COSC 404 Midterm

Key Concepts:

- Insert and Delete from B+ tree (4 marks)
  - Insert with linear hashing (2 marks)
  - Perform RAID and index calculations (6 marks)
  - Code an iterator in Java (lab 4) (5 marks)
  - Create relational query plans in Java (5 marks)

1GB = 10^9bytes

Topics:

I) **Storage**
   a. **Memory performance calculations**
      i. Transfer time = memoryTransferSize/bandwidth
      ii. Read time = readDataSize/readBandwidth
      iii. Write bandwidth = IOPS * writeRequests
   b. **Storing records in memory**
      i. A record consists of one or more fields grouped together
         1. Each Tuple of a relation is a record
      ii. Two main types of records:
         1. Variable length: size of the record varies
         2. Fixed length: all records are the same size
   c. **Variable formats**
      i. Useful cases:
         1. the data does not have a regular structure in most cases
         2. the data values are sparse in the records
         3. there are repeating fields in the records
         4. the data evolves quickly so schema evolution is challenging
      ii. Disadvantages:
         1. Space is wasted by repeating schema information for every record
         2. allocating variable-sized records efficiently is challenging
         3. query processing is more difficult and less efficient when the structure of the data varies
      iii. JSON & XML are best described as variable format, variable size
      iv. A VARCHAR field is best described as fixed format, variable size
   d. **Storing records in blocks**
      i. Issues related to storing records in blocks

1. Separation: how are adjacent records separated?
    a. variable length records can be separated by:
        i. a special separator marker in the block
        ii. storing the size of the record at the start of each record
        iii. store the length or offset of each record in the block header
2. **Spanning**: can records cross block boundaries?
    a. unspanned: do not allocate records across blocks and waste space
        i. Blocks needed for unspanned records = 1record/block
        ii. utilization = recordSize/blockSize*100%
    b. spanned: start a record at the end of a block and continue on to the next (round up)
        i. each piece must have a pointer to its other part
        ii. blocks needed (numBlocks) = numRecords*recordSize/blockSize
        iii. utilization = (numRecords*recordSize)/(blockSize*numBlocks)
3. **Clustering**: how many records can a block store?
    a. Allocating records of different types together on the same block/file because they are frequently accessed together
    b. Beneficial when DB commonly gets queries with where clause
    c. Harmful for select * queries because the system reads more blocks and each block read has information that doesn't answer the query
    d.
4. **Splitting**: how many blocks are records allocated in?
    a. Split record: a record where portions of the record are allocated on multiple blocks for reasons other than spanning
        i. Used with or without spanning
    b. Hybrid records
        i. Fixed portions of records are stored with other fixed records on their own block(s)
        ii. Variable parts are stored with other variable record parts on their own block(s)
    c. Efficient & simplifies allocation
    d. Fixed part of record is easier to allocate and optimize for access

5. **Ordering:** are the records sorted in any way?
    a. Records in a block are sorted based on the value of 1+ fields
    b. Allows searching for keys and performing joins to be made faster
    c. Physical ordering: records are allocated in their blocks in sorted order
    d. Logical ordering: records are not physically sorted but each has a pointer to the next record in sorted order
6. **Addressing:** how each record is referenced
    a. Method for defining unique values or addresses to reference particular records
    b. **Physical addressing:** a record has a physical address based on the device where it's stored
        i. Better performance because record can be accessed directly w/o lookup cost
    c. **Logical addressing:** records have a key value or other identifier that can be used to lookup their physical address in a table
        i. Does not provide method for locating record directly
        ii. More flexible because records can be moved on the physical device and only the mapping table needs to be updated
        iii. Easier to move, update, change
7. **Pointer Swizzling**: the process for converting disk pointers to memory pointers and vice versa when blocks move between memory and disk

e. Objectives

i. Compare/contrast volatile versus non-volatile memory.

1. **Volatile memory**: retains data only when power is on
    a. DRAM
    b. Main memory
2. N**on-volatile/permanent memory**: stores data even when power is off
    a. SSD
    b. Flash memory
    c. Hard drive

ii. Compare/contrast random access versus sequential access.

1. Random access: allows retrieval of any data location in any order

2. Sequential access: requires visiting all previous locations in sequential order to retrieve a given location

iii. Use both metric and binary units for memory sizes.

1. 1 byte = 8 bits
    a. 1 bit = 1 or 0
2. 1 byte = 1 character

iv. List the benefits of RAID and common RAID levels.

