Transaction management

What is a transaction?

- ACID properties of transactions.

The need for concurrency control

- Lost Update problem
- Uncommitted update
- Inconsistent analysis

Serializability

- Locking (deadlock & tow-phase locking)

Recovery

- Deferred updates
- Checkpoints

Learning Outcomes

- Demonstrate understanding of transactions and the properties of transactions
- Understand the principles of simultaneous processing
- Be able to identify the problems with concurrency control
- Be able to identify the problems with serializabilty and the techniques to manage
- Describe problems and facilities for data recovery

What is a Transaction?

“An operation that is occurring on the database”.

What's a transaction?

- Logical unit of work/program/single command operations occurring within the database.
- A single SQL command.
- A single transaction can involve a number of operations being performs on the database.

BEGIN TRANSACTION

INSERT
UPDATE → **SELECT DISTINCT saledate**
SELECT **FROM avon_sales**
ORDER BY saledate;

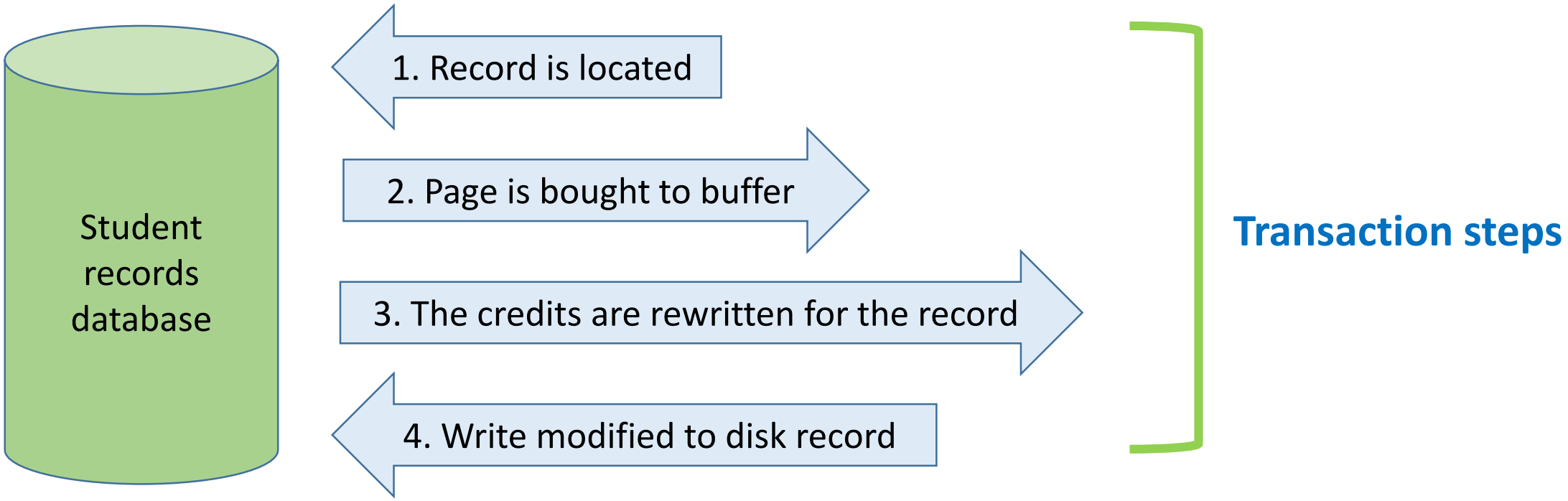
END TRANSACTION

ROLLBACK
ABORT

COMMIT

What's a transaction?

