

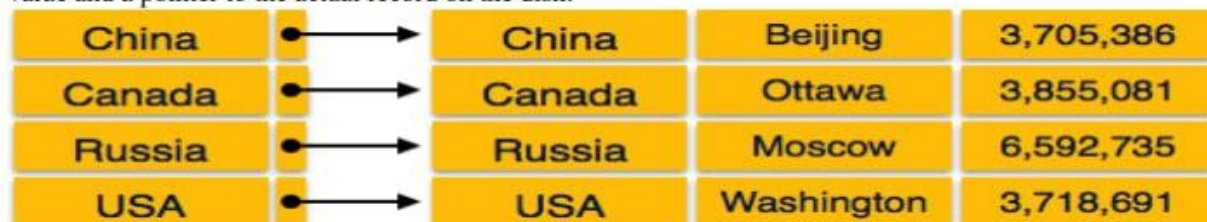
Rakib&ALfahaz

3b) When assessing the quality of an index, various evaluation metrics can be used to gauge its effectiveness and performance. These metrics help in understanding the index's ability to accurately represent the underlying market, its level of diversification, and its performance relative to a benchmark or other competing indices

A clustering index determines the physical order of the table's rows based on the values of one or more columns. It rearranges the table data to match the index, ensuring that the rows are stored in a specific order.

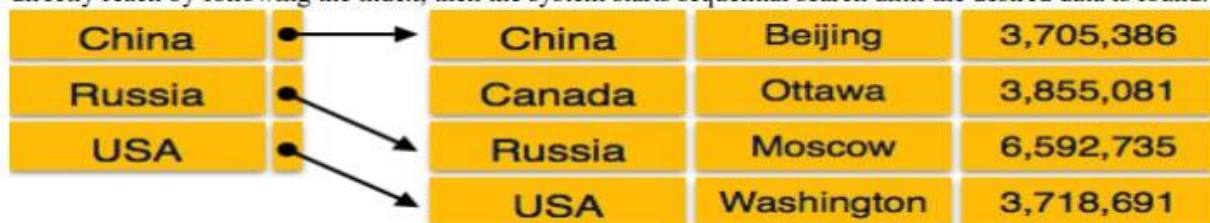
A non-clustering index does not dictate the physical order of the table's rows. It is a separate structure that provides a quick lookup mechanism to locate specific rows based on the indexed column(s).

Dense Index: In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.



China	→	China	Beijing	3,705,386
Canada	→	Canada	Ottawa	3,855,081
Russia	→	Russia	Moscow	6,592,735
USA	→	USA	Washington	3,718,691

Sparse Index: In sparse index, index records are not created for every search key. An **index record here contains a search key and an actual pointer to the data on the disk**. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.



China	→	China	Beijing	3,705,386
Russia	→	Canada	Ottawa	3,855,081
USA	→	Russia	Moscow	6,592,735
	→	USA	Washington	3,718,691

4a)

ACID Properties: A transaction is a very small unit of a program and it may contain several low-level tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability – commonly known as ACID properties – in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** – This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
- **Consistency** – The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** – The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** – In a database system where more than one transaction is being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed

as if it is the only transaction in the system. No transaction will affect the existence of any other transaction. *[More in the Following...]*

4b) Lock Based Protocols in DBMS is a mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock. Lock based protocols help to eliminate the concurrency problem in DBMS for simultaneous transactions by locking or isolating a particular transaction to a single user.

A lock is a data variable which is associated with a data item. This lock signifies that operations that can be performed on the data item. Locks in DBMS help synchronize access to the database items by concurrent transactions.

All lock requests are made to the concurrency-control manager. Transactions proceed only once the lock request is granted.

Starvation

Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.

Following are the reasons for Starvation:

When waiting scheme for locked items is not properly managed

In the case of resource leak

The same transaction is selected as a victim repeatedly

Deadlock

Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

Or

4a) The term "Phantom Problem" typically refers to a phenomenon that can occur in the context of database transactions, specifically when dealing with concurrency control. Concurrency control ensures that multiple transactions running simultaneously on a database do not interfere with each other and maintain data consistency.

The Phantom Problem specifically arises in situations where a transaction's result set is affected by the insertion or deletion of rows by other concurrent transactions. It is called the "Phantom Problem" because it seems as if new rows magically appear or disappear between successive reads within a transaction, like phantoms.

In the scenario you described, where the set of database objects is fixed and only the values of objects can be changed, the Phantom Problem does not apply. The Phantom Problem typically occurs when concurrent transactions can insert or delete rows, and these changes affect the result sets of other transactions. If the set of objects is fixed and cannot be modified in terms of structure (adding or deleting objects), then the Phantom Problem does not arise.

