Transactions

Exam Grades

Stats (Out of 90+2)

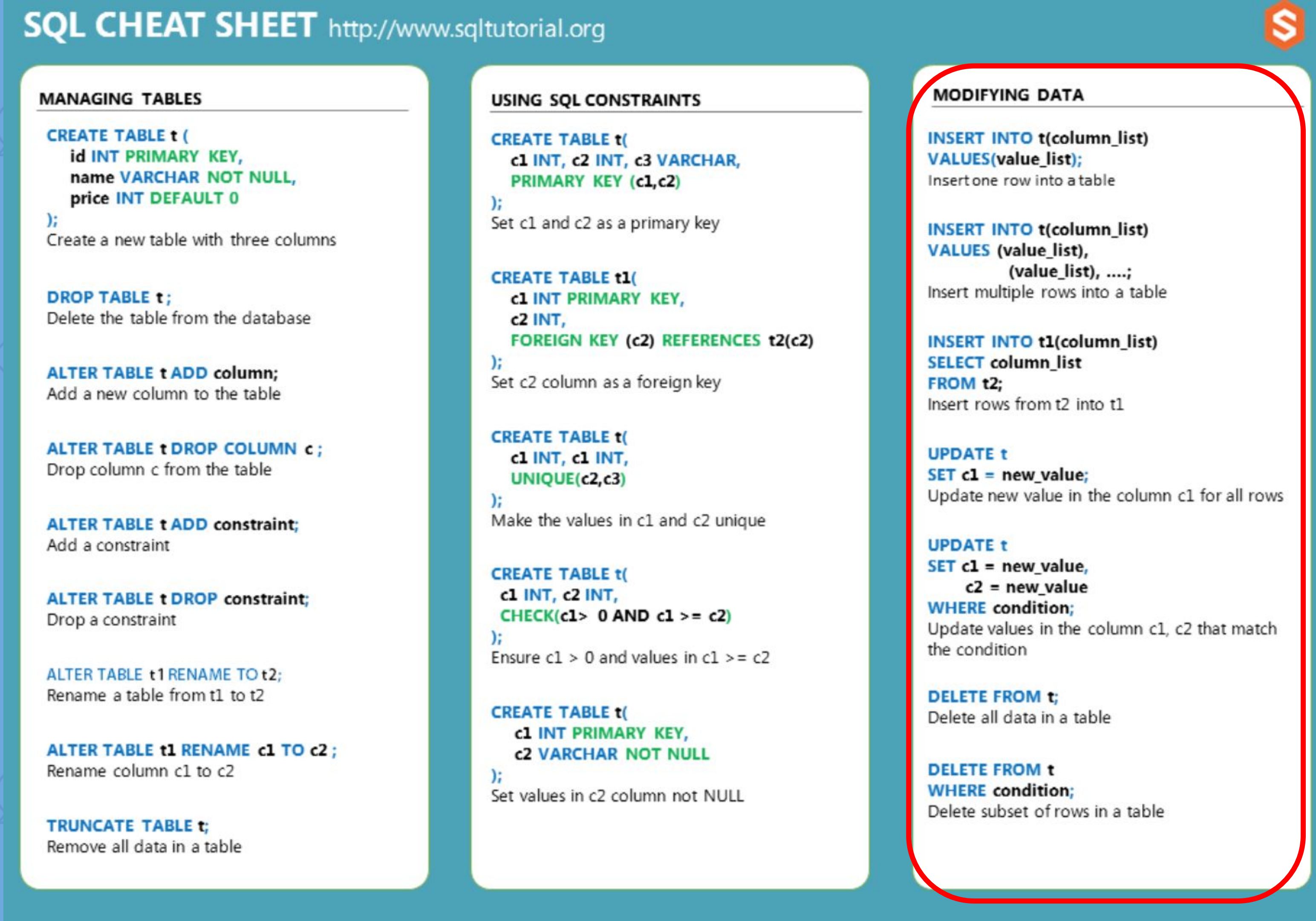
- Median: 76.5, Mean: 73.9, High: 92, Standard Deviation: 10.5
- Regrade requests open for **1 week** (next Tuesday)

Feedback Themes

- TAs were ultra-responsive during the “24 hours”
- Test was too long. We’ll recalibrate for Finals
- Most liked 24 hour flex time

