18.4. Discuss the different algorithms for implementing each of the following relational operators and the circumstances under which each algorithm can be used: SELECT, JOIN, PROJECT, UNION, INTERSECT, SET DIFFERENCE, CARTESIAN PRODUCT.

SELECT - a search algorithm is needed for selecting records from a file (file scans). This could be implemented as *linear search*, where every record is retrieved and tested on whether it satisfies the select condition; *binary search*, where if the select is ordered by some key attribute, this can be used; *primary index*, where if the select is choosing an index for a key attribute (ex. Ssn = '123456789'), an indexing method can be used to retrieve the record; *hash key*, where if the select is choosing a hash key for a key attribute, such as the ssn example above, a hashing method can be used; *primary index for multiple records*, where if the comparison condition is >, >=, < or <= on a key attribute, use the index to find the record with the corresponding equality (ex. > would find a record with =), then retrieve all previous or subsequent records; *clustering index to retrieve multiple records*, where if the select involves an equality on a nonkey attribute with a clustering index (ex. Dno = 5), use the index to retrieve all records with that condition; *secondary (B+-tree) index on an equality comparison*, which can be used to retrieve a single record if the indexing field is a key, or multiple if not a key, or used with comparisons. It is useful for range queries; *bitmap index*, where if the select involves a set of values for an attribute, the bitmaps for each value can be OR-ed to give the set of records; *functional index*, where the selection involves a functional expression, an index can be created that is used to retrieve the matching records.

Conjunctive Select - if a select is made up of several simple conditions connected with AND, the DBMS can use *individual index*, where if an attribute used in any single simple condition has a path that allows for use of any of *binary search, primary index, hash key, clustering index,* or *secondary index*, use that to retrieve the records and then check that they match all the remaining simple conditions; *composite index*, where if an index has been created on a composite key, it can be used to index directly; *intersection of record pointers*, where if secondary indexes or other paths of access are available on more than one of the fields of simple conditions, and if they contain record pointers rather than block pointers, then each index can be used to retrieve the set of record pointers for the corresponding condition, and the intersection of these gives the final select.

JOIN - Join can be implemented by *nested-loop* or *nested-block* join, where it is a brute force method that checks each record in the outer loop against each record in the inner loop for whether they satisfy the condition; *index-based nested-loop join,* where if an index or hash exists for one of the join attributes, it can get all the records in the outer loop, then match them to the records found using the index or hash; *sort-merge join*, where if the records of the two sets are ordered by value of attributes A and B, both files can be scanned concurrently, matching records that have same values for A and B, the most efficient method; *hash-join*,

where the records of each set is partitioned into smaller files in a partitioning phase, where the records of A are hashed into buckets, then using the same function, the records of B are used to probe the bucket, combining with all matching records of R.

PROJECT - Can be implemented easily if attribute list contains a key relation of R, in which case the result of the operation will have the same number of tuples as R. If the attribute list does not contain a key relation of R, then duplicate tuples need to be deleted by sorting the results of the operation and deleting the duplicates.

Union, Intersection, Set Difference- These three set operations are type-compatible relations where each have the same number of attributes and same attribute domains. Sort-merge technique, is used to implement the operations. In Sort-merge technique, two relations are sorted on the same attributes and after sorting, a single scan through each relation can produce the result. To implement Union, first has the record of R and then hash the records of S. Duplicate records are not inserted into records. To implement Intersection, partition the records of R to hash file. Then while hashing each records of S, probe to check id an identical record from R is found in the bucket and if so, add record to result file. To implement set difference, first hash the record of R to the hash file buckets. While hashing, S, if an identical record is found for S, it is removed from the bucket.

Cartesian Product- The Cartesian product operation is an operation like R X S. It is expensive because the result includes records from both R and S and the attributes of result includes attributes form both R and S. When doing the Cartesian operation, if R have attributes and n,j and S have attributes m and k, the results will be n*m and j+k attributes. Instead of using Cartesian product, it is recommended to use join during query optimization.

17.1
Define the following terms: indexing field, primary key field, clustering field, secondary key field, block anchor, dense index, and nondense (sparse) index.

- **Indexing Field**- For a file with a given record structure consisting of several fields (or attributes), an index access structure is usually defined on a single field of a file, called an indexing field (or indexing attribute). The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain records with that field value. The values in the index are ordered so that we can do a binary search on the Index

- **Primary key**- The primary index is an ordered file whose records are of fixed length with two fields, and it acts like an access structure to efficiently search for and access the data records in a data file. The first field is of the same data type as the ordering key field is called the **primary key** of the data file, and the second field is a pointer to a disk block (a block address).

- **Clustering field**- If file records are physically ordered on a non-key field that field is called the **clustering field**. There is one entry in the clustering index for each distinct value of the clustering field, and it contains the value and a pointer to the first block in the data file that has a record with that value for its clustering field.

- **Secondary key field**- A secondary index access structure on a key (unique) field has a distinct value for every record. Such a field is called a **secondary key**.

- **Block anchor**- The first record in each block of the data file is called the **block anchor**.

- **Dense Index**- Indexes can be characterized as dense or sparse. A **dense index** has an index entry for every search key value in the data file.

- **Sparse Index**- **Sparse indexes** has index entries for only some of the search values. A sparse index has fewer entries than the number of records in the file.

Chapter 21

Question 1: What is the two-phase locking protocol? How does it guarantee serializability?

A two-phase locking protocol employs the technique of locking data items to prevent the multiple transactions frim accessing the item concurrently. In other words, all locking operations must precede the first unlock operation in the transaction.

It guarantees serializability by making it so that every transaction in a schedule follows the two-phase locking protocol. The locking protocol also has serializability, by enforcing two-phase locking rules.

Question 2: What are some variations of two-phases locking protocol? Why is strict or rigorous two-phase locking often preferred?

Some variations of the two-phase protocol include:

- **Basic -** the most basic version of the protocol, does not permit all possible serializable schedules.
- **Conservative -** requires a transaction to lock all the items it accessed before the transaction begins execution, by pre declaring it's read and write sets.
- **Strict -** guaranteed strict schedules. A transaction does not release any exclusive locks, writes, until after it commits or aborts.
- **Rigorous -** a more restrictive version of Strict. A transaction does not release any of its locks, exclusive or shared, until after commits or aborts.

They are often preferred because they allow more flexibility in what locks do not release until after program commits or aborts, meaning that the program can have more reliability compared to conservative and basic.

Question 3:
Deadlocks: when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set.
-Use a deadlock prevention protocol
-Transaction timestamp: cautious waiting
-Deadlock detection
-Timeouts

Starvation: when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally. It can also occur during victim selection, if the algorithm selects the same transaction as victim repeatedly
-First-come-first-served queue
-Allows some transactions to have priority over others, but increases priority of the transaction the longer it waits