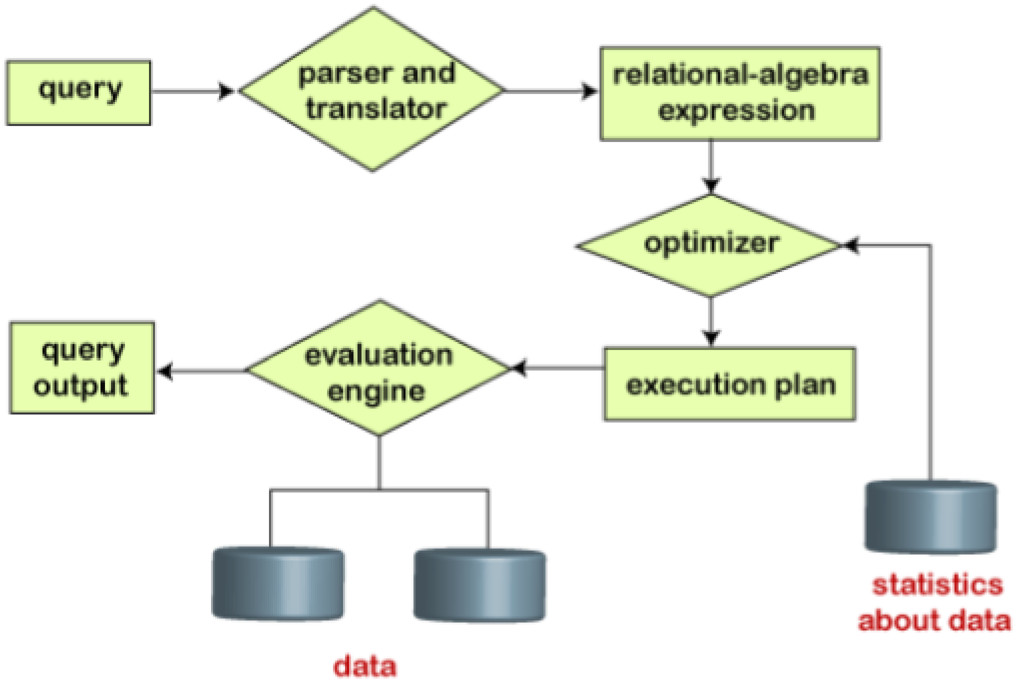
QUERY PROCESSING IN DBMS.

Query Processing is the activity performed in extracting data from the database. In query processing, it takes various steps for fetching the data from the database. The steps involved are:

- Parsing and translation
- Optimization
- Evaluation



Steps in query processing

The query processing works in the following way:

Parsing and Translation

- The scanning, parsing, and validating module produces an internal representation of the query. The query optimizer module devises an execution plan which is the execution strategy to retrieve the result of the query from the database files.
- A query typically has many possible execution strategies differing in performance, and the process of choosing a reasonably efficient one is known as query optimization.
- The code generator generates the code to execute the plan. The runtime database processor runs the generated code to produce the query result.
- Relational algebra is well suited for the internal representation of a query.

The translation process in query processing is similar to the parser of a query. When a user executes any query, for generating the internal form of the query, the parser in the system checks the syntax of the query, verifies the name of the relation in the database, the tuple, and finally the required attribute value. The parser creates a tree of the query, known as 'parse-tree.' Further, translate it into the form of relational algebra. With this, it evenly replaces all the use of the views when used in the query.

It is done in the following steps:

Step-1:

Parser: During parse call, the database performs the following checks- Syntax check, Semantic check and Shared pool check, after converting the query into relational algebra.

Parser performs the following checks as (refer detailed diagram):

1. **Syntax check** – concludes SQL syntactic validity.
   Example: SELECT * FORM employee
   Here error of wrong spelling of FROM is given by this check.
2. **Semantic check** – determines whether the statement is meaningful or not. Example: query contains a table name which does not exist is checked by this check.
3. **Shared Pool check** – Every query possess a hash code during its execution. So, this check determines existence of written hash code in shared pool if code exists in shared pool then database will not take additional steps for optimization and execution.

Hard Parse and Soft Parse –

If there is a fresh query and its hash code does not exist in shared pool then that query has to pass through from the additional steps known as hard parsing otherwise if hash code exists then query does not passes through additional steps. It just passes directly to execution engine (refer detailed diagram). This is known as soft parsing.

