Unit -8

Introduction to Transaction Processing Concepts and Theory

- Ankit Pargeni

DATE

Introduction to Transaction Processing.

Transaction processing is a logical technique of unit processing that involves one or a collection of database access operations. Transaction processing systems are the systems with large databases & hundreds of concurrent users executing database transactions. Ex:- airline reservations, banking, stock market. A transaction is typically implemented by a computer program that includes database commands such as retrievals, insertions, deletions & update. There are two operations of transaction: Read(X):- to read the X's value from DB server & keeps it in a buffer in main memory & write(X):- to write the X's value back to the DB server from buffer.

Single user vs Multiuser system

Single user	Multi user.
i) A DBMS is a single-user if at most one user at a time can use the system	i) A DBMS is a multi-user if many users can use the system & hence access the database concurrently.
ii) Single-user DBMS are mostly restricted to personal computer systems	ii) Most DBMSs are multiuser like airline, banking DB so on.
iii) The data is neither integrated nor shared among any other user.	iii) Data is integrated & shared among other users
iv) Only one task at a time gets performed.	iv) Schedules different tasks for performance at any rate.
v) It is simple	v) It is complex.

PAGE

DATE

vii)	A super user gets all the powers of maintaining the system & making changes to ensure the system runs smoothly.	vii) Super user doesn't exist when it comes to a multi user system as each entity has control over their working
viii)	Ex: Windows, apple mac.	Ex: UNIX, LINUX

Transactions

A transaction is an executing program that forms a logical unit of database processing. It is typically implemented by a computer program that includes database commands such as retrievals, insertions, deletions & updates. The computer program is usually written in a high-level data-manipulation language (typically SQL) or programming language (C++, or Java).

A transaction is delimited by statements of the form begin transaction and end transaction. If the database operations in a transaction do not update the database but only retrive data, the transaction is called a read-only transaction; Otherwise it is known as a read-write transaction

DATE

* Database Items

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 attribute value of some record in the database.
Each data item has a unique name, but this name is not typically used by the programmer; rather, it is just a means to uniquely identify each data item

* Read and write operation

There are two basic database access operations that a transaction can include

- Read(X):- This read operation is applied to read the X's value from the database server and keeps it in a buffer in main memory
- Write(X):- This write operation is applied to write the X's value back to the database server from the buffer.

Ex:- R(X); X = X-1000; W(X);
Assuming that the value of X before initial transaction is 5000

- The initial operation reads the value of X from the database & supplies it in a buffer.
- The next operation will reduce X's value by 1000. Hence buffer will include 4000.
- Final operation will perform write command to write buffer value to DB. So, X's absolute value will be 4000.

CLASSMATE
PAGE

DATE

* DBMS Buffers

A database buffer is a temporary storage area in the main memory. It allows storing of data temporarily when moving from one place to another. It stores a copy of disk blocks But the version of block copies on the disk may be older than the version in the buffer.

The DBMS maintains a number of data buffers in main memory in the database cache. 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, an additional database disk blocks must be copied into memory. Buffer replacement policies like Least recently used (LRU) are used to choose which of the current occupied buffers is to be replaced.

* Why do we need concurrency control?

⇒ Concurrency control is a procedure in DBMS which helps to manage two simultaneous processes to execute without conflicts between each other. These conflicts occur in multi user systems.

Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access

classmate

PAGE

DATE

The same data, then inconsistency arises. Concurrency control is required to maintain consistency data.
Ex. If we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency.

We need concurrency control for following reasons:

→ To apply isolation through mutual exclusion between conflicting transaction
→ To resolve read-write and write-write conflict issues
→ To preserve database consistency through Constanty preserving execution obstructions.
→ The system needs to control the interaction among the concurrent transactions. This control is achieved using Concurrent-control schemes
→ It helps to ensure serializability.

Problems like The Lost Update problem, the incorrect Summary problem, unrepeatable read problem will occur if we do not control concurrency.

DATE

Why do we need recovery?

=) Database systems, like any other computer system are subject to failures but the data stored in it must be available when required. Some failures like system crash, system or transaction error, concurrency control enforcement, disk failure and physical problems may occur.

