, then retrieve all leaf pages
    5. Unclustered B+ Tree sorting
       1. In general one I/O per data record!!
    6. 
13. Lecture 16: MongoDB
    1. Traditional databases use Vertical Scaling (more CPUs, RAM, storage)
    2. NoSQL databases use horizontal scaling or sharding
       1. Decompose data across multiple computers and do distributed queries
    3. 
    4. 
    5. 
    6. 
    7. 
    8. 
    9. 
14. Lecture 17: Evaluation of Relational Operations
    1. Selection (
       1. Heap file: O(N)
       2. Sorted File: O(log2N) + …
       3. Index
          1. Hash: O(1) + …
          2. B+ Tree: Clustered/Unclusterd: O(logFN) + …
       4. 
       5. Index Matching?
          1. 
    2. 
    3. 
    4. Join Strategies
       1. 
       2. 
       3. 
15. Lecture 18: Advanced Join Strategies
    1. 
    2. 
    3. 
    4. 
    5. 
    6. 
16. Lecture 19: Projections
    1. 
17. Lecture 20: Optimization