



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David



TRANSACTIONS MANAGEMENT CONCURRENCY

14-Nov-2023

Week 7: Database Systems (DBS)
Dr Suresh Shanmugasundaram

So far...



1. ER Modeling
2. ER Models to Relational Tables
3. Functional Dependencies
4. Normalisation
5. Transactions & its properties

Learning Objectives



1. Concurrency & its problems
2. Schedules – serial and non-serial

Recap - Quiz



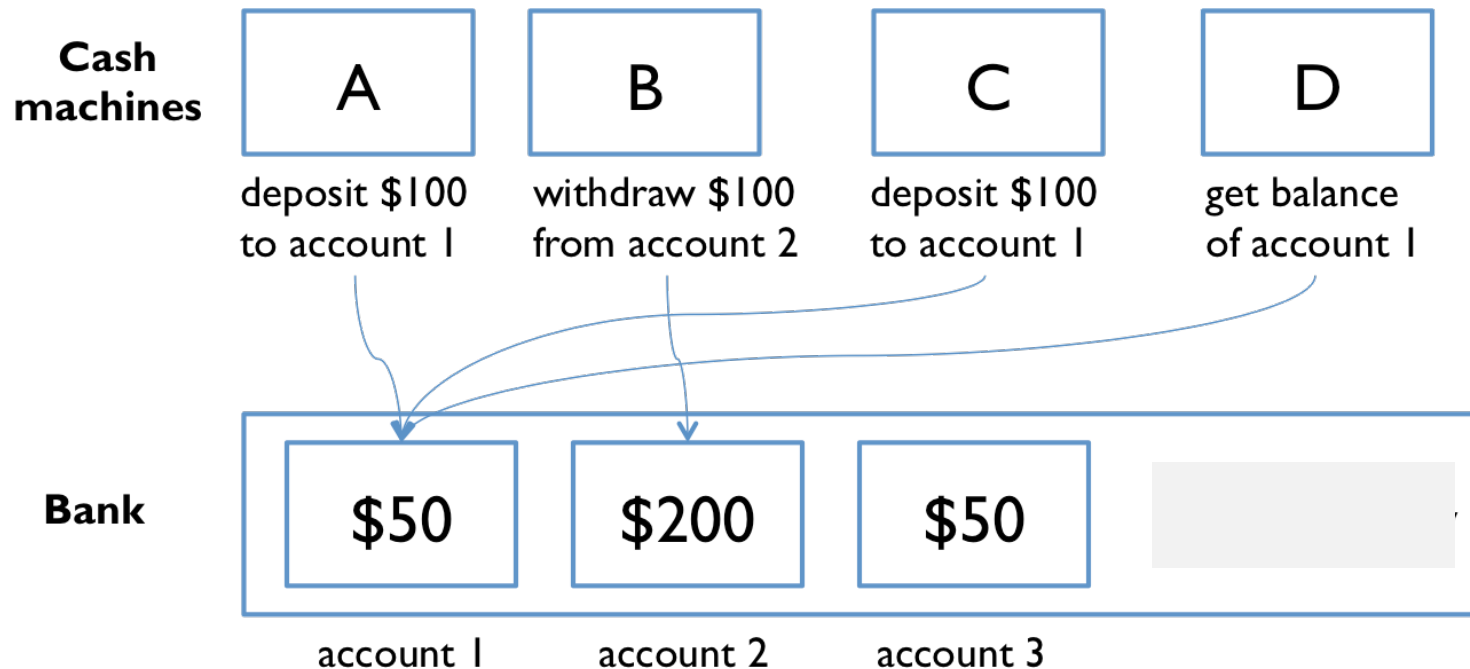
<https://forms.office.com/e/pw0jCuH3xT>

Transaction States and DML (MySQL)

- ❑ The following are the MySQL's DML (Data Manipulation Language) related to the Transaction Management.
- ❑ **BEGIN TRANSACTION:** *indicates the start of a new transaction.*
- ❑ **COMMIT:** *commits the current transaction, making the changes permanent.*
- ❑ **ROLLBACK:** *rolls back the current transaction, cancelling the changes made.*
- ❑ **SET autocommit={0|1}:** *disables/enables the default autocommit mode for the current session. Default is set to 1 (hence, it is set to autocommit)*