PAST
FUTURE

SQL Writes



SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING TABLES

```
CREATE TABLE t (
    id INT PRIMARY KEY,
    name VARCHAR NOT NULL,
    price INT DEFAULT 0
);
```

Create a new table with three columns

```
DROP TABLE t ;
```

Delete the table from the database

```
ALTER TABLE t ADD column;
```

Add a new column to the table

```
ALTER TABLE t DROP COLUMN c ;
```

Drop column c from the table

```
ALTER TABLE t ADD constraint;
```

Add a constraint

```
ALTER TABLE t DROP constraint;
```

Drop a constraint

```
ALTER TABLE t1 RENAME TO t2;
```

Rename a table from t1 to t2

```
ALTER TABLE t1 RENAME c1 TO c2 ;
```

Rename column c1 to c2

```
TRUNCATE TABLE t;
```

Remove all data in a table

USING SQL CONSTRAINTS

```
CREATE TABLE t(
   c1 INT, c2 INT, c3 VARCHAR,
   PRIMARY KEY (c1,c2)
);
```

Set c1 and c2 as a primary key

```
CREATE TABLE t1(
   c1 INT PRIMARY KEY,
   c2 INT,
   FOREIGN KEY (c2) REFERENCES t2(c2)
);
```

Set c2 column as a foreign key

```
CREATE TABLE t(
   c1 INT, c1 INT,
   UNIQUE(c2,c3)
);
```

Make the values in c1 and c2 unique

```
CREATE TABLE t(
  c1 INT, c2 INT,
  CHECK(c1> 0 AND c1 >= c2)
);
```

Ensure c1 > 0 and values in c1 >= c2

```
CREATE TABLE t(
    c1 INT PRIMARY KEY,
    c2 VARCHAR NOT NULL
);
```

Set values in c2 column not NULL

MODIFYING DATA

```
INSERT INTO t(column_list)
VALUES(value_list);
```

Insert one row into a table

```
INSERT INTO t(column_list)
VALUES (value_list),
            (value_list), ....;
```

Insert multiple rows into a table

```
INSERT INTO t1(column_list)
SELECT column_list
FROM t2;
```

Insert rows from t2 into t1

```
UPDATE t
SET c1 = new_value;
```

Update new value in the column c1 for all rows

```
UPDATE t
SET c1 = new_value,
     c2 = new_value
WHERE condition;
```

Update values in the column c1, c2 that match the condition

```
DELETE FROM t;
```

Delete all data in a table

```
DELETE FROM t
WHERE condition;
```

Delete subset of rows in a table

SQL Writes

```
UPDATE Product
SET Price = Price – 1.99
WHERE pname = ‘Gizmo’
```

```
INSERT INTO SmallProduct(name, price)
        SELECT pname, price
        FROM Product
        WHERE price <= 0.99
```

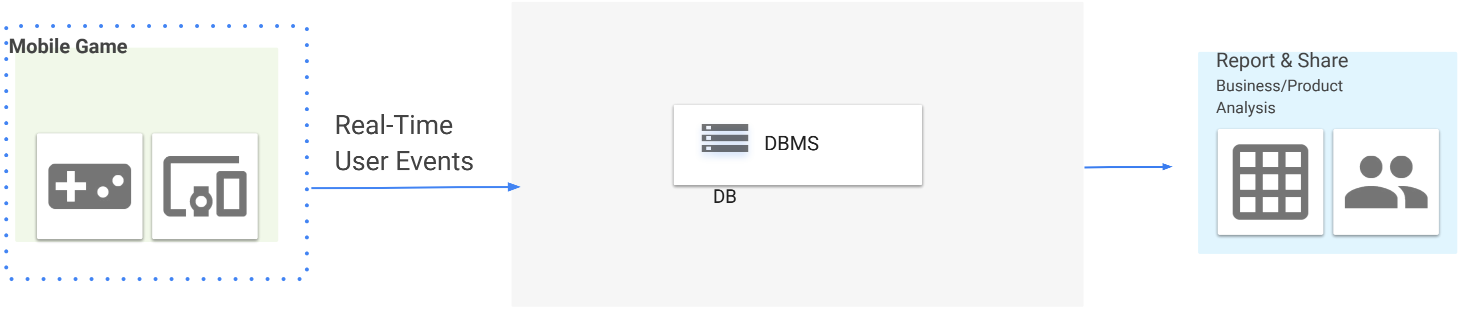
```
DELETE Product
        WHERE price <=0.99
```

How?

