Chapter – 17

Disk Storage Devices:

- ✓ A track is divided into smaller blocks or sectors because it usually contains a large amount of information.
- ✓ The division of a track into sectors is hard-coded on the disk surface and cannot be changed.
- ✓ A track is divided into blocks.
- ✓ The block size B is fixed for each system.
- ✓ Typical block sizes range from B=512 bytes to B=4096 bytes.
- ✓ Whole blocks are transferred between disk and main memory for processing.
- ✓ A read-write head moves to the track that contains the block to be transferred.
- ✓ Disk rotation moves the block under the read-write head for reading or writing.
- ✓ A physical disk block (hardware) address consists of:
  - a cylinder number (imaginary collection of tracks of same radius from all recorded surfaces)
  - the track number or surface number (within the cylinder)
  - and block number (within track).

Records

- ✓ Fixed and variable length records
- ✓ Records contain fields which have values of a particular type
  E.g., amount, date, time, age
- ✓ Fields themselves may be fixed length or variable length
- ✓ Variable length fields can be mixed into one record:
  Separator characters or length fields are needed so that the record can be "parsed."

Blocking

- ✓ Blocking : Refers to storing a number of records in one block on the disk.
- ✓ Blocking factor (bfr) refers to the number of records per block.
- ✓ There may be empty space in a block if an integral number of records do not fit in one block.
- ✓ Spanned Records: Refers to records that exceed the size of one or more blocks and hence span a number of blocks

Files of record

- ✓ A file is a sequence of records, where each record is a collection of data values (or data items).
- ✓ A file descriptor (or file header) includes information that describes the file, such as the field names and their data types, and the addresses of the file blocks on disk.
- ✓ Records are stored on disk blocks.

- ✓ The blocking factor bfr for a file is the (average) number of file records stored in a disk block.
- ✓ A file can have fixed-length records or variable-length records.
- ✓ File records can be unspanned or spanned
  Unspanned: no record can span two blocks
  Spanned: a record can be stored in more than one block
- ✓ The physical disk blocks that are allocated to hold the records of a file can be contiguous, linked, or indexed.

Operation on Files Typical file operations include:

- ✓ OPEN: Readies the file for access, and associates a pointer that will refer to a current file record at each point in time.
- ✓ FIND: Searches for the first file record that satisfies a certain condition, and makes it the current file record.
- ✓ FINDNEXT: Searches for the next file record (from the current record) that satisfies a certain condition, and makes it the current file record.
- ✓ READ: Reads the current file record into a program variable.
- ✓ INSERT: Inserts a new record into the file & makes it the current file record.
- ✓ DELETE: Removes the current file record from the file, usually by marking the record to indicate that it is no longer valid.
- ✓ MODIFY: Changes the values of some fields of the current file record.
- ✓ CLOSE: Terminates access to the file.
- ✓ REORGANIZE: Reorganizes the file records.
- ✓ READ_ORDERED: Read the file blocks in order of a specific field of the file.

انتبهوا للمقارنة بين:

Unordered Files called a heap or a pile file.

- ✓ New records are inserted at the end of the file.
- ✓ A linear search through the file records is necessary to search for a record.
- ✓ This requires reading and searching half the file blocks on the average, and is hence quite expensive.
- ✓ Record insertion is quite efficient.
- ✓ Reading the records in order of a particular field requires sorting the file records.

Ordered Files

- ✓ Also called a sequential file.
- ✓ File records are kept sorted by the values of an ordering field.
- ✓ Insertion is expensive: records must be inserted in the correct order.
- ✓ It is common to keep a separate unordered overflow (or transaction) file for new records to improve insertion efficiency; this is periodically merged with the main ordered file.
- ✓ A binary search can be used to search for a record on its ordering field value.
- ✓ This requires reading and searching log2 of the file blocks on the average, an improvement over linear search.
- ✓ Reading the records in order of the ordering field is quite efficient.