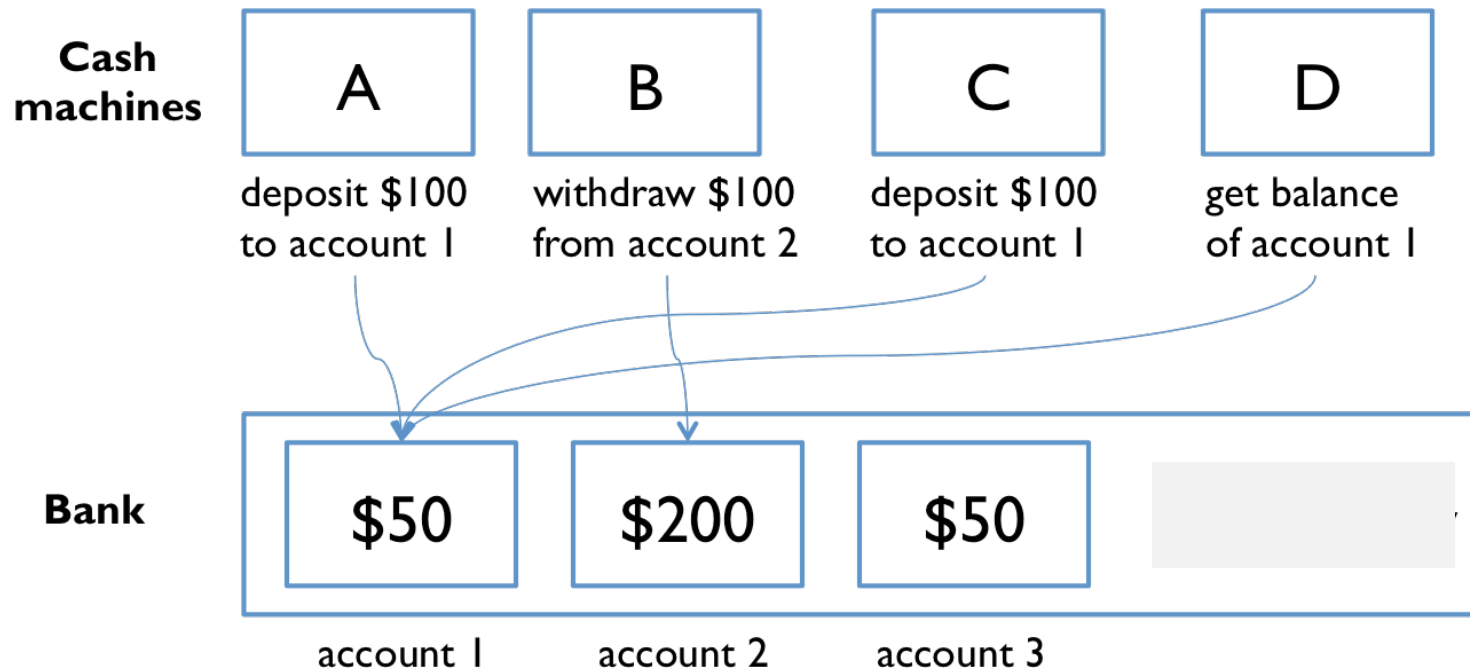
Concurrency

- What is concurrency in the context of Database Management Systems?



Concurrency

- Why is concurrency important in a database system?



Concurrency Control

- ❑ **What?** The process of managing simultaneous operations on the database
- ❑ **Why?** when two or more users are accessing the database simultaneously and at least one is updating data, there may be interference that can result in inconsistencies.
- ❑ **How?** The operations of the two transactions are **interleaved**

Concurrency Control

- **What?** The process of managing simultaneous operations on the database
- **Why?** when two or more users are accessing the database simultaneously and at least one is updating data, there may be interference that can result in inconsistencies.
- **How?** The operations of the two transactions are **interleaved**

Transaction T_1

read_item (X);

X:=X-N;

write_item (X);

Y:=Y+N;

write_item (Y);

Transaction T_2

read_item (X);

X:=X+M;

write_item (X);

Concurrency Control

□ Read and write operations

Reads a database item named X into a program variable also named X

Transaction T_1

read_item (X);

X:=X-N;

write_item (X);

Y:=Y+N;

write_item (Y);

Transaction T_2

read_item (X);

X:=X+M;

write_item (X);

Writes the value of program variable X into the database item named X.

