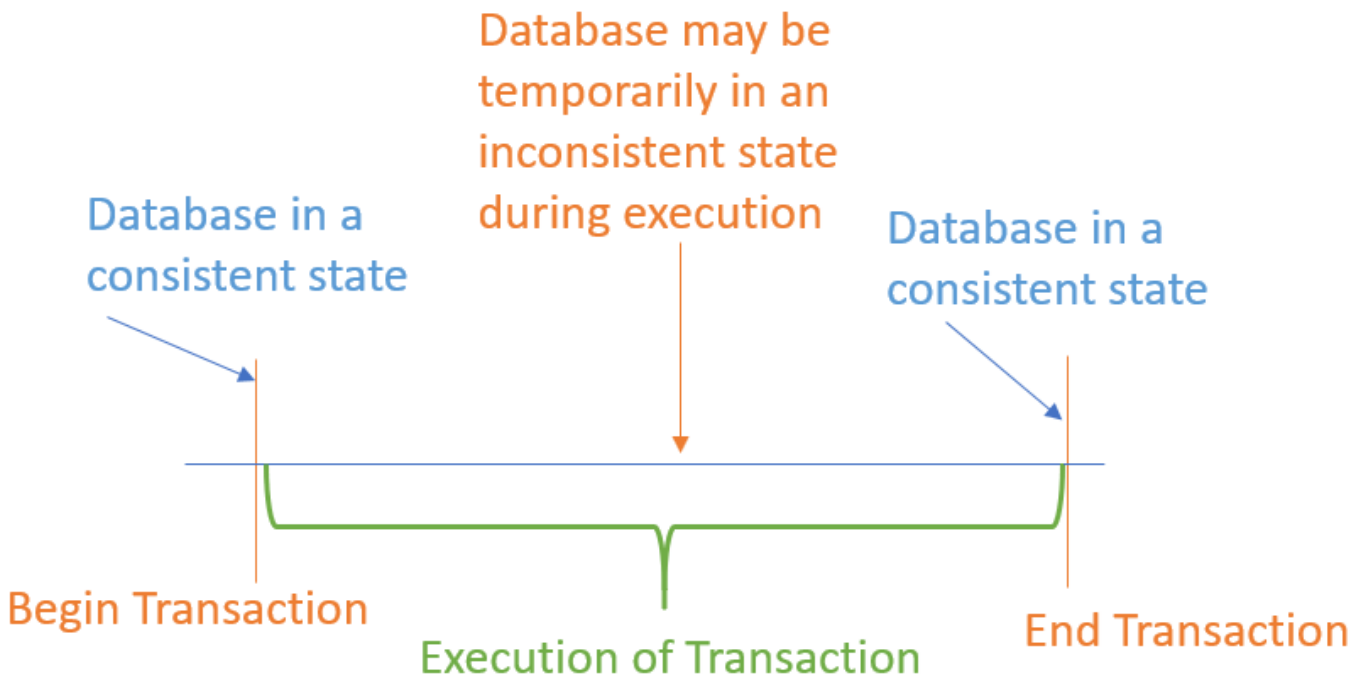
TRANSACTION MANAGEMENT

A **Database Transaction** is a logical unit of processing in a DBMS which entails one or more database access operation. They are set of operations used to perform a logical set of work. In a nutshell, database transactions represent real-world events of any enterprise.

One of the major uses of DBMS is to protect the user's data from system failures. It is done by ensuring that all the data is restored to a consistent state when the computer is restarted after a crash.

The transaction is any one execution of the user program in a DBMS. Executing the same program multiple times will generate multiple transactions.

All types of database access operation which are held between the beginning and end transaction statements are considered as a single logical transaction in DBMS. During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.



Facts about Database Transactions

- **A good DBMS should ensure that** reliability and consistency is maintained in the presence of failures of both hardware and software components, and when multiple users are accessing the database.
- A **transaction** is any action that reads from and/or writes to a database.
- A transaction may consist of a simple SELECT statement to generate a list of table contents; it may consist of a series of related UPDATE statements to change the values of

attributes in various tables; it may consist of a series of INSERT statements to add rows to one or more tables, or it may consist of a combination of SELECT, UPDATE, and INSERT statements.

- A transaction is a *logical* unit of work that must be entirely completed or entirely aborted; no intermediate states are acceptable.
- In other words, a multicomponent transaction, such as the previously mentioned sale, must not be partially completed. Updating only the inventory or only the accounts receivable is not acceptable.
- All of the SQL statements in the transaction must be completed successfully. If any of the SQL statements fail, the entire transaction is rolled back to the original database state that existed before the transaction started.
- A successful transaction changes the database from one consistent state to another. A **consistent database state** is one in which all data integrity constraints are satisfied.
- To ensure consistency of the database, every transaction must begin with the database in a known consistent state.
- If the database is not in a consistent state, the transaction will yield an inconsistent database that violates its integrity and business rules.
- Most real-world database transactions are formed by two or more database requests. A **database request** is the equivalent of a single SQL statement in an application program or transaction.
- For example, if a transaction is composed of two UPDATE statements and one INSERT statement, the transaction uses three database requests.
- In turn, each database request generates several input/output (I/O) operations that read from or write to physical storage media.

