Transaction Models

As the transaction environment evolves from the centralized environment to distributed and mobile environments, the properties and the structure of transactions change. However, several basic transaction models are indispensable. In this section, we will review the following transaction models:

1. Flat transaction model.
2. Nested transaction model.
3. Multilevel transaction model.
4. Sagas transaction model.
5. Split and Join transaction model.

Flat Transaction Model

The flat transaction model presents the simplest transaction structure that fully meets the TPS properties. Figure 16 illustrates the structure of a flat transaction. The building block of a flat transaction, between Begin and Commit /Abort operations, contains all the database operations that are tightly coupled together as one atomic database operation. The flat transaction begins at one consistent database state, and either ends in another consistent state, i.e., the transaction commits, or remains in the same consistent state, i.e., the transaction aborts.

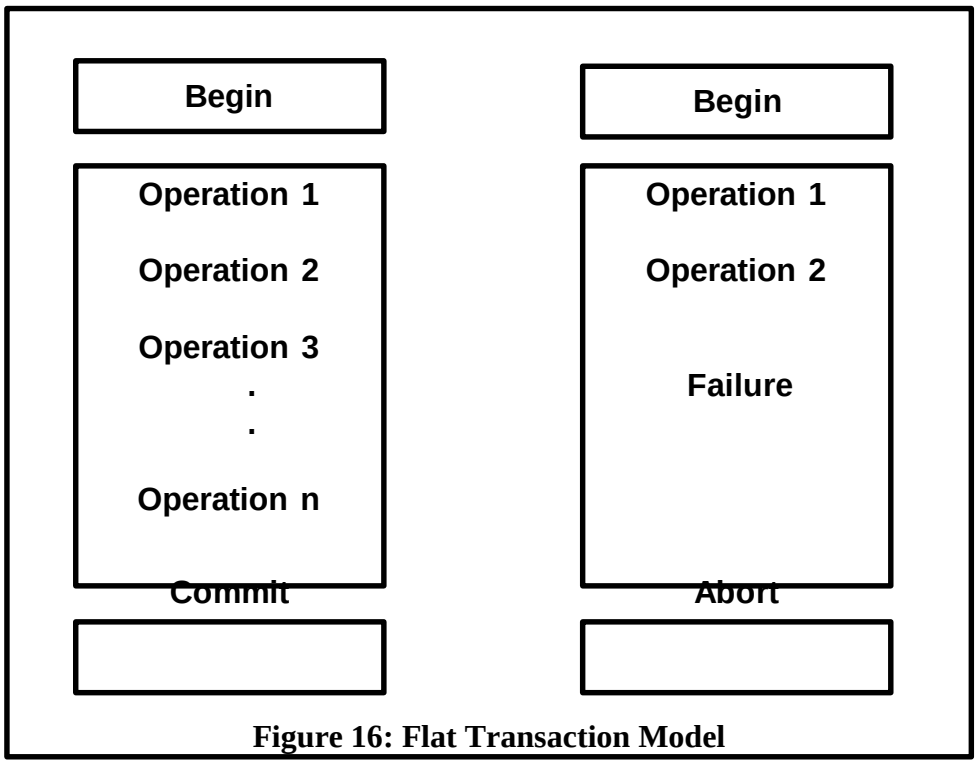


Figure 16: Flat Transaction Model

Nested Transaction Model

The nested transaction model defines the concepts and the mechanisms for breaking up the large building block of a flat transaction into a set of smaller transactions, called sub-transactions. Thus, the nested transaction model has a hierarchical tree structure that includes a top-level transaction and a set of sub-transactions (either parent or children transactions). Sub-transactions at the leaf level of the transaction tree are flat transactions.

