COMP90050
Advanced Database Systems

THE UNIVERSITY OF MELBOURNE

Semester 2, 2024

Lecturer: Farhana Choudhury (PhD)

Live lecture – Week 5

Continuation from last week's live lecture

Flat Transaction

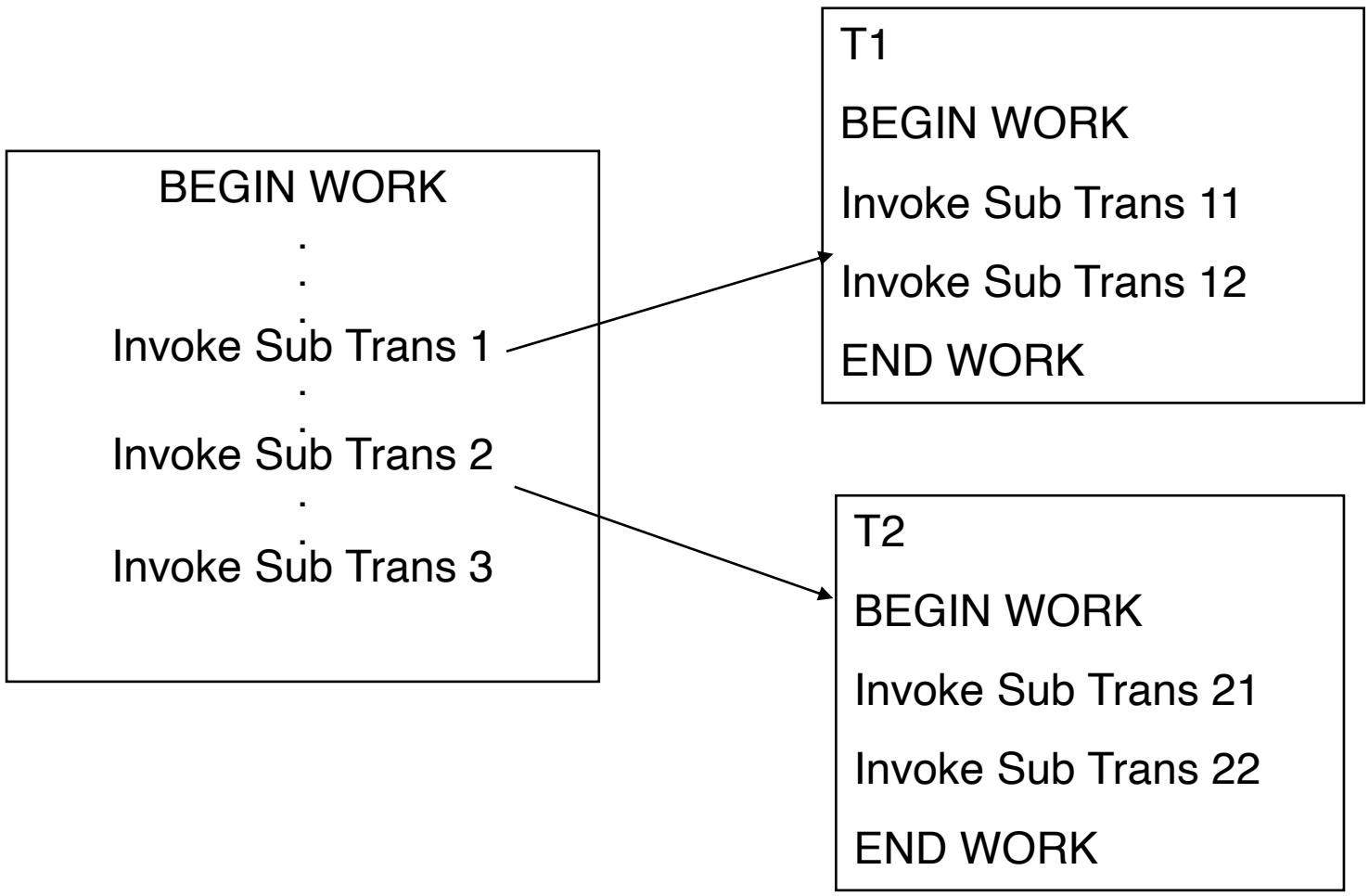
Everything inside BEGIN WORK and COMMIT WORK is at the same level; that is, the transaction will either survive together with everything else (commit), or it will be rolled back with everything else (abort)

```
exec sql BEGIN WORK;
      AccBalance = DodebitCredit(BranchId, TellerId, AccId, delta);
      send output msg;
exec sql COMMIT WORK;
```

Can be a very long running transaction with many operations

Limitations of Flat Transactions?

Nested Transactions



Nested Transactions

Commit rule

- A subtransaction can either commit or abort, however, **commit cannot take place unless the parent itself commits.**

- Subtransactions have A, C, and I properties but not D property unless all its ancestors commit.

- Commit of a sub transaction makes its results available only to its parents.

Roll back Rules

If a subtransaction rolls back, all its children are forced to roll back.

