W7

Transaction part1

A transaction is a unit of program execution that accesses and possibly updates various data items

两个主要问题：

1. 并发
2. 恢复

ACID

原子性（Atomicity）：事务中的所有操作要么全部发生，要么全都不发生。

一致性（Consistency）：数据库与其约束保持一致。

隔离性（Isolation）：看起来好像我们一次只运行 1 个事务（即使它们实际上是并发运行的）

持久性（Durability）：事务完成后，其结果将持久保存在数据库中。

example

The following is a transaction to transfer $50 from account A to account B:

1. read(A)
2. A := A - 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)

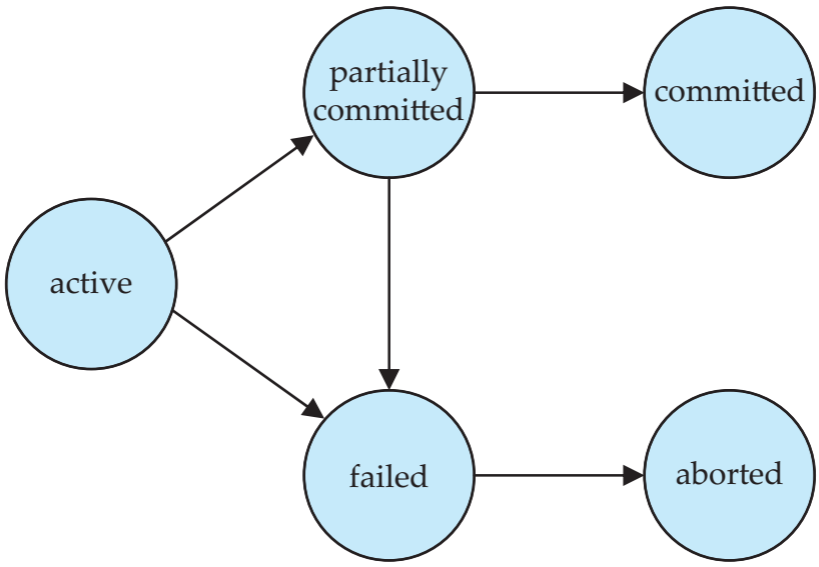
Atomicity : 3-6之间失败了就不执行全部

Consistency : a+b不变

Isolation : 3-6之间另一个事务进来数据库是不一致的

Durability: 已经commit的就会更新到database里

Transaction States



- Active：初始状态
- Partially committed：没有commit
- Failed: 停止执行
- Aborted: 回到之前的状态
- Committed : 成功执行

事务成功执行后会有commit指令作为最后一句语句，反之是abort

concurrent

需要并行控制机制来实现隔离

Schedules

a sequence of instructions that specifies the chronological order in which concurrent transactions are executed

指定并发事务的顺序

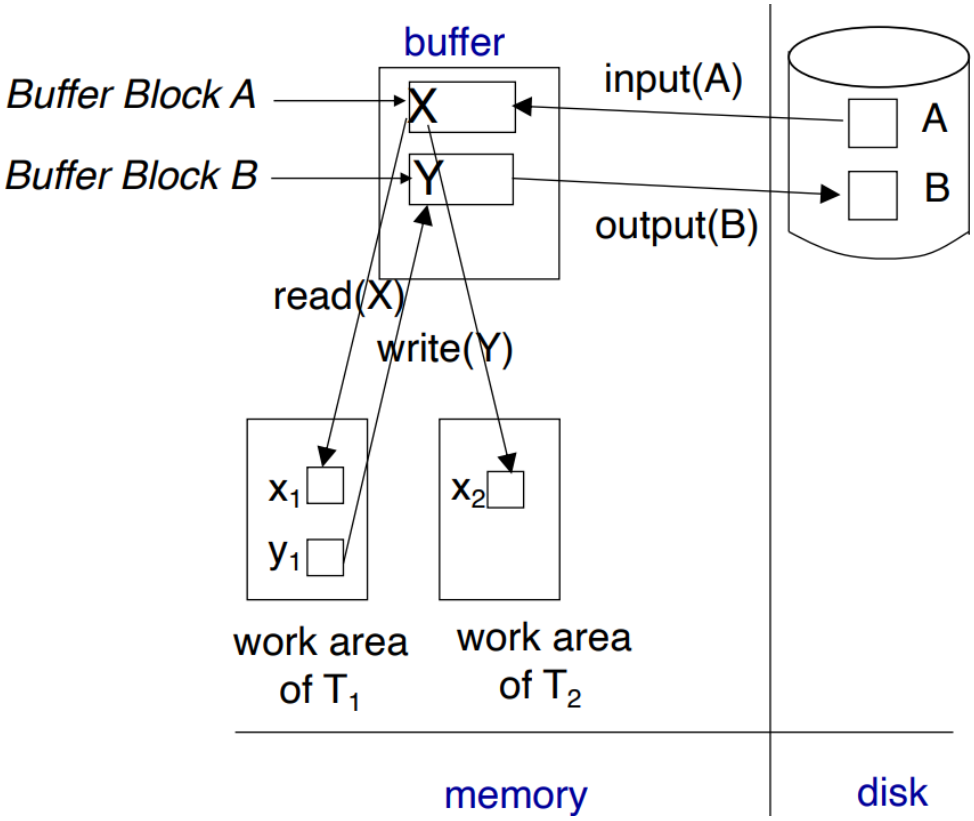
data access

分为物理块和buffer块，磁盘和主存之间的传输：

- input(B) 从物理块B传输到buffer
- output(B) 从buffer块B传输到磁盘并替换block

每个事务有各自的工作区但共享一个buffer

- read(X) 从buffer读取X到工作区
- write(Y) 从工作区写入Y到buffer



Transaction part2

Serialisability

假设：每个事务保持数据库的一致性

Def: A concurrent schedule is **serialisable** if it is **equivalent to a serial schedule**.