Hashed Files

- ✓ Hashing for disk files is called External Hashing
- ✓ The file blocks are divided into M equal-sized buckets, numbered bucket0, bucket1, ..., bucketM-1.
- ✓ Typically, a bucket corresponds to one (or a fixed number of) disk block.
- ✓ One of the file fields is designated to be the hash key of the file.
- ✓ The record with hash key value K is stored in bucket i, where i=h(K), and h is the hashing function.
- ✓ Search is very efficient on the hash key.
- ✓ Collisions occur when a new record hashes to a bucket that is already full.
- ✓ An overflow file is kept for storing such records.
- ✓ Overflow records that hash to each bucket can be linked together.

There are numerous methods for collision resolution, including the following: مهمة جداً

1) Open addressing: Proceeding from the occupied position specified by the hash address, the program checks the subsequent positions in order until an unused (empty) position is found.
2) Chaining: For this method, various overflow locations are kept, usually by extending the array with a number of overflow positions. A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location.

- ✓ Multiple hashing: The program applies a second hash function if the first results in a collision.
- ✓ Main disadvantages of static external hashing:
  - Fixed number of buckets M is a problem if the number of records in the file grows or shrinks.
  - Ordered access on the hash key is quite inefficient (requires sorting the records).

Dynamic And Extendible Hashed Files

- ✓ Hashing techniques are adapted to allow the dynamic growth and shrinking of the number of file records.
- ✓ These techniques include the following: dynamic hashing, extendible hashing, and linear hashing.
- ✓ Both dynamic and extendible hashing use the binary representation of the hash value h(K) in order to access a directory.
- ✓ In dynamic hashing the directory is a binary tree.
- ✓ In extendible hashing the directory is an array of size 2d where d is called the global depth.
- ✓ Dynamic and extendible hashing do not require an overflow area.
- ✓ Linear hashing does require an overflow area but does not use a directory.
- ✓ Blocks are split in linear order as the file expands.

Parallelizing Disk Access using RAID Technology.

- ✓ The main goal of RAID is to even out the widely different rates of performance improvement of disks against those in memory and microprocessors.
- ✓ Raid level 0 has no redundant data and hence has the best write performance at the risk of data loss
- ✓ Raid level 1 uses mirrored disks.
- ✓ Raid level 2 uses memory-style redundancy by using Hamming codes. Level 2 includes both error detection and correction.
- ✓ Raid level 3 uses a single parity disk relying on the disk controller to figure out which disk has failed.
- ✓ Raid Levels 4 and 5 use block-level data striping, with level 5 distributing data and parity information across all disks.
- ✓ Raid level 6 applies the so-called P + Q redundancy scheme using Reed-Soloman codes to protect against up to two disk failures by using just two redundant disks.

Chapter – 18

Indexes as Access Paths

- ✓ A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- ✓ The index is usually specified on one field of the file (<u>although it could be specified on several fields</u>)
- ✓ One form of an index is a file of entries <field value, pointer to record>, which is ordered by field value
- ✓ The index is called an access path on the field.
- ✓ Indexes can also be characterized as dense or sparse (مهم)

  A dense index has an index entry for every search key value (and hence every record) in the data file.

  A sparse (or nondense) index, on the other hand, has index entries for only some of the search values
- ✓ Example: Given the following data file EMPLOYEE (NAME, SSN, ADDRESS, JOB, SAL, ...)
- ✓ Suppose that:
- ✓ record size R=150 bytes block size B=512 bytes r=30000 records
- ✓ Then, we get:
- ✓ blocking factor Bfr= B div R= 512 div 150= 3 records/block
- ✓ number of file blocks b= (r/Bfr)= (30000/3)= 10000 blocks
- ✓ For an index on the SSN field, assume the field size VSSN=9 bytes, assume the record pointer size PR=7 bytes. Then:
- ✓ index entry size RI=(VSSN+ PR)=(9+7)=16 bytes
- ✓ index blocking factor BfrI= B div RI= 512 div 16= 32 entries/block

- ✓ number of index blocks b= (r/ Bfrl)= (30000/32)= 938 blocks
- ✓ binary search needs log2bI= log2938= 10 block accesses
- ✓ This is compared to an average linear search cost of:
- ✓ (b/2)= 30000/2= 15000 block accesses
- ✓ If the file records are ordered, the binary search cost would be:
- ✓ log2b= log230000= 15 block accesses

