1. Good Evening

2. Lecture begins at 9:05 pm

3. Topic - Transactions

Agenda

1. Intro → 2 case studies
2. ACID properties
3. keywords = start, commit, rollback, demo
4. ISOLATION Levels (4 isolation levels)
5. Deadlocks
6. Back log = DataTypes, Builtin Fns.

INTRODUCTION TO TRANSACTIONS

Case Study 1 : Transfer of money from one bank account to another bank account

A → 200 → B

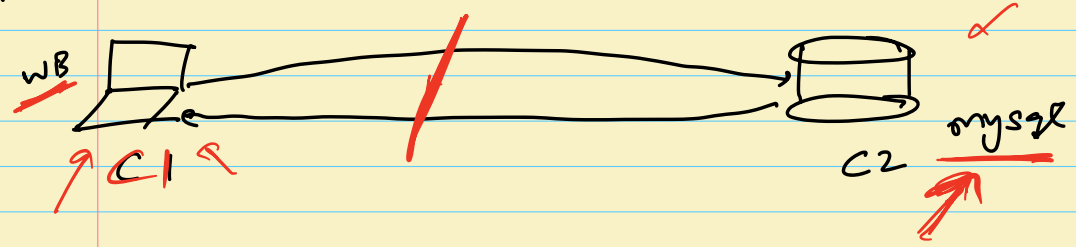
~~1000~~ 800

~~2000~~ 2200

Start transaction

1. x = Read(A). 1000
2. x = x - 200 800 ✓
3. Write(A, x) ← 800 ✓
4. x = Read(B) 2000 ✓
5. x = x + 200 2200 ✓
6. Write(B, x) ← 2200 ✓

commit

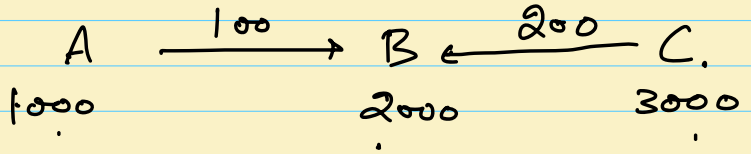


Data inconsistency can happen because of a failure / crash / network outage during a set of related operations.

→ Revert to original state.

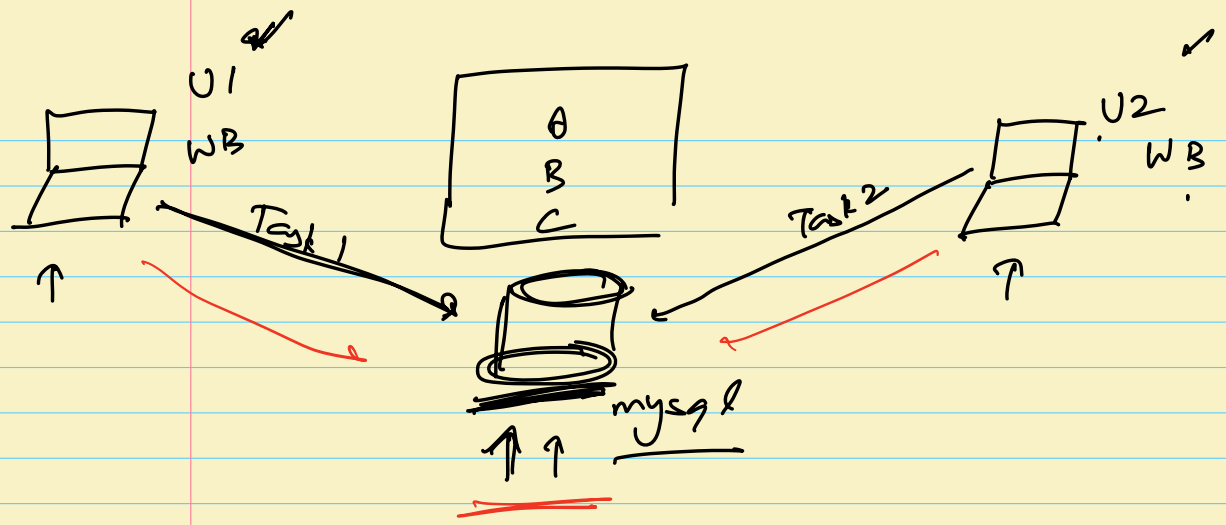
Transaction → A set of operations grouped / clubbed together to make one execution unit. [All or Nothing — Either all operations belonging to transaction will happen or none of them]

