Introduction to transaction processing

An interesting experiment

```
$sqlplus jrg@csci
```

```
$sqlplus jrg@csci
```

```
SQL> SELECT COUNT(*)
  2  FROM SKILL;

  COUNT(*)
----------
        19
```

```
SQL> SELECT COUNT(*)
  2  FROM SKILL;

  COUNT(*)
----------
        19
```


An interesting experiment

```
SQL> INSERT INTO SKILL
  2  VALUES('singing');

1 row created.
```

```
SQL> SELECT COUNT(*)
  2  FROM SKILL;

  COUNT(*)
----------
        20
```

```
SQL> SELECT COUNT(*)
  2  FROM SKILL;

  COUNT(*)
----------
        19
```


An interesting experiment

```
SQL> COMMIT;

Commit complete.
```

```
SQL> SELECT COUNT(*)
  2  FROM SKILL;

  COUNT(*)
----------
        20
```

```
SQL> SELECT COUNT(*)
  2  FROM SKILL;

  COUNT(*)
----------
        20
```


Transaction ? What is it ?

A partially ordered set of *read*, *write* operations on the database items is called as a transaction

Principles of transaction processing

Users interact with the database by executing programs

Execution of a program is equivalent to execution of a partially ordered set of *read*, *write* operations

Database is visible to transactions as a collection of data items

Concurrently running transactions interleave their operations

Transactions have no impact on execution of their operations

Principles of transaction processing

Each transaction terminates by either *commit* or *abort* operation

Each transaction arrives at a consistent database state and must leave a database in a consistent state as well

So, where is a problem ?

T_1	T_2	x = $100
read(x)		x = $100
	read(x)	x = $100
write(x,x-10)		x = $90
	write(x,x+20)	x = $120
	commit	x = $120
commit		x = $120

Correctness condition

What makes concurrent execution of database transaction incorrect ?

How do we define a correct concurrent execution of database transactions ?

Conflicting operations

	read	write
read	NO	YES
write	YES	YES

T_1	T_2
write(x, x-10)	
	read(x)
read(y)	
write(y,y+10)	
	read(y)

Conflict serializability condition

Concurrent execution of database transactions is <u>conflict serializable</u> if there exists a possible serial execution of the same set of transactions such that in both executions <u>the order of conflicting operations</u> is the same

Conflict serializable execution

T_1	T_2
read(x)	
write(x, x-10)	
	read(x)
read(y)	
write(y,y+10)	
	read(y)

Order of conflicting operations: T_1 before T_2

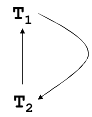
Conflict nonserializable execution

T_1	T_2	x = $100
read(x)		x = $100
	read(x)	x = $100
write(x,x-10)		x = $90
	write(x,x+20)	x = $120
	commit	x = $120
commit		x = $120

Order of conflicting operations:
T_1 before T_2 and T_2 before T_1 **impossible to serialize**

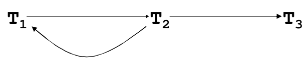
Serialization graph

T_1	T_2
read(x)	
	read(x)
write(x,x-10)	
	write(x,x+20)



Serialization graph

T_1	T_2	T_3
write(x, 10)		
	write(x, 20)	
		write(x,30)
	write(y,10)	
read(y)		



Serialization graph testing protocol (SGT)

Principles

Scheduler maintains and tests serialization graph

If an operation issued by a transaction violates conflict serializability (i.e. it creates a cycle in serialization graph) then such transaction is aborted

Problems

Cascading aborts

performance (testing acyclicity of serialization graph has $O(n^2)$ complexity)

Two-phase locking (2PL) protocol

Principle

Each transaction must acquire all locks before releasing any lock

Problems

Deadlocks

Unnecessary locks when execution is conflict serializable

Two-phase locking (2PL) protocol

T_1	T_2
lock(u) read(u)	
	lock(v) write(v,v+1)
write(u,u+2)	
lock(v) wait	
	lock(x) read(x)
	unlock(v)
	write(x,x+2)

...

Two-phase locking (2PL) protocol

...

T_1	T_2
lock(v)	
	unlock(x)
write(v,v+1)	
unlock(v)	
unlock(u)	

Deadlock

T_1	T_2
lock(u) read(u)	
	lock(v) write(v,v+1)
lock(v) wait	
	lock(u) wait

Timestamp ordering (TO) protocol

Principles

Each transaction obtains a timestamp at the start point

Data items are stamped each time any transaction accesses data items in a read or write mode

Access to data items is permitted in increasing order of timestamps

Problems

Cascading aborts

Unnecessary aborts when execution is conflict serializable

Timestamp ordering (TO) protocol

T_1	T_2	x
timestamp(t1)		
read(x)		x:t1
write(x,x-10)		
	timestamp(t2)	
	write(x)	x:t1:t2
	read(y)	y:t2
read(y)		y:t2:t1
write(y,y+1)		y:t2:t1
abort		

Timestamp ordering (TO) protocol

T_1	T_2	x
timestamp(t1)		
read(x)		x:t1
write(x,x-10)		
	timestamp(t2)	
	read(x)	x:t1:t2
fail		
	forced abort	

Cascading abort !

References

Elmasri R., Navathe S., Fundamentals of Database Systems, 6th edition, chapter 21 Introduction to Transaction Processing Concepts and Theory, pp. 747-779

Elmasri R., Navathe S., Fundamentals of Database Systems, 6th edition, chapters 22.1, 22.3 Concurrency Control Techniques, pp. 780-794