Types of Single-Level Indexes

- ✓ Primary Index Defined on an ordered data file
- ✓ The data file is ordered on a key field
- ✓ Includes one index entry for each block in the data file; the index entry has the key field value for the first record in the block, which is called the block anchor
- ✓ A similar scheme can use the last record in a block.
- ✓ A primary index is a nondense (sparse) index, since it includes an entry for each disk block of the data file and the keys of its anchor record rather than for every search value
- ✓ Clustering Index Defined on an ordered data file
- ✓ The data file is ordered on a non-key field unlike primary index, which requires that the ordering field of the data file have a distinct value for each record.
- ✓ Includes one index entry for each distinct value of the field; the index entry points to the first data block that contains records with that field value.
- ✓ It is another example of nondense index where Insertion and Deletion is relatively straightforward with a clustering index.
- ✓ Secondary Index A secondary index provides a secondary means of accessing a file for which some primary access already exists.
- ✓ The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- ✓ The index is an ordered file with two fields.
- ✓ The first field is of the same data type as some non-ordering field of the data file that is an indexing field.
- ✓ The second field is either a block pointer or a record pointer.
- ✓ There can be many secondary indexes (and hence, indexing fields) for the same file.
- ✓ Includes one entry for each record in the data file; hence, it is a dense index

Multi-Level Indexes

- ✓ Because a single-level index is an ordered file, we can create a primary index to the index itself;

- ✓ In this case, the original index file is called the first-level index and the index to the index is called the second-level index.
- ✓ We can repeat the process, creating a third, fourth, ..., top level until all entries of the top level fit in one disk block
- ✓ A multi-level index can be created for any type of first-level index (primary, secondary, clustering) as long as the first-level index consists of more than one disk block
- ✓ Such a multi-level index is a form of search tree
- ✓ However, insertion and deletion of new index entries is a severe problem because every level of the index is an ordered file.

Dynamic Multilevel Indexes Using B-Trees and B+-Trees

- ✓ Most multi-level indexes use B-tree or B+-tree data structures because of the insertion and deletion problem
- ✓ This leaves space in each tree node (disk block) to allow for new index entries
- ✓ These data structures are variations of search trees that allow efficient insertion and deletion of new search values.
- ✓ In B-Tree and B+-Tree data structures, each node corresponds to a disk block
- ✓ Each node is kept between half-full and completely full
- ✓ An insertion into a node that is not full is quite efficient
- ✓ If a node is full the insertion causes a split into two nodes
- ✓ Splitting may propagate to other tree levels
- ✓ A deletion is quite efficient if a node does not become less than half full
- ✓ If a deletion causes a node to become less than half full, it must be merged with neighboring nodes

Difference between B-tree and B+-tree

1) In a B-tree, pointers to data records exist at all levels of the tree
2) In a B+-tree, all pointers to data records exists at the leaf-level nodes
3) A B+-tree can have less levels (or higher capacity of search values) than the corresponding B-tree

Chapter – 19

- ✓ Query optimization: The process of choosing a suitable execution strategy for processing a query.
- ✓ Query block: The basic unit that can be translated into the algebraic operators and optimized.
- ✓ A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block.
- ✓ Nested queries within a query are identified as separate query blocks.
- ✓ Aggregate operators in SQL must be included in the extended algebra.
- ✓ External sorting: Refers to sorting algorithms that are suitable for large files of records stored on disk that do not fit entirely in main memory, such as most database files.
- ✓ Query tree: A tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as leaf nodes of the tree, and represents the relational algebra operations as internal nodes.
- ✓ An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.
- ✓ Query graph: A graph data structure that corresponds to a relational calculus expression. It does not indicate an order on which operations to perform first. There is only a single graph corresponding to each query.

Using Heuristics in Query Optimization (9)

General Transformation Rules for Relational Algebra Operations:

1) Cascade of σ : A conjunctive selection condition can be broken up into a cascade (sequence) of individual s operations:

$$\sigma_{c1 \text{ AND } c2 \text{ AND } \dots \text{ AND } cn}(R) = \sigma_{c1}(\sigma_{c2}(\dots(\sigma_{cn}(R))\dots))$$