CASE STUDY 2 :

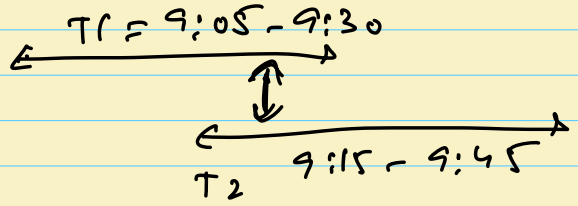


Task 1 = A $\xrightarrow{100}$ B.]

Task 2 = C $\xrightarrow{200}$ B]



Task1 & Task2 are concurrent.



Task1 & Task2 has some data which is shared.

→ If concurrent tasks (T1 & T2) are accessing shared data (DB), race conditions can happen leading to inconsistent data.

A $\xrightarrow{100}$ B $\xleftarrow{200}$ C

~~1000~~ 900 | ~~2000~~ ~~2100~~ 2200 | ~~3000~~ 2800

Task1 (A $\xrightarrow{100}$ B)		Task2 (C $\xrightarrow{200}$ B)	
a) 1	x = Read(A) 1000	a) 4	x = Read(C) 3000
b) 2	x = x - 100 900	b) 5	x = x - 200 2800
c) 3	Write(A, x) ~ 900	c) 6	Write(C, x) ~ 2800
d) 7	x = Read(B) 2000	d) 9	x = Read(B) 2000
e) 8	x = x + 100 2100	e) 10	x = x + 200 2200
f) 11	Write(B, x) ~ 2100	f) 12	Write(B, x) ~ 2200

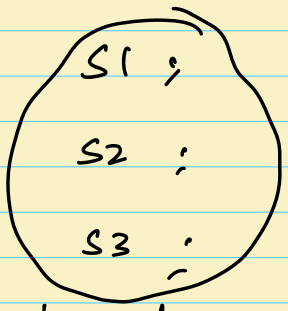
Race conditions might occur when multiple concurrent tasks access shared data, leading to data inconsistency.

- CASE1 → Crash/Failure during a set of related operations might lead to data inconsistency.
- CASE2 → Multiple concurrent task accessing shared data might lead to race conditions leading to inconsistent data.

Transactions: A set of operation grouped together to form one execution unit is called a transaction.

Keywords = start, commit, rollback [WB]

start transaction;



commit; / rollback;

ACID : Properties to be followed by transactions.

Atomic → A transaction should be atomic. All or none. {Either all statements of txn will run as none will}

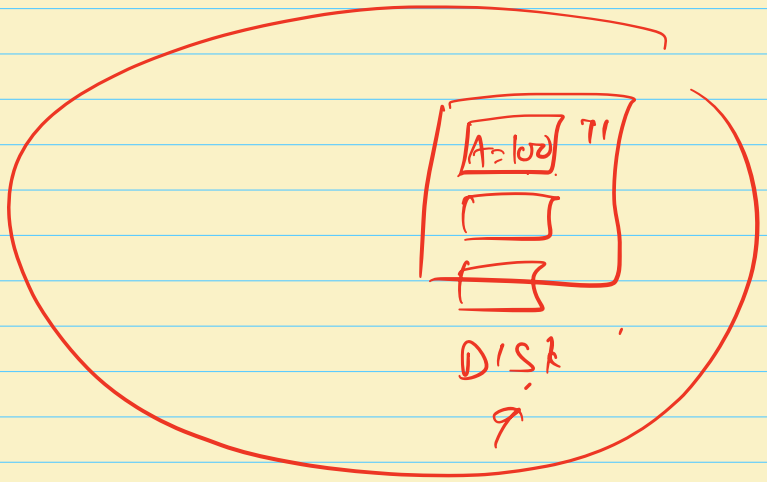
If any failure happens during set of operations of a txn, changes will be reverted.

Durability → Once the trn. is committed data changes are durable. Even if mysql fails the changes won't be lost.