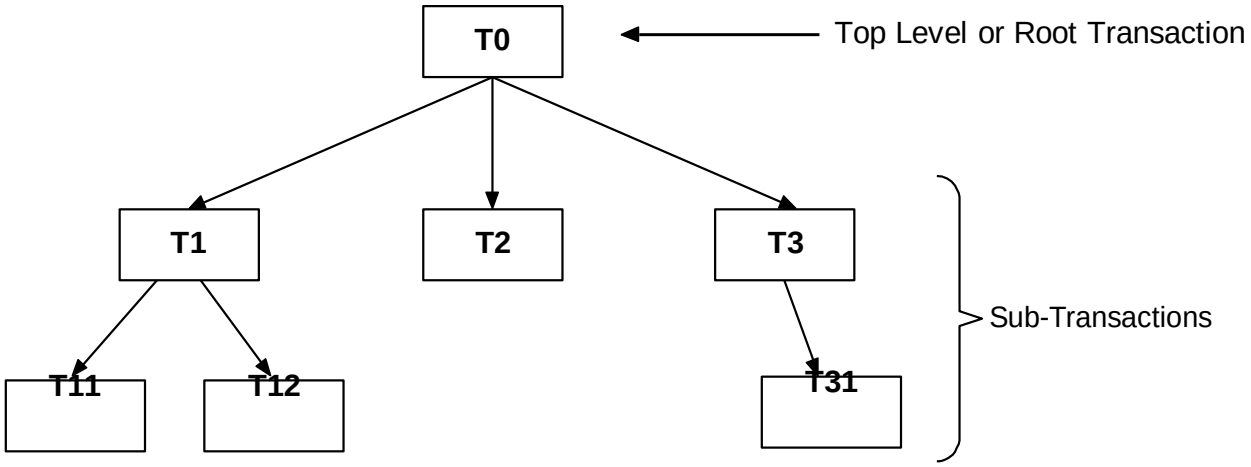


Figure 17: Nested Transaction Model

Multilevel Transaction Model

The multilevel transaction model is looser than the nested transaction model in terms of the relationship between parent and children transactions. Sub-transactions in the multilevel transaction can commit or abort independently of their parents. This is supported by the concepts of compensating transactions. We will briefly discuss the concept of compensating transactions, and its opposed contingency transactions (see Figure 18).

- Compensating Transactions are designed to undo the effect of the original transactions that have aborted. The compensating transactions are triggered and started when the original transactions fail. Otherwise, the compensating transactions are not initiated. Once a compensating transaction has started, it must commit. In other words, the compensating transactions can not abort. If a compensating transaction fails, it will be restarted.

- Contingency Transactions are designed to replace the task of the original transactions that have failed. Contingency transactions are also triggered by the failures of the original transactions. Note that it is not always possible to specify the compensating or contingency transactions for an original transaction.

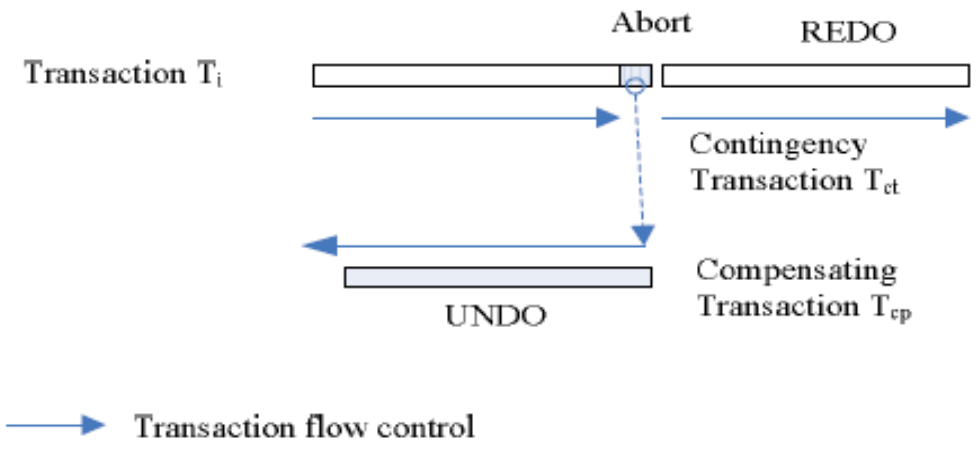
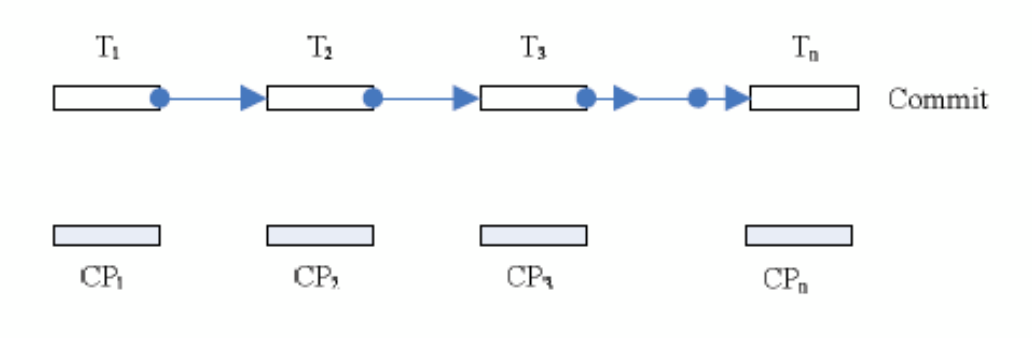