Operation: update the number of credits for student 216987123 to 30 credits



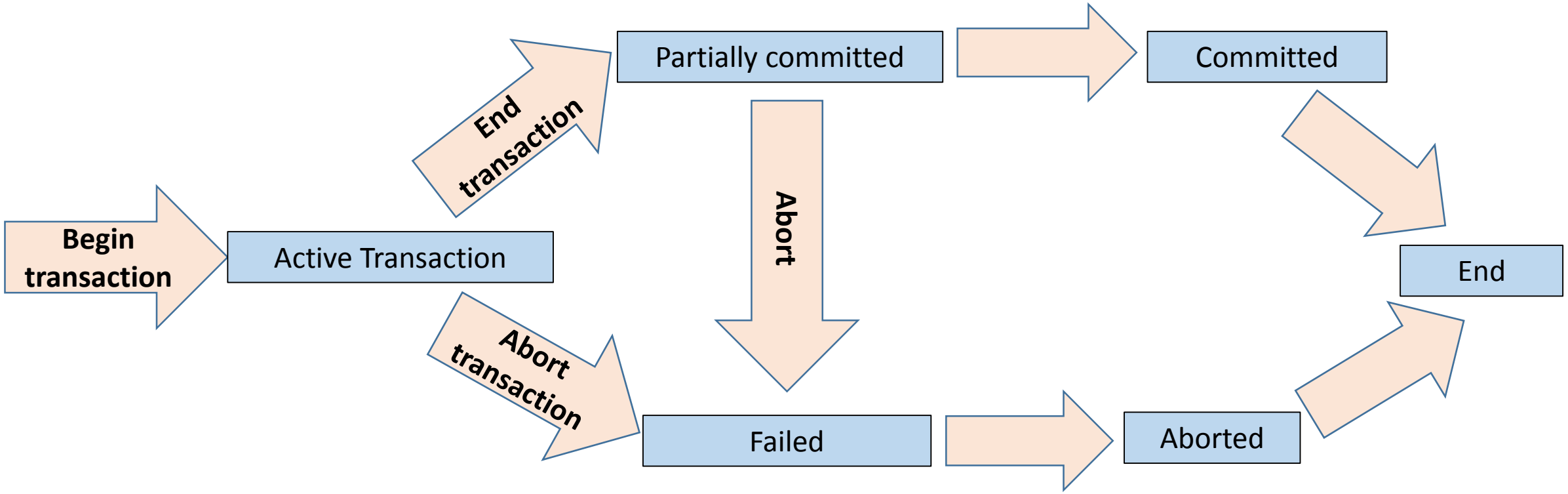
!A transaction must bring the database from one consistent state to another.

Transaction rules

- At the end of the transaction all occurrences in the database **must agree**.
- A transaction is the **entire series of steps** necessary to accomplish a logical unit of work.
- The transaction must bring the database from and to a **new consistent state**.
- While the transaction is in mid-operation the database can be in an **inconsistent state**.
- A transaction is an **atomic process**: a single all or none unit.
- Partial executions are not permitted as this leaves the database in an inconsistent state.
- If a transaction was committed and this was an error a **compensating transaction** can be run to reverse the effects.
- A transaction that has executed successfully is said to be **committed.**
- If a transaction cannot complete successfully the transaction is **aborted.**

