
Transactions Concurrency Methods

Outline

- Database transactions
- Concurrency control
- Concurrency problems
- Concurrency methods
- Database recovery

What is a Database Transaction?

- Transaction – A **logical** unit of work
 - A logical sequence of operations that must be executed as a whole, while preserving data consistency and accuracy
 - An indivisible unit of processing – partial transaction is not acceptable
 - A database must be in a consistent and correct state both before and after a transaction

What is a Database Transaction?

- Transaction – transfer \$500 from Fred to Sue
- Operations
 1. Update Fred's account
 2. Update Sue's account
 3. Commit transaction

Begin Transaction

Fred's Account \$1000
Sue's Account \$0

Db in consistent state

Db may be in an
inconsistent state

End Transaction

Fred's Account \$500
Sue's Account \$500

Db in consistent state

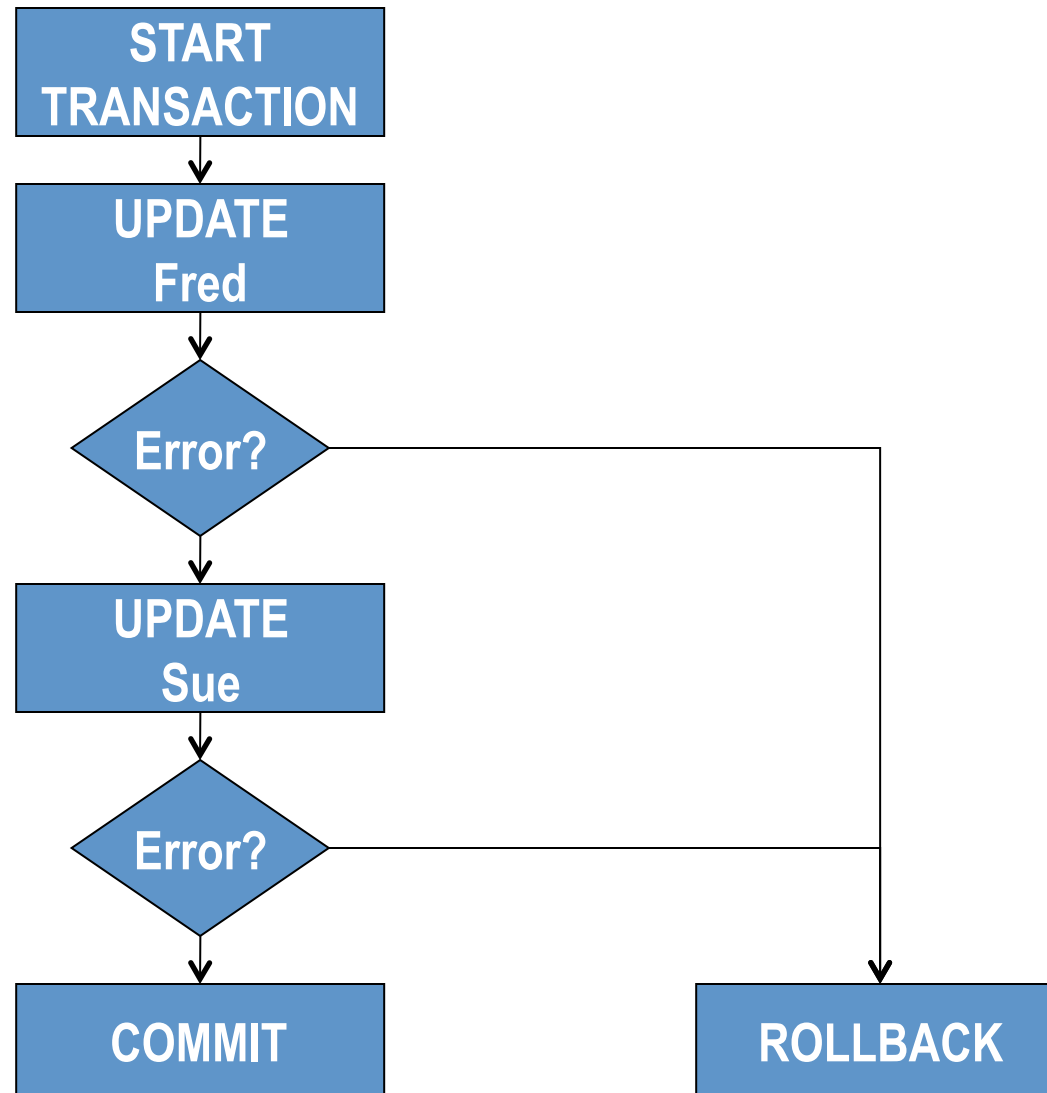
ACID Properties of Transactions

- **Atomtic**
 - A transaction is a complete unit of work - either performed entirely or not at all
- **Consistent**
 - A transaction's correct execution takes the database from one consistent state to another
- **Isolation**
 - Each transaction must appear to execute by itself without interference from other concurrent transactions
- **Durable**
 - Changes made to the database must be permanent




