UNIT-05

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 −

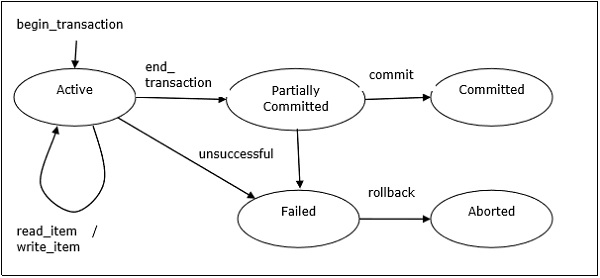
* **begin\_transaction** − A marker that specifies start of transaction execution.
* **read\_item or write\_item** − Database operations that may be interleaved with main memory operations as a part of transaction.
* **end\_transaction** − A marker that specifies end of transaction.
* **commit** − A 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.

* **Atomicity** − This property states that a transaction is an atomic unit of processing, that is, either it is performed in its entirety or not performed at all. No partial update should exist.
* **Consistency** − A transaction should take the database from one consistent state to another consistent state. It should not adversely affect any data item in the database.
* **Isolation** − A transaction should be executed as if it is the only one in the system. There should not be any interference from the other concurrent transactions that are simultaneously running.
* **Durability** − If a committed transaction brings about a change, that change should be durable in the database and not lost in case of any failure.

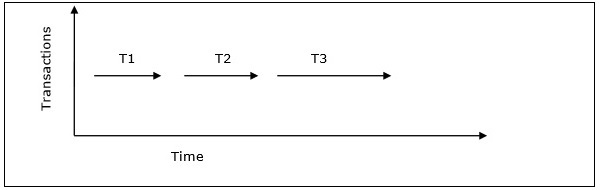
## Schedules and Conflicts

In a system with a number of simultaneous transactions, a **schedule** is the total order of execution of operations. Given a schedule S comprising of n transactions, say T1, T2, T3………..Tn; for any transaction Ti, the operations in Ti must execute as laid down in the schedule S.

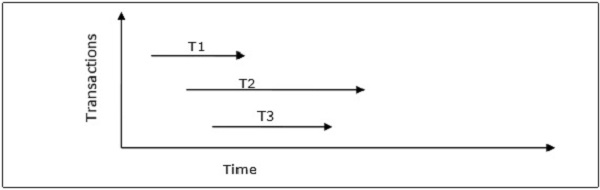
### Types of Schedules

There are two types of schedules −

* **Serial Schedules** − In a serial schedule, at any point of time, only one transaction is active, i.e. there is no overlapping of transactions. This is depicted in the following graph −



* **Parallel Schedules** − In parallel schedules, more than one transactions are active simultaneously, i.e. the transactions contain operations that overlap at time. This is depicted in the following graph −



### Conflicts in Schedules

In a schedule comprising of multiple transactions, a **conflict** occurs when two active transactions perform non-compatible operations. Two operations are said to be in conflict, when all of the following three conditions exists simultaneously −

* The two operations are parts of different transactions.
* Both the operations access the same data item.
* At least one of the operations is a write\_item() operation, i.e. it tries to modify the data item.

## Serializability

A **serializable schedule** of ‘n’ transactions is a parallel schedule which is equivalent to a serial schedule comprising of the same ‘n’ transactions. A serializable schedule contains the correctness of serial schedule while ascertaining better CPU utilization of parallel schedule.

### Equivalence of Schedules

Equivalence of two schedules can be of the following types −

* **Result equivalence** − Two schedules producing identical results are said to be result equivalent.
* **View equivalence** − Two schedules that perform similar action in a similar manner are said to be view equivalent.
* **Conflict equivalence** − Two schedules are said to be conflict equivalent if both contain the same set of transactions and has the same order of conflicting pairs of operations.

# Concurrency Control Techniques

[Concurrency control](https://www.geeksforgeeks.org/concurrency-control-in-dbms/) is provided in a database to:

* (i) enforce isolation among transactions.
* (ii) preserve database consistency through consistency preserving execution of transactions.
* (iii) resolve read-write and write-read conflicts.

Various concurrency control techniques are:

**1.** Two-phase locking Protocol

**2.** Time stamp ordering Protocol

**3.** Multi version concurrency control

**4.** Validation concurrency control

These are briefly explained below.

**1.**[Two-Phase Locking Protocol](https://www.geeksforgeeks.org/two-phase-locking-protocol/)**:**  
Locking is an operation which secures: permission to read, OR permission to write a data item. Two phase locking is a process used to gain ownership of shared resources without creating the possibility of deadlock.

The 3 activities taking place in the two phase update algorithm are:

**(i).** Lock Acquisition

**(ii).** Modification of Data

**(iii).** Release Lock

Two phase locking prevents deadlock from occurring in distributed systems by releasing all the resources it has acquired, if it is not possible to acquire all the resources required without waiting for another process to finish using a lock. This means that no process is ever in a state where it is holding some shared resources, and waiting for another process to release a shared resource which it requires. This means that deadlock cannot occur due to resource contention.

