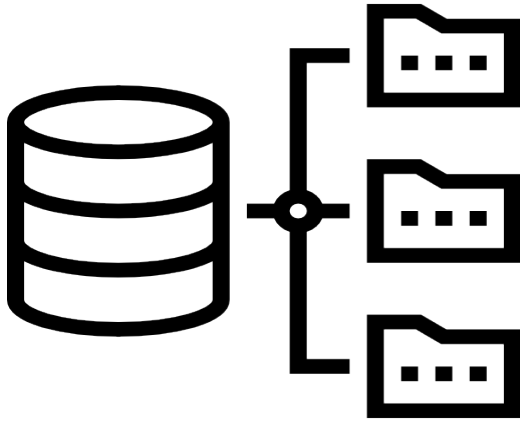
Advanced Database- IS411

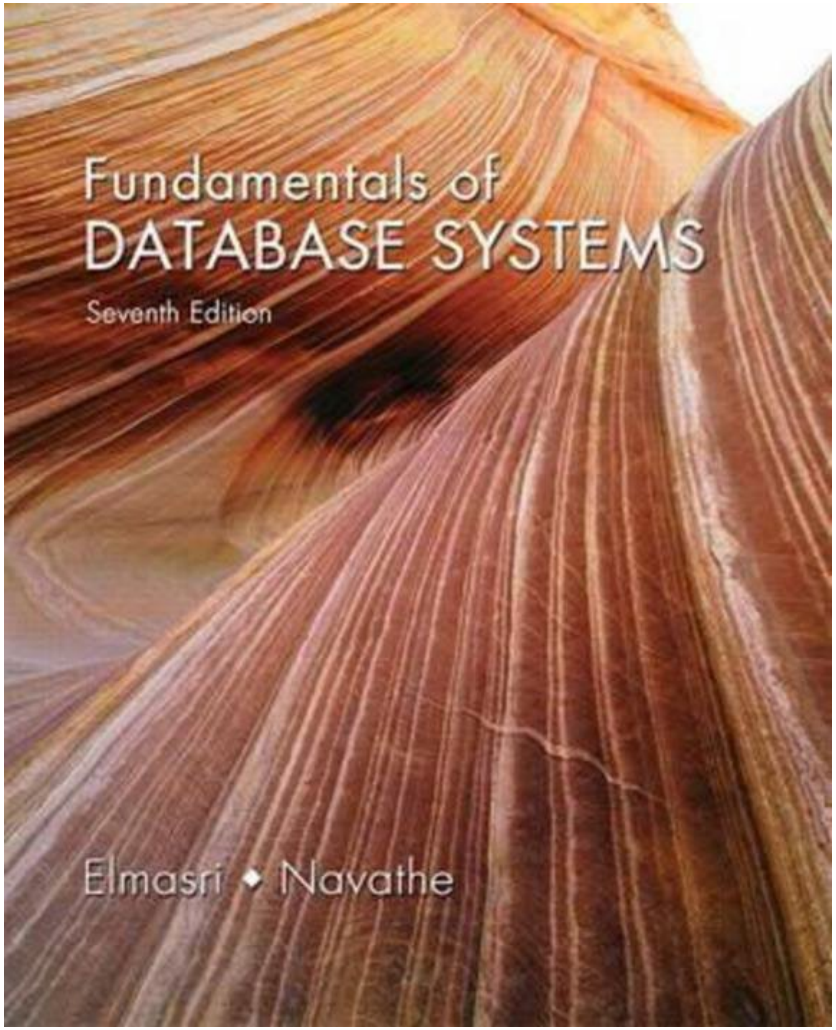


Introduced by

Dr. Ebtsam Adel

Lecturer of Information Systems,
Information Systems department,
Faculty of computers and information,
Damanhour university

Materials



https://www.amazon.com/Fundamentals-Database-Systems-RamezElmasri/dp/0133970779

Topics

- ✓ **Chapter 20** Introduction to Transaction Processing Concepts and Theory
- ✓ **chapter 24** NOSQL Databases and Big Data Storage Systems
- ✓ **chapter 25** Big Data Technologies Based on MapReduce and Hadoop
- ✓ **chapter 27** Introduction to Information Retrieval and Web Search
- ✓ **chapter 29** Overview of Data Warehousing and OLAP
- ✓ **chapter 30** Database Security

