CCI 4301 - Advanced Database Management Systems

Site:	TUM E-Learning Portal	Printed by:	DANIEL MWANGANGI KALUTU
Course:	Advanced Database Management Systems	Date:	Monday, 27 February 2023, 9:46 PM
Book:	CCI 4301 - Advanced Database Management Systems		

Description

Table of contents

1. Transaction Management

Welcome to Session one on Transaction Management

1.1. Learning Outcomes

After completing Sessions 1 in this course unit successfully, you should be able to:

- Define transaction and transaction management
- Discuss the ACID properties of a transaction
- Describe the states of transaction in DBMS
- Discuss the components of a DBMS

1.2. Introduction to Transaction Processing

Traditionally, database systems are classified according to the number of users who can use the system concurrently. There are two types of DBMS: **Single-user DBMS –** these are database systems that support at most one user at a time. Single-user DBMSs are mostly restricted to personal computer systems. **Multiuser DBMS** are database systems that can be used by many users at the same time thus allowing concurrent access the database. Most DBMSs are multiuser. Example of multiuser DBMS is an airline reservations system which is used by hundreds of travel agents and reservation clerks concurrently. Such systems are called transaction processing systems.

Transaction processing systems are systems with large databases and hundreds of concurrent users executing database transactions. Examples of such systems include airline reservations, banking, credit card processing, online retail purchasing, stock markets, supermarket checkouts, and many other applications. These systems require high availability and fast response time for hundreds of concurrent users.

When multiple transactions are submitted by various users concurrently, it is possible for them to interfere with one another in a way that produces incorrect results, or some transactions may fail. These problems lead to what we call concurrency control problem. In order to ensure the correct executions of database transactions, we therefore need concurrency control to manage concurrent users and transactions. These is broadly referred to as transaction management. In this session we shall define what is a transaction and discuss the concepts of transaction management.

1.3. Transactions Management

A transaction is defined as a series of actions, carried out by a single user or application program, which reads or updates the contents of a database. A transaction is typically implemented by a computer program, which includes database operations such as retrievals, insertions, deletions, and updates. These 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. Example of the basic database access operations that a transaction can include are:

- **read_item(*X*) -** Reads a database item named *X* into a program variable.
- **write_item(*X*) -** Writes the value of program variable *X* into the database item named *X*.

A transaction includes *read_item* and *write_item* operations to access and update the database. However, during the execution of a transaction, the database can (and usually is) temporarily inconsistent. The most important point is that the database should be consistent when the transaction terminates. Transactions Management deals with the problem of always keeping the database in a consistent state even when concurrent accesses and failures occur.

Other definitions of transaction include:

- A transaction is a ‘logical unit of work’ on a database, i.e., each transaction does something in the database.
- A transaction is a unit of consistent and reliable computation.
- Transactions are the unit of recovery, consistency, and integrity of a database

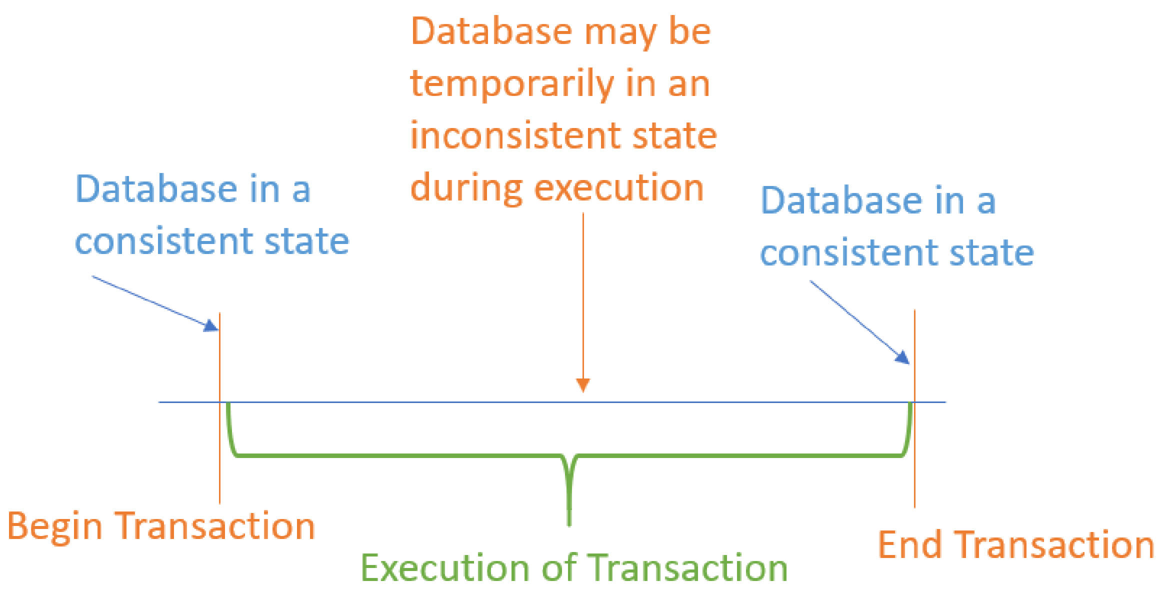
Example of a transaction
Suppose a bank employee transfers Kes. 10,000.00 from A's account to B's account. In this transaction, the tasks involved are as follows:
A’s Account

1. Open_Account(A)
2. read(A) = Old_Balance
3. New_Balance = Old_Balance - 10000
4. write(A) = New_Balance
5. Close_Account(A)

B’s Account

1. Open_Account(B)
2. read(B) = Old_Balance
3. New_Balance = Old_Balance + 10000
4. write(A) = New_Balance
5. Close_Account(B)

1.4. Properties of a Transaction (ACID)


Database may be temporarily in an inconsistent state during execution
Database in a consistent state
Database in a consistent state
Begin Transaction
Execution of Transaction
End Transaction

Database Transactions

There are four desired properties that a transaction should possess. These are: Atomicity, Consistency, Isolation and Durability (ACID). Each of these properties is discussed below. The properties are usually enforced by the concurrency control and recovery methods of the DBMS.

i. **Atomicity**

A transaction is an atomic (indivisible) unit of processing; it should either be performed in its entirety or not performed at all. This means that:

- Transactions are either done or not done
- They are never left partially executed

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.

ii. **Consistency**

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 consistent state. Therefore, a transaction is an atomic program that executes on the database and preserves the consistency of the database. This means the input to a transaction is a consistent database, AND the output of the transaction must also be a consistent database.