Example Game App

DB v0

(Recap lectures)



Q1: 1000 users/sec?
Q2: Offline?
Q3: Support v1, v1' versions?

Q7: How to model/evolve game data?
Q8: How to scale to millions of users?
Q9: When machines die, restore game state gracefully?

Q4: Which user cohorts?
Q5: Next features to build?
Experiments to run?
Q6: Predict ads demand?

App designer

Systems designer

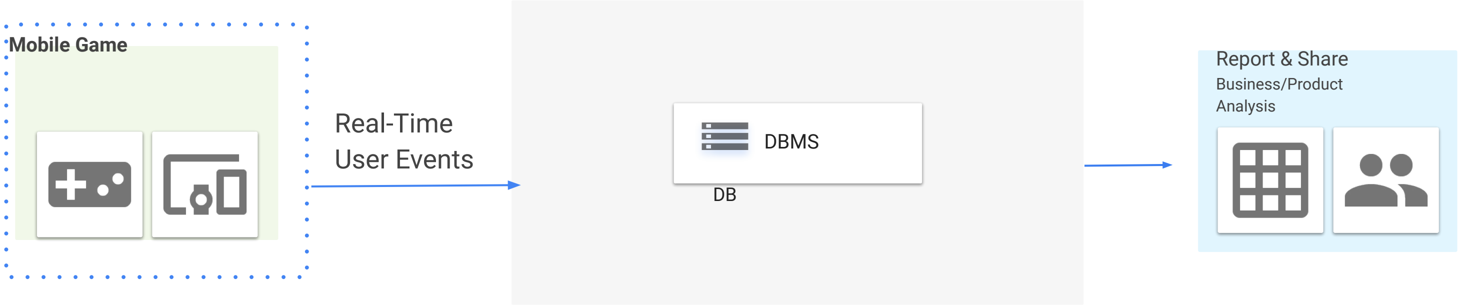
Product/Biz designer

How?

Example Game App

DB v0

(Recap lectures)



Q1: 1000 users/sec?
Q2: Offline?
Q3: Support v1, v1’ versions?

App designer

Q7: How to model/evolve game data?
Q8: How to scale to millions of users?
Q9: When machines crash, restore game state gracefully?

Systems designer

Q4: Which user cohorts?
Q5: Next features to build?
Experiments to run?
Q6: Predict ads demand?