1. **RAID Benefits**
    a. Improves reliability through redundantly storing extra data that is used to recover from a disk failure
    b. Parallelism allows for increased throughput and large accesses to reduce response time
2. **RAID Levels**
    a. 0: is for high-performance where data loss is not critical (parallelism)
        i. Striping at the block level
        ii. Non-redundant
    b. 1: has redundancy and protection from disk failures with minimum cost
        i. Requires at least 2 disks
        ii. Mirrored disks
    c. 5: offers reliability and increased performance
        i. No single disk bottleneck like RAID 4
        ii. Block interleaved distributed parity
        iii. Partitions data and parity among N+1 disks
    d. 6: offers extra redundancy compared to RAID 5 and is best for mitigating multiple drive failures
        i. P+Q redundancy scheme
        ii. Stores extra info than RAID 5

RAID Summary

Level	Performance	Protection	Capacity (for N disks)
0	Best (parallel read/write)	Poor (lose all on 1 failure)	N
1	Good (write slower as 2x)	Good (have drive mirror)	N / 2
5	Good (must write parity block)	Good (one drive can fail)	N - 1
6	Good (must write multiple parity blocks)	Better (can have as many drives fail as dedicated to parity)	N – X (where X is # of parity drives such as 2)

31

v. **List different ways for representing strings in memory.**
   1. Null-terminated String: last byte value of 0 indicates end
   2. Byte-length String: length of string in bytes is specified in first few bytes before string starts
   3. Fixed-length String: always the same size

vi. **List different ways for representing date/times in memory.**
   1. Date representations
      a. Integer representation: number of days passed since a given date
      b. String representation: show a date's components as individual characters of a string (YYYYMMDD)
   2. Time representations
      a. Integer representation: number of seconds since a given time
      b. String representation: hours, minutes, seconds, fractions (HH:MM:SS:FF)

vii. **Explain the difference between fixed and variable length records.**
   1. Fixed-length records: all records have the same size
   2. Variable-length records: all records have the same size

viii. **Compare/contrast the ways of separating fields in a record.**
   1. *No separator*: store length of each field, so do not need a separate separator (fixed length field). Simple but wastes space within a field.
   2. *Length indicator*: store a length indicator at the start of the record (for the entire record) and a size in front of each field. Wastes space for each length field and need to know length beforehand.

3. *Use offsets*: at start of record store offset to each field
4. *Use delimiters*: separate fields with delimiters such as a comma (comma-separated files). Must make sure that delimiter character is not a valid character for field.
5. *Use keywords*: self-describing field names before field value (XML and JSON). Wastes space by using field names

ix. **Define and explain the role of schemas.**
1. *Schema*: description of the record layout
2. Contains…
   a. Names and number fields
   b. Size and type of each field
   c. Field ordering in record
   d. Description/meaning of each field

x. **Compare/contrast variable and fixed formats.**
1. *Fixed record format*: every record has the same fields with the same types
   a. Relational schemas
2. *Variable record format*: not all records have the same fields or organization
   a. Record data must be *self-describing*
   b. XML/JSON

xi. **List and briefly explain the six record placement issues in blocks.**
1. ***Separation:*** how do we separate adjacent records?
2. ***Spanning:*** can a record cross a block boundary?
3. ***Clustering***: can a block store multiple record types?
4. ***Splitting***: are records allocated in multiple blocks?
5. ***Ordering***: are the records sorted in any way?
6. ***Addressing***: how do we reference a given record?

II) Indexing

a. Objectives

i. Explain the types of indexes

ii. Perform calculations on how fast it takes to retrieve one record or answer a query given a certain data file and index type

iii. **Define:**
1. Index file: the file that stores the index information
2. Data file: the file that actually contains the records
3. Search key: the set of attributes stored by the index to find the records in the data file
   a. Does not have to be unique
   b. More than one record may have the same search key value