The preservation of consistency is generally considered to be the responsibility of the DBMS module that enforces integrity constraints and application developers (programmers) who write the database programs. 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. Consistency property ensures that only valid data is written in the database.

iii. **Isolation**

Even though many transactions are executed concurrently, a transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently. Therefore, transactions should execute as though they are executed independent of one another. Partial effects of incomplete transactions should not be visible to other transactions.

The isolation property is enforced by the concurrency control subsystem of the DBMS. 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 (as we will see in the coming sessions) but does not eliminate all other problems.

iv. **Durability**

The changes applied to the database by a committed transaction must persist in the database and must not be lost because of any failure even if the system crashes. It’s the duty of the recovery subsystem of the DBMS to ensure durability. It ensures that the effects of completed transactions are resilient or permanently recorded in the database against failures.

1.5. States of Transaction in DBMS

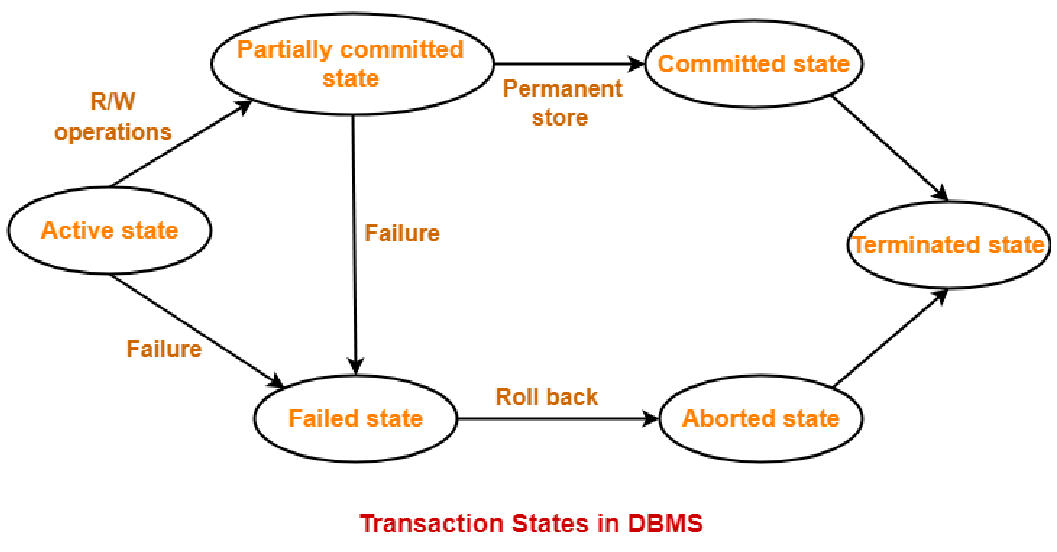


Figure 1: Life of a transaction

A transaction in DBMS can be in one of the following states:

Active State

When the instructions of the transaction are running then the transaction is in active state. If all the ‘read and write’ operations are performed without any error then it goes to the “partially committed state”; if any instruction fails, it goes to the “failed state”.

Partially Committed

After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the DataBase then the state will change to “committed state” and in case of failure it will go to the “failed state”.

Failed State

When any instruction of the transaction fails, it goes to the “failed state” or if failure occurs in making a permanent change of data on Data Base.

Aborted State

After having any type of failure the transaction goes from “failed state” to “aborted state” and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.

Committed State

It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the “terminated state”.

Terminated State

If there isn’t any roll-back or the transaction comes from the “committed state”, then the system is consistent and ready for new transaction and the old transaction is terminated.

1.6. Transaction States

A transaction is an atomic unit of work that should either be completed in its entirety or not done at all. Every active transaction must terminate by either aborting or committing.

Example: Let us take a debit transaction from an account which consists of following operations:

1. R(A);
2. A=A-1000;
3. W(A);

Assume A's value before starting of transaction is 5000.

The operations involved include:

1. The first operation reads the value of A from database and stores it in a buffer.
2. Second operation will decrease its value by 1000. So, buffer will contain 4000.
3. Third operation will write the value from buffer to database. So, A's final value will be 4000

But it may also be possible that transaction may fail after executing some of its operations. The failure can be because of **hardware, software or power** etc. For example, if debit transaction discussed above fails after executing operation 2, the value of A will remain 5000 in the database which is not acceptable by the bank. To avoid this, a database transaction has two important operations or outcomes:

1. **Committed:** if the transaction completes successfully, the changes made by transaction are made permanent(committed) in the database and the database reaches a new consistent state.
2. **Aborted:** if the transaction does not execute successfully, the transaction is aborted and the database restored back(undone) to the consistent state it was before the transaction started (rolled back).

A transaction may fail to complete successfully either because the (database) system crashes or the transactions are aborted by either the system or the user (or application). For recovery purposes, the recovery manager of the DBMS needs to keep track of when each transaction starts, terminates, and commits or aborts.