Assessment Summary (Grading Policy)

The Assessment for this subject consists of four components with the following weightings (grading breakdown):

Allocation of Marks	
Midterm examination	**15%**
Practical exam	**25%**
Unannounced quizzes	**10%**
Final examination	**50%**
Total	**100%**

chapter 20: Introduction to Transaction Processing Concepts and Theory

Introduction to Transaction Processing

- **Transaction:** An executing program (process) that includes one or more database access operations
  - Read operations (database retrieval, such as SQL SELECT)
  - Write operations (modify database, such as SQL INSERT, UPDATE, DELETE)
  - Transaction: A logical unit of database processing
  - Example: Bank balance transfer of $100 dollars from a checking account to a saving account in a BANK database.

Introduction to Transaction Processing

- A **transaction** (set of operations) may be:
  - stand-alone, specified in a high level language like SQL submitted interactively, or
  - consist of database operations embedded within a program (most transactions)
- **Transaction boundaries**: Begin and End transaction.
  - Note: An **application program** may contain several transactions separated by Begin and End transaction boundaries.

Single-User Versus Multiuser Systems

Single-User versus Multiuser Systems

- One criteria for **classifying** a database system is according to the number of users who can use the system **concurrently**.

- A DBMS is **single-user** if at most one user at a time can use the system, and it is **multiuser** if many users can use the system— and hence access the database—concurrently "at the same time".

- Single-user DBMSs are mostly restricted to personal computer systems "Microsoft access".
- most other DBMSs are multiuser such as:
  - an airline reservations system,
  - Database systems used in banks,
  - insurance agencies,
  - Stock exchanges,
  - Supermarkets
  - .....

Single-User versus Multiuser Systems

- Multiple users can access databases—and use computer systems—simultaneously because of the concept of **multiprogramming**, which allows the operating system of the computer to execute multiple programs—or **processes—**at the same time.

- A single central processing unit (CPU) can only execute at most one process at a time. However, **multiprogramming operating systems** execute some commands from one process, then **suspend** that process and execute some commands from the next process, and so on.

Single-User versus Multiuser Systems

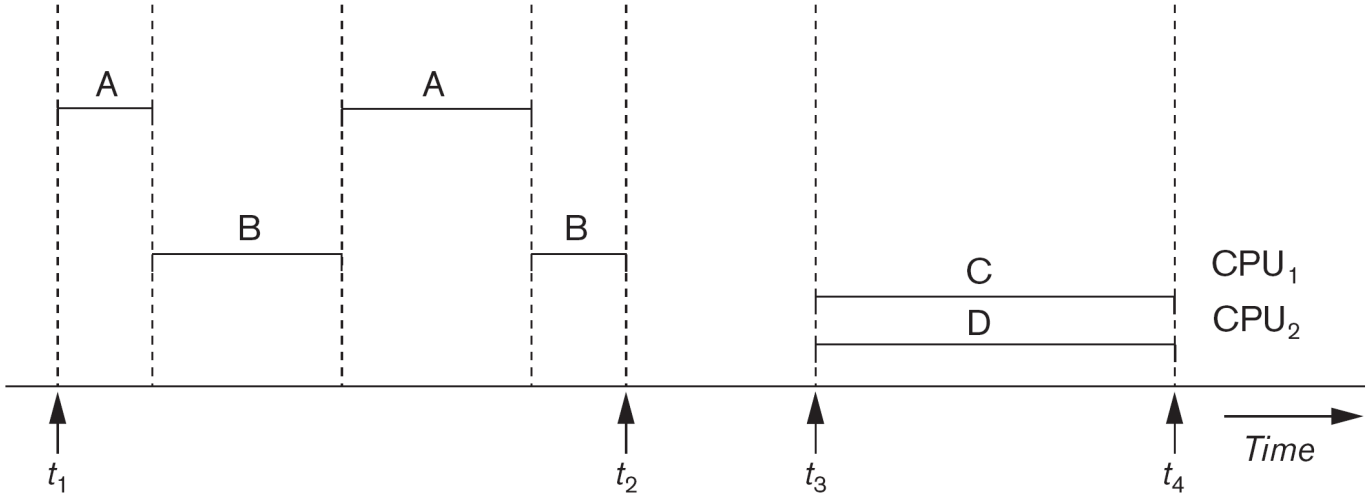


Figure 21.1
Interleaved processing versus parallel processing of concurrent transactions.