Transaction States

- For recovery purposes the system needs to keep track of when a transaction is started, and terminated or committed (saved)
 - **START TRANSACTION** – marks the start of a transaction
 - **COMMIT** – successful completion of the transaction
 - All changes are saved to the database
 - **ROLLBACK** – unsuccessful end of the transaction
 - Any changes made are undone

Transaction Flow



Transaction Example

 ACCT_NUMBER	 HOLDER	 AMOUNT
1000	Fred	1000
2000	Sue	0

Transaction Example – cont.

START TRANSACTION;

UPDATE account

SET amount = amount - 500

WHERE holder = 'Fred';

SELECT * FROM account; -- Won't see changes

UPDATE account

SET amount = amount + 500

WHERE holder = 'Sue';

COMMIT; -- Changes written to Db after this is run

Autocommit

- By default many DBMSs autocommit transactions
 - Each individual statement is a transaction
 - If no errors, commit
- Autocommit does not help with multi-statement transactions
 - You cannot rollback already committed statements
- Oracle
 - See status: **SHOW auto_commit;**
 - Set **SET autocommit off/on**
- SQL Developer
 - Tools → Preferences → Database → Advanced

Autocommit Example

UPDATE account

SET amount = amount - 750

WHERE holder = 'Fred';

UPDATE account

SET amount = amount + 750

WHERE holder = 'Sue';

ROLLBACK; -- Nothing to rollback

Autocommit Example

DELETE FROM account;

SELECT * FROM account; -- All connections will see

ROLLBACK; -- Nothing to rollback

Concurrency Control

- What is it?
 - Controlling simultaneous access/update of a single database by multiple users
- Why concurrency?
 - Overall increase in system throughput
 - Improved response time
 - Simultaneous data access ...
- Why concurrency control?
 - Transactions submitted by one user interfering with another transaction will generate incorrect or inconsistent results
 - Concurrency isolation (ACID)

Schedule

- Schedule – a sequential ordering of operations within multiple transactions
 - ❑ Schedule must maintain the order of operations within each participating transaction
 - ❑ Serial
 - ❑ Non-Serial

Serial Schedule

- All operations within participating transaction are executed consecutively
 - One after another
 - Not concurrent
- A transactions schedule is serializable if its outcome (e.g., the resulting database state) is equal to the outcome of its transactions executed serially, i.e., sequentially without overlapping in time.

Non-Serial Schedule

- Non-serial schedules – operations within transactions are interleaved
 - Two operations conflict if:
 - They belong to different transactions,
 - AND access the same data item,
 - AND one of them is a write

Multiple Transactions Example

- Sample order process (transaction)
- Why not do in one step?
 - Additional processing at different points in process
 - Verify enough quantity to fulfill order
 - Verify customer has sufficient credit
 - Determine if product should be reordered

Multiple Transactions Example – Serial

- Beginning quantity = 100
 - Transaction 1 → purchase 50
 - Transaction 2 → purchase 10

Time	Transaction 1	Transaction 2	Quantity (qty)
T1	Read qty		100
T2	qty = 100 - 50		
T3	Write qty (50)		50
T4	COMMIT		
T5		Read qty	50
T6		qty = 50 - 10	
T7		Write qty (40)	40
T8		COMMIT	

Concurrency Problems

- Lost updates
 - Two transactions trying to update the same database item at about the same time. Results if one transaction overwrites the results of the other
- Dirty reads
 - One transaction reading data that is being modified by a second trans
- Non-repeatable reads (fuzzy)
 - One trans reads data and finds another committed trans has modified or deleted the data.
- Phantom reads
 - A trans rereads a query returns rows satisfying criteria and finds another committed trans has inserted rows that satisfy criteria.