4. Index entry: one index record that contains a search key value and a pointer to the location of the record with that value

iv. **List the index evaluation metrics/criteria:**

1. Functionality: measured by the types of queries it supports
    a. Exact match on search key
    b. Query on a range of search key values
2. Performance: measured by the time required to excecute queries and update the index
    a. Access time
    b. Update
    c. Insert
    d. Delete time
3. Efficiency: measured by the amount of space required to maintain the index structure

v. **Explain the difference between the different types of indexes**

1. Dense index: has an index entry for every record in the data file
2. Sparse index: has index entries for only some of the data file records (often by block)
3. Primary index (clustering indexes): sorts the data file by its search key (doesn't have to be the same as primary key)
4. secondary index: does not determine the organization of the data file
5. single-level index: has only one index level
6. multi-level index: has several levels of indexes in the same file

vi. **List the techniques for indexing with duplicate search keys**

1. Create an index entry for each record
    a. Wastes space (key/value are repeated for each record)
2. Use buckets/blocks to store records with the same key
    a. Index entry points to the first record in the bucket
    b. All other matching records are retrieved from the bucket

vii. **Discuss some of the issues in index maintenance**

1. as the data file changes, the index must be updated as well
2. similar to ordered file maintenance but index files are smaller

3. techniques for managing data file (dense & sparse indexes):
    a. using overflow blocks
    b. re-organizing blocks by shifting records
    c. adding or deleting new blocks in the file

viii. Compare/contrast single versus multi-level indexes

1. Multi-level index: has more than one index level for the same data
    a. Each level of the multi-level index is smaller
    b. Can process each level more efficiently
    c. First level can be sparse or dense
    d. All higher levels must be sparse
    e. Index maintenance time increases with each level
2. Single-level index has only one index level
    a. Can be sparse or dense

ix. Explain the benefit of secondary indexes on query performance

1. Simpler maintenance of secondary index
    a. Secondary index changes only when primary index changes, not when the data file changes

x. Index calculations

1. Index entries/block = blockSize/idxEntrySize
2. Records/block = blockSize/recordSize
3. Number of index blocks = numRecords/(records/block)/(idxEntries/block)
4. Number of disk blocks (unspanned) = numRecords/(records/block)
5. Number of disk blocks (spanned) = ciel(numRecords/(records/block))
6. Number of blocks = numRecords/blockingFactor
7. Binary search blocks retrieved = ciel(log2(numBlocks))+1
    a. numBlocks = numDiskBlocks if no index
    b. else numBlocks = numIdxBlocks
8. Linear search blocks retrieved = numBlocks/2
9. sparse index:
    a. searching: N=numIndexBlocks
10. dense index:
    a. searching: N=numRecords/(idxEntries/block)
11. **Blocking Factor (index/datablock)**
    a. Unspanned = floor(blockSize/recordSize)
    b. Spanned = blockSize/recordSize
    c.

12. **Utilization** = (records/block)*(bytes/record)/blockSize
13. **Transfer time/average time to retrieve (s)** = numBlocks*blockSize/readBandwidth

III) B-Trees

- **a. Objectives**
  - **i. Insert and delete from a B-tree and a B+ tree**
    1. Inserting
       a. find leaf node where the new key belongs (it will have 1-2 keys)
       b. if key = 1 insert new key in the node in sorted order
       c. if keys = 2 insert node in sorted order (overflow)
          i. move middle key to parent node (split node)
          ii. if parent keys > 3 repeat node split
       d. continue until some ancestor has only 1 node or until all ancestors are full
          i. if all ancestors are full split root node and grow tree by 1 level
    2. Splitting
       a. given a node that is overflowing, split node into 2 nodes
       b. middle value gets passed to parent node
       c. repeat until a node with room for passed value is found
       d. if root node has 2 keys, split and make a new root with the middle node
    3. Deleting
       a. Locate node N containing key to delete K
          i. if K isn't found, algorithm is complete
       b. if N = interior node, find in-order successor of K and swap with K
          i. deletion always begins at leaf node L
       c. if L contains a value in addition to K, delete K from L
          i. no underflow: algorithm is done
          ii. underflow is when the # of nodes is < minimum # of keys
       d. if underflow occurs, merge node with its neighboring nodes
          i. check L siblings
             1. if sibling has max number of keys, redistribute them

            ii. else merge L with an adjacent sibling and bring down a value from L's parent
            iii. if parent(L) has underflow, recursively merge
            iv. if underflow occurs at root, tree shrinks a level
        e. Think pushing the value out of a leaf and redistribute any holes left
    ii. **Calculate the maximum order of a B-tree**
        1. one node size = keySize*numKeys+dataPtrSize*numKeys+childPtrSize*(numKeys+1) <= block Size
        2. max order = keySize/2
        3.
    iii. **Calculate query access times using B-tree indexes**
        1.
    iv. **Compare/contrast B-trees and B+ trees**
        1. B+-trees are similar to B-trees except all key values stay in the leaves of a B+-tree
        2. key values removed/promoted to parent nodes form leaves are copies