Hard Parse includes following steps – Optimizer and Row source generation.

Step-2:

Optimizer: During optimization stage, database must perform a hard parse atleast for one unique DML statement and perform optimization during this parse. This database never optimizes DDL unless it includes a DML component such as subquery that require optimization.

It is a process in which multiple query execution plan for satisfying a query are examined and most efficient query plan is satisfied for execution. Database catalog stores the execution plans and then optimizer passes the lowest cost plan for execution.

Step-3:

Execution Engine: Finally runs the query and display the required result.

Thus, we can understand the working of a query processing in the below-described diagram:

Suppose a user executes a query. As we have learned that there are various methods of extracting the data from the database. In SQL, a user wants to fetch the records of the employees whose salary is greater than or equal to 10000. For doing this, the following query is undertaken:

SELECT EMP_NAME FROM EMPLOYEE WHERE SALARY>10000;

Thus, to make the system understand the user query, it needs to be translated in the form of relational algebra. We can bring this query in the relational algebra form as:

- **σsalary>10000 (πEmp_Name(Employee))**
- **πEmp_Name(σsalary>10000 (Employee))**

After translating the given query, we can execute each relational algebra operation by using different algorithms. So, in this way, a query processing begins its working.

Evaluation

For this, with addition to the relational algebra translation, it is required to annotate the translated relational algebra expression with the instructions used for specifying and evaluating each operation. Thus, after translating the user query, the system executes a query evaluation plan.

Query Evaluation Plan

- In order to fully evaluate a query, the system needs to construct a query evaluation plan.
- A query evaluation plan defines a sequence of primitive operations used for evaluating a query. The query evaluation plan is also referred to as **the query execution plan**.
- A **query execution engine** is responsible for generating the output of the given query. It takes the query execution plan, executes it, and finally makes the output for the user query.

Optimization

- The cost of the query evaluation can vary for different types of queries. Although the system is responsible for constructing the evaluation plan, the user does need not to write their query efficiently.

- Usually, a database system generates an efficient query evaluation plan, which minimizes its cost. This type of task performed by the database system and is known as Query Optimization.
- For optimizing a query, the query optimizer should have an estimated cost analysis of each operation. It is because the overall operation cost depends on the memory allocations to several operations, execution costs, and so on.

Finally, after selecting an evaluation plan, the system evaluates the query and produces the output of the query.

Example:

SELECT LNAME, FNAME FROM EMPLOYEE WHERE SALARY > (SELECT MAX (SALARY) FROM EMPLOYEE WHERE DNO=5);

The inner block

(SELECT MAX (SALARY) FROM EMPLOYEE WHERE DNO=5)

 Translated in: ∏ MAX SALARY (σDNO=5(EMPLOYEE)) The Outer block

SELECT LNAME, FNAME FROM EMPLOYEE WHERE SALARY > C

 Translated in: ∏ LNAZME, FNAME (σSALARY>C (EMPLOYEE)) (C represents the result returned from the inner block.)

- The query optimizer would then choose an execution plan for each block.
- The inner block needs to be evaluated only once. (Uncorrelated nested query).
- It is much harder to optimize the more complex correlated nested queries.

Query Evaluation Plans

A query evaluation plan (Oracle calls it an "execution plan") is a program for an abstract machine (interpreter) inside the DBMS.

It is produced by the query optimizer.

Another name is "access plan" (the DBMS has to decide how to access the rows, e.g. whether to use an index).

In most systems, query evaluation plans (QEPs) are similar to relational algebra expressions (very system dependent).

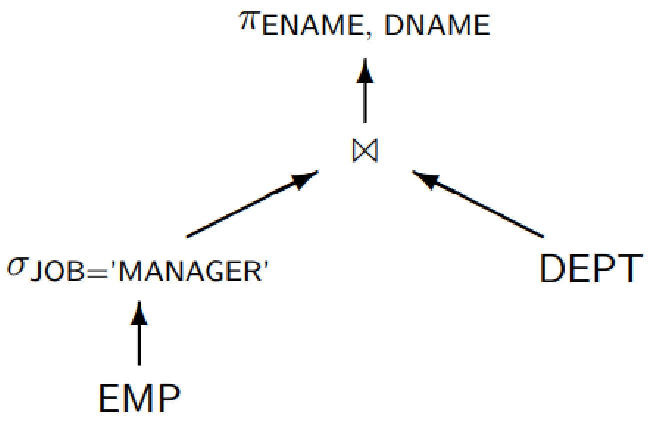
E.g. consider the following SQL query:

