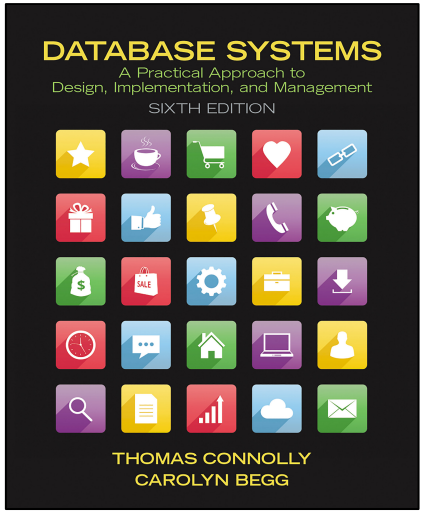
Transactions and its properties

Topic 6, Lesson 1
ACID and the algorithms to support it

Adapted from Chapter 22



Transaction

Action, or series of actions, carried out by a user or an application, which reads or updates contents of the database.

- Application program is a series of transactions with non-database processing in between.
- A transaction is a **'logical unit of work'** on a database
  - Each transaction does something in the database
  - No part of the transaction alone achieves anything of use or interest to a user
- Transforms database from one consistent state to another, although consistency may be violated during transaction

Transaction commands in SQL

START TRANSACTION;
Declare the start of a transaction

ROLLBACK;
Rollback signals the unsuccessful end of a transaction
Returns the system to the state it was in before the transaction began
System state must be the same as if the transaction had never existed
Must abort any transactions that depend on the outcome of the aborting transaction

COMMIT;
COMMIT signals the successful end of a transaction
Any changes made by the transaction should be saved
These changes are now visible to other transactions

With the COMMIT operation, the application declares that the transaction is permanently complete
If you commit:
No actions should be able to move the DBMS to a state not containing the results of the transaction
All operations must be forever persistent in the database

SAVEPOINT : flush changes to the database – can rollback to a save point

Outcome from a transaction

A transaction can have one of two outcomes:

Success - transaction commits and database reaches a new consistent state.

Failure - transaction aborts, and database must be restored to consistent state before it started.

Such a transaction is rolled back or undone.

Committed transaction cannot be aborted or undone.

Aborted transaction that is rolled back can be restarted later.

Transaction Properties

What are the properties of a transaction?

Atomicity: either the entire set of database operations happens or none of it does

Consistency: the set of operations taken together should move the database system from one consistent state to another consistent state.

Isolation: each system perceives the database system as if no other transactions were running concurrently (even though odds are there are other active transactions)

Durability: results of a completed transaction must be permanent - even IF the system crashes

Transaction example

```
START TRANSACTION
READ(A)
A-=50
WRITE(A)
READ(B)
B+=50
WRITE(B)
COMMIT
```

- **Atomicity** - shouldn't take money from A without giving it to B
- **Consistency** - money isn't lost or gained
- **Isolation** - other queries shouldn't see A or B change until transaction is completed (so both values updated)
- **Durability** - the money does not go back to A

Discussion

How can a database management system guarantee the ACID properties?

Focus on the atomicity and the isolation property.

Concurrency control

- Process of managing simultaneous operations on the database without having them interfere with one another.
- Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.
- Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.

Highlighting need for concurrency control

Problems caused by concurrency:

- Lost update problem.
- Uncommitted dependency problem.
- Inconsistent analysis problem.
- Nonrepeatable read problem
- Phantom read problem.

Concurrency control problem

Lost Update Problem:

Successfully completed update is overridden by another user.

T_1 withdraw £10 from an account with bal_x, initially £100.

T_2 deposit £100 into same account.

Serially, final balance would be £190.

Lost update problem

Schedule for the two transactions

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	read(bal_x)	100
t_3	read(bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write(bal_x)	200
t_5	write(bal_x)	commit	90
t_6	commit		90

- **Loss of T_2's update avoided by preventing T_1 from reading bal_x until after update.**

Uncommitted dependency problem

- Occurs when one transaction can see intermediate results of another transaction before it has committed.
- One transaction reads values that another transaction are in the process of modifying
- Referred to as a dirty read
- T_4 updates bal_x to £200 but it aborts, so bal_x should be back at original value of £100.
- T_3 has read new value of bal_x (£200) and uses value as basis of £10 reduction, giving a new balance of £190, instead of £90.

Uncommitted Dependency Problem

Time	T_3	T_4	bal_x
t_1		begin_transaction	100
t_2		read(bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	begin_transaction	write(bal_x)	200
t_5	read(bal_x)	⋮	200
t_6	$bal_x = bal_x - 10$	rollback	100
t_7	write(bal_x)		190
t_8	commit		190

Problem avoided by preventing T_3 from reading bal_x until after T_4 commits or aborts.