如果一个调度的执行结果等价于某个串行执行的事务结果，则该调度是可序列化的。

Conflicts

两个事务中的操作会发生冲突，当且仅当两个操作access同一个数据块，并且其中一个为write

写-读冲突（Write-Read Conflict, WR）

reading uncommitted data

当一个事务读取另一个未提交事务写入的数据时，会导致“脏读”。

会导致操作的覆盖

T1	T2
read(A)	
A:= A – 50	
write(A)	
	read(A)
	A:= A + A*10%
	write(A)
	read(B)
	B:= B + B*10%
	write(B)
	commit
read(B)	
B:= B + 50	
write(B)	
Rollback	

读-写冲突（Read-Write Conflict, RW)

unrepeatable reads

如果一个事务读取数据后，另一个事务写入相同数据，可能导致读取不一致。

T1	T2
read(A)	
	read(A)
	A:=A-1
	write(A)
	commit
read(A)	
A:=A-1	
write(A)	
commit	

Let A=1 and assume that there is an integrity constraint that prevents A from becoming negative. T1 will get an error.

写-写冲突（Write-Write Conflict, WW）

overwriting uncommitted data

如果两个事务对相同数据进行未提交的写操作，会导致“盲写”。

T1	T2
write(*Steven*)	
	write(*Paul*)
	write(*Steven*)
	commit
write(*Paul*)	
commit	

Conflict Serialisability

冲突可序列化指的是通过交换不冲突的操作，可以将一个非串行调度转换为一个串行调度。

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

swaps of non-conflicting instructions. →

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

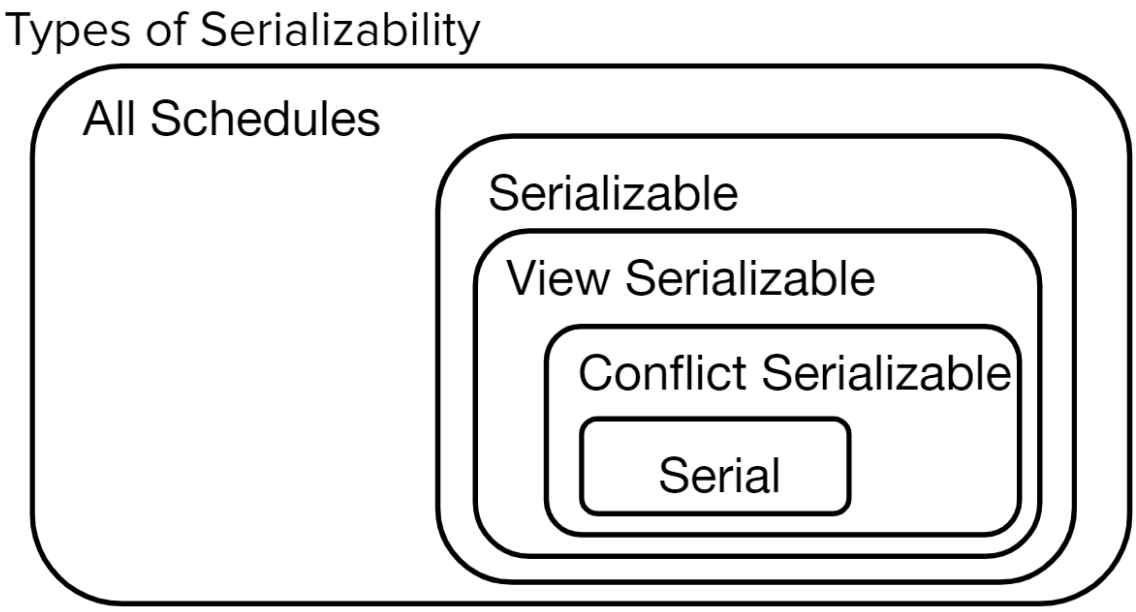
测试冲突可串行化的方法，两种图示

Precedence Graph

构建前序图以测试冲突可序列化性，如果图中无环(acyclic)，则该调度是冲突可序列化的。

要为计划绘制冲突依赖关系图，请将每个事务表示为一个节点。然后，如果**这两个操作在不同的事务中完成，并且至少有一个操作是写入，**则从前面的操作到后面的操作绘制一个箭头。

要查找等效计划，请对所有涉及的图形运行拓扑排序。**所有冲突可序列化计划都有一个无环依赖关系图。**因此，如果图形具有循环，则它不是冲突可序列化的。



View Serialisability

两个调度在以下情况下视图等价：

1. 初始读相同。
2. 中间读匹配相同的写。
3. 最终写相同。

Every conflict serialisable schedule is also view serialisable but not vice versa

Testing view serialisability is NP-complete

Recovery

if a transaction Tj reads a data item previously written by a transaction Ti , then the commit operation of Ti appears before the commit operation of Tj

Cascadeless

比recovery更严格

一个事务不会读取另一个未提交事务的写入数据。

Tj reads a data item previously written by Ti , the commit operation of Ti appears before the read operation of Tj

Ti write, Ti commit, Tj read.

这意味着如果一个事务失败并回滚，不会影响到其他事务，因为其他事务没有依赖于未提交的数据。