For this, the system must keep sufficient information to quickly recover from the failure. It must also have atomicity ie, either transaction are completed successfully and committed or the transaction should have no effect on the database. So, to prevent data loss, recovery techniques based on immediate update or backing up data can be used. The concept of transaction is fundamental to many techniques for concurrency control & recovery from failures.

DATE

Transaction and System Concepts

* Transaction states and additional operation

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 need to keep track of the following operations:

- **BEGIN_TRANSACTION**: This marks the beginning of transaction execution
- **READ or WRITE**: This specifies that READ and WRITE transaction operations have ended & marks the end of transaction execution.
- **END_TRANSACTION**:

- **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 & marks the end of transaction execution.
- **COMMIT_TRANSACTION**: This signals a successful end of the transaction so that any changes executed by the transaction can be safely committed to the DB & won't be undone
- **ROLLBACK (or ABORT)**: This signals that transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the DB must be undone

CLASSMATE PAGE

DATE

Recovery techniques use the following operator

- UNDO:- Similar to rollback except that it applies to a single operation rather than to a whole transaction.

- REDO:- This specifies that certain transactions must be redone to ensure that all the operations of a committed transaction have been applied successfully to the DB

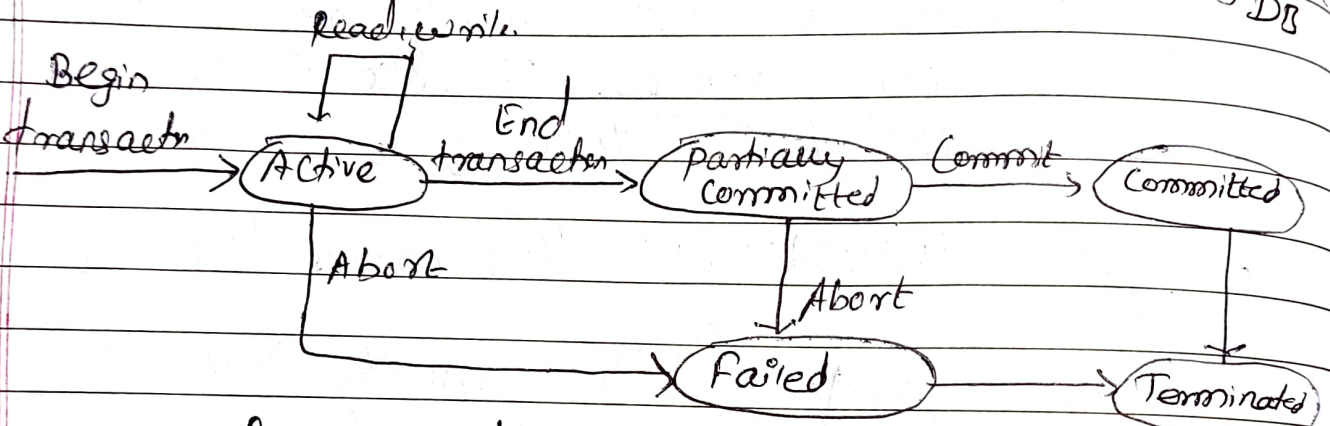


fig: State diagram of transaction

* The System log:

The log keeps track of all transaction operations that affect the values of database items. This information may be needed to permit recovery from transaction failures. The log is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure. In addition, the log is periodically backed up to guard against such failures. The following are the types of entries called log records:

PAGE

DATE

- [start_transaction, T]: Records that transaction T has started execution.
- [write_item, T, X, old_value, new_value]: Records that transaction T has changed the value of database item X from old value to new value.
- [read_item, T, X]: Records that transaction T has read the value of database item X.
- [commit, T]: Records that transaction T has completed successfully, and affirms that its effect can be committed to DB.
- [abort, T]: Records that transaction T has been aborted.

* Commit Point of a Transaction

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 has been recorded in the log. Beyond the commit point, the transaction is said to be committed, & its effect is assumed to be permanently recorded in the database. The transaction then writes an entry [commit, T] into the log.

- Roll back of transactions: Needed for transactions that have a [start_transaction, T] entry into the log but no commit entry [commit, T] into the log.
- Force writing a log: Before the transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk. This process is called force-writing the log file before committing a transaction.

PAGE

classmate

DATE

* Buffer replacement policies.