1. There are two ways in which a transaction can end – committed or aborted.

2. If a transaction cannot commit is must be aborted to return the database back to a consistent state.



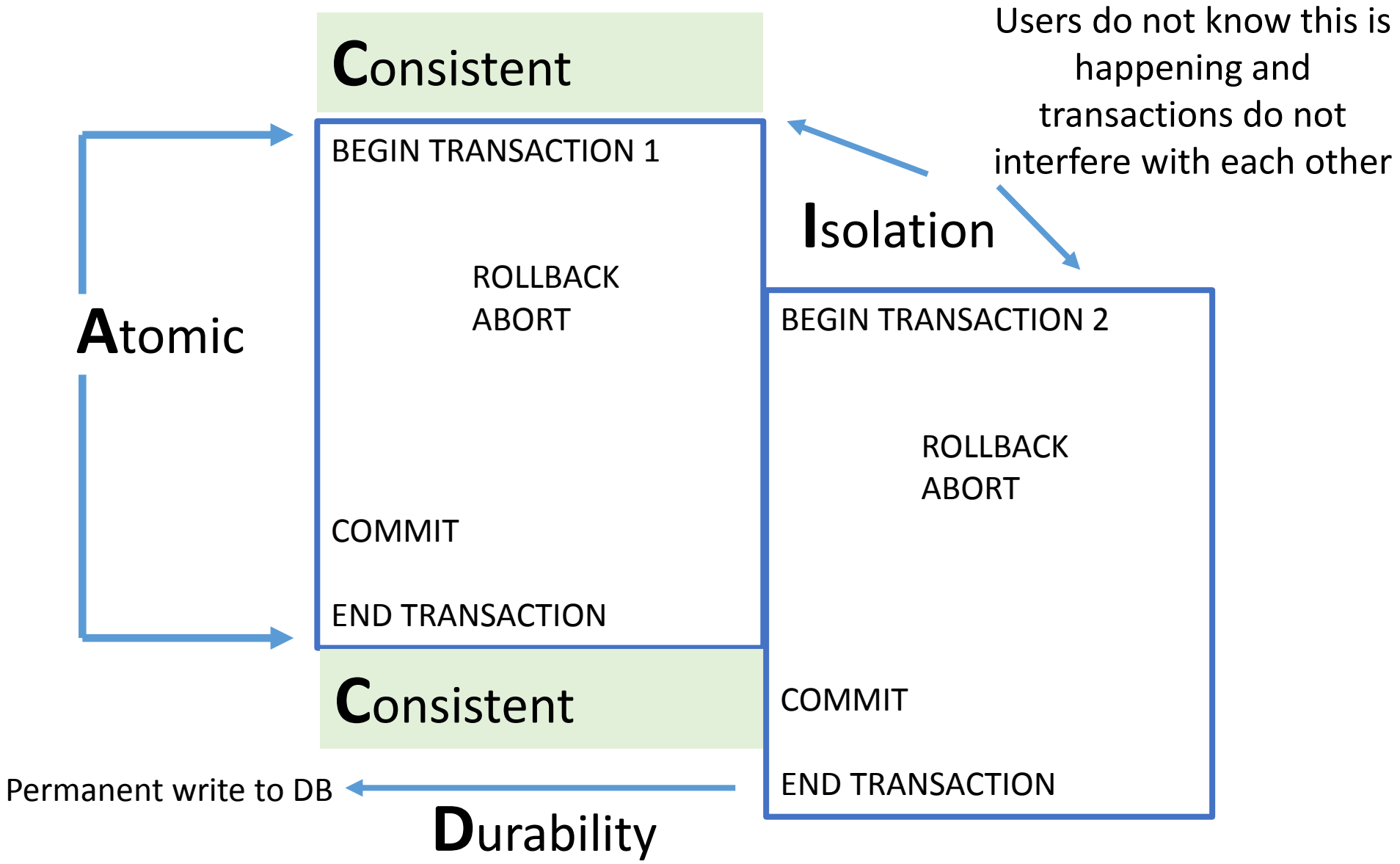
3. Once a transaction is committed it cannot be aborted.

4. If a transaction has been committed and it was a mistake a compensating transaction must be performed to reverse the effects.

5. A rolled back transaction can be re-run and could execute and commit successfully.

Ricardo&Urban (2017) p396

ACID properties of transactions



ACID properties of transactions

Atomicity: all or none unit. Either the entire set of database update operations are carried out or none. To ensure this the DBMS maintains a log, so transactions can be rolled back.

Consistency: ensure consistency of multiple transactions when they execute at the same (concurrency subsystem part of DBMS).

Isolation: several transactions executing at the same time, with interleaving operations. Changes made to one transaction is not seen by another transaction until it has committed. This gives the effect that one transaction is being executed after another (concurrency control subsystem part of DBMS).

Durability: once a transaction is committed that the changes are permanent in the database even if the database crashes before all the writes have been initiated (recovery subsystem part of DBMS).

