Transaction Management

In a database management, a transaction represents a logical unit of work performed by a program or user during database execution. A transaction may involve reading data, updating data, or performing both operations together. What distinguishes a transaction from ordinary database commands is that all the operations within a transaction are treated as a single, indivisible unit. This means the database system must ensure that the transaction is completed correctly and safely, even when multiple users are accessing the database at the same time or when system failures occur.

To guarantee reliable transaction processing, every transaction in a DBMS must satisfy four fundamental properties known as ACID properties: atomicity, consistency, isolation, and durability. These properties ensure that database operations produce correct results and that the database remains trustworthy and are discussed below.

a) Atomicity

Atomicity ensures that a transaction is treated as a whole, and either all the operations in the transaction are successfully completed, or none of them are applied to the database. If a transaction contains several SQL statements and one of them fails, the DBMS aborts the entire transaction and reverses any partial changes. This prevents situations where only some updates are applied, which could leave the database in an incorrect or inconsistent state.

b) Consistency

Consistency ensures that a transaction moves the database from one valid state to another valid state. Before a transaction begins, the database must satisfy all defined rules and constraints, such as primary key constraints, foreign key constraints, and integrity rules. After the transaction completes, these rules must still be satisfied. If a transaction would violate any constraint, the DBMS does not allow it to commit.

c) Isolation

Isolation becomes especially important when multiple transactions are executing at the same time, a situation known as concurrent execution. In real-world systems, many users access the database simultaneously. For example, several students may register for courses at the same time, or multiple bank customers may perform transactions concurrently. Isolation ensures that each transaction behaves as though it is executing alone, even though many transactions may be running at the same time.

Without isolation, concurrent transactions could interfere with one another, leading to problems such as lost updates, dirty reads, or inconsistent results. The DBMS handles this by carefully controlling how transactions access shared data, ensuring that intermediate changes made by one transaction do not incorrectly affect others.

d) Durability

Durability ensures that once a transaction has been successfully committed, its changes become permanent. Even if the system crashes, power fails, or hardware problems occur immediately after the commit, the DBMS guarantees that the effects of the transaction will not be lost. This is typically achieved through logging and recovery mechanisms that allow the database to restore committed changes after a failure.

Transaction states

Transactions can assume any of the following states.

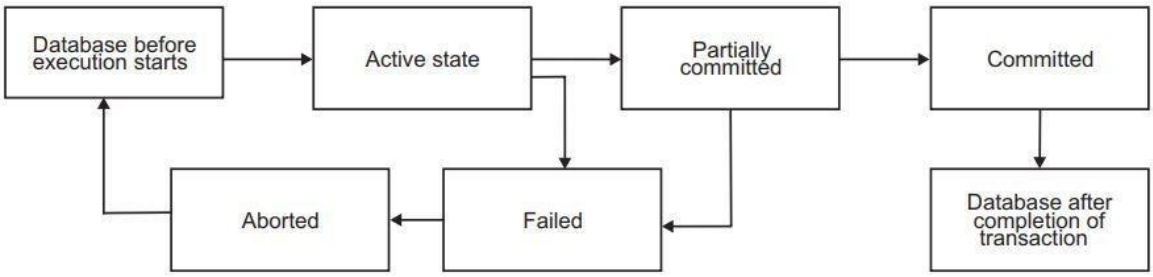

Fig 1: Transaction States

During execution, a transaction passes through several states. It begins in the active state while its SQL statements are being executed. Once all statements have been executed, the transaction enters a partially committed state, meaning it is ready to be finalized but not yet permanently recorded. If an error occurs at any point, the transaction moves into the failed state. From there, it is aborted, and all its changes are rolled back to restore the database to its previous consistent state. If everything completes successfully, the transaction reaches the committed state, at which point its changes are permanently saved.

The DBMS provides explicit support for transaction control through SQL commands such as COMMIT and ROLLBACK. When a COMMIT command is issued, all changes made during the transaction are permanently written to the database, and the transaction ends

successfully. When a ROLLBACK command is issued, all changes made during the transaction are undone, and the database returns to its earlier consistent state.