DBMS cache will hold the disk pages that contain information currently being processed in main memory buffers. If all buffers in the DBMS cache are copied & new disk pages are required to be loaded into main memory from disk, a new page replacement policy is needed to select particular buffers to be replaced.

1. Domain Separation (DS) Method:
   Here, DBMS cache is divided into separate domains (set of buffers). Each domain handles one type of disk pages, & page replacements within each domain are handled via the basic LRU.

2. Hot set method:
   This is useful in queries that have to scan a set of pages repeatedly, such as when a join operation is performed using the nested loop method. It doesn't replace disk pages until their processing is complete.

3. The DBMIN method.
   This policy uses a model known as QLSM (query locality set model), which predetermines the pattern of page references for each algorithm for a particular type of database operation.

classmate

PAGE

DATE

Desirable properties of transactions: (ACID properties)

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:

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

- **Consistency preservation:** A correct execution of the transaction must take the database from one consistent state to another.

- **Isolation:** A transaction should not make its updates visible to other transactions until it is committed; this property, when enforced strictly, solves the temporary update problem & makes cascading rollbacks of transactions unnecessary.

- **Durability or Permanency:** Once the transaction changes the database & the changes are committed, these changes must never be lost because of subsequent failure.

PAGE

classmate

DATE

Characterizing Schedules Based on Serializability Recoverability

* Concept of schedule

When several transactions are executing concurrently then the order of execution of various instructions is known as schedule. They represent the chronological order in which instructions are executed in the system.
A schedule can have many transactions in it, each comprising of a number of instructions/tasks

* Characterizing schedules based on Recoverability

It may be easy or difficult to recover transaction from system failure. The different types of schedules based on recovability are

1. Recoverable Schedule.

A schedule S is recoverable if no transaction T in S commits until all transactions T' that have written an item that T reads have committed

Here, transaction T_2 is reading value written by transaction T_1 and the commit of T_2 occurs after commit of T_1. Hence, it is a recoverable schedule

T_1	T_2
R(X)	
W(X)	
	W(X)
Commit	R(X)
	Commit

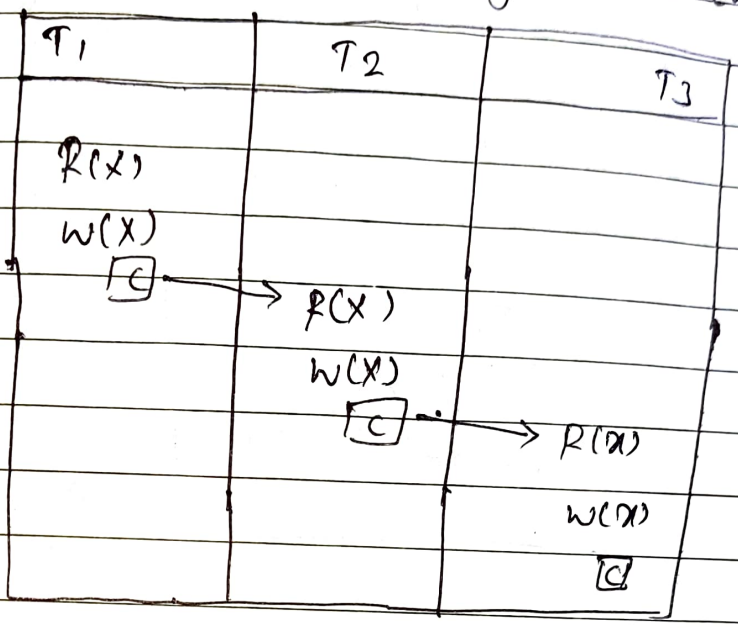
classmate

PAGE

DATE

2. Cascadeless schedule

It is the one where every transaction reads only the items that are written by committed transaction.



3. Schedules requiring cascaded rollback:-

A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back. If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a cascading Rollback.

T1	T2	T3
R(A)		
W(A)		
	R(A)	
	W(A)	
		R(A)
		W(A)
Failure		

It simply leads to the wastage of CPU time.

DATE

4. Strict schedule.

If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as strict schedule

T_1	T_2
W(A)	
Commit / Roulback	
	R(A) / W(A)

Types of Schedules:

1. Serial Schedule

A schedule in which transactions are aligned in such a way that one transaction is executed 1st. When the first transaction completes its cycle then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called Serial schedule as transactions are executed in a Serial manner. It is always a Serializable Schedule.