SELECT ENAME, DNAME FROM EMP, DEPT WHERE EMP.DEPTNO = DEPT.DEPTNO AND JOB = 'MANAGER'

In relational algebra, the example query would be written as:

πENAME, DNAME σJOB='MANAGER'(EMP) ✶ DEPT

Complex relational algebra expressions are best displayed as "operator trees" (showing the flow of data)



Transaction Processing Systems

Transactions

A transaction is a program including a collection of database operations, executed as a logical unit of data processing.

The operations performed in a transaction include one or more of database operations like insert, delete, update or retrieve data.

It is an atomic process that is either performed into completion entirely or is not performed at all.

A transaction involving only data retrieval without any data update is called **read-only** transaction.

Each high level operation can be divided into a number of low level tasks or operations. For example, a data update operation can be divided into three tasks −

- **read_item()** − reads data item from storage to main memory.
- **modify_item()** − change value of item in the main memory.
- **write_item()** − write the modified value from main memory to storage.

Database access is restricted to read_item() and write_item() operations. Likewise, for all transactions, read and write forms the basic database operations.

Transaction Operations

The low level operations performed in a transaction are −

- **begin_transaction** − A marker that specifies start of transaction execution.
- **read_item or write_item** − Database operations that may be interleaved with main memory operations as a part of transaction.
- **end_transaction** − A marker that specifies end of transaction.
- **commit** − A signal to specify that the transaction has been successfully completed in its entirety and will not be undone.
- **rollback** − A signal to specify that the transaction has been unsuccessful and so all temporary changes in the database are undone. A committed transaction cannot be rolled back.

Transaction States

A transaction may go through a subset of five states, active, partially committed, committed, failed and aborted.

- **Active** − The initial state where the transaction enters is the active state. The transaction remains in this state while it is executing read, write or other operations.

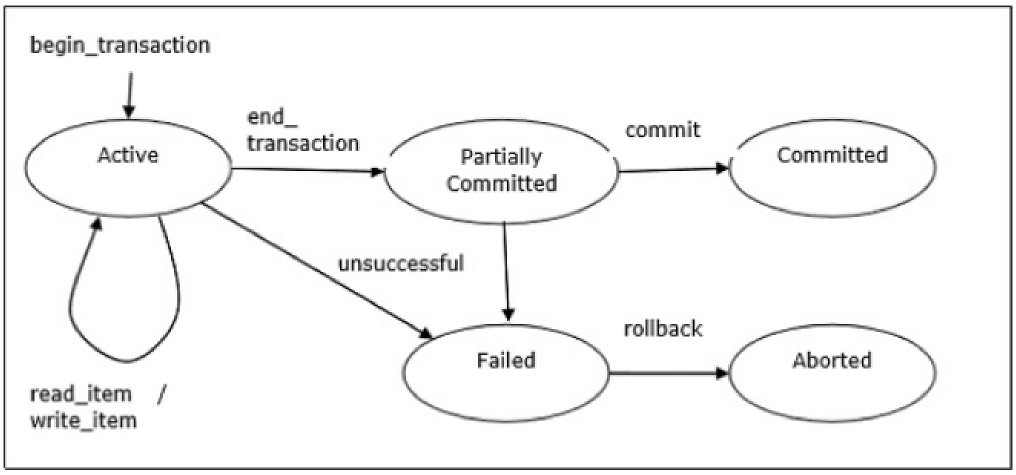
- **Partially Committed** − The transaction enters this state after the last statement of the transaction has been executed.

- **Committed** − The transaction enters this state after successful completion of the transaction and system checks have issued commit signal.

- **Failed** − The transaction goes from partially committed state or active state to failed state when it is discovered that normal execution can no longer proceed or system checks fail.

- **Aborted** − This is the state after the transaction has been rolled back after failure and the database has been restored to its state that was before the transaction began.

