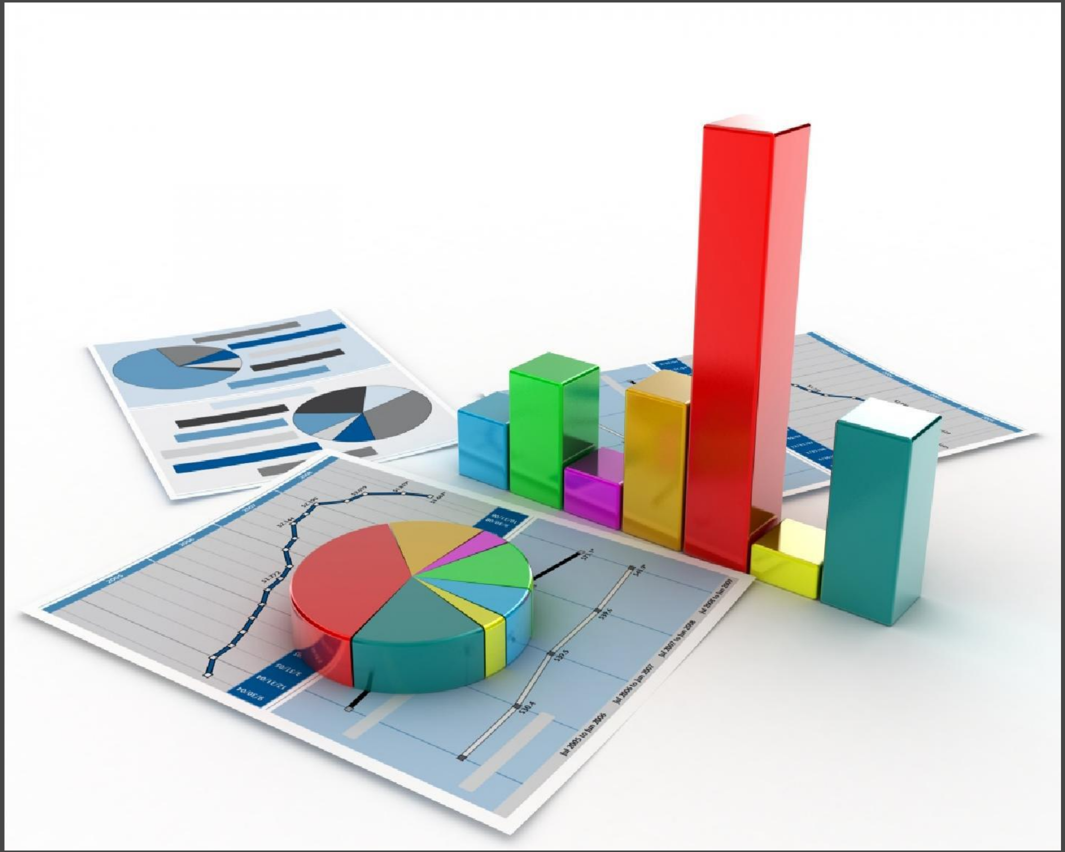


ADVANCE DATABASE SYSTEMS DEVELOPMENT [CC6001] WEEK - 06

Database Concurrency

There are three sides of ACID.

