Concurrency in postgres

- Kripa Lamichhane

Concurrency Control

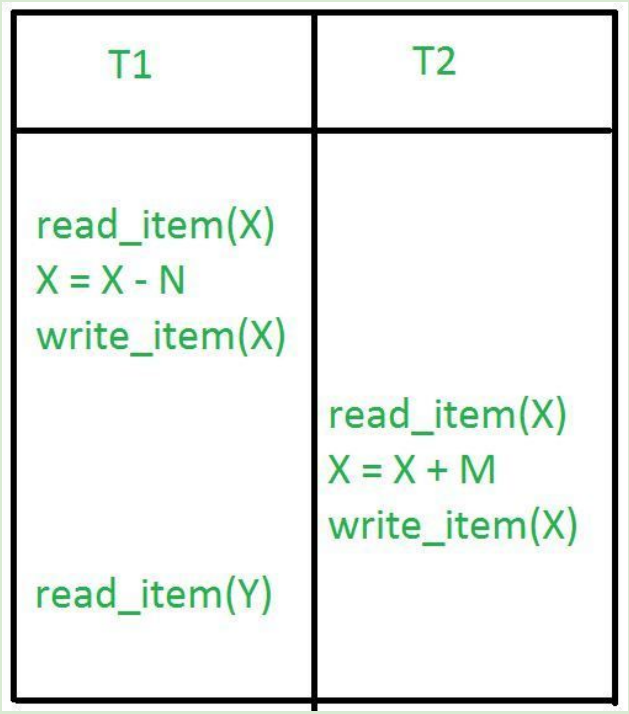
Concurrency Control is the process of managing simultaneous operations on the database without having them interfere with one another. It enhances overall CPU utilization.

- A Venmo App Transaction Condition
- Flight Booking

Problems if there isn’t Concurrency Control

- Temporary Read Or Dirty Read Problem
- Incorrect Summary
- Lost Update Problem
- Unrepeatable read problem
- Phantom read problem