1.7. Components of a Database Management Systems

a. **Transactions Manager**

It is the responsibility of the Transactions Manager of a DBMS to ensure transactions do not interfere with one another and corrupt the database. It coordinates transactions on behalf of application program.

b. **Scheduler**

Responsible for implementing a particular strategy for concurrency control. Also called the Lock Manager (for those systems that use locking as the means of managing concurrency). The objective of the scheduler is to maximize concurrency control without allowing concurrently executing transactions to interfere with one another and compromise the integrity of the database.

c. **Recovery Manager**

If a failure occurs during the transaction, then the database could be inconsistent. Recovery Manager ensures that the database is restored to the state it was in before the start of the transaction and therefore a consistent state.

d. **Buffer Manager**

Responsible for the transfer of data between disk storage and memory.

1.8. Session 1 Webnar

Session 1's webinar focuses on the basic concepts of transaction management. This is a live event which you are required to participate.

To join the lived webinar, click the Link that corresponds to your class as listed below:

1. **Tuesdays 7:00AM - 9:00AM** - BSCS/SEP2020/J&S-FT and BSCS/JAN2021/J&S-FT Link to join: https://elearning.tum.ac.ke/mod/bigbluebuttonbn/view.php?id=34599
2. **Wednesdays 7:00AM - 9:00AM** - BTIT/SEP2020/J&S-FT and BTIT/JAN2021/J&S-FT Link to join: https://elearning.tum.ac.ke/mod/bigbluebuttonbn/view.php?id=34635
3. **Thursdays 7:00AM - 9:00AM** - BSIT/SEP2020/J&S-FT and BSIT/JAN2021/J&S-FT Link to join: https://elearning.tum.ac.ke/mod/bigbluebuttonbn/view.php?id=34637

1.9. Reflection

In this session we have discussed that database systems are either single-user DBMS that support at most one user at a time or multiuser DBMS which are database systems that support more than one user at the same time.

We have also discussed that transactions Management deals with the problem of always keeping the database in a consistent state even when concurrent accesses and failures occur. We saw that a transaction in DBMS can be in one of the following states: Committed, Aborted, partially committed or failed.

We also discussed the desirable properties of a transaction that include atomicity, consistency, isolation and durability and, finally discussed the components of a DBMS

1.10. Session 1: Quiz

This end of session quiz will enable you to check your understanding of what you have learned in this Session. Please ensure that you have read all the notes in this section before you start attempt. You must try to obtain a score of at least 50% to proceed to the next session.

Take Note of the following:

- The marks obtained in this quiz will contribute to your Continuous Assessment Tests (CAT).
- You have **ONLY ONE (1) attempt** for this quiz.
- once you start attempt, the timer will start counting
- Time per question is: **1 minute**

When you have finished the quiz, click on ‘Finish attempt and submit..’ to review your ‘Summary of attempt’.

Grading method: Highest grade

Click this to Start Session 1 Quiz now

1.11. References

Core Books

i. Coronel, C., & Morris, S. (2017). *Database Systems: Design, Implementation, & Management* (12th ed.). Boston, MA: Cengage Learning. ISBN: 1305627482.
ii. Elmasri, R., & Navathe, S. B. (2017). *Fundamentals of Database Systems* (7th ed.). Hoboken, NJ: Pearson Education Ltd. ISBN: 0133970779.
iii. Connolly, T. M., & Begg, C. E. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Boston, MA: Pearson Education Ltd. ISBN: 0132943263.

Core Journals

i. Journal of Database Management. ISSN: 1063-8016.
ii. Database Management & Information Retrieval. ISSN: 1862-5347.
iii. International Journal of Information Technology and Database Systems. ISSN: 2231-1807.

Recommended Text Books

i. Hernandez, M. J. (2013). *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design* (3rd ed.). Harlow, UK: Addison-Wesley. ISBN: 0321884493.
ii. Rankins, R., Bertucci, P., Gallelli, C., & Silverstein, A. (2015). *Microsoft SQL Server 2014 Unleashed*. Indianapolis, IN: Sams Publishing. ISBN: 0672337290.
iii. Comeau, A. (2016). *MySQL Explained: Your Step By Step Guide to Database Design*. Bradenton, FL: OSTraining. ISBN: 151942437X.

Recommended Journals

i. International Journal of Intelligent Information and Database Systems. ISSN: 1751-5858.
ii. Database Systems Journal. ISSN: 2069-3230.
iii. Distributed and Parallel Databases. ISSN: 0926-8782.
iv. International Journal of Database Management Systems. ISSN: 0975 - 5985.
v. Journal of Database Management. ISSN:1063-8016.

2. Concurrency Control

Welcome to Session Two on Concurrency Control

2.1. Learning Outcomes

By the end of this session, you should be able to:

- Define concurrency control
- Identify the problems associated with concurrent access
- Discuss the problems associated with concurrent transactions

2.2. Introduction to concurrency control

The main objective of developing database is to enable many users share data concurrently. Concurrent access is relatively easy if all users are only reading data. However, when multiple (more than one) transactions/users are running simultaneously in an interleaved manner and are updating the same data item, then there may be interference (conflicts) that can result to inconsistencies. To handle these conflicts, we need concurrency control in DBMS. Concurrency control are techniques that allows transactions to run simultaneously but handles them in such a way that they don’t interfere with each other hence preserving the integrity of data in the database. The basic terminologies used in concurrency control are defined below:

Concurrency

In computer science, concurrency is a property of systems in which several computations are executing simultaneously and potentially interacting with each other.

Concurrency control

It is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

Interleaving transactions

Interleaving allows multiple users of the database to access it at the same time. While one transaction is performing an I/O task, another one can perform a CPU-intensive task thus maximizing the throughput of the system, i.e., Interleaving keeps the CPU busy when a process requires an input or output (I/O) operation, such as reading a block from disk. The CPU is switched to execute another process rather than remaining idle during I/O time. Interleaving also prevents a long process from delaying other processes.