A transaction in the Two Phase Locking Protocol can assume one of the 2 phases:

* **(i) Growing Phase:**  
  In this phase a transaction can only acquire locks but cannot release any lock. The point when a transaction acquires all the locks it needs is called the Lock Point.
* **(ii) Shrinking Phase:**  
  In this phase a transaction can only release locks but cannot acquire any.

**2.**[Time Stamp Ordering Protocol](https://www.geeksforgeeks.org/timestamp-based-concurrency-control/)**:**  
A timestamp is a tag that can be attached to any transaction or any data item, which denotes a specific time on which the transaction or the data item had been used in any way. A timestamp can be implemented in 2 ways. One is to directly assign the current value of the clock to the transaction or data item. The other is to attach the value of a logical counter that keeps increment as new timestamps are required.

The timestamp of a data item can be of 2 types:

* **(i) W-timestamp(X):**  
  This means the latest time when the data item X has been written into.
* **(ii) R-timestamp(X):**  
  This means the latest time when the data item X has been read from. These 2 timestamps are updated each time a successful read/write operation is performed on the data item X.

**3. Multiversion Concurrency Control:**

Multiversion schemes keep old versions of data item to increase concurrency.

**Multiversion 2 phase locking:**  
Each successful write results in the creation of a new version of the data item written. Timestamps are used to label the versions. When a read(X) operation is issued, select an appropriate version of X based on the timestamp of the transaction.

**4.**[Validation Concurrency Control](https://www.geeksforgeeks.org/validation-based-protocol-in-dbms/)**:**  
The optimistic approach is based on the assumption that the majority of the database operations do not conflict. The optimistic approach requires neither locking nor time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through 2 or 3 phases, referred to as read, validation and write.

* **(i)** During read phase, the transaction reads the database, executes the needed computations and makes the updates to a private copy of the the database values. All update operations of the transactions are recorded in a temporary update file, which is not accessed by the remaining transactions.
* **(ii)** During the validation phase, the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to a write phase. If the validation test is negative, he transaction is restarted and the changes are discarded.
* **(iii)** During the write phase, the changes are permanently applied to the database.

In a local database system, for committing a transaction, the transaction manager has to only convey the decision to commit to the recovery manager. However, in a distributed system, the transaction manager should convey the decision to commit to all the servers in the various sites where the transaction is being executed and uniformly enforce the decision. When processing is complete at each site, it reaches the partially committed transaction state and waits for all other transactions to reach their partially committed states. When it receives the message that all the sites are ready to commit, it starts to commit. In a distributed system, either all sites commit or none of them does.

The different distributed commit protocols are −

* One-phase commit
* Two-phase commit
* Three-phase commit

Distributed One-phase Commit

Distributed one-phase commit is the simplest commit protocol. Let us consider that there is a controlling site and a number of slave sites where the transaction is being executed. The steps in distributed commit are −

* After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site.
* The slaves wait for “Commit” or “Abort” message from the controlling site. This waiting time is called **window of vulnerability**.
* When the controlling site receives “DONE” message from each slave, it makes a decision to commit or abort. This is called the commit point. Then, it sends this message to all the slaves.
* On receiving this message, a slave either commits or aborts and then sends an acknowledgement message to the controlling site.

Distributed Two-phase Commit

Distributed two-phase commit reduces the vulnerability of one-phase commit protocols. The steps performed in the two phases are as follows −

**Phase 1: Prepare Phase**

* After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site. When the controlling site has received “DONE” message from all slaves, it sends a “Prepare” message to the slaves.
* The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a “Ready” message.
* A slave that does not want to commit sends a “Not Ready” message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

**Phase 2: Commit/Abort Phase**

* After the controlling site has received “Ready” message from all the slaves −
  + The controlling site sends a “Global Commit” message to the slaves.
  + The slaves apply the transaction and send a “Commit ACK” message to the controlling site.
  + When the controlling site receives “Commit ACK” message from all the slaves, it considers the transaction as committed.
* After the controlling site has received the first “Not Ready” message from any slave −
  + The controlling site sends a “Global Abort” message to the slaves.
  + The slaves abort the transaction and send a “Abort ACK” message to the controlling site.
  + When the controlling site receives “Abort ACK” message from all the slaves, it considers the transaction as aborted.

Distributed Three-phase Commit

The steps in distributed three-phase commit are as follows −

**Phase 1: Prepare Phase**

The steps are same as in distributed two-phase commit.

**Phase 2: Prepare to Commit Phase**

* The controlling site issues an “Enter Prepared State” broadcast message.
* The slave sites vote “OK” in response.

**Phase 3: Commit / Abort Phase**

The steps are same as two-phase commit except that “Commit ACK”/”Abort ACK” message is not required.