Product/Biz designer

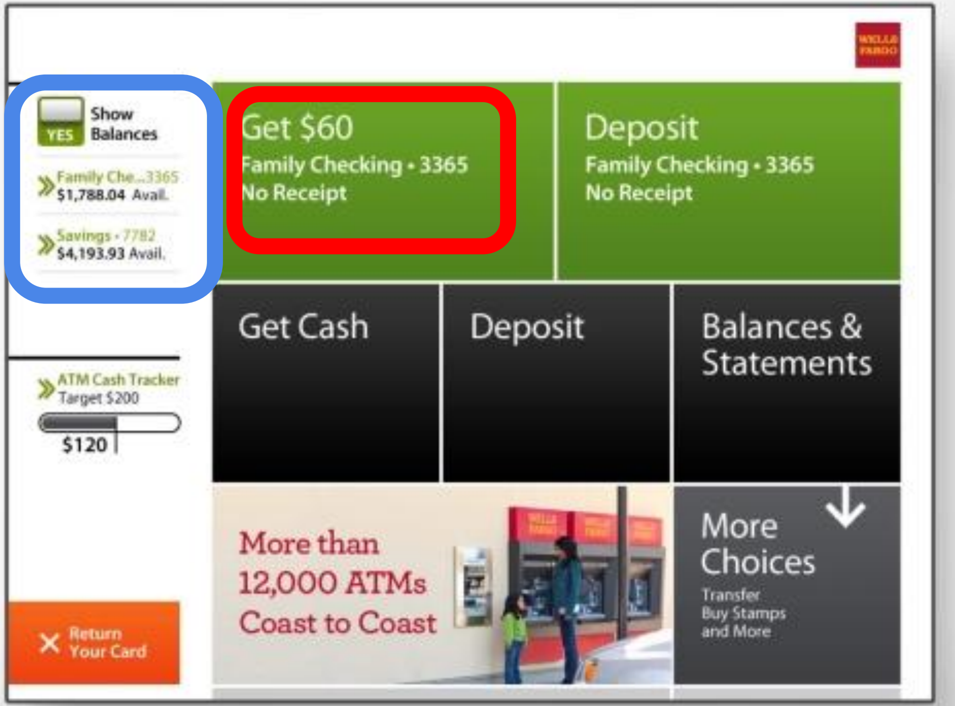
Today's Lecture

1. Why Transactions?
2. Transactions
3. Properties of Transactions: ACID
4. Logging

Example

Unpack ATM DB:

Transaction



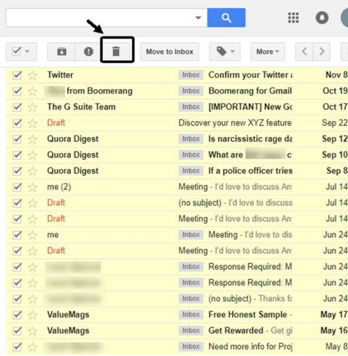
Read Balance
Give money
Update Balance

vs

Read Balance
Update Balance
Give money

Visa does > 60,000 TXNs/sec with users & merchants

Want your 4$ Starbucks transaction to wait for a stranger’s 10k$ bet in Las Vegas ?
⇒ Transactions can (1) be quick or take a long time, (2) unrelated to you



☑	☆	Twitter	Inbox	Confirm your Twitter	Nov 8
☑	☆	[redacted] from Boomerang	Inbox	Boomerang for Gmail	Oct 19
☑	☆	The G Suite Team	Inbox	[IMPORTANT] New G	Oct 17
☑	☆	Draft		Discover your new XYZ feature	Sep 22
☑	☆	Quora Digest	Inbox	Is narcissistic rage d	Sep 12
☑	☆	Quora Digest	Inbox	What are [redacted] c	Sep 10
☑	☆	Quora Digest	Inbox	If a police officer tries	Sep 8
☑	☆	me (2)		Meeting - I'd love to discuss An	Jul 14
☑	☆	Draft		(no subject) - I'd love to discuss	Jul 14
☑	☆	Draft		Meeting - I'd love to discuss An	Jul 14
☑	☆	me	Inbox	Meeting - I'd love to dis	Jun 24
☑	☆	Draft		Meeting - I'd love to discuss An	Jun 24
☑	☆	[redacted]	Inbox	Response Required: M	Jun 24
☑	☆	[redacted]	Inbox	Response Required: M	Jun 24
☑	☆	[redacted]	Inbox	(no subject) - Thanks f	Jun 24
☑	☆	ValueMags	Inbox	Free Honest Sample -	May 17
☑	☆	ValueMags	Inbox	Get Rewarded - Get gi	May 16
☑	☆	[redacted]	Inbox	Need more info for Proj	May 8

Transactions are at the core of

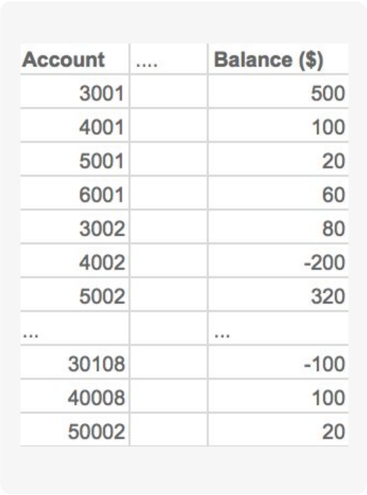
-- payment, stock market, banks, ticketing

-- Gmail, Google Docs (e.g., multiple people editing)