2) Commutativity of σ : The s operation is commutative:

$$\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$$

3) Cascade of π: In a cascade (sequence) of p operations, all but the last one can be ignored:

$$\pi_{List1}(\pi_{List2}(\dots(\pi_{Listn}(R))\dots)) = \pi_{List1}(R)$$

4) Commuting σ with π: If the selection condition c involves only the attributes A1, ..., An in the projection list, the two operations can be commuted:

$$\pi_{A1, A2, \dots, An}(\sigma_c(R)) = \sigma_c(\pi_{A1, A2, \dots, An}(R))$$

5) Commutativity of ⋈ (and x): The ⋈ operation is commutative as is the x operation: $R \bowtie_c S = S \bowtie_c R$; $R \times S = S \times R$

6) Commuting σ with (or $\bowtie$): If all the attributes in the selection condition c involve only the attributes of one of the relations being joined—say, R—the two operations can be commuted as follows:

$$s_c (R \bowtie S) = (s_c (R)) \bowtie S$$

- ✓ Alternatively, if the selection condition c can be written as (c1 and c2), where condition c1 involves only the attributes of R and condition c2 involves only the attributes of S, the operations commute as follows:

$$\sigma_c (R \bowtie S) = (\sigma_{c1} (R)) \bowtie (\sigma_{c2} (S))$$

7) Commuting π with (or $\bowtie$): Suppose that the projection list is L = {A1, ..., An, B1, ..., Bm}, where A1, ..., An are attributes of R and B1, ..., Bm are attributes of S. If the join condition c involves only attributes in L, the two operations can be commuted as follows:

$$\pi_L (R \bowtie_c S) = (\pi_{A1, ..., An} (R)) \bowtie_c (\pi_{B1, ..., Bm} (S))$$

- ✓ If the join condition C contains additional attributes not in L, these must be added to the projection list, and a final π operation is needed.

8) Commutativity of set operations: The set operations υ and ∩ are commutative but "–" is not.

9) Associativity of , x, $\bowtie$, and ∩ : These four operations are individually associative; that is, if Θ stands for any one of these four operations (throughout the expression), we have (R Θ S) Θ T = R Θ (S Θ T)

10) Commuting s with set operations: The σ operation commutes with υ , ∩ , and – . If q stands for any one of these three operations, we have

$$\sigma_c (R \; q \; S) = (\sigma_c (R)) \; q \; (\sigma_c (S))$$

- ✓ The π operation commutes with υ.

$$\pi_L (R \cup S) = (\pi_L (R)) \cup (\pi_L (S))$$

- ✓ Converting a (σ, x) sequence into $\bowtie$: If the condition c of a σ that follows a x Corresponds to a join condition, convert the (σ, x) sequence into a as $\bowtie$ follows:

$$(\sigma_C (R \times S)) = (R \bowtie_C S)$$

- ✓ Other transformations
- ✓ Query Execution Plans
- ✓ An execution plan for a relational algebra query consists of a combination of the relational algebra query tree and information about the access methods to be used for each relation as well as the methods to be used in computing the relational operators stored in the tree.
- ✓ Materialized evaluation: the result of an operation is stored as a temporary relation.

- ✓ Pipelined evaluation: as the result of an operator is produced, it is forwarded to the next operator in sequence.

Using Selectivity and Cost Estimates in Query Optimization (1)

- ✓ Cost-based query optimization:
  - Estimate and compare the costs of executing a query using different execution strategies and choose the strategy with the lowest cost estimate.
  - (Compare to heuristic query optimization)
- ✓ Issues
  - Cost function
  - Number of execution strategies to be considered

Using Selectivity and Cost Estimates in Query Optimization (2)

- ✓ Cost Components for Query Execution
  1) Access cost to secondary storage
  2) Storage cost
  3) Computation cost
  4) Memory usage cost
  5) Communication cost
- ✓ Note: Different database systems may focus on different cost components.