T_1	T_2
R(A)	
A:=A-50	
W(A)	
R(B)	
B:=B+50	
W(B)	
	R(A)
	temp: A*0.1
	W(A)

DATE

2. Non-Serial Schedule:

A schedule S is called non-serial if for any two transactions T_i and T_j participating in S, the operation of each transaction are executed non-consecutively with interleaved operations from the other transaction.

T_1	T_2
R(A)	
A:=A-50	
W(A)	
	R(A)
	Temp:=A*0.1
	A:A-temp
	W(A)
R(B)	
B:=B+50	
W(B)	
	R(B)
	B:=B+temp
	W(B)

PAGE

DATE

Characterizing Schedules Based on Serializability

A serializable schedule always leaves the database in consistent state. A concurrent execution of N transaction is called serializable, if the execution is computationaly equivalent to a serial execution. When more than one transactions are being executed concurrently, we must have serializability in order to have same effect on the database as same serial execution does.

There are two types of schedule based on serializability

- Conflict serializable.
- View serializable.

Conflict serializable

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

Two operations are said to be conflicting if all conditions satisfy:

i) Both operations should belong to a different transactions.
ii) They should act on same database variable.
iii) At least one of the operation should be write

DATE

Testing for conflict serializability of a schedule:

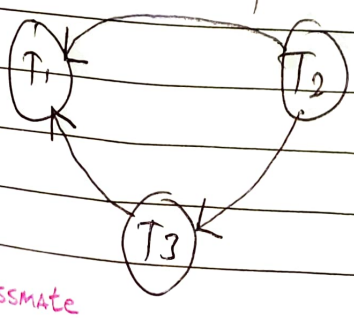
Algorithm:

- Look at only read_item(X) and write_item(X) operations
- Construct a precedence graph - a graph with directed edges
- An edge is created from T_i to T_j if one of the operation in T_i appears before a conflicting operation in T_j.
- The schedule is serializable iff the precedence graph has no cycles

Ex: Check whether given schedule is conflict serializable or not.

T_1	T_2	T_3
R(x)		
		R(y)
		R(x)
	R(y)	
	R(z)	
		W(y)
	W(z)	
R(z)		
W(x)		
W(z)		

Let's construct precedence graph. We check conflict pairs in other transactions & draw edges



Since, there is not a loop/cycle, so it is a conflict serializable. So, it is serializable and it is consistent also.

classmate

Sequence is: $T_2 \rightarrow T_3 \rightarrow T_1$

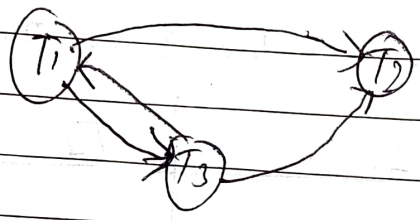
PAGE

DATE

Ex:

T_1	T_2	T_3
R(x)		
		R(z)
		W(z)
	R(y)	
R(y)		
	W(y)	
		W(x)
	W(z)	
W(x)		

=>



Since, it has cycle. So it is non-conflict serializable.

DATE

2. View serializable

A schedule is called view serializable if its view (equivalent) equal to a serial schedule (no overlapping transactions). A conflict schedule is a view serializable but if the serializability contains blind write, then the view serializable is not conflict serializable.

View Equivalence:-

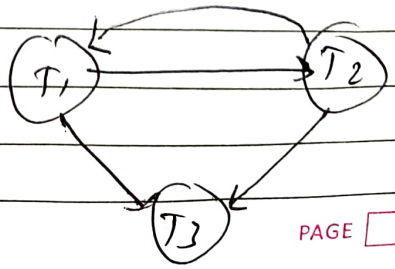
Two schedules A & B are said to be view equivalent if they follow the following condition:

- 1st read should be performed by same transaction.
- Last write should be performed by same transaction
- Producer-Consumer sequence should be maintained in both S_1 and S_2: ie: If $(W(A))$ followed by $T_2(R(A))$ in S_1 and $T_1(W(A))$ followed by $T_2(R(A))$ in S_2