Lost Update

- Beginning quantity = 100
 - Transaction 1 → purchase 50
 - Transaction 2 → purchase 10

Time	Transaction 1	Transaction 2	Quantity (qty)
T1	Read qty		100
T2		Read qty	100
T3	qty = 100 - 50		
T4		qty = 100 - 10	
T5	Write qty (50)		50
T6		Write qty (90)	90

Dirty Read

- Beginning quantity = 100
 - Transaction 1 → purchase 50
 - Transaction 2 → purchase 10

Time	Transaction 1	Transaction 2	Quantity (qty)
T1	Read qty		100
T2	qty = 100 - 50		
T3	Write qty (50)		50
T4		Read qty	50
T5		qty = 50 - 10	
T6	ROLLBACK		100
T7		Write qty (40)	40

Nonrepeatable Read

Time	Transaction 1	Transaction 2
T1	Read quantities	
T2		Update product A (+50)
T3	Read quantities	

Product	Time T1	Time T2	Time T3
A	100	150	150
B	200	200	200
C	300	300	300
D	400	400	400

Phantom Read

Time	Transaction 1	Transaction 2
T1	Read quantities	
T2		INSERT product D
T3	Read quantities	

Product	Time T1	Time T2	Time T3
A	100	100	100
B	200	200	200
C	300	300	300
D		400	400

Isolation Levels

- Trade-off between Consistency and Concurrency
- Read Uncommitted - BAD
 - Each query can see uncommitted data
- Read Committed – Default
 - Each query in a trans only sees data committed before query began (not the trans)
 - Used where few trans will conflict
 - Provides higher potential throughput

Isolation Levels

- Serializable
 - Sees only changes committed at the time trans began Plus those changes made by trans itself by I/U/D
 - Large DBs & short trans that update only a few rows
 - 2 concurrent trans will modify the same rows is relatively low
 - Relatively long-running trans are primarily read only
- Repeatable Read
 - Same as Serializable but doesn't allow I/U/D

Preventable Concurrency Problems

Isolation Level	Dirty Read	Nonrepeatable	Phantom
Read Uncommit	Possible	Possible	Possible
Read Commit	Not Possible	Possible	Possible
Serializable	Not Possible	Not Possible	Not Possible
Repeatable Read	Not Possible	Not Possible	Possible

Locking

- Lock – a mechanism to control concurrent access to a data item by competing transactions
- Locking protocol – a set of rules followed by all transactions while requesting and releasing locks

Locking

■ How it works

- ❑ Transactions wanting to read/change a data item request the lock manager to lock the data they want to read/change
- ❑ Lock manager locks the requested data items for the use of the requesting transaction
 - OR, makes the transaction wait in queue if the requested data is already locked by a previous transaction
- ❑ When the first transaction is done with the lock the lock is released. Data is now available for the next transaction
- ❑ Lock manager maintains a “lock table” to record granted locks and pending requests, which is used for lock management

Types of Locks

- Granularity of locks:
 - Specific value (column & row), row, table, database
 - Depends on DBMS
 - Finer granular locks provides better concurrency, but increases locking overhead since more lock/unlock operations are needed
- Types of locks:
 - Read lock or shared lock (lock-R): Data item can be read but not changed by the locking and other transactions
 - Write lock or exclusive lock (lock-W): Data items can be read and changed by the locking transaction, but not by other transactions
 - Concurrency control manager can upgrade read locks to write locks, or downgrade write locks to read locks, as requested.

Types of Locks

Requested Lock	Current Lock	
	Read (lock-R)	Write (lock-W)
	Read (lock-R)	Conflict
Write (lock-W)	Conflict	Conflict

Serializable Lock Example

Transaction 1	Transaction 2
Lock-W(A)	Lock-W(B)
Read(A = 10)	Read(B = 20)
Update(A = A * 2)	Update(B = B + 10)
Unlock-W(A)	Unlock-W(B)

Serializable Lock Example

Transaction 1	Transaction 2
Lock-W(A)	
Read(A = 10)	Lock-W(A) → Wait
Update(A = A * 2)	
Unlock-W(A)	
	↓
	Lock-W(A)
	Read(A = 20)
	Lock-R(B)
	Read(B = 20)
	Update(A = A + B)
	Unlock-W(A), R(B)

Nonserializable Lock Example

Initial values:

A = 30, B = 20

If transaction 1 → 2:

A = 80; B = 50

If transaction 2 → 1:

A = 50; B = 70

As shown:

A = 50; B = 50

Released lock
too early!