Visibility Rules

Changes made by a subtransaction are visible to the parent only when the subtransaction commits. All objects of parent are visible to its children. Implication of this is that the **parent should not modify objects while children are accessing them.** This is not a problem as parent does not run in parallel with its children.

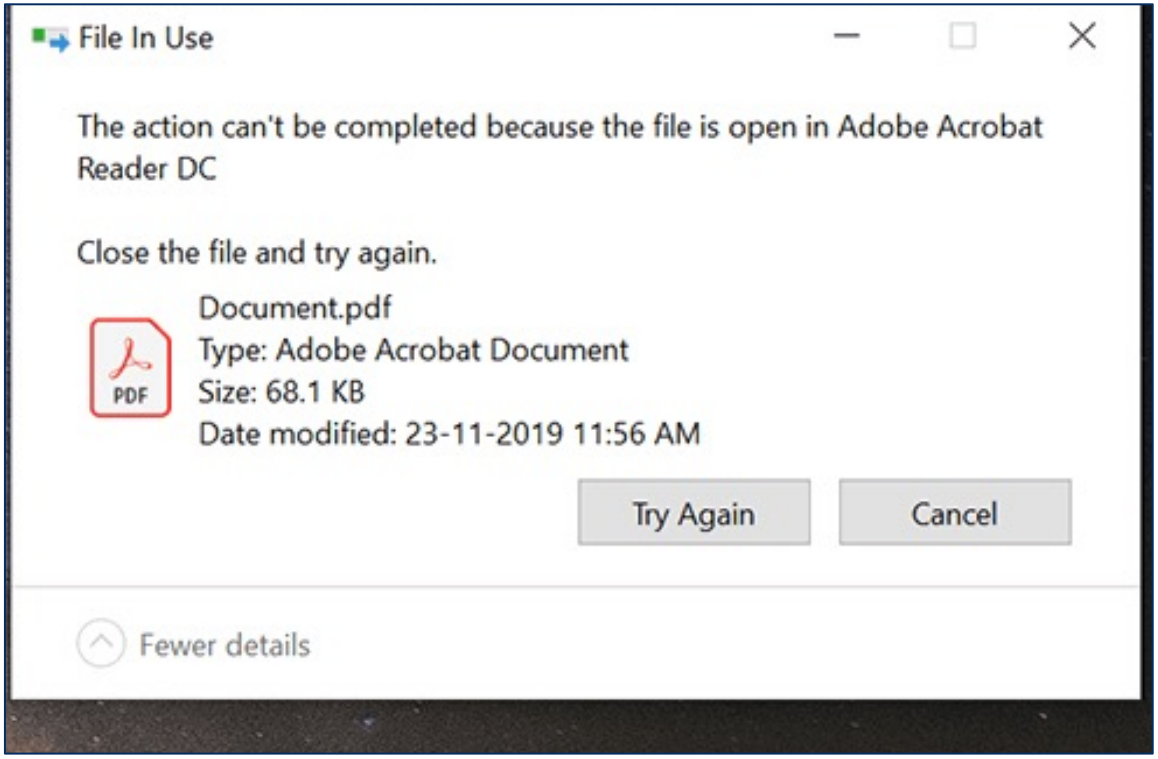
Advantages of nested transactions?

Time for a poll - Pollev.com/farhanachoud585

Book chapter: 4.7

This week's contents

Concurrency problem



Concurrent transactions can cause issues in Database – need concurrency control

8

Concurrency Problems

Transaction 1	Transaction 2
balance = balance + 50;	balance = balance - 80;

Account balance = 100;

Both transactions are running concurrently. What are the possible outcomes if no concurrency control in place? Which ones are correct sequence of actions?

Write by a transaction gets lost/overwritten

a) balance == 150

Sequence of actions

T1: Reads balance == 100

T2: Reads balance == 100

T2: Writes balance == 100-80

T1: Writes balance == 100+50

← not desirable

c) balance == 70;

Sequence of actions

T1: Reads balance == 100

T1: Writes balance == 100+50

T2: Reads balance == 150

T2: Writes balance == 150-80

→ get extra more money

b) balance == 20

Sequence of actions

T1: Reads balance == 100

T2: Reads balance == 100

T1: Writes balance == 100+50

T2: Writes balance == 100-80

← not desirable

lose money

→ last write can not overwrite the first write.

d) balance == 70;

Sequence of actions

T2: Reads balance == 100

T2: Writes balance == 100-80

T1: Reads balance == 20

T1: Writes balance == 20+50

对于 a. b 而言：我可以先 withdraw 再 deposit，或先 deposit 再 withdraw ⇒ 但 These two transaction get intertwined ⇒ they both read at the same time, one is only change the other one is unaware of change that we made ⇒ there is no concurrency control

How two transaction should work on the same value, so that one does not impact the other?

one solution: no interleavy of actions

another solution: { 先存再取
先取再存.

