TRANSACTIONS & CONCURRENCY CONTROL

Adapted from K. Goldberg (UC Berkeley) and Ramakrishnan & Gherke

1

Introduction

2

- Concurrent execution of user programs is essential for good DBMS performance.
- Because disk accesses are frequent, and relatively slow, it is important to keep the CPU going by working on several user programs concurrently.

2

Transactions

3

- A user's program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned about what data is read/written from/to the database.
- A *transaction* is the DBMS's abstract view of a user program: a sequence of reads and writes.

3

Concurrency

4

- What is Concurrent Process (CP)?
  - Multiple users access databases and use computer systems simultaneously.
- Example: Airline reservation system.
  - An airline reservation system is used by hundreds of travel agents and reservation clerks concurrently.
  - Banking system: you may be updating your account balances the same time the bank is crediting you interest.

4

Concurrency

5

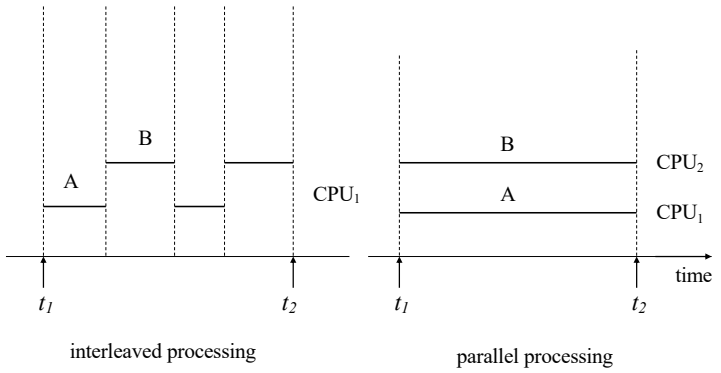
- Why Concurrent Process?
- Better transaction throughput and response time
- Better utilization of resource

- Concurrency
  - **Interleaved processing:**
    - Concurrent execution of processes is interleaved in a single CPU
  - **Parallel processing:**
    - Processes are concurrently executed in multiple CPUs.

5

Concurrent Transactions

6



6

Transactions

7

- What is a transaction?

  A sequence of many actions which are considered to be one unit of work.

  Basic operations a transaction can include "actions":
  - Reads, writes
  - Special actions: commit, abort

7

Concurrency (cont'd)

8

- Users submit transactions, and can think of each transaction as executing by itself.
  - Concurrency is achieved by the DBMS, which interleaves actions (reads/writes of DB objects) of various transactions.
  - Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins.
    - DBMS will enforce some constraints
    - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
- *Issues:* Effect of *interleaving* transactions, and *crashes.*

8

Atomicity of Transactions

9

- A transaction might *commit* after completing all its actions, or it could *abort* (or be aborted by the DBMS) after executing some actions.
- A very important property guaranteed by the DBMS for all transactions is that they are *atomic.* That is, a user can think of a Xact as always executing all its actions in one step, or not executing any actions at all.
  - DBMS *logs* all actions so that it can *undo* the actions of aborted transactions.

9

Transaction Properties (ACID)

10

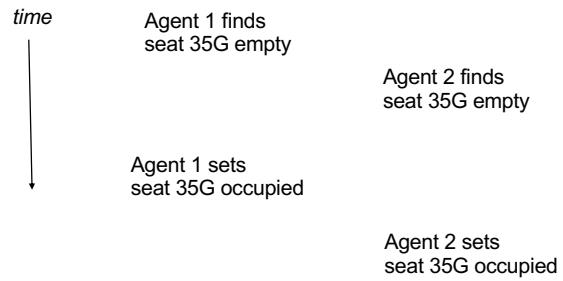
- Atomicity
  - Transaction is either performed in its entirety or not performed at all, this should be DBMS' responsibility
- Consistency
  - Transaction must take the database from one consistent state to another.
- Isolation
  - Transaction should appear as though it is being executed in isolation from other transactions
- Durability
  - Changes applied to the database by a committed transaction must persist, even if the system fails before all changes reflected on disk

10

Oops, Something's Wrong

11

- Reserving a seat for a flight
- In concurrent access to data in DBMS, two users may try to book the same seat simultaneously



11

Example

12

- Consider two transactions (*Xacts*):

```
T1:   BEGIN  A=A+100,   B=B-100   END
T2:   BEGIN  A=1.06*A,  B=1.06*B  END
```

- Intuitively, the first transaction is transferring $100 from B's account to A's account. The second is crediting both accounts with a 6% interest payment.
- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect *must* be equivalent to these two transactions running serially in some order.

12

Example (Contd.)

13

- Consider a possible interleaving (*schedule*):

T1:	A=A+100,		B=B-100
T2:		A=1.06*A,	B=1.06*B

- This is OK. But what about:

T1:	A=A+100,		B=B-100
T2:		A=1.06*A, B=1.06*B	

- The DBMS's view of the second schedule:

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	

13

What Can Go Wrong?

14

- Concurrent process may end up violating Isolation property of transaction if not carefully scheduled
- Transaction may be aborted before committed
  - undo the uncommitted transactions
  - undo transactions that see the uncommitted change before the crash

14

Schedule

15

- A schedule S of n transactions T1,T2,…Tn is an ordering of the operations of the transactions subject to the constraint that,
  - for each transaction Ti that participates in S, the operations of Ti in S must appear in the same order in which they occur in Ti.
- Informally, a schedule is a sequence of interleaved actions from all transactions
- Example:

S_a: R1(A),R2(A),W1(A),W2(A), Abort1,Commit2;

T1	T2
Read(A)	
	Read(A)
Write(A)	
	Write(A)
Abort T1	
	Commit T2

15

Scheduling Transactions

16

- *Serial schedule:* Schedule that does not interleave the actions of different transactions.
- *Equivalent schedules:* For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.
- *Serializable schedule:* A schedule that is equivalent to some serial execution of the transactions.

16

Serial Schedule

17

T1	T2
R(A)	
A := A+100	
R(B)	
B := B+100	
	R(A)
	A: = A* 2
	R(B)
	B:= B*2

S: R1(A),W1(A),R1(B), W1(B), R2(A), W2(A), R2(B), W2(B)

T1 T2

Adapted from M. Balazinska

17

A Serializable Schedule

18

T1	T2
R(A)	
A := A+100	
	R(A)
	A: = A* 2
R(B)	
B := B+100	
	R(B)
	B:= B*2

Notice: this is not a serial schedule, i.e., there is interleaving of operations

Net effect is the same as the serial schedule

S: R1(A),W1(A), R2(A), W2(A), R1(B), W1(B), R2(B), W2(B)

18

A Non-Serializable Schedule

19

T1	T2
R(A)	
A := A+100	
	R(A)
	A: = A* 2
	R(B)
	B:= B*2
R(B)	
B := B+100	

What's different?

S: R1(A),W1(A), R2(A), W2(A), R2(B), W2(B), R1(B), W1(B)

19

Conflict operations

20

- Two operations in a schedule are said to be *conflict* if they satisfy all three of the following conditions:
  (1) They belong to different transactions
  (2) They access the same item A;
  (3) At least one of the operations is a write(A)

Example in Sa: R1(A), R2(A), W1(A), W2(A), A1, C2;

- R1(A),W2(A) conflict, so do R2(A),W1(A),
- R1(A), W1(A) do not conflict because they belong to the same transaction,
- R1(A),R2(A) do not conflict because they are both read operations.

20