However, it's important to note that even in a scenario where only the values of objects can be changed, other concurrency-related issues such as dirty reads, non-repeatable reads, or lost updates may still occur. These issues pertain to inconsistent or incorrect data access due to concurrent transactions modifying the same data concurrently. Implementing appropriate concurrency control mechanisms like locks, isolation levels, or optimistic concurrency control can help mitigate such issues and maintain data consistency.

4b)

Write-Read Conflict:

Schedule:

T1: Read(X)

T1: Read(Y)

T1: Write(Y)

T2: Read(X)

T2: Read(Y)

In this schedule, transaction T1 writes object Y after reading it, and then transaction T2 reads the same object Y. This results in a write-read conflict, as T2 reads a value of Y that has been modified by T1.

Read-Write Conflict:

Schedule:

T1: Read(X)

T1: Read(Y)

T2: Read(X)

T2: Write(Y)

In this schedule, transaction T1 reads objects X and Y, and then transaction T2 reads object X. Afterward, T2 writes object Y. This results in a read-write conflict because T1 reads object Y before T2 writes to it.

Write-Write Conflict:

Schedule:

T1: Read(X)

T2: Read(X)

T1: Write(X)

T2: Write(X)

In this schedule, both transactions T1 and T2 read object X. Then, T1 writes to object X followed by T2 also writing to object X. This results in a write-write conflict, as both transactions modify the same object.

Strict 2PL (Two-Phase Locking) disallows the schedule by preventing conflicts. In Strict 2PL, a transaction must acquire all locks it needs during its execution and release them only after completing its transaction. To show that the given schedules are disallowed by Strict 2PL, we analyze each schedule:

In the first schedule, T1 writes to Y after reading it, but T2 attempts to read Y before T1 releases the lock. This violates the Strict 2PL rule, where a transaction

cannot read an object that has been modified by another uncommitted transaction.

In the second schedule, T1 reads X and Y, and T2 reads X. Then, T2 writes to Y, but T1 has not released its locks on X and Y yet. This violates the Strict 2PL rule, where a transaction cannot write to an object that has been read by another uncommitted transaction.

In the third schedule, both T1 and T2 read X, and then T1 writes to X followed by T2 writing to X. This violates the Strict 2PL rule, as a transaction cannot write to an object that has been written by another uncommitted transaction.

Strict 2PL enforces serializability by preventing conflicts between transactions, ensuring that a transaction's modifications are atomic and isolated from other transactions' operations.

5a)

States Diagram of Transactions: A transaction in a database can be in one of the following states –



- **Active** – In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** – When a transaction executes its final operation, it is said to be in a partially committed state.
- **Failed** – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
- **Aborted** – If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction abort –
 - Re-start the transaction
 - Kill the transaction
- **Committed** – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

Shadow copy technique is a technique which is based on making copies of the database.

→ Let us assume that only one transaction is active at a time.

→ A pointer called db-pointer always points to the current consistent copy of the database.

→ All updates ^{are} made on a shadow copy of the database, and db-pointer is made to point to the updated shadow copy only after the transaction reaches partial commit and all updated pages have been flushed to disk.

→ In case transaction fails, old consistent copy pointed to by db-pointer can be used and the shadow copy can be deleted.

These procedures ensure Atomicity, Consistency, Durability but not the Isolation property.

5b)

differences between log-based recovery and checkpoint-based recovery:

Approach: Log-based recovery relies on recording individual transaction actions in a log, while checkpoint-based recovery periodically captures the database's state.

Recovery process: Log-based recovery uses the transaction log to undo or redo operations, while checkpoint-based recovery uses checkpoints to reduce the amount of work required during recovery.

Granularity: Log-based recovery provides a more fine-grained recovery mechanism as it records individual actions, whereas checkpoint-based recovery offers a coarser recovery mechanism based on checkpoints.

Overhead: Log-based recovery incurs additional overhead due to the need to write and manage the transaction log, while checkpoint-based recovery incurs overhead during the checkpoint creation process.

Use cases: Log-based recovery is commonly used in systems requiring high durability and transaction integrity. Checkpoint-based recovery is suitable for systems with large databases where frequent checkpoints can help reduce recovery time.

lock based recovery

Example: Consider a transaction that transfers funds from one bank account to another. In log-based recovery, the transaction's actions (debiting one account and crediting another) are recorded in the transaction log. If a system failure occurs before the transaction completes, the log can be used during recovery to undo the credit operation and redo the debit operation, ensuring that the transfer is properly recorded or rolled back.

checkpoints

Example: Suppose a database takes periodic checkpoints every hour. If a system failure occurs, the recovery process can start from the most recent checkpoint, as it represents a known consistent state of the database. The system then applies the transactions from the transaction log since the last checkpoint to bring the database up to date.