Example

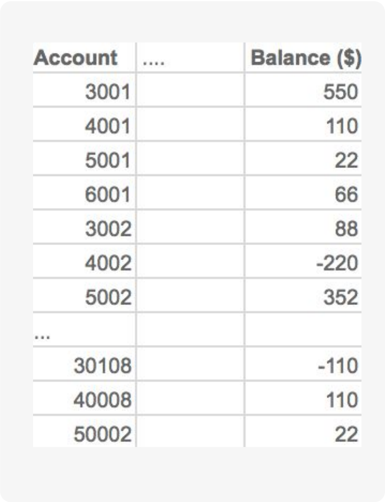
Monthly bank interest transaction

Money



Account	....	Balance ($)
3001		500
4001		100
5001		20
6001		60
3002		80
4002		-200
5002		320
...		...
30108		-100
40008		100
50002		20

Money (@4:29 am day+1)



Account	....	Balance ($)
3001		550
4001		110
5001		22
6001		66
3002		88
4002		-220
5002		352
...		
30108		-110
40008		110
50002		22

‘T-Monthly-423’

Monthly Interest 10%

4:28 am Starts run on 100M bank accounts

Takes 24 hours to run

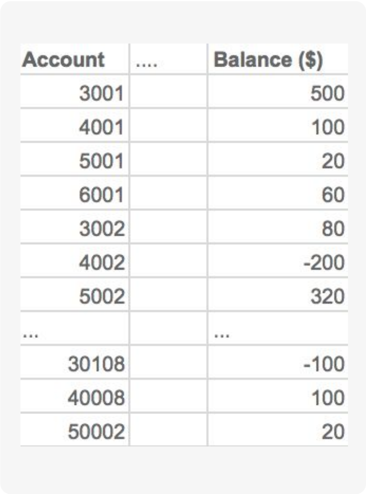
```
UPDATE Money
SET Balance = Balance * 1.1
```

Example

Monthly bank interest transaction

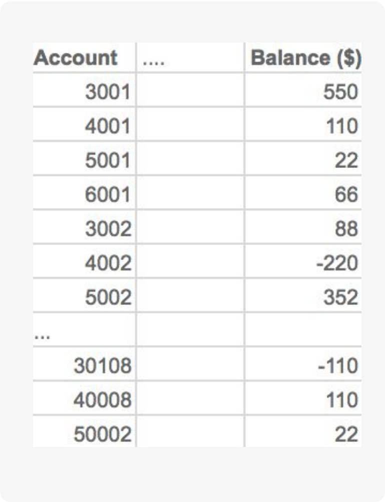
Performance

Money



Account	....	Balance ($)
3001		500
4001		100
5001		20
6001		60
3002		80
4002		-200
5002		320
...		...
30108		-100
40008		100
50002		20

Money (@4:29 am day+1)

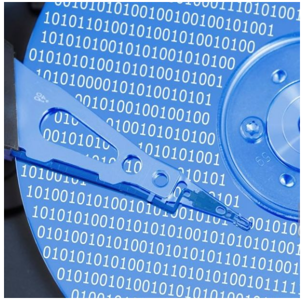


Account	....	Balance ($)
3001		550
4001		110
5001		22
6001		66
3002		88
4002		-220
5002		352
...		
30108		-110
40008		110
50002		22

Cost to update all data

100M bank accounts → 100M seeks? (worst case)

(@10 msec/seek, that's 1 million secs)



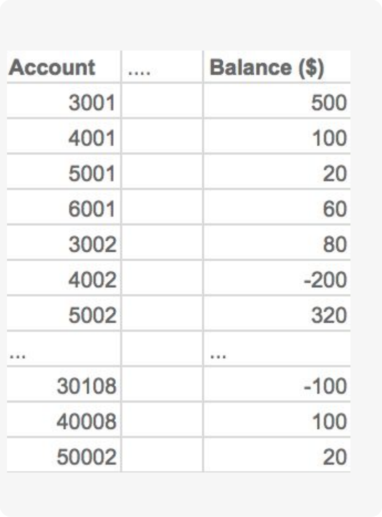
Problem1: SLOW :(

Example

Monthly bank interest transaction

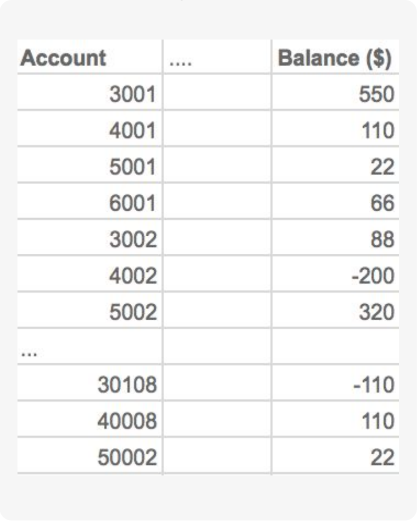
With crash

Money



Account	....	Balance ($)
3001		500
4001		100
5001		20
6001		60
3002		80
4002		-200
5002		320
...		...
30108		-100
40008		100
50002		20