IV) R-Trees

- **a. Objectives**
    - **i. Explain the difference between an R-tree and a B+ tree**
        1. R-trees can handle multidimensional data
    - **ii. List some types of spatial data**
        1. multidimensional points
        2. lines
        3. rectangles
        4. geometric objects
    - **iii. List some types of spatial queries**
        1. Spatial range queries
            - a. query has associated region and asks to find matches within that region
            - b. answer may include overlapping/contained regions
        2. Nearest neighbor queries
            - a. find closest region to a given region
            - b. results are ordered by proximity from given region
        3. Spatial join queries
            - a. join two types of regions
            - b. expensive to compute
            - c. involves regions & proximity
    - **iv. List some applications of spatial data and queries**
        1. Geographic information systems (GIS)
            - a. use spatial data for modeling terrain

2. Computer-aided design (CAD)
    a. process spatial objects when designing systems
    b. spatial constraints
3. Multimedia databases
    a. storing images, text, and video requires spatial data management

v. **Explain the idea of insertion in an R-tree**
1. start at root and go down to best fit leaf L
    a. best fit L: child whose box needs least enlargement to cover B, resolve ties by going to smallest area child
2. if best fit L has space insert and stop
3. else split L into L1 and L2
    a. existing entries in L + newly inserted entry must be distributed between L1 and L2

V) Hashing

a. Objectives

i. Perform open address hashing with linear probing
1. computes hash function(y=f(x)) and attempts to put key in location y
2. if y is occupied, scan array to find next open location
    a. array is treated as circular

ii. **Perform linear hashing**
1. insert record with key K by computing its hash value H
2. take the last d bits of H where d is the current # of bits used
3. find the bucket m where K would belong using d
4. if m<n, bucket exists, go back to that bucket
    a. insert K if bucket has space
    b. else use an overflow block
5. if m>=n, put K in bucket m-2^(d-1)
6. after each insert, check to see if load factor lf<threshold
7. if lf>= threshold, perform split
    a. add new bucket n (may increase directory size d)
    b. divide records between new bucket n=1bd and bucket 0bd
    c. bucket split may not be the bucket where the record was added
    d. update n and d to reflect new bucket

iii. **Define:**
1. Hashing: a technique for mapping key values to locations

2. Collision: when two different keys are trying to be stored in the same location
3. Perfect hash function: a function that doesn't allow for any two keys to map to the same location

iv. **Calculate load factor of a hash table**
1. load factor = (# of records stored)/(# of possible storage locations)
2. # of possible storage locations (s) = numBlocks*(records/block)

v. **Compare/contrast external hashing and main memory hashing**
1. external hashing allocates records with keys to blocks on disk rather than locations in memory

vi. **Compare/contrast B+ trees and linear hashing**
1. hashing is better at retrieving records with a specified value for the key
2. B+-trees are better for range queries

VI) SQL Indexing

a. Objectives

i. **Use index structures in SQL using CREATE/ALTER commands**
1. CREATE [UNIQUE] INDEX indexName ON tableName (colName [ASC|DESC][,…])
   a. UNIQUE: each value in index is unique
   b. ASC/DESC: specifies sort order for index
   c. syntax varies between systems
2. DROP INDEX indexName

ii. **Perform insertions and searches using partitioned hashing**
1. partitioned hashing: the overall hash location is a combination of the hash values from each key
2. key1 ->hash function-> hash1 (h1), key2 ->hash function-> hash2 (h2)
   a. hash location (L) = 12 bits long
   b. 1^{st} 6 bits = h1
   c. 2^{nd} 6 bits = h2

iii. **Perform searches using grid files (where lo < x < hi and lo < y < hi)**
1. find number of partitions for each variable
2. number of partitions from x * number from y

iv. **Understand how bitmap indexes are used for searching and why they provide a space and speed improvement in certain cases**
1. useful for indexing attributes that have small number values