Concurrency Problems

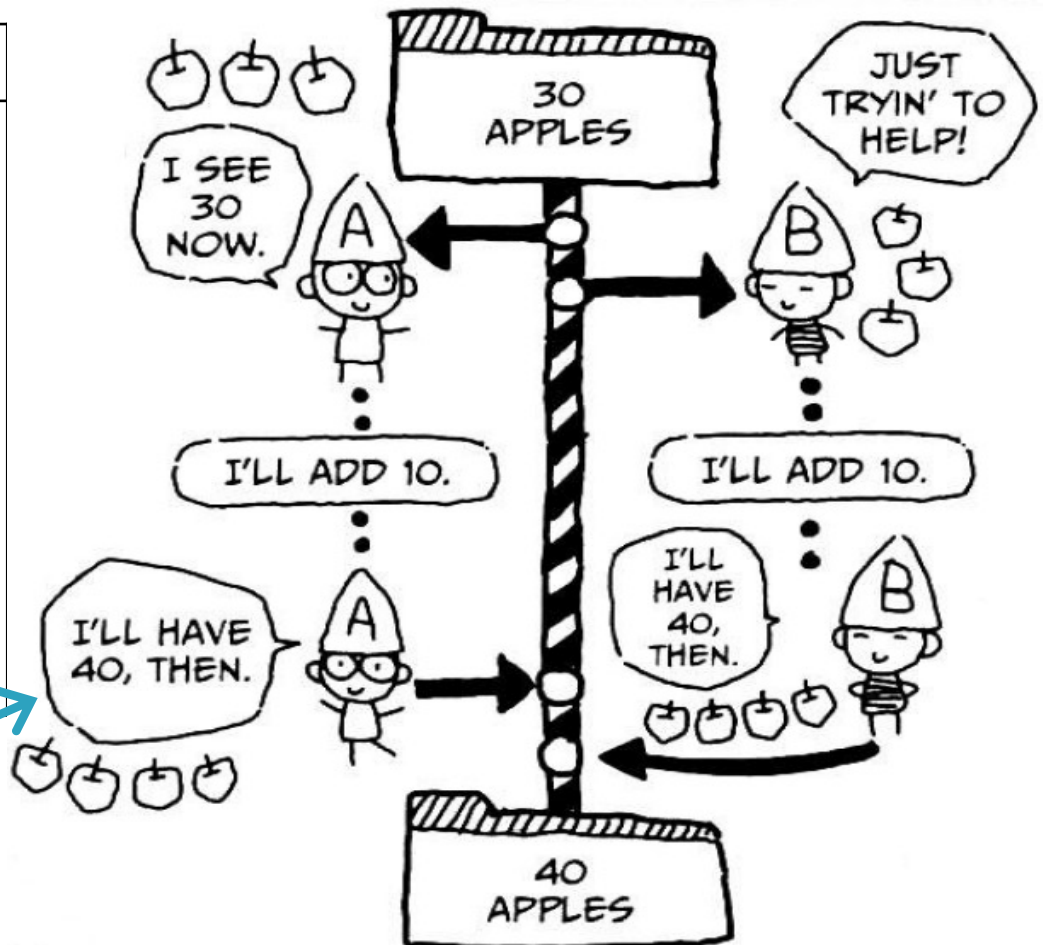
- ❑ The interleaving of operations in this way may produce an incorrect result, thus compromising the integrity and consistency of the database.
- ❑ The three examples of potential problems caused by concurrency:
 - a. **Lost update problem**
 - b. **Uncommitted dependency problem (aka Dirty Read problem)**
 - c. **Inconsistent Analysis problem**
- ❑ All arise when isolation is broken.

Lost Update Problem - Commit

- Successfully completed update operation by one user can be overridden by another user.

A (T1)	B (T2)
Read(X) $X = X + 10$	
	Read(X) $X = X + 10$
Write(X)	Write(X)
COMMIT	COMMIT

This update is lost!



there should be 50 APPLES afterwards!

Lost Update Problem - Commit

- Two joint account holders A and B with $X = £500$.

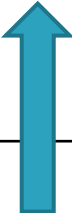
A (T1)	B (T2)
Read(X) $X = X - 100$	
	Read(X) $X = X - 50$
Write(X)	
	Write(X) COMMIT
COMMIT	

Initial State	Bank account (x) = £500
User A's Transaction	Checks the balance Withdraws 100£
User B's Transaction	Simultaneously checks the balance Withdraws 50£
Lost Update Problem	
A assumes the balance as 500	B assumes the balance as 500
Withdraws 100 balance - 400	Withdraws 50 (not realizing A's update) Balance - 450

there should be £350 afterwards!