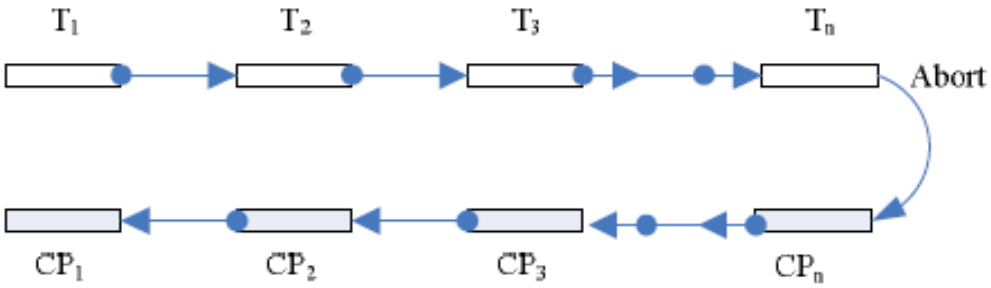
Figure 18: Compensating and Contingency Transactions

Sagas Transaction Model

The Sagas transaction model also makes use of the concept of compensating transactions to support transactions whose execution time is long. A Sagas transaction consists of a consecutive chain of flat transactions Si that can commit independently. For each flat transaction Si, there is a compensating transaction CPi that will undo the effect of the transaction Si if the transaction Si aborts. A compensating transaction CPi in the Sagas chain is triggered by the associated transaction Si or the compensating transaction CPi+1. If the Sagas transaction commits, no compensating transaction CPi is initiated (see Figure 19-a), otherwise the chain of compensating transactions is triggered (see Figure 19-b).



a : A successful Sagas



b: An unsuccessful Sagas

Figure 19: Successful and Unsuccessful Sagas

Split and Join Transaction Model

The Split and Join transaction model was proposed to support the open ended activities that associate with transactions. The Split and Join transaction model focuses on activities that have uncertain duration, uncertain developments, and are interactive with other concurrent activities. The main idea is to divide an on- going transaction into two or more serializable transactions, and to merge the results of several transactions together as one atomic unit. In other words, the Split and Join transaction model supports reorganizing the structure of transactions (as illustrated in Figure 20).

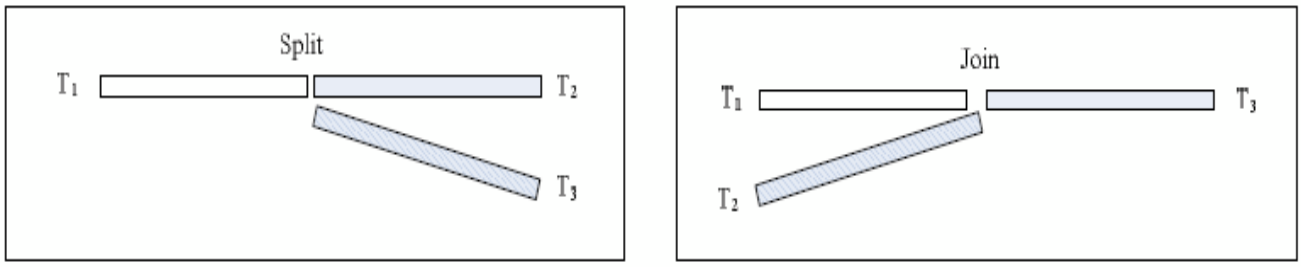


Figure 20: Split and Join Transaction Model

Transactions on a Transaction-Oriented Database System

A transaction processing system plays a role as a mediator that accepts transaction requests from users, dispatches these requests to the database system, coordinates the execution of the involved transactions, and forwards transaction results to the original acquirers.