Concurrency control

"The ability to manage simultaneous processes involving the database"

Need for concurrency control

- Without care a database can be easily damaged – concurrency control can help avoid this situation.
- Concurrency control – managing the simultaneous processes that are occurring (serializability).
- Users access the data simultaneously.
- Control transactions so they don’t interfere with one another

Concurrency control is needed to avoid loss of information or corruption of the database.

Concurrency control

- In a multi-user environment, simultaneous access to data can result in interference and data loss where transactions conflict
- Concurrency Control - the process of managing simultaneous transactions
- Problems caused by concurrency:
  - lost update problem,
  - uncommitted update and
  - inconsistent analysis.

Lost update problem

John Transaction	Time	Marsha Transaction
BEGIN TRANSACTION	T1	
Read BAL (£1000)	T2	BEGIN TRANSACTION
	T3	Read BAL (£1000)
DEBIT (£200) (Balance = £800)	T4	
WRITE balance (£800)	T5	
COMMIT	T6	
	T7	CREDIT (£100) (Balance = £1100)
	T8	…
	T9	WRITE balance (£1100)
	T10	COMMIT

The final balance should be £900

- Each transaction is unaware of the other transaction and the changes
- Each transaction reads the balance and uses this for the calculation.
- Marsha's transaction overwrites Jacks update to the account balance by withdrawal of £200.
- According to the schedule John's transaction is lost.

Lost update problem

TIME	Jack's Transaction	Jill's Transaction	Account Balance
T1	BEGIN TRANSACTION		
T2	Read BAL (£1000)		£1000
T3		BEGIN TRANSACTION	
T4		Read BAL (£1000)	£1000
T5	BAL = BAL-£50 (£950)		£950
T6		BAL = BAL+100 (£1100)	
T7	WRITE BAL (£950)		£950
T8	COMMIT		
T9		WRITE BAL (£1100)	£1100
T10		COMMIT	£1100

- **Can you describe the problem with the transaction?**
- **Can you calculate the correct balance of the account?**

Lost update problem

1. The balance at start of schedule is £1000.
2. The Time column shows the time at which the transaction operation is taking place, and because transactions are interleaved we can see that Jack's transaction and Jill's transaction are being handled simultaneously.
3. The transaction for Jack commences at time T1 with the BEGIN TRANSACTION.
4. Jack's transaction reads the balance correctly as £1000
5. However at time T3 Jill's transaction begins and also reads the account balance as £1000, at this point not commit has occurred for Jacks transaction - so the state is still consistent and correct.
6. At T5 Jack's transaction debits the account with £50 but this transaction does not commit at this point in time.
7. At T6 Jill's transaction interleaves and credits the account with £100
8. At T7 Jack's transaction writes the balance of £950 after the £10 withdrawal.
9. At T8 Jack's transaction COMMITS
10. At T9 and T10 Jill's transaction WRITE's the balance and COMMITS and overwrites Jack's transaction amount with £1100.
11. This is called lost update because Jacks transaction is lost, the correct balance should be £1050 after both transactions have completed Jack (1000 – 50 = £950) and Jill's (950 + 100 = £1050).

Uncommitted update explained

TIME	Transaction 1	Transaction 2	BALANCE
t1	Read BAL (£5000)		£5000
T2	Debit £500 (5000-500 =£4500)		£5000
T3	Write BAL £4500		£4500
T4		Read BAL (£4500)	£4500
T5		Credit £1000 (4500+1000= £5500)	£4500
T6		Write BAL (£5500)	£5500
T7		COMMIT	£5500
T8	ROLLBACK		

- **This issue occurs when the first transaction is permitted to modify a value, which is read by a second transaction, and the first transaction does not commit but rolls backs. The data that the second transaction is using is invalidated. Also called dirty read problem.**

Uncommitted update

Time	DEPOSIT	INTEREST	BALANCE
T1	BEGIN TRANSACTION		£1000
T2	READ BAL (£1000)		£1000
T3	BAL = BAL + 1000		£1000
T4	WRITE BAL (£2000)		£2000
T5		BEGIN TRANSACTION	
T6		READ BAL (£2000)	
T7		BAL = BAL * 1.01	£2000
T8	ROLLBACK		£1000
T9		WRITE BAL (£2020)	£2020
T10		COMMIT	£2020