Changes are written to hard disk

Consistency → Data should be consistent before & after every transaction.

S1 ✓
S2 ✓
S3 ✓
S4
S5
S6
commit



A $\xrightarrow{100}$ B

~~1000~~ 900 | 2000

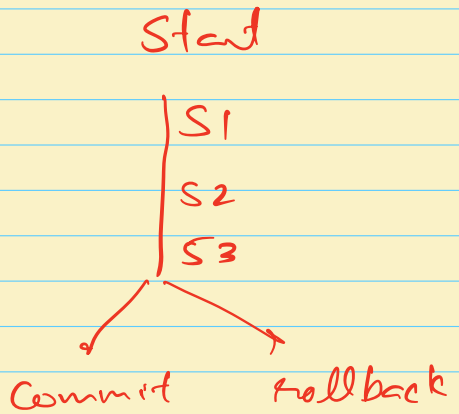
Start

1. x = Read(A) 1000
2. x = x - 100 900
3. Write (A, x) ← 900
4. isActive = Read(B)

✓ → 5. x = Read(B) 2000

X → [rollback;]

6. x = x + 100 2100
7. Write(B, x) ← 2100
8. Commit



BREAK = 10:39 to 10:49

1. ISOLATION (4 levels)
2. Deadlock
3. Backlog

ISOLATION LEVELS

1. READ UNCOMMITTED
2. READ COMMITTED
3. REPETABLE READ
4. SERIALIZABLE

SET SESSION TRANSACTION ISOLATION LEVEL " _____ "

start transaction

S1

S2

S3

commit / rollback

∀ all isolation levels → What?
→ Problem?
→ Benefits?

READ UNCOMMITTED

A = 1500

Current Transaction (set A to 1000)

1. x = Read (A) ✓ 2000

2. Write (A, 1000) ✓ 1000

READ UNCOMMITTED my Transaction (Inc A by 500)

Dirty Read ↓

3. x = Read(A) 1000

5. x = x + 500 1500

6. Write(A, x) 1500

7. COMMIT

4. ROLLBACK

What is Read Committed?

Problem ⟹ Dirty Read

Scenario in world where it is ok to use "Read Uncommitted"

Read Uncommitted has the highest performance of all 4 isolation levels.

But it suffer from data inconsistency problem called dirty read.

Financial Appn. → ✗

Booking.com → ✗

Social Media → ✓

Data accuracy is less important than performance & scale.

READ COMMITTED → What?
→ Benefit?
→ Problem?

will not suffer from "dirty reads"

o my trans will read only committed data.

A = 2500

currentTrans (Set A to 1000)		myTrans (Inc A by 500) READ COMMITTED
1. x = Read(A) ✓	2000	
2. Write(A, 1000)	A = 1000	
		3. x = Read(A) ✓ 2000
4. COMMIT ✓		
		5. x = x + 500 ✓ 2500
		6. Write(A, x) A = 2500
		7. COMMIT T

Dirty Read is solved.

* [Lost updates]. *

Non Repeatable Reads.:

Read Committed solves dirty read, but suffers from another

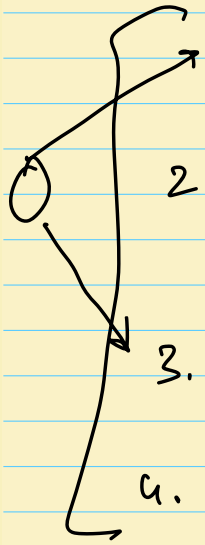
data inconsistency problem called

non repeatable read.