The purpose of Concurrency Control is therefore to: enforce isolation among conflicting transactions, preserve database consistency and resolve conflicts among conflicting transactions.

2.3. Interference

Several problems can occur when concurrent transactions execute in an uncontrolled manner. Before we discuss the problems caused by interference, we will first illustrate how transactions executing simultaneously without concurrent control interfere/conflict with one another.

Example on conflict/interference
Assume you run a joint bank account with your friend and each of you can access and withdraw money. If the current balance in your joint account is Kes 9000. Now let's say you both go to different branches of the same bank at the same time and try to withdraw Kes 7000 each. Now if the bank doesn't have concurrency control in place you both get Kes 7000 at the same time but once both transactions finish the account balance would be Kes - 5000 which is not possible and leaves the database in inconsistent state.

As earlier discussed, the basic database access operations that a transaction can include are:

- **read_item(*X*) -** reads a database item named *X* into a program variable and,
- **write_item(*X*) -** writes the value of program variable *X* into the database item named *X*.

Two operations on the same data item in different transactions conflict if one of them is a write operation. If we denote database read and write operations on a data item x as r(x) and w(x), then we say the operations conflict:

1. If the operations belong to different transactions (One transaction cannot conflict)
2. If the operations access the same data item and
3. If at least one of the operations is a write

We can summarize conflict/interference by saying

> *"conflict (can) occurs when two or more transactions update the same data item at the same time without concurrency control".*

Examples on how two operations on the same data item conflict if at least one of the operations is a write:

i. r(x) and w(x) conflict
ii. w(x) and r(x) conflict
iii. w(x) and w(x) conflict
iv. r(x) and r(x) do not
v. r/w(x) and r/w(y) do not

The order of conflicting operations matters. For example, let's assume T_1 represents Transaction 1 and T_2 represents transaction 2, r(a) is a read operation on data item a and w(a) is a write operation on data item a, the if T_1 . r(a) precedes T_2 .w(a), then conceptually, T_1 should precede $T_{2.}$

2.4. Concurrency problems in DBMS Transaction

There are several problems that may occur when concurrent transactions execute in an uncontrolled or unrestricted manner (without concurrency control). These problems are commonly referred to as concurrency problems in database environment and include:

- Lost update problem
- Uncommitted dependency problem
- Inconsistency analysis problem

We shall discuss each of these problems in details

2.5. Lost update problem

Lost update problem occurs when two different transactions are trying to update the same data item within a database at the same time. Typically, one transaction updates the value of a particular data item x while another transaction that began very shortly afterward did not see this update on that data item x before updating the same value itself. The result of the first transaction is then "lost," as it is simply overwritten by the second transaction. This problem is known as the Lost Update Problem.

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

Example 1: Consider this situation.

Time	Transaction A	Transaction B
t_1	Read (x)	-
t_2	-	Read (x)
t_3	Write (x)	-
t_4	-	Write (x)

1. Transaction A retrieves/reads some record x at time t_1.
2. Transaction B retrieves/reads the same record x at the time t_2.
3. Transaction A updates/write the record at time t_3 on the basis of valves read at time t_1.
4. Transaction B updates/writes the same record at time t_4 on the basis of values read at time t_2.
5. Update at t_3 is lost i.e., Transaction A’s update is lost at time t_4 because transaction B overwrites it without even looking at it.

Example 2: Consider the below diagram where two transactions $T_1$and T_2, are performed on the same account x where the balance of account x is 100.

Time	T 1	T 2	Bal_x
t_1	-	begin_transaction	100
t_2	begin_transaction	read(bal_x)	100
t_3	read(bal_x)	bal_x= bal_x +100	100
t_4	bal_x= bal_x -10	write(bal_x)	200
t_5	write(bal_x)	Commmit	90
t_6	Commmit	-	90

Transaction T_1 updates the same record updated earlier by T_2 at time t_5 on the basis of values read at t_3.

Question: How do we solve the lost update problem?

2.6. Uncommitted Dependency Problem (Dirty Read Problems)

The uncommitted dependency problem occurs when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions. This leads to violations of integrity constraints governing the database that arise when two transactions are allowed to execute concurrently without being synchronized.

Uncommitted dependency problem occurs when one transaction is allowed to see the intermediate results of another transaction before it has committed. Consider:

Transaction A	Time	Transaction B
__	t_1	Read(R)
__	t_2	Write(R)
Read(R)	t_3	__
	t_4	Roll back

- Transaction B reads R at t_1 and updates it at t_2.
- Transaction A reads an uncommitted update at time t3, and then the update is undone at time t_4.

Note: Transaction A is therefore operating on false assumption. Transaction A becomes dependent on an uncommitted update at time t_2.

Example 1: Uncommitted dependency problem
Consider two transactions T_3 and T_4 in the below diagram performing read/write operations on account x where the available balance in account A is Kes 100:

Time	T_3	T_4	Bal_x
t_1	-	begin_transaction	100
t_2	-	read(bal_x)	100
t_3	-	bal_x= bal_x +100	100
t_4	begin_transaction	write(bal_x)	200
t_5	read(bal_x)	:	200
t_6	bal_x= bal_x -10	Rollback	100
t_7	write(bal_x)	-	190
t_8	Commit	-	190

- At time t_2, transaction T_4 reads the value of account x, i.e., 100.
- At time t_3, transaction T_4 adds 100 to account x that becomes 200.
- At time t_4, transaction T_4 writes the updated value in account x, i.e., 200.
- Then at time t_5, transaction T_3 reads uncommitted value of account x that will be read as 200 (Dirty Read).
- Then at time t_6, transaction T_4 rollbacks due to server problem, and the value changes back to 100 (as initially). But the value for account x remains 200 for transaction T_3 as committed, At same time t_6, transaction T_3 subtracts 10 from account x that becomes 190.
- At time t_7, transaction T_3 writes the updated value in account x, i.e., 190.
- At time t_8, transaction T_3 commits value in account x, i.e., 190.
- Transaction T_3 is therefore operating on false assumption i.e.Transaction T_3 becomes dependent on an uncommitted update at time T_4

Question: How do we solve the uncommitted dependency problem?