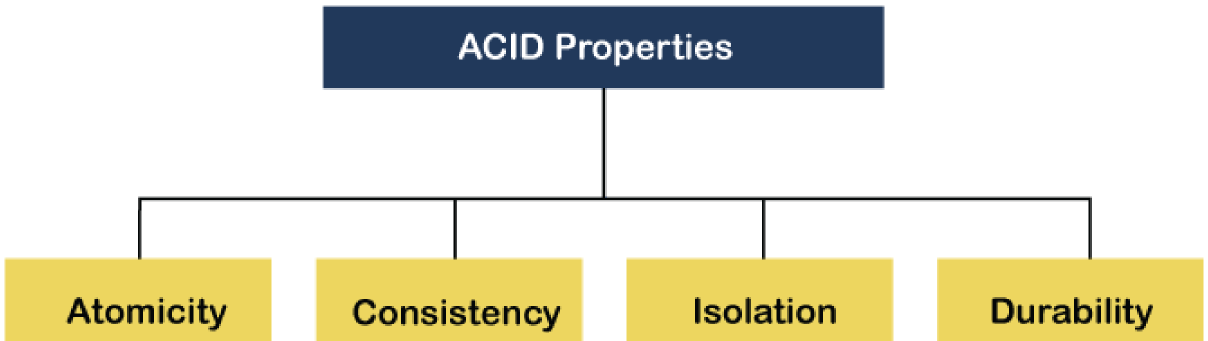
The following state transition diagram depicts the states in the transaction and the low level transaction operations that causes change in states.



Desirable Properties of Transactions

Any transaction must maintain the ACID properties, viz. Atomicity, Consistency, Isolation, and Durability.

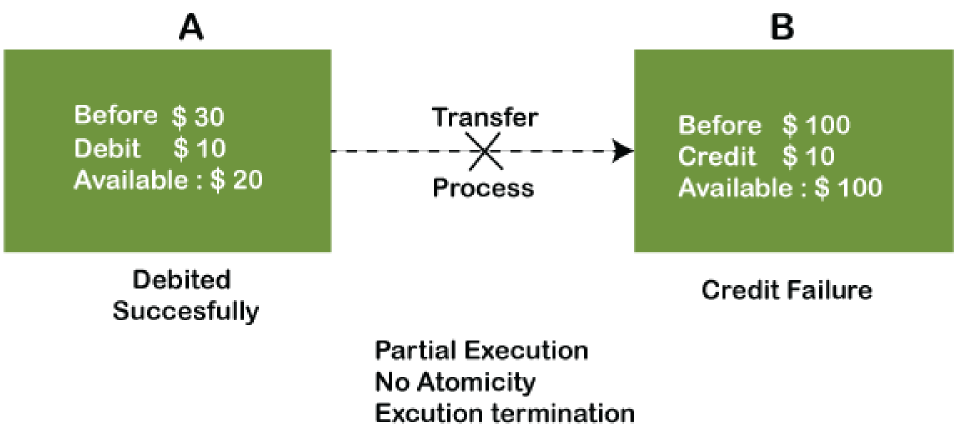
ACID Properties
The expansion of the term ACID defines for:



) Atomicity

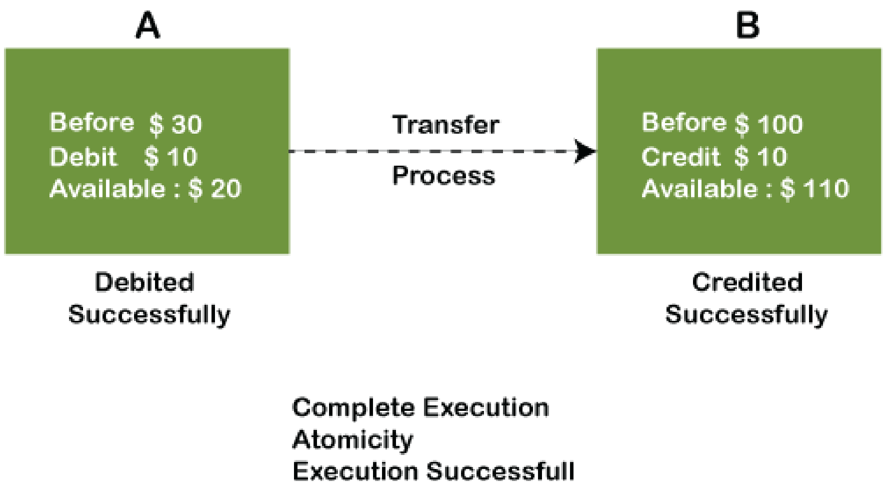
The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

Example: If Rahim has account A having $30 in his account from which he wishes to send $10 to Rama's account, which is B. In account B, a sum of $ 100 is already present. When $10 will be transferred to account B, the sum will become $110. Now, there will be two operations that will take place. One is the amount of $10 that Rahim wants to transfer will be debited from his account A, and the same amount will get credited to account B, i.e., into Rama's account. Now, what happens - the first operation of debit executes successfully, but the credit operation, however, fails. Thus, in Rahim's account A, the value becomes $20, and to that of Rama's account, it remains $100 as it was previously present.