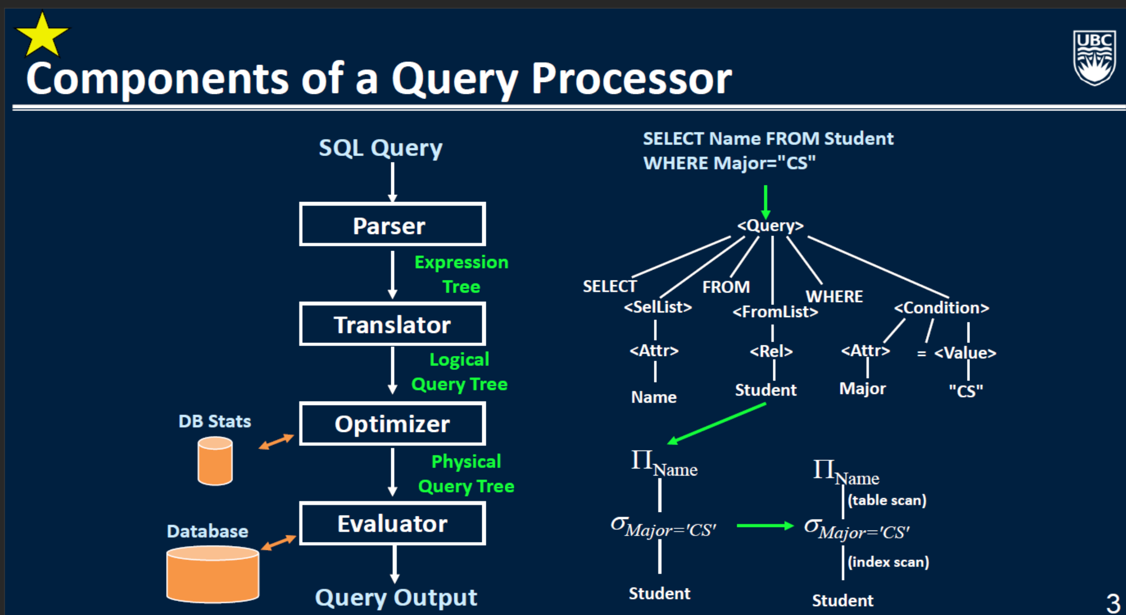
VII) Query Processing

a. Variable definitions

i. M-the number of buffer blocks available to the algorithm

            1. always less than the size of memory
        ii. B(R)-the number of blocks on disk used to store all the tuples of R
            1. assume that R is clustered and algorithm can only read 1 block at time
            2. ignore free space in blocks
        iii. T(R)-the number of tuples in R
        iv. V(R,a)-the number of distinct values of column a in R

b. **Objectives**
    i. **Diagram the query processor components and explain their function**



    ii. **Calculate block access for one-pass algorithms**
        1. number of blocks accessed = B(R)+B(S)
        2. one pass can be completed as long as B(S)<=M-1|B(S)<M
        3. where B=buffer, M=memory
    iii. **Calculate block accesses for tuple and block nested joins**
        1. R=bigger relation, S=smaller relation
        2. **Tuple Based:**
            a. worst case = T(R)*T(S)
            b. if there's an index on the join attribute of R, the entire relation R does not have to be read
                i. T(S)
        3. **Block-Based:**
            a. worst case = B(S)+B(R)*ciel(B(S)/M-1))
            b. if R is in outer loop = B(S)+B(R)*ciel(B(R)/M-1))
            c. buffers smaller relation into memory
            d.
    iv. **Perform two-pass sorting methods including all operators, sort-join and sort-merge-join and calculate performance**

v. **Perform two-pass hashing methods including all operators, hash-join and hybrid-hash-join and calculate performance**

vi. **Explain the goal of query processing**

1. **return the answer to a SQL query in the most efficient way possible given the organization of the database**

vii. **List the relational and set operators**

1. **Relational:**

• selection	σ	- return subset of rows
• projection	π	- return subset of columns
• Cartesian product	$\times$	- all combinations of two relations
• join	$\bowtie$	- combines σ and $\times$
• duplicate elimination	δ	- eliminates duplicates

2. **Set:**

• Union	$\cup$	- tuple in output if in either or both
• Difference	-	- tuple in output if in 1st but not 2nd
• Intersection	$\cap$	- tuple in output if in both

• Union compatibility means relations must have the same number of columns with compatible domains.

viii. **Explain how index and table scans work and calculate the block operations performed**

