
DATABASE

CSE3103 : Database
FALL 2020

Nazmus Sakib
Assistant Professor
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

Concurrent Executions

- Multiple transactions are allowed to run concurrently in the system.
- Advantages are:
  - **Increased processor and disk utilization**, leading to better transaction *throughput*
  - **Reduced average response time** for transactions: short transactions need not wait behind long ones.
- **Concurrency control schemes** – mechanisms to achieve isolation
  - That is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

Schedules

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - A schedule for a set of transactions must consist of all instructions of those transactions.
  - Must preserve the order in which the instructions appear in each individual transaction.
- A transaction that successfully completes its execution will have a **commit** instructions as the last statement
  - By default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an **abort** instruction as the last statement

Schedule 1

- T_1 transfer \$50 from A to B, and
- T_2 transfer 10% of the balance from A to B.
- An example of a **serial** schedule in which T_1 is followed by T_2 :

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

T1	T2
A=100	A=50
A=100-50	TEMP=50*10%=5
A=50	A=50-TEMP =45
B= 200	B= 250
B= 200+50	B= 250+5
B=250	B=255

After Schedule 1	
A = 45	B= 255

Schedule 2

- T_1 transfer \$50 from A to B, and
- T_2 transfer 10% of the balance from A to B.
- An example of a **serial** schedule in which T_2 is followed by T_1 :

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	

After Schedule 1	
A = 45	B= 255

T1	T2
A=90	A=100
A=90-50	TEMP=100*10%=10
A=40	A=100-TEMP =90
B= 210	B= 200
B= 210+50	B= 200+10
B=260	B=210

After Schedule 2	
A = 40	B= 260

Schedule 3

- Let T_1 and T_2 be the transactions defined previously.
- The following schedule is not a serial schedule,
- but it is **equivalent** to Schedule 1.

T_1	T_2
read (A) $A := A - 50$ write (A)	
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A)
read (B) $B := B + 50$ write (B) commit	
	read (B) $B := B + temp$ write (B) commit

Note -- In schedules 1, 2 and 3, the sum "A + B" is preserved.

After Schedule 1	
A = 45	B= 255

After Schedule 2	
A = 40	B= 260

T1	T2
A=100	A=50
A=100-50	TEMP=50*10%=5
A=50	A=50-TEMP =45
B= 200	B= 250
B= 200+50	B= 250+5
B=250	B=255

After Schedule 3	
A = 45	B= 255

Schedule 4

- Let T_1 and T_2 be the transactions defined previously.
- The following schedule is not a serial schedule,

Note -- In schedules 4 the sum "A + B" is not preserved.

T_1	T_2
read (A) $A := A - 50$	
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B)
write (A) read (B) $B := B + 50$ write (B) commit	
	$B := B + temp$ write (B) commit

After Schedule 1	
A = 45	B= 255

After Schedule 2	
A = 40	B= 260

After Schedule 3	
A = 45	B= 255

T1	T2
A=100	A=100
A=100-50	TEMP=100*10%=10
A=50	A=100-TEMP =90
B= 200	B= 200
B= 200+50	B= 200+10
B=250	B=210

After Schedule 4	
A = 50	B= 210


DBMS
data
database
distributed
parallel
shared
system
architecture
systems
network
processing
performance
access
centralized
processors
DBMSs
Distributed
DDBMS
multiple
sites
memory
applications
computers
nothing
Shared
site
require
physically
users
within
machines
like
Database
control
scalability
approach
improve
distinction
logical
databases
development
definition
nodes
technology
capable
computer
disk
stored
logically
global
associated
provides
collection
structure
incompatible
way
consists
partitioned
used
least
must
System
processor
also
Parallel
many
replicated
provide
make
may
major
fact
fragments
typically
across
reliability
eliminates
distribution
communication
sometimes
computing
single
one
split
order
scalable
architectures
operational
referred
local
decentralized
directly
com
Date
disks
large
islands
resources
remote
defined
controlled
based
often
Management
number
dis
user
bottleneck
interconnection
Figure
administered
mode
organizational
application
management
transparent
makes
per
discuss
independently