Concurrency in Transactions

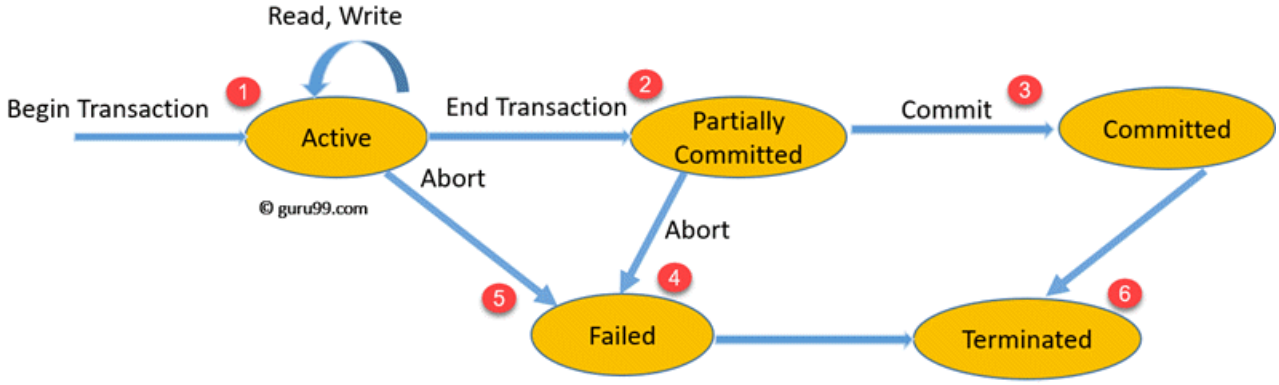
- A database is a shared resource accessed. It is used by many users and processes concurrently. For example, the banking system, railway, and air reservations systems, stock market monitoring, supermarket inventory, and checkouts,
- Not managing concurrent access may create issues like:
- Hardware failure and system crashes
- Concurrent execution of the same transaction, deadlock, or slow performance

States of Transactions

The various states of a transaction concept in DBMS are listed below:

State	Transaction types
Active State	A transaction enters into an active state when the execution process begins. During this

	state read or write operations can be performed.
Partially Committed	A transaction goes into the partially committed state after the end of a transaction.
Committed State	When the transaction is committed to state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently.
Failed State	A transaction considers failed when any one of the checks fails or if the transaction is aborted while it is in the active state.
Terminated State	State of transaction reaches terminated state when certain transactions which are leaving the system can't be restarted.



State Transition Diagram for a Database Transaction

Let's study a state transition diagram that highlights how a transaction moves between these various states.

- Once a transaction states execution, it becomes active. It can issue READ or WRITE operation.
- Once the READ and WRITE operations complete, the transactions becomes partially committed state.

- Next, some recovery protocols need to ensure that a system failure will not result in an inability to record changes in the transaction permanently. If this check is a success, the transaction commits and enters into the committed state.
- If the check is a fail, the transaction goes to the Failed state.
- If the transaction is aborted while it's in the active state, it goes to the failed state. The transaction should be rolled back to undo the effect of its write operations on the database.
- The terminated state refers to the transaction leaving the system.

Transaction Properties

- Each individual transaction must display *atomicity*, *consistency*, *isolation*, and *durability*. These properties are sometimes referred to as the ACID test.
- In addition, when executing multiple transactions, the DBMS must schedule the concurrent execution of the transaction's operations.
- The schedule of such transaction's operations must exhibit the property of *serializability*.

1. **Atomicity** requires that *all* operations (SQL requests) of a transaction be completed; if not, the transaction is aborted. If a transaction T1 has four SQL requests, all four requests must be successfully completed; otherwise, the entire transaction is aborted. In other words, a transaction is treated as a single, indivisible, logical unit of work.

- It's also known as 'all or nothing' property. A transaction is an indivisible unit that is either performed in its entirety or is not performed at all. It is the responsibility of the recovery subsystem of the DBMS to ensure atomicity.

2. **Consistency** indicates the permanence of the database's consistent state. A transaction takes a database from one consistent state to another consistent state. When a transaction is completed, the database must be in a consistent state; if any of the transaction parts violates an integrity constraint, the entire transaction is aborted.