The common programming model for a transaction-oriented database system is the client-server model. Users or clients interact with the database system by submitting their transaction processes that consist of one or many database operations to the transaction processing system. The transaction processing system will coordinate and manage the execution of these transaction processes by subsequently sending these database operations to the database system. The database system will carry out the actual execution of the submitted database operations. Finally, the transaction results that reflect the consistent states of the database system are returned to the clients.

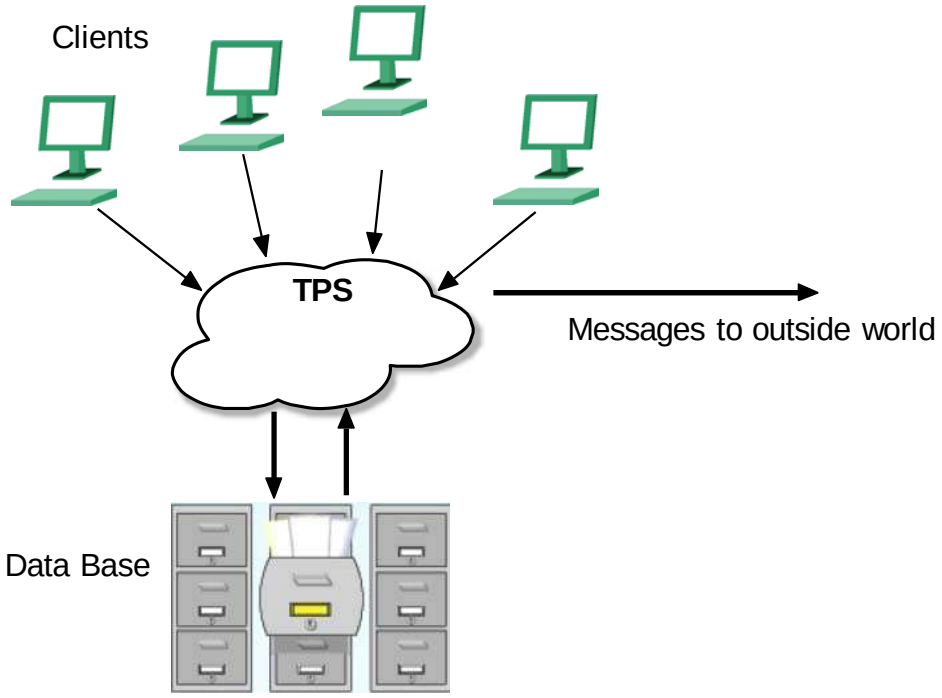


Figure 21: Transaction-oriented database system

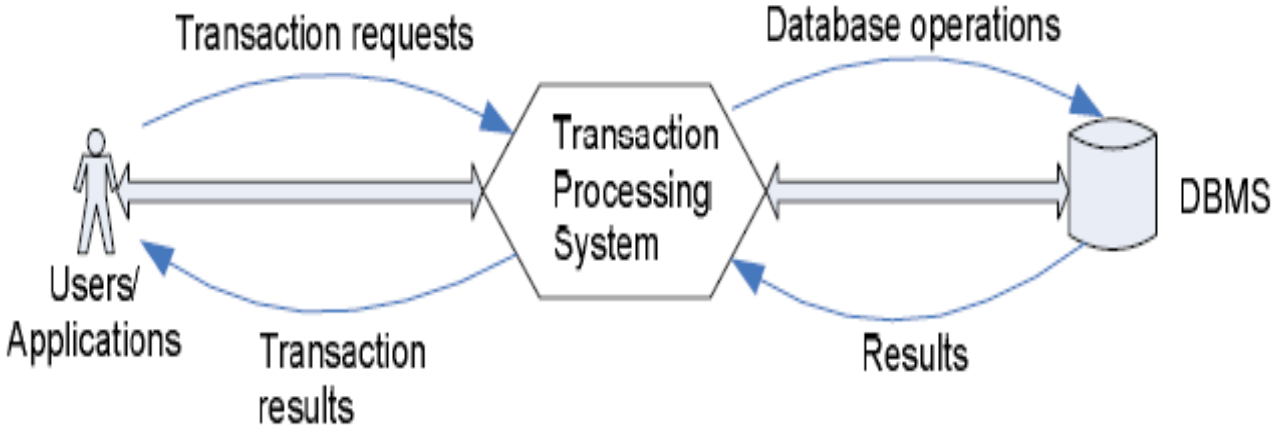


Figure 22: Dataflow of transaction-oriented database systems

To protect the integrity constraint of the database system, a set of essential components that includes a transaction manager, a scheduling manger and a log manger are deployed. Additional components such as communication manger or other resource managers can also be employed by the transaction processing system.

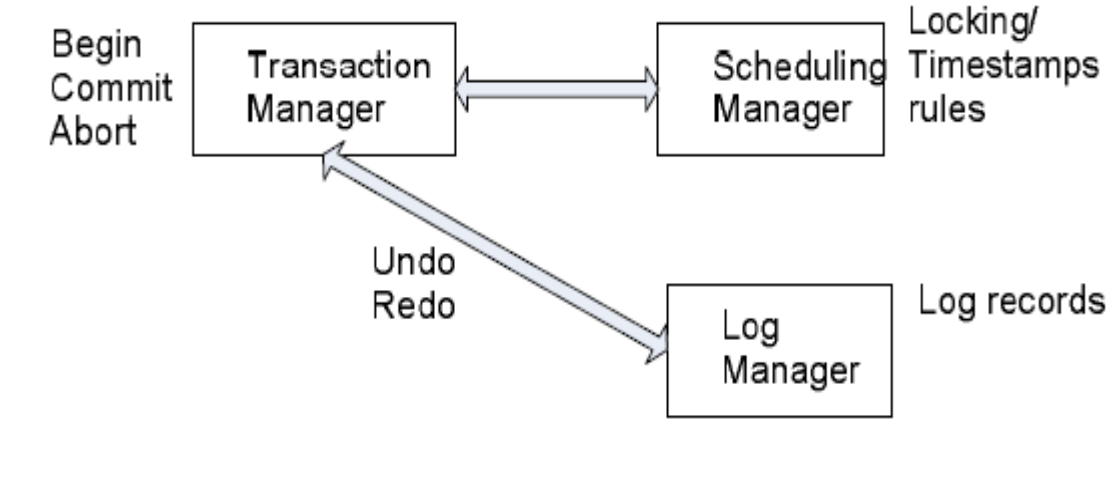


Figure 23: Transaction processing system components

The role of each transaction processing component is described as follows:

1. **Transaction Manager**. The role of the transaction manager is to orchestrate the execution of transactions. Via the help of the scheduling and log managers (explained below), the transaction manger takes care of all important operations of transactions such as begin, read, write, commit, and abort (or rollback). If the execution of a transaction is distributed to many different resource managers, the transaction manager will act as the coordinator of the involved participants.

2. **Scheduling Manager**. The scheduling manger manages the order of execution of the database operations. Usually, the scheduling manager makes use of concurrency control protocols, for example locking or timestamp protocols, in order to control the execution of transactions. Thus, the scheduling manger supports the isolation and consistency properties of transactions. Based on the applied concurrency control protocol, the scheduling manager will determine an execution order in which the submitted database operations will be carried out. For example, if a locking protocol is used, the scheduling manager will decide whether a lock request will be granted to the acquired transaction, or if a timestamp protocol is applied, the scheduling manager will assess if a submitted operation will be allowed to be carried out.

3. **Log Manager**. The role of the log manager is to support the database system to recover from failures. The log manager keeps track of the changes of the database states by recording the history of transaction execution. Depending on the deployed recovery strategies, for example undo and/or redo, the log manger will record necessary information in a non-volatile logbook. The log manager ensures the atomicity and the durability properties of transactions.