- A and B, executing concurrently in an interleaved fashion. Interleaving keeps the CPU busy when a process requires an input or output (I/O) operation.
- If the computer system has multiple hardware processors (CPUs), **parallel processing** of multiple processes is possible.

✓ Most of the theory concerning concurrency control in databases is developed in terms of **interleaved concurrency**.

Transactions, Database Items, Read and Writ Operations, and DBMS Buffers

Transactions, Database Items, Read and Write Operations, and DBMS Buffers

- A **transaction** is an executing program that forms a logical unit of database processing. A transaction includes one or more database access operations.
- Those can include insertion, deletion, modification (update), or retrieval operations.
- The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL.

- One way of specifying the transaction boundaries is by specifying explicit **begin transaction** and **end transaction** statements in an application Program.

- the transaction is called a **read-only transaction**; otherwise it is known as a **read-write transaction**.

Transactions, Database Items, Read and Write Operations, and DBMS Buffers

- **database** is basically represented as a collection of named data items.

- The size of a data item is called its **granularity** تقسيمات.
- A **data item** can be a database record, but it can also be a larger unit such as a whole disk block, or even a smaller unit such as an individual field (attribute) value of some record in the database.

❖ Using this simplified database model, the basic database access operations that a transaction can include are as follows:

■ **read_item(X).** Reads a database item named X into a program variable. we assume that the *program variable* is also named X.

■ **write_item(X).** Writes the value of *program variable* X into the database item named X.

Introduction to Transaction Processing (cont.)

READ AND WRITE OPERATIONS (cont.):

- **read_item(X) command includes the following steps:**

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable named X.

Introduction to Transaction Processing (cont.)

READ AND WRITE OPERATIONS (cont.):

- **write_item(X) command includes the following steps:**

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if it is not already in some main memory buffer).
3. Copy item X from the program variable named X into its correct location in the buffer.
4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Introduction to Transaction Processing (cont.)

- The DBMS will maintain in the **database cache** a number of **data buffers** in main memory. Each buffer typically holds the contents of one database disk block, which contains some of the database items being processed. When these buffers are all occupied, and additional database disk blocks must be copied into memory, some **buffer replacement policy** is used to choose which of the current occupied buffers is to be replaced.

- If the chosen buffer has been modified, it must be written back to disk before it is reused.

Introduction to Transaction Processing (cont.)

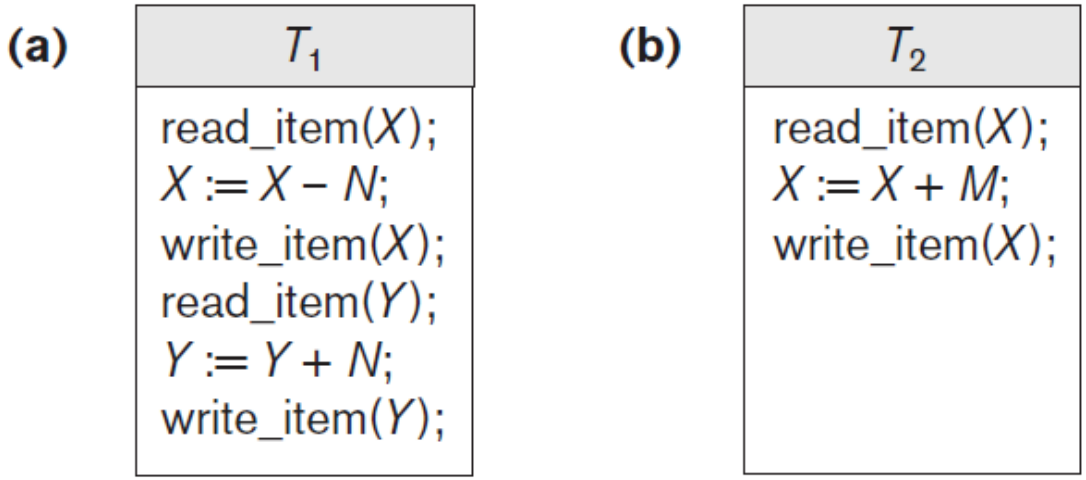


Figure 20.2
Two sample transactions.
(a) Transaction T_1.
(b) Transaction T_2.