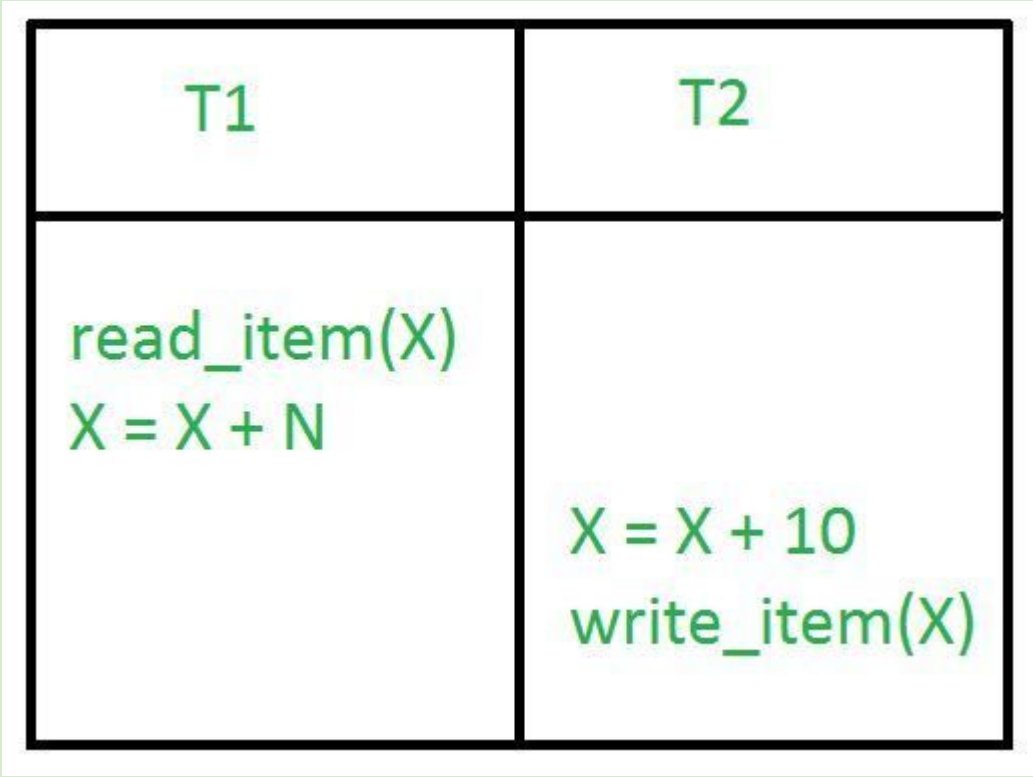
Temporary Update Problem



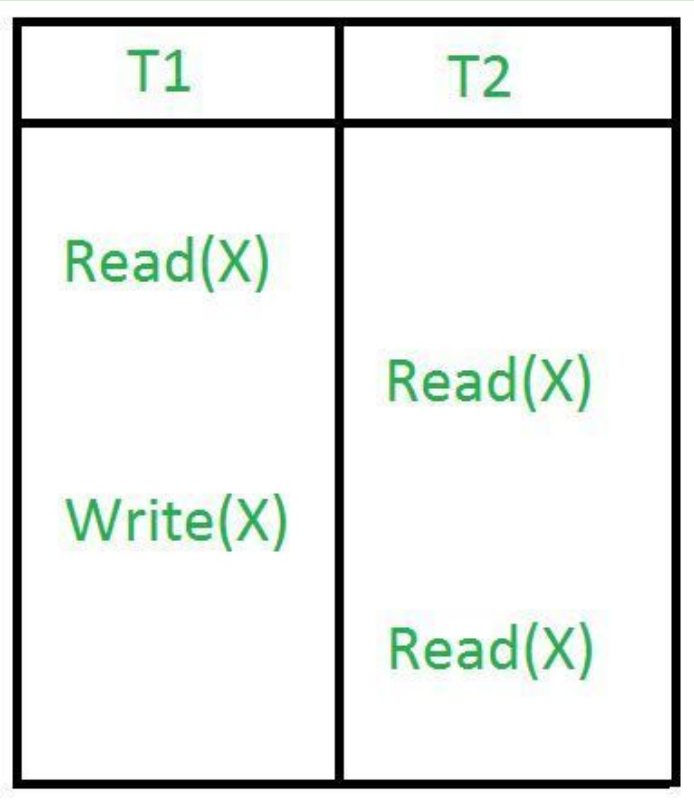
Incorrect Summary

T1	T2
	sum = 0 read_item(A) sum = sum + A
read_item(X) X = X - N write_item(X)	
	read_item(X) sum = sum + X read_item(Y) sum = sum + Y
read_item(Y) Y = Y + N write_item(Y)	