2.7. Inconsistency Analysis Problem

The problem of inconsistent analysis occurs when a transaction reads several values from the database but a second transaction updates some of them during the execution of the first. Transactions that only read the database can obtain the wrong result if they're allowed to read partial result or an incomplete transaction, which has simultaneously updated the database.

Example: Inconsistency Analysis Problem
Consider two transactions A and B operating on bank account records. Transaction B is summing account balances. Transaction A is transferring an amount 10 from account 1 to account 3. Initial account balances are as follows: Account 1 = 100; Account 2 = 50; Account 3 = 25

Time	**Transaction A**	**Transaction B**	**Acc1**	**Acc2**	**Acc3**	**Sum**
t_1	Read Acc1	Read Acc1	100	50	25	0
t_2	Acc1 - 10	Sum + Acc1	100	50	25	100
t_3	Write Acc1	Read Acc2	90	50	25	100
t_4	Read Acc3	Sum + Acc2	90	50	25	150
t_5	Acc3 + 10	-	90	50	25	150
t_6	Write Acc3	-	90	50	35	150
t_6	Commit	Read Acc3	90	50	35	150
t_7	-	Sum + Acc3	90	50	35	185
t_8	-	Commit	90	50	35	185

* **Sum = 185 not 175!!**

Question: How do we solve the inconsistent analysis problem?

2.8. Reflection

In this session we have discussed concurrency transactions and the problems associated with uncontrol concurrent transactions. In particular, we have discussed three types of problems: lost update problem, uncommitted dependency problem and inconsistent analysis problem. In our next session, we will discuss about scheduling and how its used in concurrency control to solve the above problems.

2.9. Session 2: Quiz

This end of session quiz will enable you to check your understanding of what you have learned in this Session. Please ensure that you have read all the notes in this section before you start attempt. You must try to obtain a score of at least 50% to proceed to the next session.

Take Note of the following:

- The marks obtained in this quiz will contribute to your Continuous Assessment Tests (CAT).
- You have ONLY ONE (1) attempt for this quiz.
- once you start attempt, the timer will start counting
- Time per question is: 1 minute

When you have finished the quiz, click on ‘Finish attempt and submit..’ to review your ‘Summary of attempt’.

Grading method: Highest grade

Click this to Start Session 2 Quiz now

3. Scheduling

Welcome to Session Three on Scheduling

3.1. Learning Outcomes

At the end of this lecture, you should be able to:

1. Define scheduling and serialization
2. Describe different types of schedules
3. Apply different types of schedules

3.2. Introduction

Hello and welcome to our third lecture in advanced database systems. Today we shall define scheduling and serialization, discuss the different types of schedules, give examples on how each schedule is applied in transaction management and finally give a summary of scheduling.

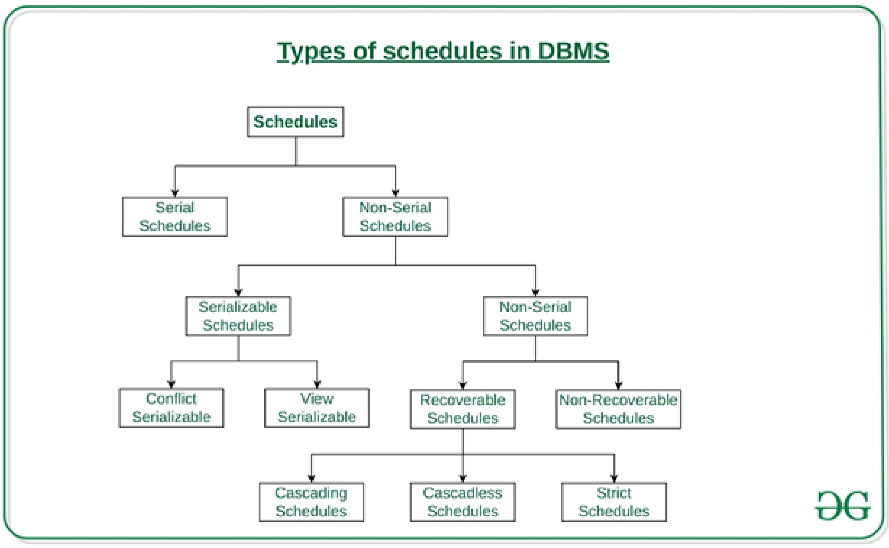
3.3. Scheduling and Serialization

In our previous session, we discussed about concurrent transactions and the problems that arise when multiple transactions run in a concurrent manner in an uncontrolled way. Though a quick fix to the problems of concurrent execution is to allow only one transaction to be executed and committed before another transaction begins to execute, however, this defeats the purpose of multi-user DBMS which is to maximize the degree of concurrency or parallelism in the

3.4. Types of Schedules in DBMS

Having understood what a schedule is, let us now discuss various types of schedules. Schedules can be categorized as being either serial schedule of concurrent schedule or non-serial schedule

The diagram below shows the different types of schedules.



3.5. Serial Schedules

Serial schedules is a type of scheduling in which the transactions are executed non-interleaved. When one transaction completely executes before starting another transaction, the schedule is called serial schedule. In a serial schedule, no transaction starts until a running transaction has ended.

A serial schedule is always consistent. e.g.; If a schedule S has debit transaction T1 and credit transaction T2, possible serial schedules are T1 followed by T2 (T1->T2) or T2 followed by T1 ((T2->T1). The disadvantages of serial schedules are that they have **low throughput** and **less resource utilization**.