Using Selectivity and Cost Estimates in Query Optimization (3)

- ✓ Catalog Information Used in Cost Functions
- ✓ Information about the size of a file
  - number of records (tuples) (r),
  - record size (R),
  - number of blocks (b)
  - blocking factor (bfr)
- ✓ Information about indexes and indexing attributes of a file
  - Number of levels (x) of each multilevel index
  - Number of first-level index blocks (bI1)
  - Number of distinct values (d) of an attribute
  - Selectivity (sl) of an attribute
  - Selection cardinality (s) of an attribute. (s = sl * r)

Using Selectivity and Cost Estimates in Query Optimization (4)

- ✓ Examples of Cost Functions for SELECT
- ✓ S1. Linear search (brute force) approach
  - $C_{S1a} = b$;
  - For an equality condition on a key, $C_{S1a} = (b/2)$ if the record is found; otherwise $C_{S1a} = b$.
- ✓ S2. Binary search:

  - $C_{S2} = \log_2 b + (s/bfr)\rceil - 1$
  - For an equality condition on a unique (key) attribute, $C_{S2} = \log_2 b$
- S3. Using a primary index (S3a) or hash key (S3b) to retrieve a single record
  - $C_{S3a} = x + 1$; $C_{S3b} = 1$ for static or linear hashing;
  - $C_{S3b} = 1$ for extendible hashing;

Using Selectivity and Cost Estimates in Query Optimization (5)

- ✓ Examples of Cost Functions for SELECT (contd.)
- ✓ S4. Using an ordering index to retrieve multiple records:
  - For the comparison condition on a key field with an ordering index, $C_{S4} = x + (b/2)$
- ✓ S5. Using a clustering index to retrieve multiple records:
  - $C_{S5} = x + \lceil (s/bfr) \rceil$
- ✓ S6. Using a secondary (B+-tree) index:
  - For an equality comparison, $C_{S6a} = x + s$;
  - For an comparison condition such as >, <, >=, or <=,

Chapter – 20

Physical Database Design in Relational Databases (1)

- ✓ **Factors that Influence Physical Database Design:**

A. Analyzing the database queries and transactions
B. Analyzing the expected frequency of invocation of queries and transactions
C. Analyzing the time constraints of queries and transactions
D. Analyzing the expected frequencies of update operations
E. Analyzing the uniqueness constraints on attributes

1. Analyzing the database queries and transactions

- ✓ For each query, the following information is needed.
  1) The attributes on which any selection conditions for the query are specified;
  2) The attributes on which any join conditions or conditions to link multiple tables or objects for the query are specified;
  3) The attributes whose values will be retrieved by the query.
- ✓ Note: the attributes listed in items 2 and 3 above are candidates for definition of access structures.
- ✓ For each update transaction or operation, the following information is needed.

1) The files that will be updated;
2) The type of operation on each file (insert, update or delete);
3) The attributes on which selection conditions for a delete or update operation are specified;
4) The attributes whose values will be changed by an update operation.

- ✓ Note: the attributes listed in items 3 above are candidates for definition of access structures. However, the attributes listed in item 4 are candidates for avoiding an access structure.
- ✓ B. Analyzing the expected frequency of invocation of queries and transactions
  - The expected frequency information, along with the attribute information collected on each query and transaction, is used to compile a cumulative list of expected frequency of use for all the queries and transactions.
  - It is expressed as the expected frequency of using each attribute in each file as a selection attribute or join attribute, over all the queries and transactions.
  - 80-20 rule
- ✓ 20% of the data is accessed 80% of the time
- ✓ C. Analyzing the time constraints of queries and transactions
  - Performance constraints place further priorities on the attributes that are candidates for access paths.
  - The selection attributes used by queries and transactions with time constraints become higher-priority candidates for primary access structure
- ✓ D. Analyzing the expected frequencies of update operations
  - A minimum number of access paths should be specified for a file that is updated frequently.
- ✓ E. Analyzing the uniqueness constraints on attributes
  - Access paths should be specified on all candidate key attributes — or set of attributes — that are either the primary key or constrained to be unique.
- ✓ Denormalization as a design decision for speeding up queries
  - The goal of normalization is to separate the logically related attributes into tables to minimize redundancy and thereby avoid the update anomalies that cause an extra processing overheard to maintain consistency of the database.
  - The goal of denormalization is to improve the performance of frequently occurring queries and transactions. (Typically the designer adds to a table

attributes that are needed for answering queries or producing reports so that a join with another table is avoided.)

- Trade off between update and query performance

An Overview of Database Tuning in Relational Systems (1)

✓ Tuning: The process of continuing to revise/adjust the physical database design by monitoring resource utilization as well as internal DBMS processing to reveal bottlenecks such as contention for the same data or devices.