Lost Update Problem - Rollback

T1	T2
Read(X) $X = X + 10$	Read(X) $X = X + 10$
Write(X)	ROLLBACK
COMMIT	



Let us assume that the initial value of X is 30

The **Rollback** causes the value of X to go back to the original value i.e. 30 instead of 40 for T1. When it is committing, it is committing to value 30 which was the original value.

Dirty Read Problem

- ❑ Occurs when one transaction can see the intermediate results of another transaction before it has committed.

T1	T2
Read(X)	
$X = X + 10$	
Write(X)	
	Read(X)
	$X = X + 10$
	Write(X)
ROLLBACK	
	COMMIT

A read that T2 should not have seen. It read a value that would never exist because of the rollback

© 2013 Pearson Education, Inc. or its affiliate(s). All rights reserved. Pearson Education, Inc., publishing as Pearson Benjamin Cummings, 101 Philip Drive, Assinippi Park, New York, NY 10986-1997. Printed in the United States of America. This publication is protected by copyright. Permission to reproduce copies may be obtained from the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. For more information, contact C.C.C. at (978) 750-8400 or www.copyright.com.

- | Transaction by User A | Transaction by User B |
|-----------------------|-----------------------|
|-----------------------|-----------------------|

Read(X) = 1000	
User A transfers 200 to a friend	
Write(X) = 800	
	Read(X) - 800
	Write(X) - 800
ROLLBACK = 1000	
	COMMIT

Inconsistent Analysis Problem

- Occurs when a transaction reads several values from the database but a second transaction updates some of them during the execution of the first.

T1 (X=5)	T2 (Y=10)
Read(X)	Read(Y) Y=Y+5 Write(Y) Read(X) X = X - 5 Write(X)
Read(Y) Sum=X+Y COMMIT	COMMIT

Summing up data while it is being updated. Happens when data items are used for analysis while another transaction is modifying it.

Other concurrency problems

- ❑ **Fuzzy Read**
- ❑ **Phantom Read**



<https://padlet.com/ss259/distinguish-fuzzy-read-and-phantom-read-concurrency-problems-qusddl4rg3fk3k8>

Concurrency Control

- schedule transactions in such a way as to avoid any interference between them, and hence prevent the types of problem that might arise.
- **allow only one transaction to execute at a time:**
- Serialising the transactions.

Schedule(s)

When transactions are executed concurrently in an interleaved fashion, the order of execution of operations from the various transactions forms a transaction schedule (or history)

time	Schedule A		Schedule B	
	T ₁	T ₂	T ₁	T ₂
1	read_item(X);			read_item(X);
2	X:=X-N;			X:=X+M;
3	write_item(X);			write_item(X);
4	read_item(Y);		read_item(X);	
5	Y:=Y+N;		X:=X-N;	
6	write_item(Y);		write_item(X);	
7		read_item(X);	read_item(Y);	
8		X:=X+M;	Y:=Y+N;	
9		write_item(X);	write_item(Y);	

Schedule(s)

When transactions are executed concurrently in an interleaved fashion, the order of execution of operations from the various transactions forms a transaction schedule (or history)

Serial Schedule – no interleaved operations from other transactions

Non-serial Schedule – has interleaved operations from other transactions

Serial Schedule

Schedule 1:

$A = \text{£}1000$, $B = \text{£}2000$, $A + B = \text{£}3000$

T_1	T_2
$\text{read}(A)$ $A := A - 50$ $\text{write}(A)$ $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$ commit	$\text{read}(A)$ $\text{temp} := A * 0.1$ $A := A - \text{temp}$ $\text{write}(A)$ $\text{read}(B)$ $B := B + \text{temp}$ $\text{write}(B)$ commit

$A = ?$, $B = ?$: $A + B = ?$

Schedule 2:

$A = \text{£}1000$, $B = \text{£}2000$, $A + B = \text{£}3000$

T_1	T_2
$\text{read}(A)$ $A := A - 50$ $\text{write}(A)$ $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$ commit	$\text{read}(A)$ $\text{temp} := A * 0.1$ $A := A - \text{temp}$ $\text{write}(A)$ $\text{read}(B)$ $B := B + \text{temp}$ $\text{write}(B)$ commit

$A = ?$, $B = ?$, $A + B = ?$

Serial Schedule

Schedule 1:

$A = \text{£}1000, B = \text{£}2000, A+B = \text{£}3000$

T_1	T_2
$\text{read}(A)$ $A := A - 50$ $\text{write}(A)$ $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$ commit	$\text{read}(A)$ $\text{temp} := A * 0.1$ $A := A - \text{temp}$ $\text{write}(A)$ $\text{read}(B)$ $B := B + \text{temp}$ $\text{write}(B)$ commit

$A = \text{£}855, B = \text{£}2145, A+B = \text{£}3000$

Schedule 2:

$A = \text{£}1000, B = \text{£}2000, A+B = \text{£}3000$

T_1	T_2
$\text{read}(A)$ $A := A - 50$ $\text{write}(A)$ $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$ commit	$\text{read}(A)$ $\text{temp} := A * 0.1$ $A := A - \text{temp}$ $\text{write}(A)$ $\text{read}(B)$ $B := B + \text{temp}$ $\text{write}(B)$ commit

$A = \text{£}850, B = \text{£}2150, A+B = \text{£}3000$

Serial Schedule

Schedule 1:

$A = £1000, B = £2000, A + B = £3000$

T_1	T_2
$\text{read}(A)$ $A := A - 50$ $\text{write}(A)$ $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$ commit	$\text{read}(A)$ $A := A - 50$ $\text{write}(A)$ $\text{read}(B)$ $B := B + \text{temp}$ $\text{write}(B)$ commit

$A = £855, B = £2145, A + B = £3000$

Schedule 2:

$A = £1000, B = £2000, A + B = £3000$

T_1	T_2
$\text{read}(A)$ $A := A - 50$ $\text{write}(A)$ $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$ commit	$\text{read}(A)$ $A := A - 50$ $\text{write}(A)$ $\text{read}(B)$ $B := B + \text{temp}$ $\text{write}(B)$ commit

$A = £850, B = £2150, A + B = £3000$

Both transactions lead to the data being in a consistent state but **NO** CONCURRENT

Non-Serial Schedule

Schedule 3:

A=£1000, B=£2000: A+B=£3000

T_1	T_2
read(A) $A := A - 50$ write(A)	
	read(A) $temp := A * 0.1$ $A := A - temp$ write(A)
read(B) $B := B + 50$ write(B) commit	
	read(B) $B := B + temp$ write(B) commit

A=?, B=? : A+B=?

Schedule 3 is not a serial one, but it is **equivalent** to the serial schedule 1.

Ensures concurrency and leaves the system in a consistent state.

Non-Serial Schedule

Schedule 3:

A=£1000, B=£2000: A+B=£3000

T_1	T_2
read(A) $A := A - 50$ write(A)	
	read(A) $temp := A * 0.1$ $A := A - temp$ write(A)
read(B) $B := B + 50$ write(B) commit	
	read(B) $B := B + temp$ write(B) commit

A=£855, B=£2145: A+B=£3000

Schedule 3 is not a serial one, but it is **equivalent** to the serial schedule 1.

Ensures concurrency and leaves the system in a consistent state.

Schedules – Not always lucky

A=£1000, B=£2000: A+B=£3000

T_1	T_2
$\text{read}(A)$ $A := A - 50$	$\text{read}(A)$ $\text{temp} := A * 0.1$ $A := A - \text{temp}$ $\text{write}(A)$ $\text{read}(B)$
$\text{write}(A)$ $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$ commit	$B := B + \text{temp}$ $\text{write}(B)$ commit

Not equivalent to any serial schedule – **PROHIBITED!**

The job of the database management system is to ensure that the state is consistent.

A=?, B=?: A+B=?

Schedules – Not always lucky

A=£1000, B=£2000: A+B=£3000

T_1	T_2
$\text{read}(A)$ $A := A - 50$	$\text{read}(A)$ $\text{temp} := A * 0.1$ $A := A - \text{temp}$ $\text{write}(A)$ $\text{read}(B)$
$\text{write}(A)$ $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$ commit	$B := B + \text{temp}$ $\text{write}(B)$ commit

Not equivalent to any serial schedule – **PROHIBITED!**

The job of the database management system is to ensure that the state is consistent.

A=£950, B=£2100: A+B=£3050

Next up...

- How to non-serialise schedules that allow transactions to execute concurrently without interfering with one another, and thereby produce a database state that could be produced by a serial execution.

Summary



- Transactions
- Properties of Transactions
- States of execution of a transaction
- Concurrency Control
- Problems associated with concurrency control

Lab Exercise



- ❑ Download Instructions from Moodle