Transaction 1	Transaction 2
Lock-R(A)	Lock-R(B)
Read(A = 30)	Read(B = 20)
Unlock-R(A)	Lock-W(A) → Wait
Lock-W(B) → Wait	Lock-W(A) ↓
	Read(A = 30)
	Update(A = A + B)
	Unlock-R(B), W(A)
Lock-W(B) ↓	
Read(B = 20)	
Update(B = B + A)	
Unlock-W(B)	

Deadlocks

Deadlocks

- Occurs when two or more transactions try to update the same data and wait for the other to release its lock
 - ❑ Detected by lock manager when it sees two transactions halted for a period of time (timeout)
- Handling deadlocks:
 - ❑ Difficult to prevent
 - ❑ When detected, one transaction is aborted (locks released) and rolled back
 - ❑ Rolled back transaction rolled forward later after the first transaction is completed
 - ❑ May require cascading rollbacks

Deadlocks

Transaction 1	Transaction 2
Lock-R(A)	
Read(A = 10)	Lock-W(B)
Lock-R(B) → Wait	Read(B = 20)
...	Update(B = B + 10)
	Lock-W(A) → Wait
	...

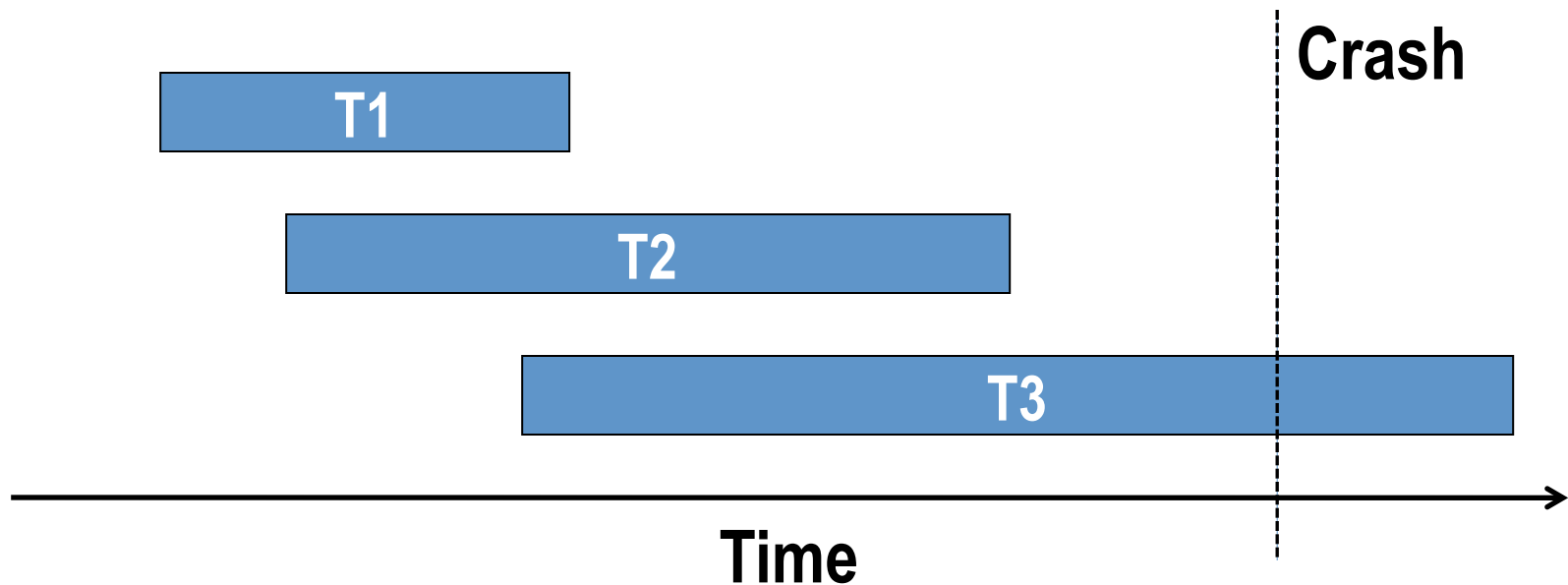
■ Solution

- ❑ Rollback Transaction 2
- ❑ Complete Transaction 1
- ❑ Roll forward Transaction 2