- **Can you describe the problem with the transaction?**
- **Can you calculate the correct balance of the account?**

Uncommitted update

1. The starting balance for the account is £1000 at T1
2. At T2 balance is read £1000
3. At T3 balance is credited with £1000 and at T4 the balance is written £2000. Note: Transactions can only be rolled back if the transaction has not been committed.
4. At T6 the second transaction reads the balance £2000 and at T7 the interest is calculated on £2000 balance and the balance written.
5. However at T8 the credit is rolled back. The interest transaction committed an interest amount which was calculated and applied to the credited account balance. Therefore the interest applied on the account is incorrect.
6. The correct interest applied on £1000 balance is £1010.10.

Inconsistent analysis problem

- This occurs when the first transaction reads a number of values, but a second transaction updates a number of the values before the first transaction has a chance to commit.

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

Mcfadden, Hoffer and Prescott (2000) p507

Concurrency control

Other problem which arise with concurrency control are:

- **Non repeatable read problem**
- **Phantom data problem**

For wider reading look at these problems in your own time.

Serializability

"Serial execution of transactions means transactions are performing one after the another without interleaving operations".

Serializability

- Serial execution are when transactions are executed one after the other.
- There is no interleaving of operations.
- For two transactions A and B, the possibility of serial execution is A followed by B or B followed by A.
- For some transactions the order of the execution is important.
- With serial execution it is assumed that the database is left in a consistent state and correct. Even if the order of the transaction produced different results.
- If a set of transactions is executed concurrently it is called serializable.

Serializability

Transaction 1	Transaction 2	Data	Conflict	Order
Reading data	Reading data	Same	No conflict	order not important
Reading data	Reading data	Different	No conflict	order not important
Writing data	Reading data	Different	No conflict	order not important
Writing data	Reading data	Same	Conflict	order is important

- Is it essential to guarantee serializability of concurrent transactions in order to ensure correctness.

Serializability

Tranasction1
BEGIN TRANSACTION
READ BALANCE
BALANCE = BALANCE + £50000)
WRITE BALANCE
COMMIT
END TRANSACTION

Transaction2
BEGIN TRANSACTION
READ BALANCE
BALANCE= BALANCE + INTEREST (2.5%)
WRITE BALANCE
COMMIT
END TRANSACTION

The starting balance is £50000. For the following two transactions what will be the balance on the account if the transactions are run serially Transaction 1 followed by Transaction 2?

Serializability

T1	
BEGIN TRANSACTION	
READ BALANCE	50000
BALANCE = BALANCE + £50000)	50000 + 50000 =100000
WRITE BALANCE	100000
COMMIT	COMMIT 100000
END TRANSACTION	
T2	
BEGIN TRANSACTION	
READ BALANCE	100000
BALANCE= BALANCE + INTEREST (2.5%)	100000 + (100000 /100 * 2.5 = 2500) = 102500
WRITE BALANCE	102500
COMMIT	COMMIT 102500
END TRANSACTION	

Serializability

Tranasction1
BEGIN TRANSACTION
READ BALANCE
BALANCE = BALANCE + £50000)
WRITE BALANCE
COMMIT
END TRANSACTION

Transaction2
BEGIN TRANSACTION
READ BALANCE
BALANCE= BALANCE + INTEREST (2.5%)
WRITE BALANCE
COMMIT
END TRANSACTION

The starting balance is £50000. For the following two transactions what will be the balance on the account if the transactions are run serially Transaction 2 followed by Transaction 1?