✓ Goal:

1) To make application run faster
2) To lower the response time of queries/transactions
3) To improve the overall throughput of transactions

✓ Statistics internally collected in DBMSs:

- Size of individual tables
- Number of distinct values in a column
- The number of times a particular query or transaction is submitted/executed in an interval of time
- The times required for different phases of query and transaction processing

✓ Statistics obtained from monitoring:

1. Storage statistics
2. I/O and device performance statistics
3. Query/transaction processing statistics
4. Locking/logging related statistics
5. Index statistic

✓ Tuning Indexes

✓ Reasons to tuning indexes

- Certain queries may take too long to run for lack of an index;
- Certain indexes may not get utilized at all;
- Certain indexes may be causing excessive overhead because the index is on an attribute that undergoes frequent changes

✓ Options to tuning indexes

- Drop or/and build new indexes
- Change a non-clustered index to a clustered index (and vice versa)
- Rebuilding the index

- ✓ Tuning the Database Design
  - Dynamically changed processing requirements need to be addressed by making changes to the conceptual schema if necessary and to reflect those changes into the logical schema and physical design.
- ✓ Possible changes to the database design
  - Existing tables may be joined (denormalized) because certain attributes from two or more tables are frequently needed together.
  - For the given set of tables, there may be alternative design choices, all of which achieve 3NF or BCNF. One may be replaced by the other.
  - A relation of the form R(K, A, B, C, D, …) that is in BCNF can be stored into multiple tables that are also in BCNF by replicating the key K in each table.
  - Attribute(s) from one table may be repeated in another even though this creates redundancy and potential anomalies.
  - Apply horizontal partitioning as well as vertical partitioning if necessary.
- ✓ Tuning Queries
- ✓ Indications for tuning queries
  - A query issues too many disk accesses
  - The query plan shows that relevant indexes are not being used.
- ✓ Typical instances for query tuning
  - In some situations involving using of correlated queries, temporaries are useful.
  - If multiple options for join condition are possible, choose one that uses a clustering index and avoid those that contain string comparisons.
  - The order of tables in the FROM clause may affect the join processing.
  - Some query optimizers perform worse on nested queries compared to their equivalent un-nested counterparts.
  - Many applications are based on views that define the data of interest to those applications. Sometimes these views become an overkill
- ✓ Additional Query Tuning Guidelines
  1) A query with multiple selection conditions that are connected via OR may not be prompting the query optimizer to use any index. Such a query may be split up and expressed as a union of queries, each with a condition on an attribute that causes an index to be used.
  2) Apply the following transformations
     - NOT condition may be transformed into a positive expression.

- Embedded SELECT blocks may be replaced by joins.
- If an equality join is set up between two tables, the range predicate on the joining attribute set up in one table may be repeated for the other table

3) WHERE conditions may be rewritten to utilize the indexes on multiple columns.

Chapter – 21

Introduction to Transaction Processing

- ✓ Single-User System: At most one user at a time can use the system.
- ✓ Multiuser System: Many users can access the system concurrently.
- ✓ Concurrency
  - Interleaved processing: Concurrent execution of processes is interleaved in a single CPU
  - Parallel processing: Processes are concurrently executed in multiple CPUs.
- ✓ A Transaction: Logical unit of database processing that includes one or more access operations (read -retrieval, write - insert or update, delete).
- ✓ A transaction (set of operations) may be stand-alone specified in a high level language like SQL submitted interactively, or may be embedded within a program.
- ✓ Transaction boundaries: Begin and End transaction.
- ✓ An application program may contain several transactions separated by the Begin and End transaction boundaries.
- ✓ SIMPLE MODEL OF A DATABASE (for purposes of discussing transactions):
  1) A database is a collection of named data items
  2) Granularity of data - a field, a record , or a whole disk block (Concepts are independent of granularity)
  3) Basic operations are read and write
- ✓ read_item(X): Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also named X.
- ✓ write_item(X): Writes the value of program variable X into the database item named X.