In the above diagram, it can be seen that after crediting $10, the amount is still $100 in account B. So, it is not an atomic transaction.

The below image shows that both debit and credit operations are done successfully. Thus the transaction is atomic.



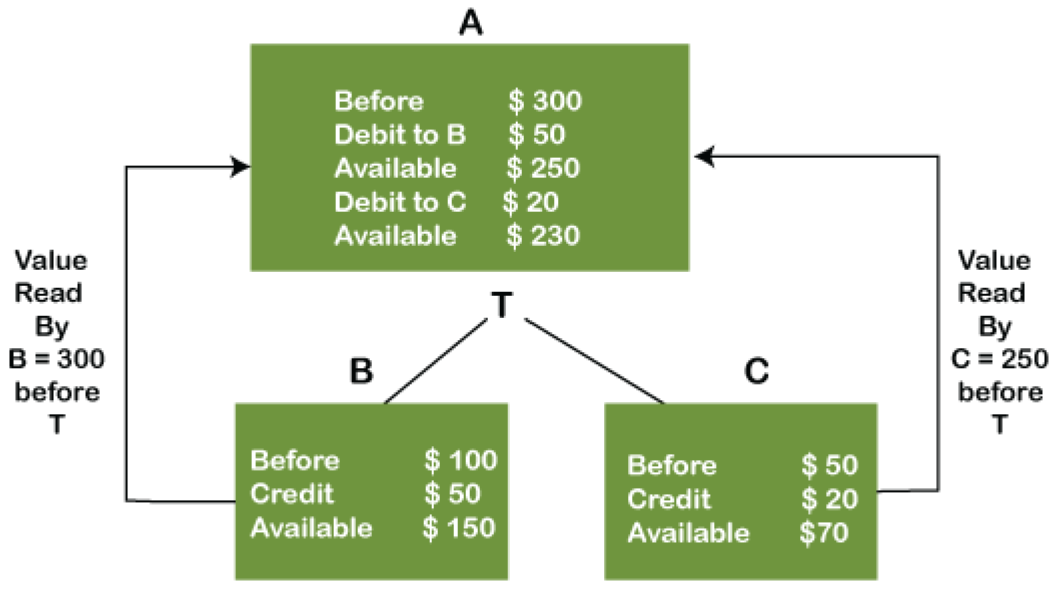
Thus, when the amount loses

atomicity, then in the bank systems, this becomes a huge issue, and so the atomicity is the main focus in the bank systems.

2) Consistency

The word **consistency** means that the value should remain preserved always. In DBMS, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

Example:



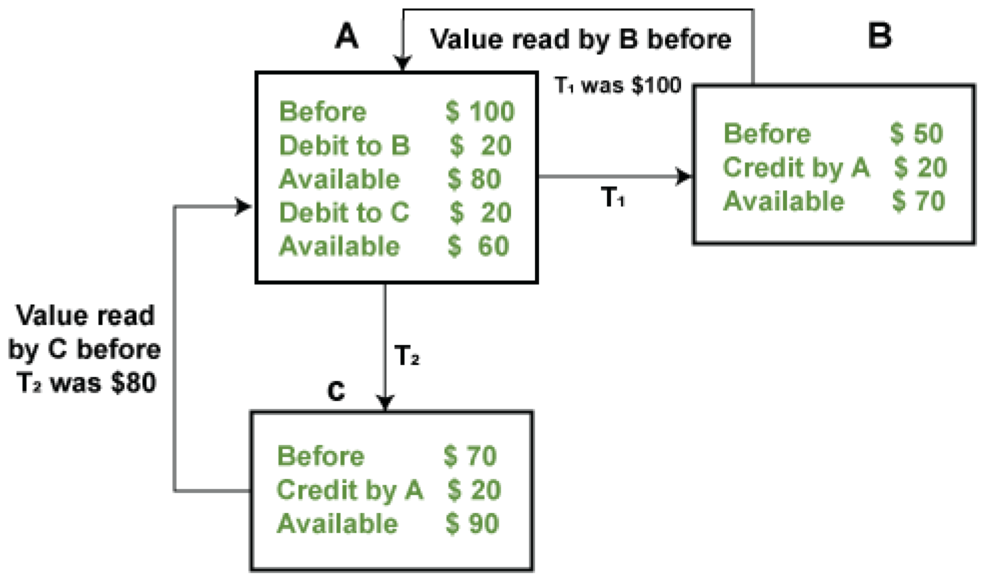
Data Consistent

In the above figure, there are three accounts, A, B, and C, where A is making a transaction T one by one to both B & C. There are two operations that take place, i.e., Debit and Credit. Account A firstly debits $50 to account B, and the amount in account A is read $300 by B before the transaction. After the successful transaction T, the available amount in B becomes $150. Now, A debits $20 to account C, and that time, the value read by C is $250 (that is correct as a debit of $50 has been successfully done to B). The debit and credit operation from account A to C has been done successfully. We can see that the transaction is done successfully, and the value is also read correctly. Thus, the data is consistent.
In case the value read by B and C is $300, which means that data is inconsistent because when the debit operation executes, it will not be consistent.

3) Isolation

The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.
Example: If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the

below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.



Isolation - Independent execution of T_1 & T_2 by A

4) Durability

Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

Thus, it was a precise introduction of ACID properties in DBMS. We have discussed these properties in the transaction section also.

Concurrency Control

Concurrency Control in Database Management System is a procedure of managing simultaneous operations without conflicting with each other.
It ensures that Database transactions are performed concurrently and accurately to produce correct results without violating data integrity of the respective Database.
Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another.
Though for any practical Database, it would have a mix of READ and WRITE operations and hence the concurrency is a challenge.

DBMS Concurrency Control is used to address such conflicts, which mostly occur with a multi-user system. Therefore, Concurrency Control is the most important element for proper functioning of a Database Management System where two or more database transactions are executed simultaneously, which require access to the same data.

Potential problems of Concurrency

Here, are some issues which you will likely to face while using the DBMS Concurrency Control method:

- **Lost Updates** occur when multiple transactions select the same row and update the row based on the value selected
- Uncommitted dependency issues occur when the second transaction selects a row which is updated by another transaction (**dirty read**)
- **Non-Repeatable Read** occurs when a second transaction is trying to access the same row several times and reads different data each time.
- **Incorrect Summary issue** occurs when one transaction takes summary over the value of all the instances of a repeated data-item, and second transaction update few instances of that specific data-item. In that situation, the resulting summary does not reflect a correct result.

Why use Concurrency method?

Reasons for using Concurrency control method is DBMS:

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues
- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability

Example

Assume that two people who go to electronic kiosks at the same time to buy a movie ticket for the same movie and the same show time.

However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control in DBMS, it is possible that both moviegoers will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Both moviegoers can still access information written in the movie seating database. But concurrency control only provides a ticket to the buyer who has completed the transaction process first.

Concurrency Control Protocols

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose. Following are the Concurrency Control techniques in DBMS:

- Lock-Based Protocols
- Two Phase Locking Protocol
- Timestamp-Based Protocols
- Validation-Based Protocols

Lock-based Protocols

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.

Binary Locks: A Binary lock on a data item can either locked or unlocked states.

Shared/exclusive: This type of locking mechanism separates the locks in DBMS based on their uses. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

1. Shared Lock (S):

A shared lock is also called a Read-only lock. With the shared lock, the data item can be shared between transactions. This is because you will never have permission to update data on the data item.

For example, consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

2. Exclusive Lock (X):

With the Exclusive Lock, a data item can be read as well as written. This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction. Transactions may unlock the data item after finishing the 'write' operation.

For example, when a transaction needs to update the account balance of a person. You can allows this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

3. Simplistic Lock Protocol

This type of lock-based protocols allows transactions to obtain a lock on every object before beginning operation. Transactions may unlock the data item after finishing the 'write' operation.

4. Pre-claiming Locking

Pre-claiming lock protocol helps to evaluate operations and create a list of required data items which are needed to initiate an execution process. In the situation when all locks are granted, the transaction executes. After that, all locks release when all of its operations are over.