Serializability

T2	
BEGIN TRANSACTION	
READ BALANCE	50000
BALANCE= BALANCE + INTEREST (2.5%)	50000 + (50000 /100 * 2.5 =) = 51250
WRITE BALANCE	51250
COMMIT	COMMIT 51250
END TRANSACTION	
T1	
BEGIN TRANSACTION	
READ BALANCE	51250
BALANCE = BALANCE + £50000)	51250 + 50000 = 101250
WRITE BALANCE	101250
COMMIT	COMMIT 101250
END TRANSACTION	

Locking

- Locking ensures serializability by allowing a transaction to lock an object to prevent another transaction from accessing or modifying it.
- Objects from the entire database to single data items can be locked.
- Locking: by flag on the data item or keeping a list.
- Two types of lock shared and exclusive.
- A transaction must acquire a lock on any item it needs to read or write.

Types of locks: shared lock

BEGIN TRANSACTION 1

s-lock at time T1

READ BALANCE
:
:
:
:
RELEASE S-LOCK
COMMIT
END TRANSACTION

BEGIN TRANSACTION 2

s-lock at time T3
e-lock denied

READ BALANCE
:
:
:
:
RELEASE S-LOCK
COMMIT
END TRANSACTION

- S-lock allows transactions to read, but not to update. Placing a s-lock will prevent another operation from placing an e-lock on the data.

- Read access a shared lock can be acquired

Types of locks – exclusive lock

BEGIN TRANSACTION 1

e-lock at time T1

READ BALANCE
:
:
:
:
RELEASE S-LOCK
COMMIT
END TRANSACTION

BEGIN TRANSACTION 2

s-lock – DENIED
e-lock - DENIED

READ BALANCE
:
:
:
:
RELEASE S-LOCK
COMMIT
END TRANSACTION

- e-lock prevent another transaction from reading the data until the write operation is complete.

- For write access a exclusive lock needs to be acquired.

Problem with locking: deadlock

- When two or more transactions have locked common resources, and each is waiting for the other to unlock their resources

TIME	Transaction 5	Transaction 6
T1	REQUEST e-lock (A)	
T2	GRANT e-lock (A)	
T3		REQUEST e-lock (B)
T4		GRANT e-lock (B)
T5	GRANT e-lock (B)	REQUEST e-lock (A)
T6	Waiting	Waiting
T7	Waiting	Waiting
T8	Waiting	Waiting
T9	…	….

FIGURE 9.9 P408 Ricardo and Urban 2017

Managing Deadlock

- Deadlock prevention:
  - Deadlock detection - lock all records required at the beginning of a transaction
  - May be difficult to determine all needed resources in advance

- Deadlock detection and recovery:
  - Allow deadlocks to occur
  - Mechanisms for detecting and breaking them
  - Need graphing to show transactions waiting for resources

Two-phase locking

- Two phases growing (acquiring locks) and shrinking (releasing locks).
- There is no requirement for all the locks to be acquired at the start.
- Normally some locks acquired, then processing occurs and then further locks acquired.
- Locks are not released until it has reached a stage where no new locks are needed.

Recovery

"The process of restoring the database to correct state in the event of failure".

Database failures

- Natural disaster: fire flood, earthquake and power outage.
- Sabotage: intentional contamination, destruction or data.
- Unintentional: deletion and overwriting data.
- Hardware malfunction: disk failure
- System crashes: loss of cache memory
- Software errors: abnormal termination and update effects.

Database recovery

- Mechanism for restoring transactions quickly and accurately after loss or damage.
- Ensuring atomicity and durability of transactions.
- Recovery facilities:
  - **Backup**
  - Deferred update protocol
  - **Checkpoints**
  - Immediate Update Protocol
  - Mirrored disks
  - Recovery manager

Back-ups

- A DBMS copy utility that produces backup copy of the entire database or subset
- Periodical backups (e.g. nightly, weekly and incremental)
- Cold backup–database is shut down during backup
- Backups stored in secure and off-site location

Checkpoints

- Periodically force writing modified pages from buffer to disk
- Writing all logs from memory to disk
- A checkpoint record for rollback

References

- Modern database management (2000) McFadden, Hoffer and Prescott
- Databases illuminated (2017) Ricardo and Urban