A transaction includes read_item and write_item operations to access and update the database. Figure 20.2 shows examples of two very simple transactions. The **read-set** of a transaction is the set of all items that the transaction reads, and the **write-set** is the set of all items that the transaction writes. For example, the read-set of T_1 in Figure 20.2 is {X, Y} and its write-set is also {X, Y}.

Why Concurrency Control Is Needed

Why Concurrency Control Is Needed

- Without Concurrency Control, problems may occur with concurrent transactions:

1) Lost Update Problem.
2) The Temporary Update Problem
3) The Incorrect Summary Problem
4) The Unrepeatable Read Problem

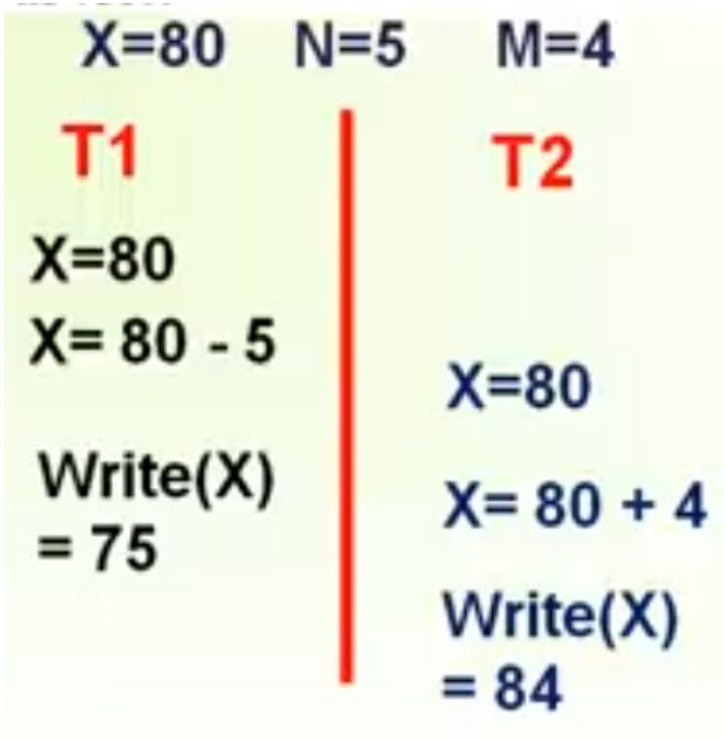
Why Concurrency Control Is Needed

1. Lost Update Problem

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect.

Time

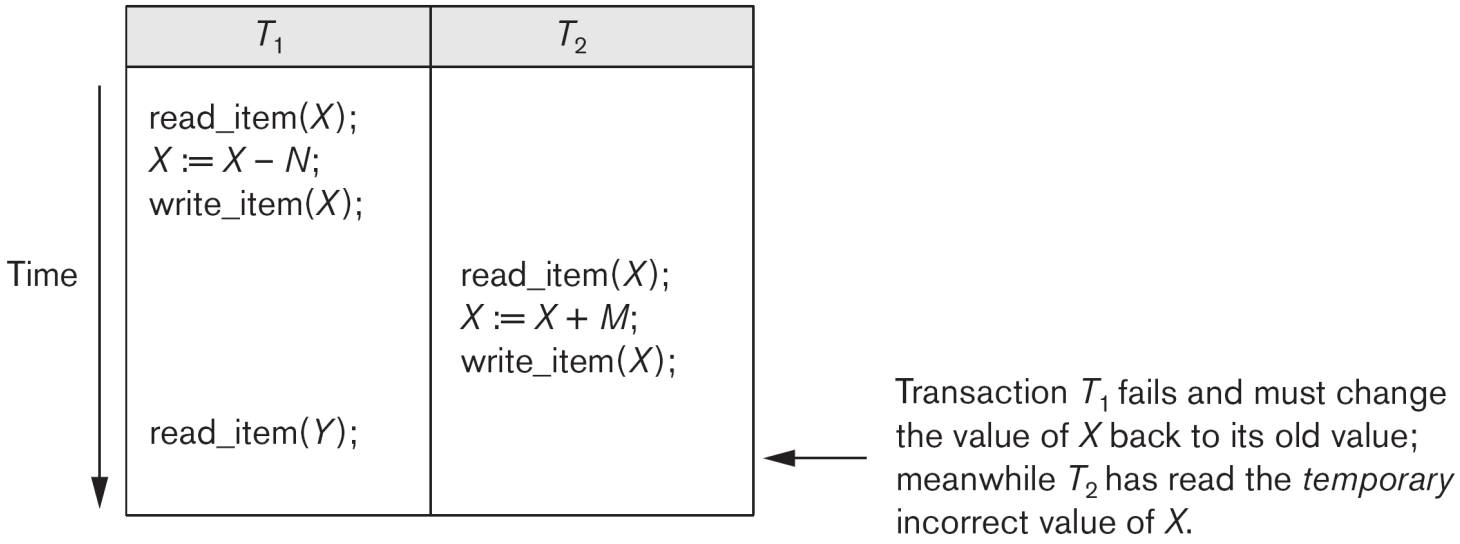
T_1	T_2
read_item(X);	
$X := X - N$;	
	read_item(X);
	$X := X + M$;
write_item(X);	
read_item(Y);	
	write_item(X);
$Y := Y + N$;	
write_item(Y);	



