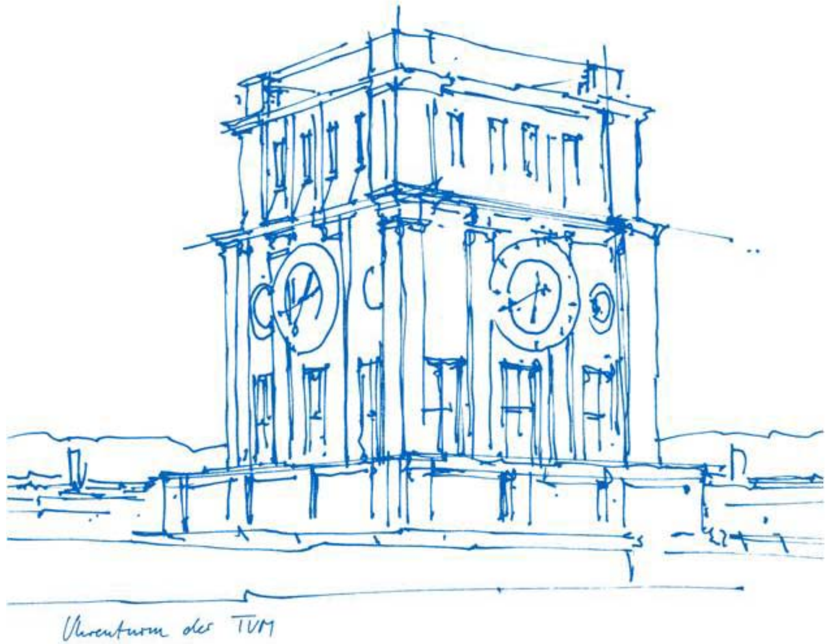
Transactions



What is a transaction?

A sequence of operations executed on the database

Ensure database integrety

Error handling

Transfer money from account A to account B	
Account A: subtract 100€	
	Account B: add 100€

← **Crash?**

ACID principle

Atomicity:

- Transaction must be non-dividable like an atom
- All individual operations must be succesful
- Should one operation fail, the data is reset to ist original state

ACID principle

Consistency:

- Before transaction is completed, the system checks if the changed data fulfills integrity constraints defined in the schema

ACID principle

Isolation:

- Simultanious transactions potentially influence each other
- If that is the case, anomalies can occur
- Isolation levels define what anomalies are allowed to occur

ACID principle

Durabulity:

- After a transaction is successfully completed, the data persists even in case of system failure
- Often by transaction logs written to non-volatile memory

Types of failure

Application error:

- Undo transaction

System Crash, Main Memory lost:

- Redo already committed transactions
- Undo active transactions

System crash, disks lost:

- Restore DB from backup on tape

Isolation

Concurrent transactions potentially influence the outcome of each other.
The isolation level determines how strong this influence is.
Depends on what influences you want to allow

➡ Anomalies

Types of anomalies (1): Lost Update

One change in the data is overwritten by another simultanious transaction

T1: Sell 5 tickets	T2: redeem 3 tickets
Read how many tickets I have: 100	
	Read how many tickets I have: 100
Sell 5 tickets: 105	
	Redeem 3 tickets:97

We end up with 97 sold tickets when in fact we sold 102

Types of anomalies (2): Dirty Read

We read data that is not confirmed to be correct

T1: Sell 5 tickets; abort	T2: redeem 3 tickets
Read how many tickets I have: 100	
Sell 5 tickets: 105	
	Read how many tickets I have: 105
	Redeem 3 tickets: 102
abort	

We end up with 102 sold tickets when in fact we sold 97

Types of anomalies (3): Non-repeatable read

The same query returns different results

T1: How many tickets do I have?	T2: redeem 3 tickets
Read how many tickets I have: 100	
	Redeem 3 tickets: 97
Read how many tickets I have: 97	

T1 runs the same query twice but gets different results

Types of anomalies (4): Phantom read

Special case of non-repeatable read.
During a select where, sum() or count() in T1 new rows are added by T2

T1: Value of all tickets sold	T2: Sell 5 more tickets
Select sum(value) from tickets	
	Sell 5 more tickets
Select sum(value) from tickets	

Query in T1 returns different results

Synchronization

→Goal: criterion for correctness (logical single usermode, i.e. avoiding all multi user anomalies)

→Formal criterion for correctness: Serializability (the outcome is equal to the outcome of its transactions executed serially, i.e. without overlapping in time)

- →BUT: Serializability restricts parallel execution of transactions (accepting anomalies enables less hindurance of transactions – use carefully!)

Locking

• With lock conflict requesting transaction has to wait until incompatible lock(s) is (are) removed

- Locks are potentially held until end of transaction

Different types of locking:

- RX-locking
- Deadlock
- Timeout

Isolation levels

- Read uncommitted (lowest level, access to unwritten data)
- Read committed (only reading definitively written values, but nonrepeatable read possible)
- Repeatable read (Np nonrepeatable read, but phantom problems)
- Serializible (Transaction sees only changes that were committed at the beginning of the transaction (and own changes))

Isolation levels	Dirty read	Nonrepeatable read	Phantom read
Read uncommitted	+	+	+
Read committed	-	+	+
Repeatable read	-	-	+
Serializable	-	-	-

MVCC – Snapshot Isolation

- Definition: each transaction sees the database in that state it was when the transaction started = reads the last committed values that existed at the time it started
- It guarantees that all reads made in a transaction will see a consistent snapshot of the database
- There are only write-write conflicts checked before commit

Advantages	Disadvantages
No reader waits for a writer	Needs more space for new versions
No writer waits for a reader	Needs cleaning

→ Good if mainly read transactions

→ Snapshot isolation may lead to non serializable schedules!

Questions (1) :

- What does ACID stand for?
- Which anomaly do we have here?

Step	T1	T2
1	read(A, a1)	
2	a1 = a1 – 300	
3	write(A, a1)	
4		read(A, a2)
5		a2 = a2 * 1,03
6		write(A, a2)
7	read(B, b1)	
8	...	
9	abort	

dirty read:
T1 is aborted, but T2 uses the values written in T1 for calculations

Questions (2):

- What means Serializability?

- What is a snapshot isolation and what are the advantages/disadvantages of a snapshot isolation?