Inconsistent analysis problem

- Occurs when a transaction reads several values while a second transaction updates some of those values during the execution of the first transaction.
- T_6 is totaling balances of account x (£100), account y (£50), and account z (£25).
- Meanwhile, T_5 has transferred £10 from bal_x to bal_z, so T_6 now has wrong result since we read one value before the transfer and one value after the transfer (£10 too high).

Schedule for an inconsistent analysis issue

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

Problem avoided by preventing T_6 from reading bal_x and bal_z until after T_5 completes its updates.

Other READ problems

Nonrepeatable read occurs, when during the course of a transaction, a row is retrieved twice and the values within the row differ between reads. A transaction must access the same tuple at different points in the transaction and retrieve two different values.

Phantom read occurs, when during the course of a transaction, new tuples are added or removed into a table from a different transaction that invalidates the read results of a running transaction.

Nonrepeatable read

Time	Transaction 1	Transaction 2	A value
t_1	Read(A)		100
t_2		Read(A)	
t_3		A+=50	
t_4		Write(A)	150
t_5	Read(A)		150

Transaction 1 sees 2 different values for A and assumes there is a bug in the application or the database.

Problem avoided by preventing transaction 2 from updating A

Phantom read

Time	Transaction 1	Transaction 2	A value
t_1	Read(A)		100
t_2		DELETE(A)	
t_3			
t_4	Read(A)		

Transaction 1 believes a copy of A is in the database; however transaction 2 deletes A.

Problem avoided by preventing transaction 2 from deleting A

Solving transaction issues

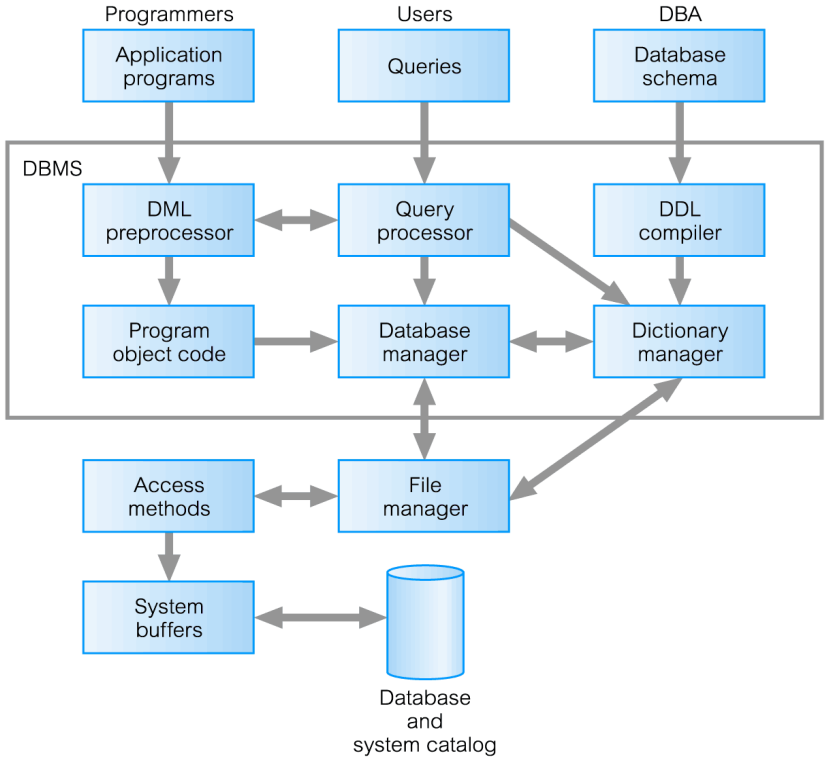
GOAL: identify methods for scheduling I/O a transaction's operations so we can safely preserve the isolation property of the transactions.

SOLUTION: must develop a concurrency control solution that handles the inconsistent analysis problem, uncommitted dependency problem, the lost update problem, the non-repeatable read problem and the phantom read problem.

Summary

- Transactions are guaranteed the ACID properties
- To ensure these properties the scheduler must ensure that the scheduled operations do not interfere with each other
- If transactions run concurrently then the following problems can occur:
  - Lost Update problem
  - Uncommitted Dependency (Dirty Read) problem
  - Inconsistent Analysis problem
  - Nonrepeatable Read problem
  - Phantom Read problem

Components of a DBMS



Components of Database Manager