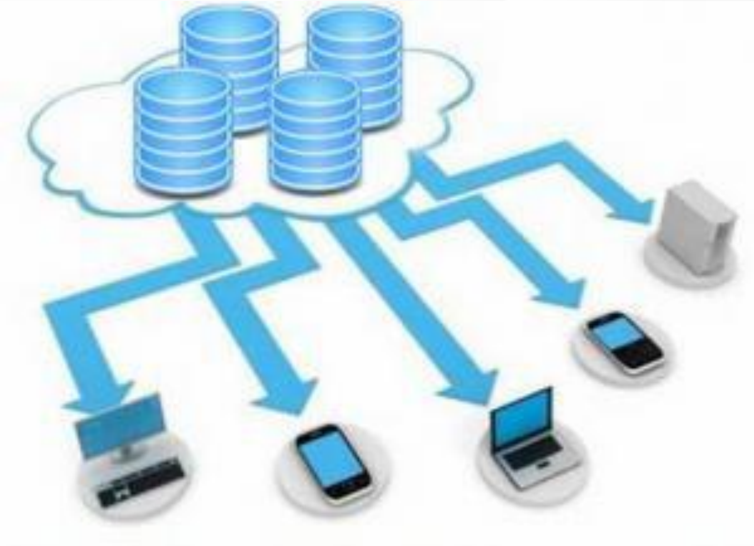
ENHANCED LONG TERM MEMORY

DECREASED SHORT TERM MEMORY

AND I FORGOT THE THIRD

- TIMOTHY LARY

Requirement

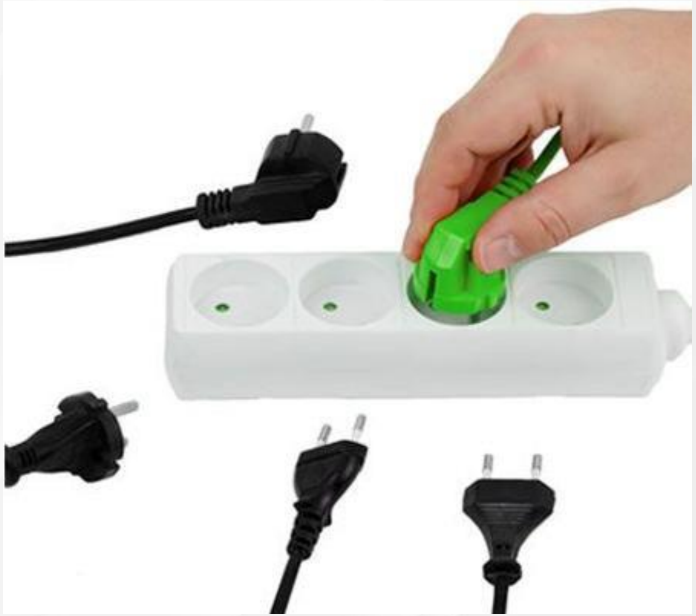


MOST OF THE CASES, DATABASE IS ACCESSED BY MULTIPLE USERS AT A TIME

Concurrency Control

ENSURES THAT CONCURRENT OPERATIONS ARE CARRIED OUT

- CORRECTLY
- EFFICIENTLY



Example

- Begin Transaction T1
- Read Balance1
- Balance1 = Balance1 – 100
- If Balance <0
  - Print insufficient fund
  - Abort T1
- End
- Write Balance1
- Read Balance2
- Balance2 = Balance2 +100
- Write Balance2
- Commit T1

Points to Remember

Effect of ABORT is to ROLLBACK the TRANSACTION and UNDO changes it has made on the DATABASE

In this example, TRANSACTION was NOT written to the DATABASE prior to ABORT hence, NO UNDO is necessary

Yet Another Scenario

Transactions T_A and T_B performs a series of processing (sub-tasks) T_{A1}, T_{A2}, T_{A3} and T_{B1}, T_{B2}, T_{B3} respectively.

CPU is shared by T_A and T_B and other transactions in the system

Yet Another Scenario

Single Transaction Environment

Multi Transaction Environment

$T_{A1} \longrightarrow T_{B1} \longrightarrow T_{A2} \longrightarrow T_{B2} \longrightarrow T_{A3} \longrightarrow T_{B3}$

$T_{A1} \longrightarrow T_{A2} \longrightarrow T_{A3}$

Interleaved Transactions

The operations of the two transactions T_A and T_B are said to be **interleaved** to achieve concurrent execution.

$$T_{A1} \longrightarrow T_{B1} \longrightarrow T_{A2} \longrightarrow T_{B2} \longrightarrow T_{A3} \longrightarrow T_{B3}$$

The operations of the two transactions T_A and T_B are said to be **interleaved** to achieve concurrent execution.

Various concurrency problems may occur if proper control is not enforced!!!

Concurrency Problems

- Lost Update
- Violation of Integrity Constraints
- Inconsistent Retrieval

Lost Update

two concurrent transactions, say T_A and T_B, are allowed to update an uncommitted change on the same data item, say x.

The earlier update of x by Transaction T_A is overwritten by a subsequent update of x by transaction T_B.

Consequently, the update value of x by T_A is lost after the update by T_B.

Lost Update - Example

Time	T_A	T_B	Balance
			500
t1	begin		500
t2	read (balance)	begin	500
t3	balance=balance+200	read (balance)	500
t4	write (balance)	balance=balance-300	700
t5	commit	write (balance)	200
t6		commit	200

Lost Update - Question

What should be the correct balance after T_A and T_B?

Lost Update - Answer

400

Violation of Integrity Constraints

Time	T_A	T_B	Balance
			500
t1	begin		500
t2	read (balance)		500
t3	balance=balance+200		500
t4	write (balance)	begin	700
t5	…….	read (balance)	700
t6	rollback	balance=balance-300	500
t7		write (balance)	400
t8		commit	400

Integrity Constraint - Question

What should be the correct balance after T_A and T_B?

Integrity Constraint - Answer

200

Inconsistent Retrieval

T_A is summing up the total number of vacant teaching rooms in 3 building (Tower, Moorgate and Eden), whilst T_B is moving 2 classes (2 rooms) from Tower to Eden.

Initial numbers of vacant

rooms: Tower: 10

Moorgate: 15

Eden 5

Total number 30

Inconsistent Retrieval

Tower	10	Moorgate	15	Eden	5

Time	T_A	T_B
t1	....	
t2	read (Tower)	
t3	sum = 10	
t4	read (Moorgate)	
t5	sum = 25 (sum+15)	......
t6		read (Tower)
t7		write Tower = 8 (10-2)
t8		read (Eden)
t9		Write Eden = 7 (5+2)
t10		commit
t11	read (Eden)	
t12	sum = 32 (sum+7)	
t13	......	