Ex S_1

T_1	T_2	T_3
R(x)		
	W(x)	
	W(y)	
W(x)		
		W(x)
		R(y)

⇒ It's precedence graph:



DATE

It Contains loop, so it is not conflict serializable. But it may be view serializable. Let's check it further

S_1

T_1	T_2	T_3
R(x)		
	W(x)	
	W(y)	
W(x)		
		W(x)
		R(y)

S_1'

T_1	T_2	T_3
	R(x)	
	W(x)	
W(x)		
W(y)		
		W(x)
		R(y)

We draw the equivalent serial schedule of S_1 as S_1'

Here S_1 and S_1' satisfies all 3 view equivalent conditions

- S_1 - R1(x) and S_1' - R1(x) (you can check y also)
- S_1 - W3(x) and S_1' - W3(x)
- S_1 - W2(y) & consumed by S_1 - R3(y)
  S_1' - W2(y) & consumed by S_1' - R3(y)

Thus the given S_1 is view serializable

Note: If it has blind write, then it may be view ser but if it hasn't, then it can't be view serializable

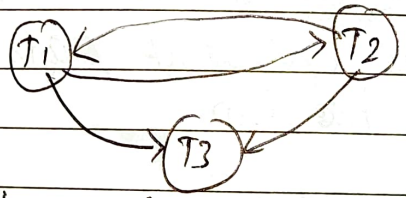
S		
T_1	T_2	T_3
	R(a)	
	W(b)	
R(b)		
		R(b)
W(a)		
	W(a)	
		W(a)

Note: If it is conflict serializable, then it is also view serializable

Q. Explain whether the below schedule is

i) Conflict Serializable
ii) View serializable
iii) Conflict serializable but not view serializable
iv) View serializable but not conflict serializable.

i) Let's draw precedence graph



It has a loop, so it is not conflict serializable.

ii) Let's draw equivalent serial schedule of S as S'

S		
T_1	T_2	T_3
	R(a)	
	W(b)	
R(b)		
		R(b)
W(a)		
	W(a)	
		W(a)

S'		
T_1	T_2	T_3
	R(a)	
	W(b)	
	W(a)	
R(b)		
W(a)		
		R(b)
		W(a)

- $S - R_2(a)$ and $S' - R_2(a)$ (you can check others too)
- $S - W_3(a)$ and $S' - W_3(a)$
- $S - R_2(b)$ ✗ Consumed by $S' - R_1(b)$
  $S' - W_2(b)$ ✗ Consumed by $S' - R_1(b)$

Here $S \not\equiv S'$ Satisfies all 3 view equivalent conditions
Hence, S is view Serializable

iii) Hence, it is not conflict Serializable but view Serializable (as it has blind write also)

iv) It is view serializable but not conflict serializable because $S \not\equiv S'$ are view equivalent and the precedency graph of S has loop

Questions asked from this chapter

Q. Define schedule and serializability. How can you test the serializability?
(2078 - 5 marks) (2076 - 5 marks)

Q. What is Serializable schedule? How can you test a schedule for conflict serializability?
(2070 - 5 marks)

Q. Discuss ACID properties of a database transaction with suitable example. (2067 - 5 marks)
(2072 - 5 marks) (2068 - 5 marks)

Q. Describe the serial & serializable schedule? Why serializable schedule is consider correct?
(2069 - 5 marks, 2067 - 5 marks)

Q. Define the concept of recoverable, cascadeless & strict schedule, & compare them in terms of their recoverability. (2068 - 5 marks)

Q. Draw a state diagram, & discuss the typical state that a transaction goes through during transaction. (2069 - 5 marks) (2067 - 5 marks)

Q. Which of the following is conflict serializable? For each serializable schedule, draw equivalent serial schedule

i) $r_1(x)$; $r_3(x)$; $w_1(x)$; $r_2(x)$; $w_3(x)$;

ii) $r_1(x)$; $r_3(x)$; $w_3(x)$; $w_1(x)$; $r_2(x)$;

iii) $r_3(x)$; $r_2(x)$; $w_3(x)$; $r_1(x)$; $w_1(x)$;

iv) $r_3(x)$; $r_2(x)$; $r_1(x)$; $w_3(x)$; $w_1(x)$;
(2067 - 5 marks)