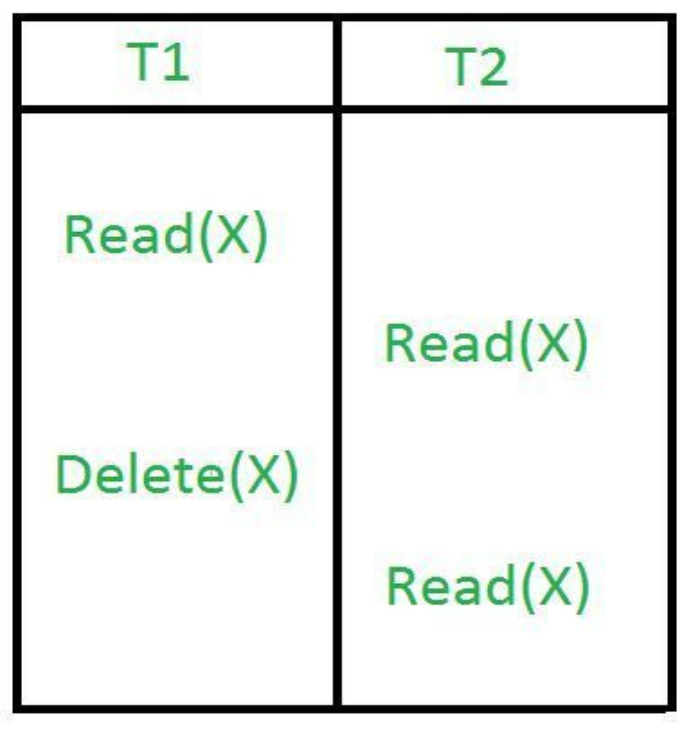
Lost Update Problem



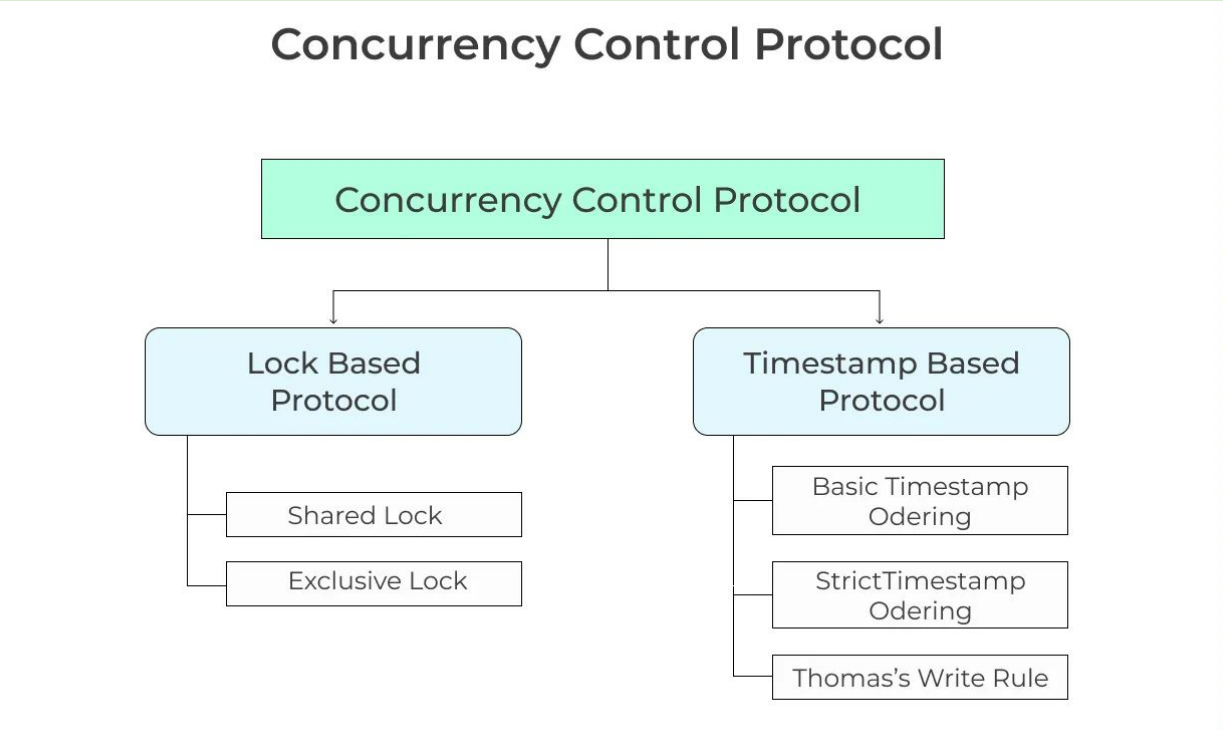
Unrepeatable Read Problem



Phantom Read Problem



Solution


Concurrency Control Protocol
Concurrency Control Protocol
Lock Based Protocol
Timestamp Based Protocol
Shared Lock
Exclusive Lock
Basic Timestamp Odering
StrictTimestamp Odering
Thomas's Write Rule

Locking

Used to control concurrent access to data when one transaction is accessing the database.

Lock are of two types:

1. Shared Lock
2. Exclusive lock

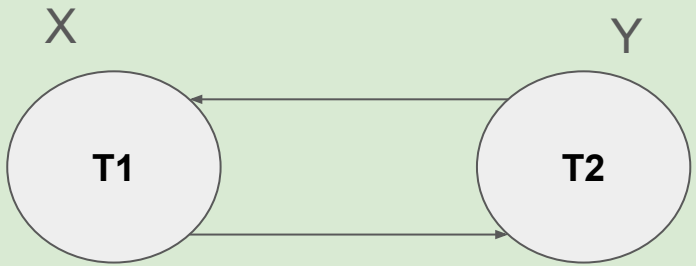
A locking protocol is a set of rules followed by all transactions while requesting and releasing locks

Compatibility of locking

In the board - upgrade and downgrade of lock

Deadlock

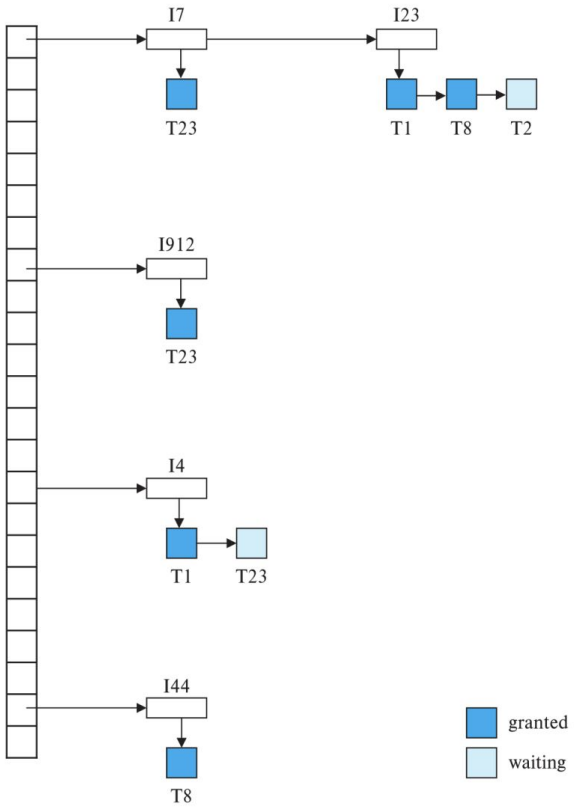
Situation in which two or more transaction are waiting for one another to give up locks.