READ AND WRITE OPERATIONS:

- ✓ Basic unit of data transfer from the disk to the computer main memory is one block. In general, a data item (what is read or written) will be the field of some record in the database, although it may be a larger unit such as a record or even a whole block.
- ✓ read_item(X) command includes the following steps:

1) Find the address of the disk block that contains item X.
2) Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3) Copy item X from the buffer to the program variable named X

- ✓ **write_item(X) command includes the following steps:**

1) Find the address of the disk block that contains item X.
2) Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3) Copy item X from the program variable named X into its correct location in the buffer.
4) Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Why Concurrency Control is needed:

- ✓ **The Lost Update Problem** This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.
- ✓ **The Temporary Update (or Dirty Read) Problem** This occurs when one transaction updates a database item and then the transaction fails for some reason (see Section 21.1.4).
  The updated item is accessed by another transaction before it is changed back to its original value.
- ✓ **The Incorrect Summary Problem** If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

(What causes a Transaction to fail)

1) **A computer failure (system crash):** A hardware or software error occurs in the computer system during transaction execution. If the hardware crashes, the contents of the computer's internal memory may be lost.
2) **A transaction or system error:** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.
3) **Local errors or exception conditions detected by the transaction:**

   Certain conditions necessitate cancellation of the transaction. For example, data for the transaction may not be found. A condition, such as insufficient account balance in a banking database, may

cause a transaction, such as a fund withdrawal from that account, to be canceled.
A programmed abort in the transaction causes it to fail.

4) Concurrency control enforcement: The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock .
5) Disk failure: Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.
6) Physical problems and catastrophes: This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator

Transaction and System Concepts

- ✓ A transaction is an atomic unit of work that is either completed in its entirety or not done at all.
- ✓ For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts.
- ✓ Transaction states: Active state, Partially committed state, Committed state, Failed state, Terminated State

Recovery manager keeps track of the following operations:

1) begin_transaction: This marks the beginning of transaction execution.
2) read or write: These specify read or write operations on the database items that are executed as part of a transaction.
3) end_transaction: This specifies that read and write transaction operations have ended and marks the end limit of transaction execution.

- At this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database or whether the transaction has to be aborted because it violates concurrency control or for some other reason.

4) commit_transaction: This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
5) rollback (or abort): This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.
6) undo: Similar to rollback except that it applies to a single operation rather than to a whole transaction.

7) redo: This specifies that certain transaction operations must be redone to ensure that all the operations of a committed transaction have been applied successfully to the database.

The System Log

- ✓ Log or Journal: The log keeps track of all transaction operations that affect the values of database items.
- ✓ This information may be needed to permit recovery from transaction failures.
- ✓ The log is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure.
- ✓ In addition, the log is periodically backed up to archival storage (tape) to guard against such catastrophic failures.
- ✓ T in the following discussion refers to a unique transaction-id that is generated automatically by the system and is used to identify each transaction:

Types of log record:

1) [start_transaction,T]: Records that transaction T has started execution.
2) [write_item,T,X,old_value,new_value]: Records that transaction T has changed the value of database item X from old_value to new_value.
3) [read_item,T,X]: Records that transaction T has read the value of database item X.
4) [commit,T]: Records that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5) [abort,T]: Records that transaction T has been aborted.

- ✓ Protocols for recovery that avoid cascading rollbacks do not require that read operations be written to the system log, whereas other protocols require these entries for recovery.
- ✓ Strict protocols require simpler write entries that do not include new_value (see Section 21.4).
- ✓ Recovery using log records: If the system crashes, we can recover to a consistent database state by examining the log and using one of the techniques described in Chapter 19.

1) Because the log contains a record of every write operation that changes the value of some database item, it is possible to undo the effect of these write operations of a transaction T by tracing backward through the log and resetting all items changed by a write operation of T to their old_values.

2) We can also redo the effect of the write operations of a transaction T by tracing forward through the log and setting all items changed by a write operation of T (that did not get done permanently) to their new_values.

Commit Point of a Transaction:

1) Definition a Commit Point:
   - A transaction T reaches its commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database has been recorded in the log.

2) Roll Back of transactions:
   - Needed for transactions that have a [start_transaction,T] entry into the log but no commit entry [commit,T] into the log.

3) Redoing transactions:
   - Transactions that have written their commit entry in the log must also have recorded all their write operations in the log; otherwise they would not be committed, so their effect on the database can be redone from the log entries.