Why Concurrency Control Is Needed

2. The Temporary Update (or Dirty Read) Problem

This occurs when one transaction T1 updates a database item X, which is accessed (read) by another transaction T2; then T1 fails for some reason ; X was (read) by T2 before its value is changed back (rolled back or UNDONE) after T1 fails.



Why Concurrency Control Is Needed

3. The Incorrect Summary Problem

If one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.

(c)

T_1	T_3

	sum := 0;
	read_item(*A*);
	sum := *sum* + *A*;
	. . .

read_item(*X*);
X := *X* − *N*;
write_item(*X*);	

	read_item(*X*);
	sum := *sum* + *X*;
	read_item(*Y*);
	sum := *sum* + *Y*;

read_item(*Y*);
Y := *Y* + *N*;
write_item(*Y*);	

← T_3 reads *X* after *N* is subtracted and reads *Y* before *N* is added; a wrong summary is the result (off by *N*).

Why Concurrency Control Is Needed

4. The Unrepeatable Read Problem

Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction T' between the two reads. Hence, T receives different values for its two reads of the same item.

Why Recovery Is Needed

Why Recovery استعادة Is Needed

- Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or that the transaction does not have any effect on the database or any other transactions. In the first case, the transaction is said to be **committed**, whereas in the second case, the transaction is **aborted**.

Why Recovery Is Needed

Types of Failures. Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

1. **A computer failure (system crash).** A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures—for example, main memory failure.

2. **A transaction or system error.** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error.

Why Recovery Is Needed

3. Local errors or exception conditions detected by the transaction. During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found.

4. Concurrency control enforcement. deadlock

5. Disk failure. Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

6. Physical problems and catastrophes. This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

- Failures of types 1, 2, 3, and 4 are more common than those of types 5 or 6.

Transaction and System Concepts

Transaction States and Additional Operations

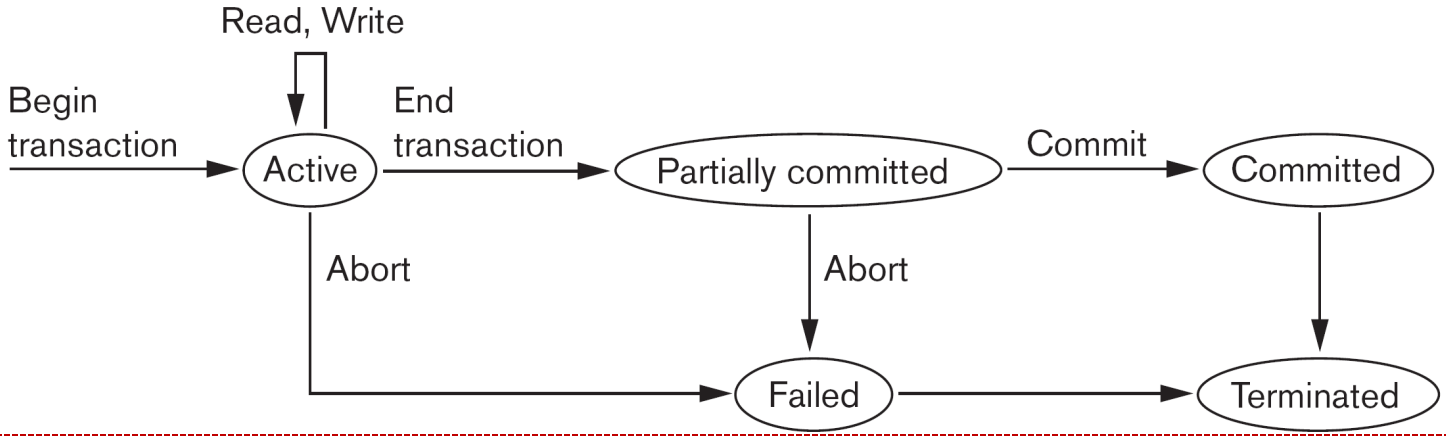
- A **transaction** is an atomic unit of work that should either be completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when each transaction starts, terminates, and commits, or aborts. Therefore, the recovery manager of the DBMS needs to keep track of the following operations:

  - BEGIN_TRANSACTION. This marks the beginning of transaction execution.
  - READ or WRITE. These specify read or write operations on the database items that are executed as part of a transaction.
  - END_TRANSACTION. This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or

  whether the transaction has to be aborted because it violates serializability (see Section 20.5) or for some other reason.

Transaction States and Additional Operations

Figure 21.4
State transition diagram illustrating the states for transaction execution.



whether the transaction has to be aborted because it violates serializability (see Section 20.5) or for some other reason.

- COMMIT_TRANSACTION. This signals a *successful end* of the transaction so that any changes (updates) executed by the transaction can be safely **committed** to the database and will not be undone.
- ROLLBACK (or ABORT). This signals that the transaction has *ended unsuccessfully*, so that any changes or effects that the transaction may have applied to the database must be **undone**.

Transaction States and Additional Operations

Figure 21.4
State transition diagram illustrating the states for transaction execution.

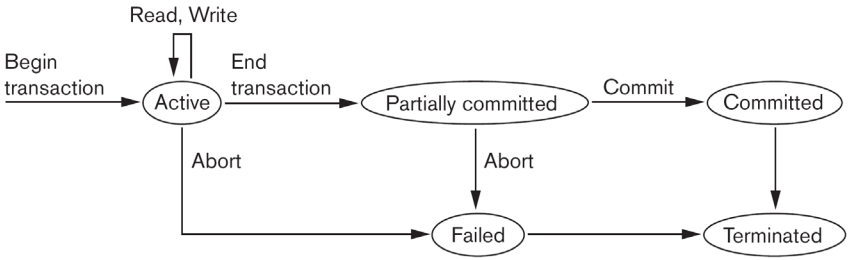


Figure 20.4 shows a state transition diagram that illustrates how a transaction moves through its execution states. A transaction goes into an **active state** immediately after it starts execution, where it can execute its READ and WRITE operations. When the transaction ends, it moves to the **partially committed state**. At this point, some types of concurrency control protocols may do additional checks to see if the transaction can be committed or not. Also, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently (usually by recording changes in the system log, discussed in the next section).[5] If these checks are successful, the transaction is said to have reached its commit point and enters the **committed state**. Commit points are discussed in more detail in Section 20.2.3. When a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database, even if a system failure occurs.

The System Log

Transaction The System Log

- To be able to recover from failures that affect transactions, the system maintains a **log** to keep track of all transaction operations that affect the values of database items, as well as other transaction information that may be needed to permit recovery from failures.

- The log is a **sequential**, append-only file that is kept on **disk**, so it is not affected by any type of failure except for disk or catastrophic failure.

Transaction The System Log

- In these entries, T refers to a unique **transaction-id** that is generated automatically by the system for each transaction and that is used to identify each transaction:

1. [**start_transaction**, T]. Indicates that transaction T has started execution.
2. [**write_item**, T, X, ***old_value***, ***new_value***]. Indicates that transaction T has changed the value of database item X from *old_value* to *new_value*.
3. [**read_item**, T, X]. Indicates that transaction T has read the value of database item X.
4. [**commit**, T]. Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5. [**abort**, T]. Indicates that transaction T has been aborted.

Transaction The System Log

- Because the log contains a record of every WRITE operation that changes the value of some database item, it is possible to **undo** the effect of these WRITE operations of a transaction *T* by tracing backward through the log and resetting all items changed by a WRITE operation of *T* to their old values.

- **Redo** of an operation may also be necessary if a transaction has its updates recorded in the log but a failure occurs before the system can be sure that all these new values have been written to the actual database on disk from the main memory buffers.