Example

T1	T2
Lock - X(X) R(X) W(X)	
	Lock - X(Y) R(Y) W(Y)
Lock - X(Y) R(Y) W(Y) {waiting for T2 to release lock in Y}	
	Lock-X(X) R(X) W(X) {waiting for T1 to release lock on X}

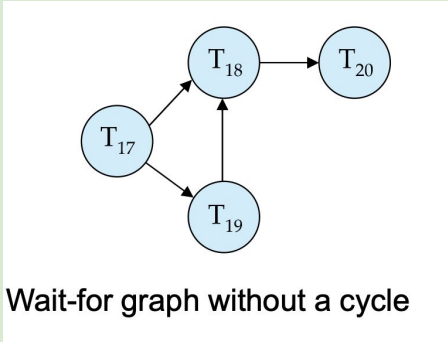
Lock table - Deadlock



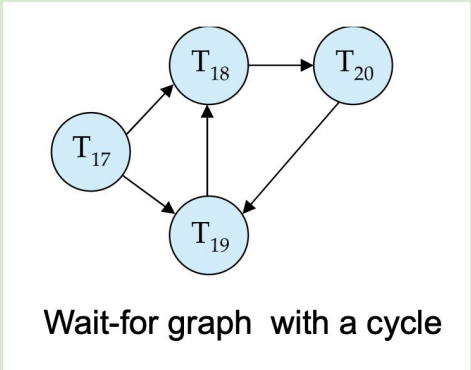
- Dark rectangles indicate granted locks, light colored ones indicate waiting requests
- Lock table also records the type of lock granted or requested
- New request is added to the end of the queue of requests for the data item, and granted if it is compatible with all earlier locks
- Unlock requests result in the request being deleted, and later requests are checked to see if they can now be granted
- If transaction aborts, all waiting or granted requests of the transaction are deleted
  - lock manager may keep a list of locks held by each transaction, to implement this efficiently

Deadlock Detection

- Wait-for graph
  - Vertices: transactions
  - Edge from Ti →Tj : if Ti is waiting for a lock held in conflicting mode byTj
- The system is in a deadlock state if and only if the wait-for graph has a cycle.
- Invoke a deadlock-detection algorithm periodically to look for cycles.



Wait-for graph without a cycle



Wait-for graph with a cycle

Deadlock Prevention

- Hold & Wait : hold and need request
- Mutual Exclusion : locked by other processes
- No preemption: holding lock even after transaction is completed
- Circular wait: waiting in circular form

Time Based Protocol

- Each transaction Ti is issued a timestamp TS(Ti) when it enters the system.
  - Each transaction has a unique timestamp
  - Newer transactions have timestamps strictly greater than earlier ones
  - Timestamp could be based on a logical counter
    - Real time may not be unique
    - Can use (wall-clock time, logical counter) to ensure
- Timestamp-based protocols manage concurrent execution such that time-stamp order = serializability order
- Several alternative protocols based on timestamps

Time-Stamp Based Protocol

- Suppose a transaction Ti issues a read(Q)

  1. If TS(Ti) ≤ W-timestamp(Q), then Ti needs to read a value of Q that was already overwritten.

     Hence, the read operation is rejected, and Ti is rolled back.

  2. If TS(Ti) ≥ W-timestamp(Q), then the read operation is executed, and R-timestamp(Q) is

     set to max(R-timestamp(Q), TS(Ti)).

Timestamp Ordering Protocol

- Maintains for each data Q two timestamp values:
  - W-timestamp(Q) is the largest time-stamp of any transaction that executed write(Q) successfully.
  - R-timestamp(Q) is the largest time-stamp of any transaction that executed read(Q) successfully.
- Imposes rules on read and write operations to ensure that
  - any conflicting operations are executed in timestamp order
  - out of order operations cause transaction rollback

Correctness of timestamp

- The timestamp-ordering protocol guarantees serializability since all the arcs in the precedence graph are of the form: Thus, there will be no cycles in the precedence graph
- Timestamp protocol ensures freedom from deadlock as no transaction ever waits.

Thomas Write-rule

Modified version of the timestamp-ordering protocol in which obsolete write operations may be ignored under certain circumstances.

Thomas' Write Rule allows greater potential concurrency.

- Allows some view-serializable schedules that are not conflict serializable

Resources :

https://db-book.com/slides-dir/PDF-dir/ch18.pdf

https://youtu.be/aeykOjWjT5Q?si=DWinQKn7INg7m00u

https://youtu.be/ee-wg9q29f0?si=q0LyViyfgznebreY

https://youtu.be/a74V14OnDvw?si=bGEzUHq37jJcYzb7

https://youtu.be/wEsPL50Uiyo?si=6oihYJ2bBz9uL50I

https://youtu.be/I8IlO0hCSgY?si=jWffK4CbVH9yjXH-

https://youtu.be/Thm0xW9oTow?si=k2cakgzkGGKO6I-q

Any Question ?