Example of serial schedule: Consider the following schedule involving two transactions T_1 and T_2 where R(A) denotes that a read operation is performed on some data item 'A' and W(A) denotes that a write operation is performed on some data item 'A'

Time	**T_1**	**T_2**
t_1	R(A)	
t_2	W(A)	
t_3	R(B)	
t_4		W(B)
t_5		R(A)
t_6		R(B)

In this example, transaction T_1 is performed from time t_1 to $t_{3,}$ then T_2 is performed from time t_4 to $t_{6.}$ This is a serial schedule since the transactions are perform serially in the order T_1 —> $T_{2.}$

3.6. Non-Serial Schedule

A non-serial schedule (concurrent schedule) is a type of scheduling where the operations of multiple transactions are interleaved. In other words, when operations of a transaction are interleaved with operations of other transactions of a schedule, the resultant schedule is called Concurrent schedule. This type of scheduling might lead to a rise in the concurrency problem that were discussed in the previous section. The transactions are executed in a non-serial manner, keeping the end result correct and same as the serial schedule. Unlike the serial schedule where one transaction must wait for another to complete all its operation, in the non-serial schedule, the other transaction proceeds without waiting for the previous transaction to complete. The Non-Serial Schedule can be divided further into Serializable and Non-Serializable Schedule.

3.7. Serializable

Serializable schedule is used to maintain the consistency of the database. It is mainly used in the Non-Serial scheduling to verify whether the scheduling will lead to any inconsistency or not. On the other hand, a serial schedule does not need the serializability because it follows a transaction only when the previous transaction is complete. The non-serial schedule is said to be in a serializable schedule only when it is equivalent to the serial schedules, for an n number of transactions. Since concurrency is allowed in this case thus, multiple transactions can execute concurrently. A serializable schedule helps in improving both resource utilization and CPU throughput. These are of two types:

i. **Conflict serializable**

A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

Recall

Two operations are said to be conflicting if they satisfy the following conditions:

- They belong to different transactions
- They operate on the same data item
- At Least one of them is a write operation

ii. **View serializable**

A Schedule is called view serializable if it is view equal to a serial schedule (no overlapping transactions). A conflict schedule is a view serializable but if the serializability contains blind writes, then the view serializable does not conflict serializable.

A blind write occurs when a transaction writes a value without reading it. For example: In particular, a write $w_i(X)$ is said to be blind if it is not the last action of resource X and the following action on X is a write $w_j(X)$.

3.8. Non-Serializable

The non-serializable schedule is divided into two types, Recoverable and Non-recoverable Schedule. Each is discussed in the next section.

3.9. Recoverable Schedule

Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules. In other words, if some transaction T_j is reading value updated or written by some other transaction T_i, then the commit of T_j must occur after the commit of T_i.

Example of Recoverable Schedule: Consider the following schedule involving two transactions T_1 and T_2.

Time	T_1	T_2
t_1	R(A)	
t_2	W(A)	
t_3		W(A)
t_4		R(A)
t_5	commit	
t_6		commit

This is a recoverable schedule since T_1 commits before T_2, that makes the value read by T_2 correct.

There can be three types of recoverable schedule:

a. Cascading Schedule:

Also called Avoids cascading aborts/rollbacks (ACA). When there is a failure in one transaction and this leads to the rolling back or aborting other dependent transactions, then such scheduling is referred to as Cascading rollback or cascading abort.

Example of Cascading rollback:

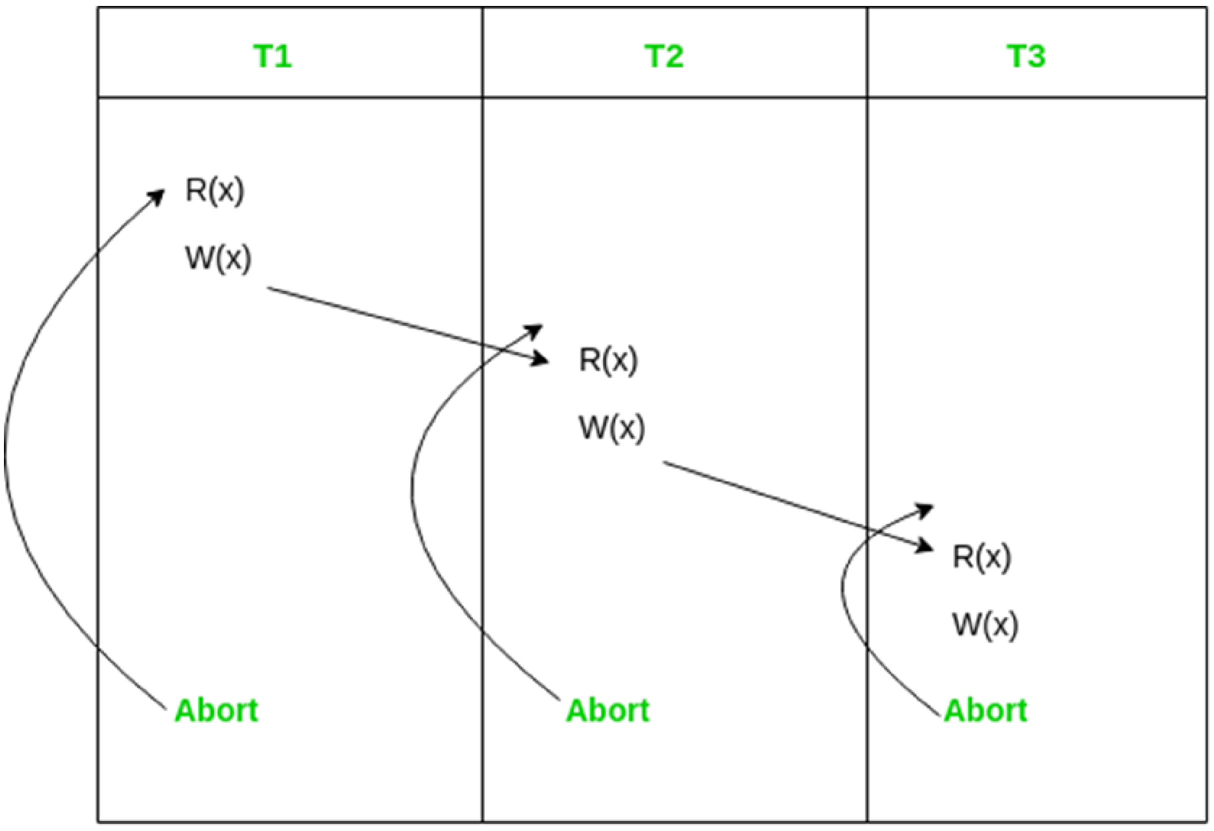


Figure - Cascading Abort

b. Cascadeless Schedule:

Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules. Cascadeless schedules avoids that a single transaction abort leads to a series of transaction rollbacks. A strategy to prevent cascading aborts is to disallow a transaction from reading uncommitted changes from another transaction in the same schedule. In other words, if some transaction T_j wants to read value updated or written by some other transaction T_i, then the commit of T_j must read it after the commit of T_i.

Example Cascadeless schedules:

Consider the following schedule involving two transactions T_1 and T_2.

Time	Time	T_1	T_2
t_1	t_1	R(A)	
t_2	t_2	W(A)	
t_3	t_3		W(A)
t_4	t_4	commit	
t_5	t_5		R(A)
t_6	t_6		commit

This schedule is cascadeless since the updated value of A is read by T_2 at time t_5 only after the updating transaction i.e., T_1 commits.
Example: Consider the following schedule involving two transactions T_1 and T_2.

Time	T_1	T_2
t_1	R(A)	
t_2	W(A)	
t_3		R(A)
t_4		W(A)
t_5	Abort	
t_6		abort

This is a recoverable schedule but it does not avoid cascading aborts. It can be seen that if T_1 aborts, T_2 will have to be aborted too in order to maintain the correctness of the schedule as T_2 has already read the uncommitted value written by T_1.

c. Strict Schedule:

A schedule is strict if for any two transactions T_i, T_j, if a write operation of T_i precedes a conflicting operation of T_j (either read or write), then the commit or abort event of T_i also precedes that conflicting operation of T_j.
In other words, T_j can read or write updated or written value of T_i only after T_i commits/aborts.

Example of Strict Schedule: Consider the following schedule involving two transactions T_1 and T_2.

Time	T_1	T_2
t_1	R(A)	
t_2		R(A)
t_3	W(A)	
t_4	commit	

Time	**T_1**	**T_2**
t_5		**W(A)**
t_6		**R(A)**
t_7		**commit**

This is a strict schedule since T_2 reads and writes A which is written by T_1 only after the commit of T_1.

3.10. Non-Recoverable Schedule

Example of Non-Recoverable Schedule: Consider the following schedule involving two transactions T_1 and T_2.

Time	T_1	T_2
t_1	R(A)	
t_2	W(A)	
t_3		W(A)
t_4		R(A)
t_5		Commit
t_6	abort	

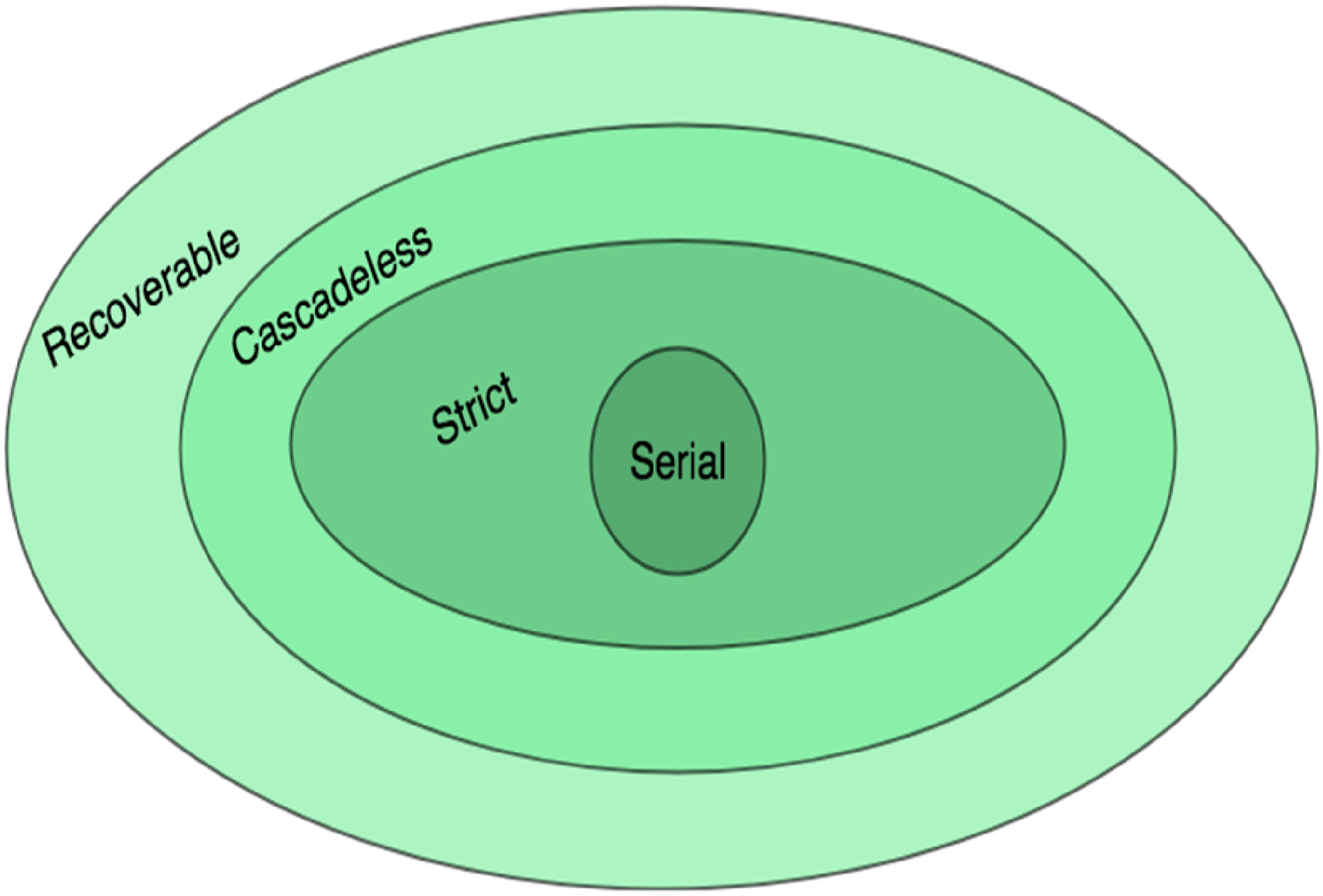
T_2 read the value of A written by T_1, and committed. T_1 later aborted, therefore the value read by T_2 is wrong, but since T_2 committed, this schedule is **non-recoverable**.