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.

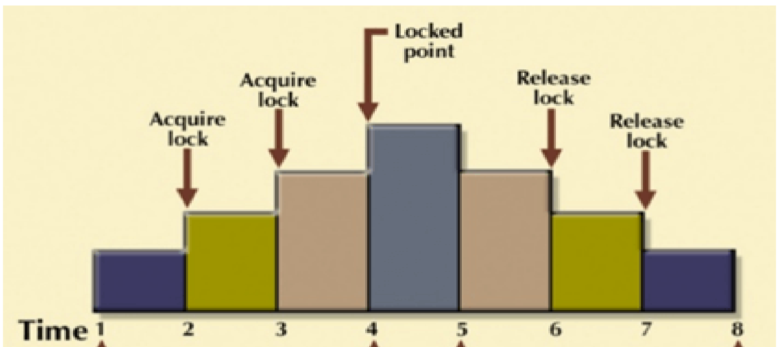
Two Phase Locking Protocol

Two Phase Locking Protocol also known as 2PL protocol is a method of concurrency control in DBMS that ensures serializability by applying a lock to the transaction data which blocks other transactions to access the same data simultaneously. Two Phase Locking protocol helps to eliminate the concurrency problem in DBMS.
This locking protocol divides the execution phase of a transaction into three different parts.

- In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
- The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.
- In this third phase, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.

The Two-Phase Locking protocol allows each transaction to make a lock or unlock request in two steps:

- **Growing Phase**: In this phase transaction may obtain locks but may not release any locks.



- **Shrinking Phase**: In this phase, a transaction may release locks but not obtain any new lock

It is true that the 2PL protocol offers serializability. However, it does not ensure that deadlocks do not happen.

In the above-given diagram, you can see that local and global deadlock detectors are searching for deadlocks and solve them with resuming transactions to their initial states.

Strict Two-Phase Locking Method

Strict-Two phase locking system is almost similar to 2PL. The only difference is that Strict-2PL never releases a lock after using it. It holds all the locks until the commit point and releases all the locks at one go when the process is over.

Centralized 2PL

In Centralized 2 PL, a single site is responsible for lock management process. It has only one lock manager for the entire DBMS.

Primary copy 2PL

Primary copy 2PL mechanism, many lock managers are distributed to different sites. After that, a particular lock manager is responsible for managing the lock for a set of data items. When the primary copy has been updated, the change is propagated to the slaves.

Distributed 2PL

In this kind of two-phase locking mechanism, Lock managers are distributed to all sites. They are responsible for managing locks for data at that site. If no data is replicated, it is equivalent to primary copy 2PL. Communication costs of Distributed 2PL are quite higher than primary copy 2PL

Timestamp-based Protocols

Timestamp based Protocol in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions. The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order. The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.

Lock-based protocols help you to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created.

Example:

Suppose there are there transactions T1, T2, and T3.
T1 has entered the system at time 0010
T2 has entered the system at 0020
T3 has entered the system at 0030
Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.

Advantages:

- Schedules are serializable just like 2PL protocols
- No waiting for the transaction, which eliminates the possibility of deadlocks!

Disadvantages:

Starvation is possible if the same transaction is restarted and continually aborted

Validation Based Protocol

Validation based Protocol in DBMS also known as Optimistic Concurrency Control Technique is a method to avoid concurrency in transactions. In this protocol, the local copies of the transaction data are updated rather than the data itself, which results in less interference while execution of the transaction.
The Validation based Protocol is performed in the following three phases:

1. Read Phase
2. Validation Phase
3. Write Phase

Read Phase

In the Read Phase, the data values from the database can be read by a transaction but the write operation or updates are only applied to the local data copies, not the actual database.

Validation Phase

In Validation Phase, the data is checked to ensure that there is no violation of serializability while applying the transaction updates to the database.

Write Phase

In the Write Phase, the updates are applied to the database if the validation is successful, else; the updates are not applied, and the transaction is rolled back.

Characteristics of Good Concurrency Protocol

An ideal concurrency control DBMS mechanism has the following objectives:

- Must be resilient to site and communication failures.
- It allows the parallel execution of transactions to achieve maximum concurrency.
- Its storage mechanisms and computational methods should be modest to minimize overhead.
- It must enforce some constraints on the structure of atomic actions of transactions.