>>

Transaction Schedules

- Transaction Schedules
- Serial Schedules
- Concurrent Schedules
- Example Update Anomaly

∧ >>

❖ Transaction Schedules

When reasoning about transactions, we consider only

- `READ` - transfer data item from database to memory
- `WRITE` - transfer data item from memory to database
- `BEGIN` - start a transaction
- `COMMIT` - successfully complete a transaction
- `ABORT` - fail a transaction and unwind effects

All other operations are ignored (e.g. addition, testing, ...)

- take place in the memory space of one transaction
- have no affect on other transactions

<< ∧ >>

❖ Transaction Schedules (cont)

Relating SQL to database reads/writes ...

- **`SELECT`** produces **`READ`** operations on the database
- **`INSERT`** produces **`WRITE`** operations
- **`UPDATE`**, **`DELETE`** produce both **`READ`** + **`WRITE`** operations

Assume: each operation involves one database item (e.g. one tuple)

Notation: items denoted **`X`**, **`Y`**, etc; operations denoted **`R`**, **`W`**, **`C`**, **`A`**

Thus, we see notation like: **`R(X)`**, **`R(Y)`**, **`W(X)`**, **`W(Y)`**, etc.

Notes:

- items with same name in different transactions refer to a shared item
- typically don't use explicit **`BEGIN`** or **`COMMIT`** or **`ABORT`**

<< ∧ >>

❖ Transaction Schedules (cont)

Showing SQL→Schedule, using bank transfer example

```
get balance in source account
get balance in destination account
if (source balance sufficient):
   update source by subtracting amount transferred
   update destination by adding amount transferred
```

If X = source account, Y = destination account, can be summarized as

```
R(X)  R(Y)  W(X)  W(Y)
```

Note: we treat the **`update`**s simply as writes ...

- assume **`UPDATE`** = **`R;W`**, and **`R;W`** is atomic, so overall effect is just **`W`**

<< ∧ >>

❖ Transaction Schedules (cont)

When multiple transactions run in parallel

- each transaction runs its own operations in a well-defined order
- but operations from different transactions interleave differently

Possible execution orders for operations of two transactions

```
-- no concurrency
T1: R(X) W(X) R(Y) W(Y)
T2:                     R(X) W(X) R(Y) W(Y)

-- with concurrent execution
T1: R(X)      W(X)      R(Y)      W(Y)
T2:      R(X)      W(X)      R(Y)      W(Y)
```

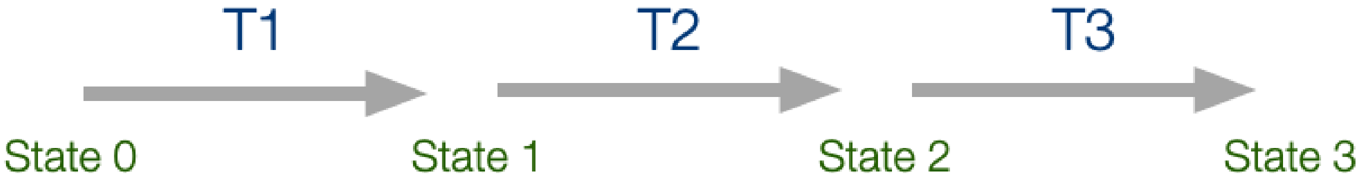
<< ^ >>

❖ Transaction Schedules (cont)

Executing a single correct transaction ...

- maps the DB from a consistent state to another consistent state

Similarly, executing transactions sequentially ...



Abribtrary interleaving of operations can cause anomalies, so that ...

- two consistency-preserving transactions, running concurrently
- produce a final state which is not consistent

<< ∧ >>

❖ Serial Schedules

Serial execution: **T1** then **T2** or **T2** then **T1**

```
T1: R(X) W(X) R(Y) W(Y)
T2:                     R(X) W(X)
```

or

```
T1:           R(X) W(X) R(Y) W(Y)
T2: R(X) W(X)
```

Serial execution guarantees a consistent final state if

- the initial state of the database is consistent
- **T1** and **T2** are consistency-preserving

<< ∧ >>

❖ Concurrent Schedules

Concurrent schedules interleave **T1**,**T2**,... operations

Some concurrent schedules are ok, e.g.

```
T1: R(X) W(X)      R(Y)      W(Y)
T2:           R(X)      W(X)
```

Other concurrent schedules cause anomalies, e.g.

```
T1: R(X)      W(X)      R(Y) W(Y)
T2:      R(X)      W(X)
```

Want the system to ensure that only valid schedules occur.

<< ∧

❖ Example Update Anomaly

Two concurrent transfers from same source account:

- T1 transfers $200 X→Y, T2 transfers $100 X→Y
- inital values: X=500, Y=100; final values: X=200, Y=400

T1	T2	X_{T1}	X_{T2}	X_{db}	Y_{T1}	Y_{T2}	Y_{db}
R(X)		500		500			100
X-200		300					
	R(X)		500				
W(X)		300		300			
	X-100		400				
	W(X)		400	400			
	R(Y)					100	
R(Y)					100		
Y+200					300		
W(Y)					300		300
	Y+100					200	
	W(Y)					200	200

Produced: 16 Nov 2020