Design for Concurrency

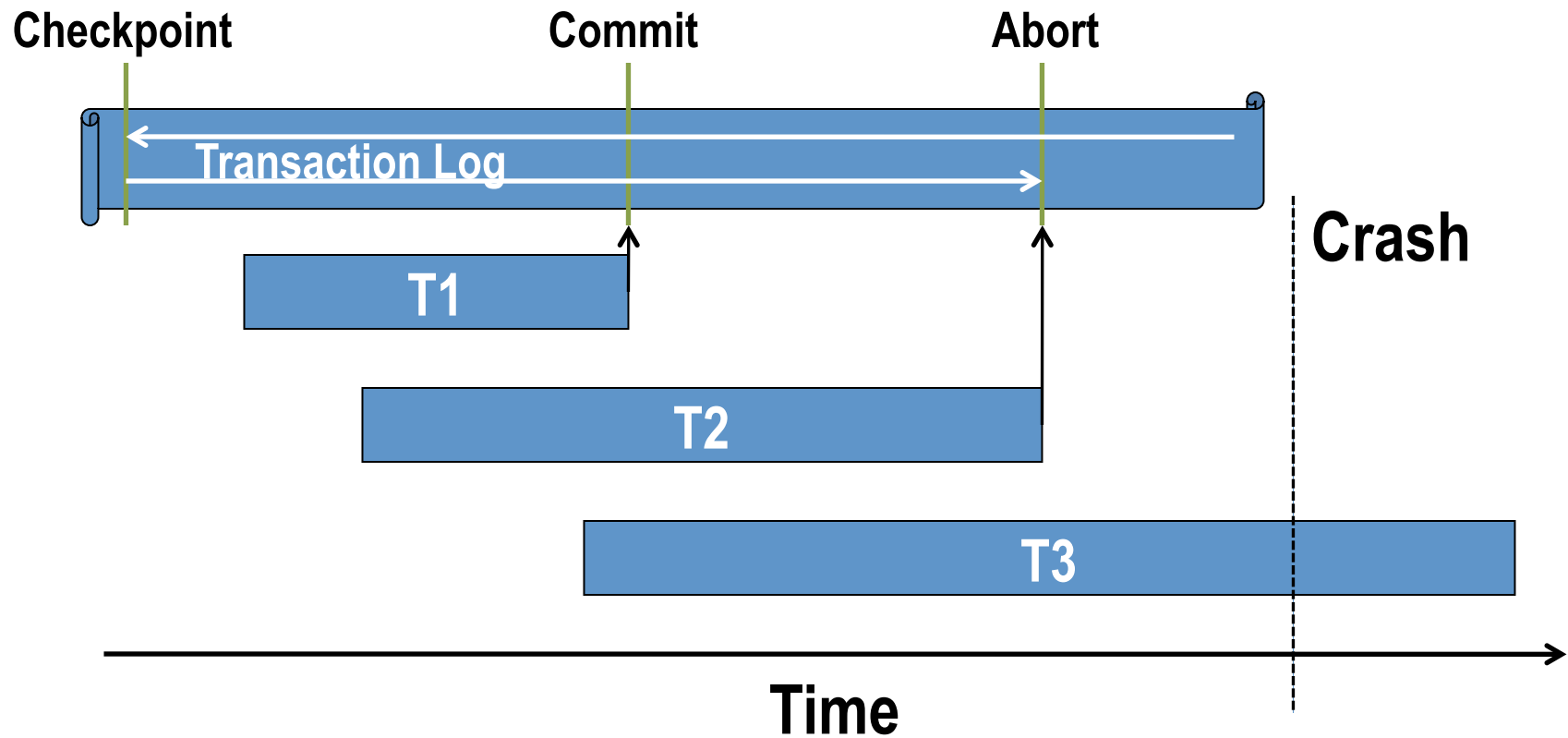
- Minimize transaction size
- Limit transaction operations
- Access resources in a consistent order
- Minimize resource access time
- Run resource-intensive operations during non-peak hours (if possible)

Database Recovery



- Non-catastrophic – system failure, disk failure, etc.
- Catastrophic – earthquake, fire, flood, etc.

Database Recovery



Database Recovery

- Transaction log – statements are written to a log before being written to database
- Checkpoint – point where all committed transactions written to db (point noted in log)

Transaction Log Example

TransId	Log_entry
...	
52	commit
*** CHECKPOINT ***	
53	start_trans
53	read_item(X)
53	write_item(X, 5, 25)
54	start_trans
53	commit
54	write_item(Y, 1, 0)
54	abort
55	start_trans
55	write_item(X, 43, 23)
*** CRASH ***	

Database Recovery

- Non-catastrophic failure:
 - Maintain a log that records all insert/update/delete operations on database
 - ROLLBACK operation – reverse changes that occurred after last checkpoint
 - ROLL FORWARD operation – redo legitimate changes that were lost

Database Recovery

- Catastrophic failure:
 - Restore a previous copy of the database from archival backup.
 - Apply transaction log to recreate current state by redoing all committed transaction up to failure point.