Concurrency Problems

Transaction 1	Transaction 2
balance = balance + 50;	balance = balance – 110;

Account balance = 100;

Both transactions are running concurrently. What are the possible outcomes if no concurrency control in place? Which ones are correct sequence of actions? (assume overdrawing not allowed)

c) balance == 40;

Sequence of actions

T1: Reads balance == 100

T1: Writes balance == 100+50

T2: Reads balance == 150

T2: Writes balance == 150-**110**

d) balance == 150;

Sequence of actions

T2: Reads balance == 100

T2: Writes balance == 100-110

(message on overdrawing denied)

T1: Reads balance == 100

T1: Writes balance == 100+50

Try withdrawing money after some time

both of sequences are correct, although the balance values are different

Concurrency Control

目的①②

- To resolve conflicts
- To preserve database consistency

Different ways for concurrency control

- **…**
- **Spin locks (using atomic lock/unlock instructions)** – most commonly used

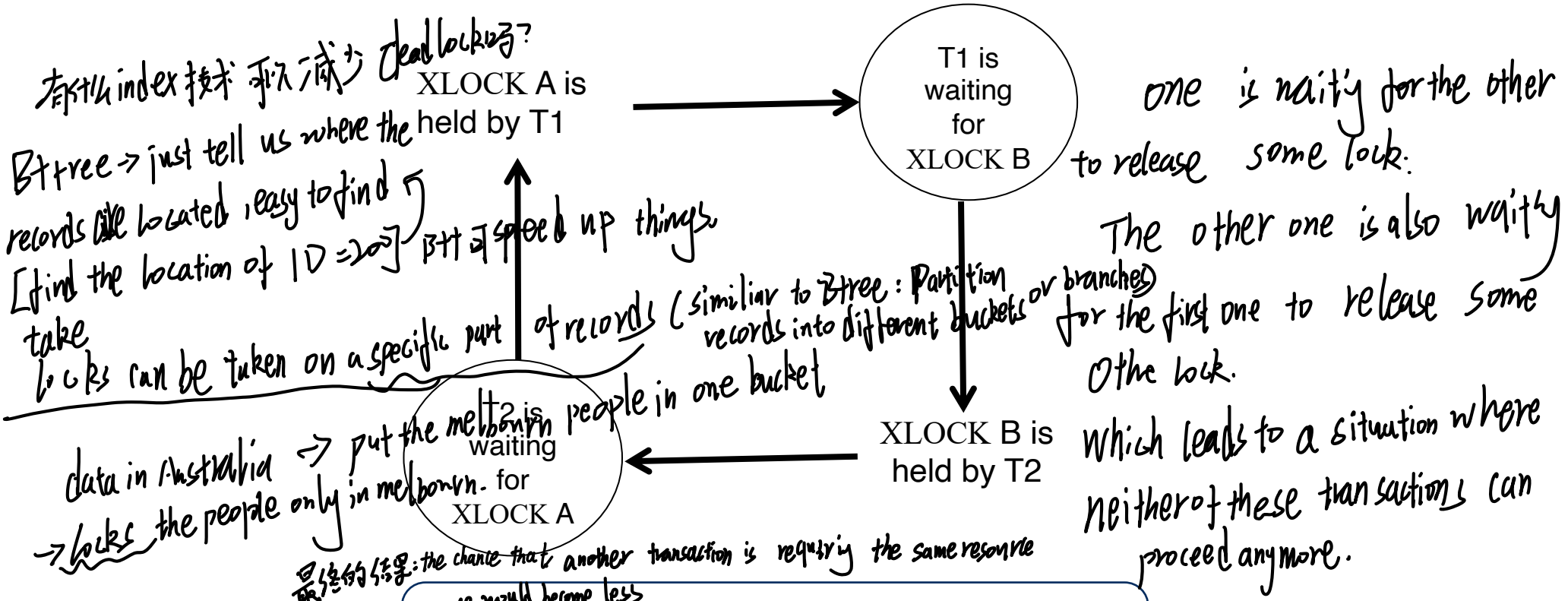
General purpose locks - semaphores

↓

who came first who got released first.

Deadlocks ...

Deadlocks are rare, however, they do occur and the database has to deal with them when they occur



Can proper indexing reduce the changes of a deadlock?

indexing can make query faster -> so the amount of time a lock will be kept that duration will become shorter

Deadlock avoidance/mitigation

- Pre-declare all necessary resources and allocate in a single request.
- Periodically check the resource dependency graph for cycles. If a cycle exists - rollback (i.e., terminate) one or more transaction to eliminate cycles (deadlocks). The chosen transactions should be cheap (e.g., they have not consumed too many resources).
- Allow waiting for a maximum time on a lock then force Rollback. Many successful systems (IBM, Tandem) have chosen this approach.

most commonly used technique -> if a transaction is waiting for a long time => force one transaction to quit/roll back.

- Many distributed database systems maintain only local dependency graphs and use time outs for global deadlocks.

Concurrency problem

Multiple concurrently running transactions may cause conflicts

- Still we try to allow concurrent runs as much as possible for a better performance, while avoiding conflicts as much as possible

Topics for next two weeks

Core Concepts of Database management system