1. table scan: read the relation R from disk one block at a time
   a. cost = B
2. index scan: read R or just its tuples that satisfy a given condition by using an index on R

ix. **Write an iterator in java for a relational operator**

1. **table scan:**

```
init() {
    b = the first block of R;
    t = first tuple of R;
}
next() {
    if (t is past the last tuple on block b) {
        increment b to the next block;
        if (there is no next block)
            return NULL;
        else    /* b is a new block */
            t = first tuple on block b;
    }
    oldt = t;
    increment t to the next tuple of b;
    return oldt;
}
close() {}
```

2. main-memory:

```
init() {
    Allocate buffer array A
    read entire relation R block-by-block into A;
    sort A using quick sort;
    tLoc = 0;   // First tuple location in A
}

next() {
    if (tLoc >= T)
            return NULL;
    else
    {   tLoc++;
        return A[tLoc-1];
    }
}

close() {}
```

How is this it
the table scan

x. List the tuple-at-a-time relational operators
xi. Illustrate how one-pass algorithms for selection, project, duplicate elimination, and binary operators work and calculate performance and memory requirements
xii. Perform and calculate performance of two-pass sorting based algorithms, sort-merge algorithm, set operators, sort-merge-join/sort-join

Operators	Approximate M required	Disk I/Os
γ,δ	$\sqrt{B}$	3*B
$\cup,-,\cap$	$\sqrt{B(R)+B(S)}$	3*(B(R) + B(S))
$\bowtie$ (sort)	$\sqrt{\max(B(R),B(S))}$	5*(B(R) + B(S))
$\bowtie$ (sort-merge)	$\sqrt{B(R)+B(S)}$	3*(B(R) + B(S))

xiii. **Perform and calculate performance of two-pass hashing based algorithms, hash partitioning, operation implementation and performance, hash join, hybrid-hash-join**

Operators	Approximate M required	Disk I/Os
γ,δ	$\sqrt{B}$	3*B
$\cup,-,\cap$	$\sqrt{B(S)}$	3*(B(R) + B(S))
$\bowtie$ (simple)	$\sqrt{B(S)}$	3*(B(R) + B(S))
$\bowtie$ (hybrid)	$\sqrt{B(S)}$	$(3-\frac{2M}{B(S)})(B(R)+B(S))$

xiv. **Compare/contrast sorting versus hashing methods**

1. hash-based algorithms for binary operations require memory based on the size of the smaller of the two relations rather than the sum of the relation sizes
2. sort-based algorithms produce the result in sorted order which may be used for later operations
3. hash-based algorithms depend on the buckets being equal size
4. sort-based algorithms may write sorted sublists to consecutive disk blocks
5. both algorithms save disk access time by writing/reading several blocks at once if memory is available

6. hash-based joins are usually the best if neither of the input relations are sorted or there are no indexes for equi-join
7. hashing performs divide and conquer
8. sorting performs conquer and merge (sort merge)

xv. **Calculate performance of index-based algorithms**
1. cost estimate
2. complicated sections
3. index joins
    a. cost = T(S)*(T(R)/V(R,Y))
    b. makes sense when V(R,Y) is large and S is small

xvi. **Explain how two-pass algorithms are extended to multi-pass algorithms**
1. two-pass algorithms based on sorting and hashing can be extended to any number or passes using recursion
2. each pass partitions the relations into smaller pieces
3. eventually the partitions will fit entirely into memory
4. for k passes…
    a. memory requirement M = (B(R))^1/k
    b. Maximum relation size B(R) <= M^k
    c. Disk operations = 2*k*B(R)
    d. w/o final pass = 2*k*B(R)-B(R)

xvii. **List some recent join algorithms**

VIII)Query Optimization **Not on Midterm**

a. Objectives

i. Convert an SQL query to a parse tree using a grammar

ii. Convert a pars tree to a logical query tree

iii. Use heuristic optimization and relational algebra laws to optimize logical query trees

iv. Convert a logical query tree to a physical query tree

v. Calculate size estimates for selection, projection, joins, and set operations

vi. Explain the difference between syntax and semantic validation and the query processor component responsible for each

vii. Define:
1. **Valid parse tree**
2. **Logical query tree**
3. **Physical query tree**
4. **Join-orders**
5. **Left-deep**
6. **Right-deep**
7. **Balanced join trees**

viii. Explain the difference between correlated and uncorrelated nested queries
ix. Define and use canonical logical query trees
x. Explain issues in selecting algorithms for selection and join
xi. Compare/contrast materialization versus pipelining and know when to use them when building physical query plans