Students

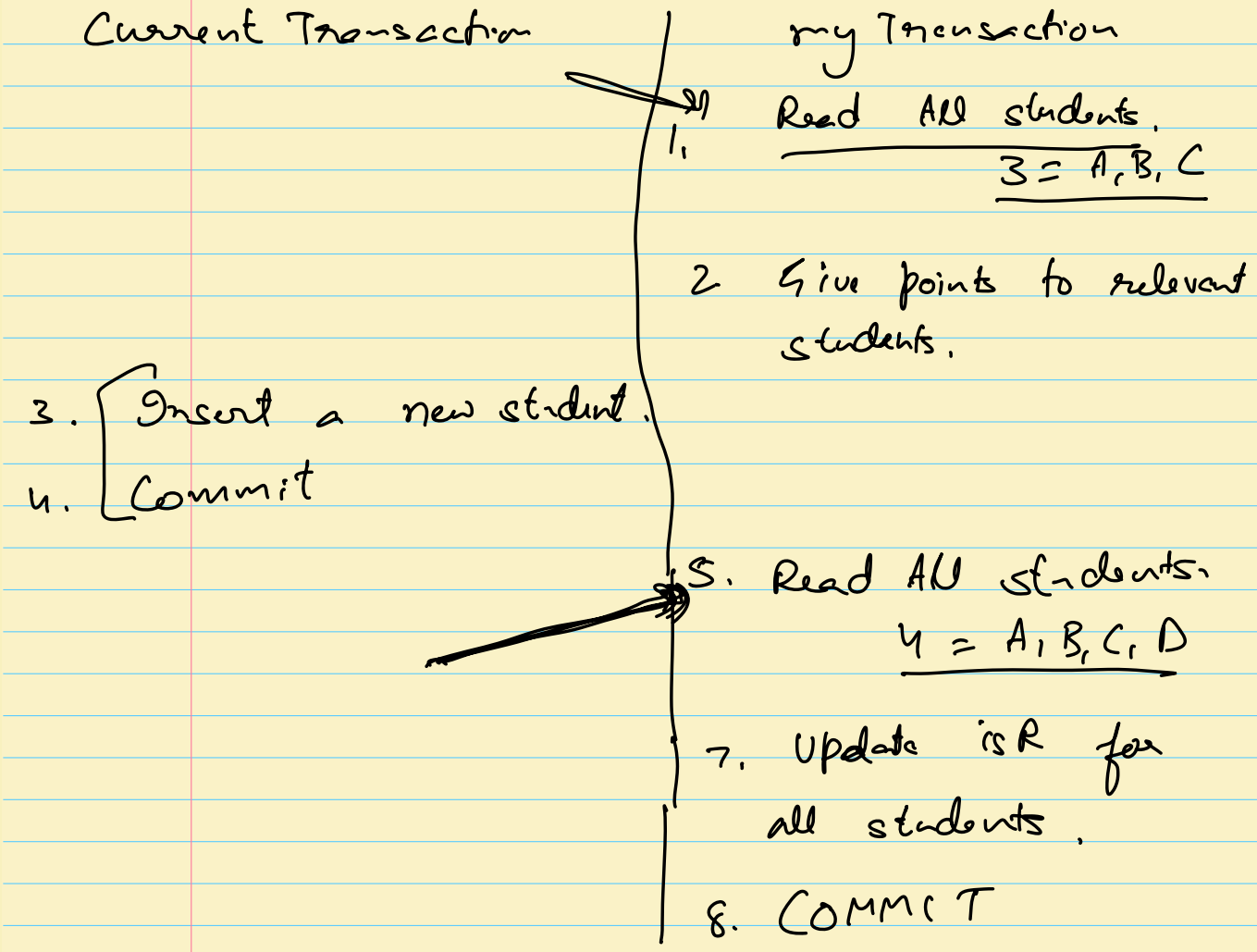
id	name	psp	points	isR
1	A	82	100 + 1000	X ✓
2	B	75	200	X ✓
3	C	90	300 + 1000	X ✓
4	D	85	0	X ✓

Task



1. Read All Students.
2. To all students with psp ≥ 80, add 1000 points.
3. Read All students.
4. Update isR to true for all students

READ COMMITTED



This called a non-repeatable read.

REPEATABLE READ = Default

CurrentTransaction

REPEATABLE READ
myTransaction.

1. Read All students
(3 = A, B, C)

time 2. Give points.

3. Insert new student

4. Commit

5. Read All students.
(3 = A, B, C)

6. Update isR

Students 7. COMMIT

id	name	psp	points	isR
1	A	82	100 → 1000	✗ ✓
2	B	78	200	✗ ✓
3	C	90	300 → 1000	✗ ✓
4	D	85	0	✗

Backlog =
1. Demo of 3 Isolation Levels.
2. SERIALIZABLE
3. Deadlocks
4. Data Types
5. Builtin Fns.