Inconsistent Retrieval

- Initial numbers of vacant rooms:
  - Tower: 10
  - Moorgate 15
  - Eden 5
- Correct numbers of vacant rooms after moving should be:
  - Tower: 8 (10 – 2)
  - Moorgate: 15
  - Eden 7 (5+2)
- But we seem to have got 32 vacant rooms now!!

WRONG TOTAL!


PROBLEM
SOLUTION

Concurrency Control - Principles

Schedule multiple transactions in such a way as to avoid any interference between them, while at the same time

Maximize the degree of concurrency or parallelism in the system

Solutions to Concurrency Problems

- Serialization
- Concurrency Control
- Transaction Scheduling

Serialization

Process

- Permits SERIAL EXECUTION of TRANSACTIONS

Disadvantage

- Inefficient for MULTI-USER environment

ONE TRANSACTION MUST COMMIT BEFORE ANOTHER CAN START.

Concurrency Control

Process

- Allows CONCURRECT EXECUTION of transactions in a CONTROLLED way.

Mechanisms

- Locking
- Optimistic Scheduling
- Time Stamping

Locking

Locking method is the most commonly used approach to ensure the serializability of concurrent transactions.

Locking synchronizes interleaved transactions so that it is **equivalent** to some serial execution of those transactions.

Locking - Idea

When a transaction requires the **assurance** that some data will not change during the course of processing (read/write) it, this transaction must request and acquire a lock, in order to prevent other transactions updating the data.

Lock Types

Exclusive

- X (Write)
- Grant read/write access to a data item to the transaction which holds the lock
- Prevent any other transactions reading or writing the same data item.

Shared

- S (Read)
- Grant read-only access to a data item in the transaction which holds the lock
- Prevent any transaction writing data item.
- Several transactions may hold an S lock on the same data item.

Locking Protocol

- A transaction must get an S lock on a data item before it wishes to READ it
- A transaction must get an X lock on a data item before it wishes to WRITE it
- A TRANSACTION already holding an S lock on a data item can promote itself to an X lock in order to WRITE it, provided there is no other transaction also holding an S lock on it already.
- If a lock request is denied, the transaction goes into a WAIT state
- A transaction in a WAIT state resumes operations only when the requested lock is released by another transaction
- X locks are held until COMMIT/ROLLBACK. S locks are normally the same.

Lock Compatibility Matrix

	S	X
S	true	false
X	false	false

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions

Lock Compatibility Matrix

	S	X
S	true	false
X	false	false

- Any number of transactions can hold shared locks on an item
  - But if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.

Lock Compatibility Matrix

	S	X
S	true	false
X	false	false

- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

Example

- Locking as above is not sufficient to guarantee serializability If A and B get updated in-between the read of A and B, the displayed sum would be wrong.
- A locking protocol is a set of rules

Followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

T_2: **lock-S**(*A*);
read (*A*);
unlock(*A*);
lock-S(*B*);
read (*B*);
unlock(*B*);
display(*A*+*B*)

Two Phased Locking

Protocol

Before operating on any object (a tuple), a transaction must acquire a lock on that object

After releasing a lock a transaction must never go on to acquire any more locks.

Theorem

If all transactions obey the "two–phased locking protocol", then all possible interleaved schedules are serializable

Lost Update – Example Revisited

Time	T_A	T_B	Balance
			500
t1	begin *acquire S lock on balance*		500
t2	read (balance)	begin *acquire S lock on balance*	500
t3	balance=balance+200 *X lock requested but denied*	read (balance)	500
t4	*wait* …… …… ……	balance=balance-300 *X lock requested but denied* *wait* ……	500

Lost Update – Example Revisited

Time	T_A	T_B	Balance
			500
t1	begin *acquire S lock on balance*		500
t2	read (balance)	begin *acquire S lock on balance*	500
t3	balance=balance+200 *X lock requested but denied*	read (balance)	500
t4	*wait*	balance=balance-300 *X lock requested but denied* *wait*	500

- neither transaction can update balance as both are waiting for other to release the lock on balance

Relational Integrity – Example Revisited

Time	T_A	T_B	Balance
			500
t1	begin		500
	acquire S lock on balance		
t2	read (balance)		500
t3	balance=balance+200		500
	promote to X lock on balance		
t4	write (balance)	begin	700
		S lock requested but denied	
	…….	*wait*	
		…….	
t5	rollback	…….	500
	release X lock on balance	…….	
		acquire S lock on balance	
t6		read (balance)	500
t7		balance=balance-300	500
		promote to X lock on balance	
t8		write (balance)	200
t9		commit	200
		release X lock on balance	

Relational Integrity – Example Revisited

Time	T_A	T_B	Balance
			500
t1	begin		500
	acquire S lock on balance		
t2	read (balance)		500
t3	balance=balance+200		500
	promote to X lock on balance		
t4	write (balance)	begin	700
		S lock requested but denied	
	…….	*wait*	
		…….	
t5	rollback	…….	<u>500</u>
	release X lock on balance	…….	
		acquire S lock on balance	
t6		read (balance)	500
t7		balance=balance-300	500
		promote to X lock on balance	
t8		write (balance)	200
t9		commit	<u>200</u>
		release X lock on balance	

CORRECT ANSWER

Thank You