4) Force writing a log:

- Before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk.
- This process is called force-writing the log file before committing a transaction.

Desirable Properties of Transactions

ACID properties:

1) Atomicity: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
2) Consistency preservation: A correct execution of the transaction must take the database from one consistent state to another.
3) Isolation: A transaction should not make its updates visible to other transactions until it is committed; this property, when enforced strictly, solves the temporary update problem and makes cascading rollbacks of transactions unnecessary (see Chapter 21).
4) Durability or permanency: Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

Characterizing Schedules Based on Recoverability

- Transaction schedule or history:
  - When transactions are executing concurrently in an interleaved fashion, the order of execution of operations from the various

transactions forms what is known as a transaction schedule (or history).

- A schedule (or history) S of n transactions T1, T2, …, Tn:
  - It is an ordering of the operations of the transactions subject to the constraint that, for each transaction Ti that participates in S, the operations of T1 in S must appear in the same order in which they occur in T1.
  - Note, however, that operations from other transactions Tj can be interleaved with the operations of Ti in S.

Schedules classified on recoverability:

1) Recoverable schedule:
   - One where no transaction needs to be rolled back.
   - A schedule S is recoverable if no transaction T in S commits until all transactions T' that have written an item that T reads have committed.
2) Cascadeless schedule:
   - One where every transaction reads only the items that are written by committed transactions.
3) Schedules requiring cascaded rollback:
   - A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back.
4) Strict Schedules:
   - A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed.

Characterizing Schedules Based on Serializability

1) Serial schedule:
   - A schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule.
   - Otherwise, the schedule is called nonserial schedule.
2) Serializable schedule:
   - A schedule S is serializable if it is equivalent to some serial schedule of the same n transactions.
3) Result equivalent:
   - Two schedules are called result equivalent if they produce the same final state of the database.
4) Conflict equivalent:
   - Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.

5) **Conflict serializable:**
   - A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S'.
6) **Being serializable is not the same as being serial**
7) **Being serializable implies that the schedule is a correct schedule.**
   - It will leave the database in a consistent state.
   - The interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution.
8) **Serializability is hard to check.**
   - Interleaving of operations occurs in an operating system through some scheduler
   - Difficult to determine beforehand how the operations in a schedule will be interleaved.

Practical approach:

- Come up with methods (protocols) to ensure serializability.
- It's not possible to determine when a schedule begins and when it ends.
  - Hence, we reduce the problem of checking the whole schedule to checking only a committed project of the schedule (i.e. operations from only the committed transactions.)
- Current approach used in most DBMSs:
  - Use of locks with two phase locking
- View equivalence:
  - A less restrictive definition of equivalence of schedules
- View serializability:
  - Definition of serializability based on view equivalence.
  - A schedule is view serializable if it is view equivalent to a serial schedule.

The premise behind view equivalence:

- As long as each read operation of a transaction reads the result of the same write operation in both schedules, the write operations of each transaction must produce the same results.
- "The view": the read operations are said to see the same view in both schedules.

Relationship between view and conflict equivalence:

- The two are same under constrained write assumption which assumes that if T writes X, it is constrained by the value of X it read; i.e., new X = f(old X)

- Conflict serializability is stricter than view serializability. With unconstrained write (or blind write), a schedule that is view serializable is not necessarily conflict serializable.
- Any conflict serializable schedule is also view serializable, but not vice versa.
- Consider the following schedule of three transactions
  T1: r1(X), w1(X); T2: w2(X); and T3: w3(X):
- Schedule Sa: r1(X); w2(X); w1(X); w3(X); c1; c2; c3;
- In Sa, the operations w2(X) and w3(X) are blind writes, since T1 and T3 do not read the value of X.
- Sa is view serializable, since it is view equivalent to the serial schedule T1, T2, T3.
- However, Sa is not conflict serializable, since it is not conflict equivalent to any serial schedule.

Testing for conflict serializability: Algorithm 21.1:

- Looks at only read_Item (X) and write_Item (X) operations
- Constructs a precedence graph (serialization graph) - a graph with directed edges
- An edge is created from Ti to Tj if one of the operations in Ti appears before a conflicting operation in Tj
- The schedule is serializable if and only if the precedence graph has no cycles.

- تم بحمد الله ..