- A transaction must transform the database from one consistent state to another consistent state. It is the responsibility of both the DBMS and the application developers to ensure consistency. The DBMS can ensure consistency by enforcing all the constraints that have been specified on the database schema, such as integrity and enterprise constraints. However, in itself this is insufficient to ensure consistency. For example, suppose we have a transaction that is intended to transfer money from one bank account to another and the programmer makes an error in the transaction logic and debits one account but credits the wrong account, then the database is in an inconsistent state. However, the DBMS would not have been responsible for introducing this inconsistency and would have had no ability to detect the error.

3. **Isolation** means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed. In other words, if a transaction T1 is being executed and is using the data item X, that data item cannot be accessed by any other transaction (T2 ... T*n*) until T1 ends. This property is particularly useful in multiuser database environments because several users can access and update the database at the same time.

- Transactions execute independently of one another. In other words, the partial effects of incomplete transactions should not be visible to other transactions. It is the responsibility of the concurrency control subsystem to ensure isolation.

4. **Durability** ensures that once transaction changes are done (committed), they cannot be undone or lost, even in the event of a system failure.
- The effects of a successfully completed (committed) transaction are permanently recorded in the database and must not be lost because of a subsequent failure. It is the responsibility of the recovery subsystem to ensure durability.

ACID Property in DBMS with example:

Below is an example of ACID property in DBMS:

Transaction 1: Begin X=X+50, Y = Y-50 END

Transaction 2: Begin X=1.1*X, Y=1.1*Y END

Transaction 1 is transferring $50 from account X to account Y.

Transaction 2 is crediting each account with a 10% interest payment.

If both transactions are submitted together, there is no guarantee that the Transaction 1 will execute before Transaction 2 or vice versa. Irrespective of the order, the result must be as if the transactions take place serially one after the other.

Types of Transactions

Based on Application areas

- Non-distributed vs. distributed
- Compensating transactions
- Transactions Timing
- On-line vs. batch

Based on Actions

- Two-step
- Restricted
- Action model

Based on Structure

- Flat or simple transactions: It consists of a sequence of primitive operations executed between a begin and end operations.
- Nested transactions: A transaction that contains other transactions.
- Workflow

What is a Schedule?

A Schedule is a process creating a single group of the multiple parallel transactions and executing them one by one. It should preserve the order in which the instructions appear in each transaction. If two transactions are executed at the same time, the result of one transaction may affect the output of other.

Example

Initial Product Quantity is 10
Transaction 1: Update Product Quantity to 50
Transaction 2: Read Product Quantity

If Transaction 2 is executed before Transaction 1, outdated information about the product quantity will be read. Hence, schedules are required.

Parallel execution in a database is inevitable. But, Parallel execution is permitted when there is an equivalence relation amongst the simultaneously executing transactions. This equivalence is of 3 Types.

Serializability

Serializability ensures that the schedule for the concurrent execution of the transactions yields consistent results.
This property is important in multiuser and distributed databases, where multiple transactions are likely to be executed concurrently.
Naturally, if only a single transaction is executed, serializability is not an issue.
A single-user database system automatically ensures serializability and isolation of the database because only one transaction is executed at a time.
The atomicity, consistency, and durability of transactions must be guaranteed by the single-user DBMSs.
Multiuser databases are typically subject to multiple concurrent transactions.
Therefore, the multiuser DBMS must implement controls to ensure serializability and isolation of transactions—in addition to atomicity and durability—to guard the database's consistency and integrity.
For example, if several concurrent transactions are executed over the same data set and the second transaction updates the database before the first transaction is finished, the isolation property is violated and the database is no longer consistent.
The DBMS must manage the transactions by using concurrency control techniques to avoid such undesirable situations.

RESULT EQUIVALENCE:

If two schedules display the same result after execution, it is called result equivalent schedule. They may offer the same result for some value and different results for another set of values. For example, one transaction updates the product quantity, while other updates customer details.

View Equivalence

View Equivalence occurs when the transaction in both the schedule performs a similar action. Example, one transaction inserts product details in the product table, while another transaction inserts product details in the archive table. The transaction is the same, but the tables are different.

CONFLICT Equivalence

In this case, two transactions update/view the same set of data. There is a conflict amongst transaction as the order of execution will affect the output.

What is Serializability?

Serializability is the process of search for a concurrent schedule who output is equal to a serial schedule where transaction ae execute one after the other. Depending on the type of schedules, there are two types of serializability:

- Conflict
- View

Summary:

- Transaction management is a logical unit of processing in a DBMS which entails one or more database access operation
- It is a transaction is a program unit whose execution may or may not change the contents of a database.
- Not managing concurrent access may create issues like hardware failure and system crashes.
- Active, Partially Committed, Committed, Failed & Terminate are important transaction states.
- The full form of ACID Properties in DBMS is Atomicity, Consistency, Isolation, and Durability
- Three DBMS transactions types are Base on Application Areas, Action, & Structure.
- A Schedule is a process creating a single group of the multiple parallel transactions and executing them one by one.
- Serializability is the process of search for a concurrent schedule whose output is equal to a serial schedule where transactions are executed one after the other.