Commit Point of a Transaction- commit

- A transaction T reaches its **commit point** when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database have been recorded in the log.
- Beyond the commit point, the transaction is said to be **committed**, and its effect must be permanently recorded in the database.
- The transaction then writes a commit record [commit, T] into the log.

Commit Point of a Transaction - failure

If a system failure occurs, we can search back in the log for all transactions T that have written a [start_transaction, T] record into the log but have not written their [commit, T] record yet; these transactions may have to be *rolled back* to *undo their effect* on the database during the recovery process. Transactions that have written their commit record in the log must also have recorded all their WRITE operations in the log, so their effect on the database can be *redone* from the log records.

Desirable Properties of Transactions

Desirable Properties of Transactions

- Transactions should possess several properties, often called the **ACID** properties; they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the **ACID** properties:

1. **Atomicity.** A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.

2. **Consistency تناسق preservation.** A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.

Desirable Properties of Transactions-atomicity

- **The atomicity property** requires that we execute a transaction to completion. It is the responsibility of the transaction recovery subsystem of a DBMS to ensure atomicity.

- If a transaction fails to complete for some reason, such as a system crash in the midst of transaction execution, the recovery technique must undo any effects of the transaction on the database. On the other hand, write operations of a committed transaction must be eventually written to disk.

Desirable Properties of Transactions-
Consistency preservation

The preservation of *consistency* is generally considered to be the responsibility of the programmers who write the database programs and of the DBMS module that enforces integrity constraints. Recall that a **database state** is a collection of all the stored data items (values) in the database at a given point in time. A **consistent state** of the database satisfies the constraints specified in the schema as well as any other constraints on the database that should hold. A database program should be written in a way that guarantees that, if the database is in a consistent state before executing the transaction, it will be in a consistent state after the *complete* execution of the transaction, assuming that *no interference with other transactions* occurs.

Desirable Properties of Transactions

3. Isolation. A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.

4. **Durability or permanency.** The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

Desirable Properties of Transactions

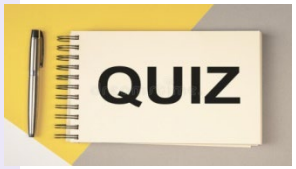
The *atomicity property* requires that we execute a transaction to completion. It is the responsibility of the *transaction recovery subsystem* of a DBMS to ensure atomicity. If a transaction fails to complete for some reason, such as a system crash in the midst of transaction execution, the recovery technique must undo any effects of the transaction on the database. On the other hand, write operations of a committed transaction must be eventually written to disk.

The preservation of *consistency* is generally considered to be the responsibility of the programmers who write the database programs and of the DBMS module that enforces integrity constraints. Recall that a **database state** is a collection of all the stored data items (values) in the database at a given point in time. A **consistent state** of the database satisfies the constraints specified in the schema as well as any other constraints on the database that should hold. A database program should be written in a way that guarantees that, if the database is in a consistent state before executing the transaction, it will be in a consistent state after the *complete* execution of the transaction, assuming that *no interference with other transactions* occurs.

Desirable Properties of Transactions

The *isolation property* is enforced by the *concurrency control subsystem* of the DBMS.[8] If every transaction does not make its updates (write operations) visible to other transactions until it is committed, one form of isolation is enforced that solves the temporary update problem and eliminates cascading rollbacks (see Chapter 22) but does not eliminate all other problems.

The *durability property* is the responsibility of the *recovery subsystem* of the DBMS. In the next section, we introduce how recovery protocols enforce durability and atomicity and then discuss this in more detail in Chapter 22.



1. Discuss the different types of failures. What is meant by catastrophic failure?

2. Discuss the actions taken by the read_item and write_item operations on a database.

Practical Part

NOSQL MongoDB

MongoDB is an open source NOSQL dataset in C++, this offers flexible data models, indexing, replication and modern applications for scalable architecture.

Please visit this website:
https://www.tutorialspoint.com/mongodb/index.htm

NOSQL MongoDB

- What is the difference between SQL database and NoSQl database?
- Advantages of MongoDB over RDBMS

✓ **Install MongoDB On Windows**

download the latest release of MongoDB from: https://www.mongodb.com/download-center.

✓ **Create a database in MongoDB**

✓ **Design Data Model**


Thank You!
Any questions?