Note – It can be seen that:

1. Cascadeless schedules are stricter than recoverable schedules or are a subset of recoverable schedules.
2. Strict schedules are stricter than cascadeless schedules or are a subset of cascadeless schedules.
3. Serial schedules satisfy constraints of all recoverable, cascadeless and strict schedules and hence is a subset of strict schedules.

The relation between various types of schedules can be depicted as:

.



Example: Consider the following schedule:

```
S:R1(A), W2(A), Commit2, W1(A), W3(A), Commit3, Commit1
```

Which of the following is true?

(A) The schedule is view serializable schedule and strict recoverable schedule
(B) The schedule is non-serializable schedule and strict recoverable schedule
(C) The schedule is non-serializable schedule and is not strict recoverable schedule.
(D) The Schedule is serializable schedule and is not strict recoverable schedule

Solution: The schedule can be re-written as: -

Time	**T_1**	**T_2**	**T_3**
t_1	R(A)		
t_2		W(A)	
t_3		Commit	
t_4	W(A)		
t_5			W(A)
t_6			Commit
t_7	Commit		

First of all, it is a view serializable schedule as it has view equal serial schedule T_1 —> T_2 —> T_3 which satisfies the initial and updated reads and final write on variable A which is required for view serializability. Now we can see there is write – write pair done by transactions T_1 followed by T_3 which is violating the above-mentioned condition of strict schedules as T_3 is supposed to do write operation only after T_1 commits which is violated in the given schedule. Hence the given schedule is serializable but not strict recoverable.

So, option (D) is correct.

3.11. Summary

In this session, we have defined a schedule and discussed the various types of schedules including serial and non-serial schedules, different types of non-serial schedules and the schedules that fall under each categorization. We have also given examples of each schedule. In our next session we shall discuss how concurrency control techniques use scheduling to resolve the problems of concurrent transactions. I wish you a fruitful week

3.12. Session 3: Quiz

This end of session quiz will enable you to check your understanding of what you have learned in this Session. Please ensure that you have read all the notes in this section before you start attempt. You must try to obtain a score of at least 50% to proceed to the next session.

Take Note of the following:

- The marks obtained in this quiz will contribute to your Continuous Assessment Tests (CAT).
- You have ONLY ONE (1) attempt for this quiz.
- once you start attempt, the timer will start counting
- Time per question is: 1 minute

When you have finished the quiz, click on ‘Finish attempt and submit..’ to review your ‘Summary of attempt’.

Grading method: Highest grade

Click this to Start Session 3 Quiz now

3.13. References and Further Reading

Core Books

i. Coronel, C., & Morris, S. (2017). *Database Systems: Design, Implementation, & Management* (12th ed.). Boston, MA: Cengage Learning. ISBN: 1305627482.
ii. Elmasri, R., & Navathe, S. B. (2017). *Fundamentals of Database Systems* (7th ed.). Hoboken, NJ: Pearson Education Ltd. ISBN: 0133970779.

iii. Connolly, T. M., & Begg, C. E. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Boston, MA: Pearson Education Ltd. ISBN: 0132943263.

Core Journals

i. Journal of Database Management. ISSN: 1063-8016.
ii. Database Management & Information Retrieval. ISSN: 1862-5347.
iii. International Journal of Information Technology and Database Systems. ISSN: 2231-1807.

Recommended Text Books

i. Hernandez, M. J. (2013). *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design* (3rd ed.). Harlow, UK: Addison-Wesley. ISBN: 0321884493.
ii. Rankins, R., Bertucci, P., Gallelli, C., & Silverstein, A. (2015). *Microsoft SQL Server 2014 Unleashed*. Indianapolis, IN: Sams Publishing. ISBN: 0672337290.
iii. Comeau, A. (2016). *MySQL Explained: Your Step By Step Guide to Database Design*. Bradenton, FL: OSTraining. ISBN: 151942437X.

Recommended Journals

i. International Journal of Intelligent Information and Database Systems. ISSN: 1751-5858.
ii. Database Systems Journal. ISSN: 2069-3230.
iii. Distributed and Parallel Databases. ISSN: 0926-8782.
iv. International Journal of Database Management Systems. ISSN: 0975 - 5985.
v. Journal of Database Management. ISSN:1063-8016.