Money (@10:45 am)



Account	....	Balance ($)
3001		550
4001		110
5001		22
6001		66
3002		88
4002		-200
5002		320
...		
30108		-110
40008		110
50002		22

??

??

??

??

Did T-Monthly-423 complete?
Which tuples are bad?

Case1: T-Monthly-423 crashed
Case2: T-Monthly-423 completed
4002 deposited 20$ at 10:45 am

'T-Monthly-423'

Monthly Interest 10%
4:28 am Starts run on 100M bank accounts
Takes 24 hours to run
Network outage at 10:29 am,
System access at 10:45 am

Problem 2: Wrong :(

15

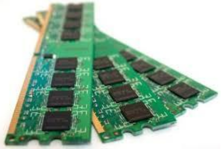
Big Scale

Roadmap

Primary data structures/algorithms

LOGS

LOCKS



?????

Today's Lecture

1. Why Transactions?

2. Properties of Transactions: ACID

3. Logging

Transactions: Basic Definition

A transaction (“TXN”) is a sequence of one or more *operations* (reads or writes) which reflects *a single real-world transition*.

In the real world, a TXN either happened completely or not at all (e.g., you withdrew 100$ from bank. Or not.)

```
START TRANSACTION
        UPDATE Product
        SET Price = Price – 1.99
        WHERE pname = ‘Gizmo’
COMMIT
```

Transactions in SQL

- In “ad-hoc” SQL, each statement = one transaction

- In a program, multiple statements can be grouped together as a transaction

```
START TRANSACTION
        UPDATE Bank SET amount = amount – 100
        WHERE name = ‘Bob’
        UPDATE Bank SET amount = amount + 100
        WHERE name = ‘Joe’
COMMIT
```

Motivation for Transactions

Group user actions (reads & writes) into *Transactions* helps with two goals:

1. Recovery & Durability: Keep the data consistent and durable. *Despite system crashes, user canceling TXN part way, etc.*

This lecture!

Idea: Use **LOGS**. Support to “commit” or “rollback” TXNs

2. Concurrency: Get better performance by parallelizing TXNs *without* creating ‘bad data.’ *Despite slow disk writes and reads*.

Next lecture

Idea: Use **LOCKS.** Run several user TXNs concurrently.

Example 1: Protection against crashes / aborts

Scenario: Make a CheapProducts table, from a Products table

```
Client 1:
        INSERT INTO CheapProduct(name, price)
        SELECT pname, price
        FROM Product
        WHERE price <= 0.99

        DELETE Product
        WHERE price <=0.99
```

Crash / abort!

What goes wrong?

```
Client 1:
        START TRANSACTION
        INSERT INTO CheapProduct(name, price)
                SELECT pname, price
                FROM Product
                WHERE price <= 0.99

        DELETE Product
                WHERE price <=0.99
        COMMIT
```

Now we’d be fine! We’ll see how / why this lecture

Example 2: Multiple users: single statements

Client 1: [at 10:01 am]

```
UPDATE Product
SET Price = Price – 1.99
WHERE pname = ‘Gizmo’
```

Client 2: [at 10:01 am]

```
UPDATE Product
SET Price = Price*0.5
WHERE pname=‘Gizmo’
```

Two managers attempt to discount products ***at same time -***

What could go wrong?

```
Client 1: START TRANSACTION
                    UPDATE Product
                    SET Price = Price – 1.99
                    WHERE pname = ‘Gizmo’
             COMMIT
```

```
Client 2: START TRANSACTION
                    UPDATE Product
                    SET Price = Price*0.5
                    WHERE pname=‘Gizmo’
          COMMIT
```

Now works like a charm- we’ll see how / why next lecture…

3. Properties of Transactions

What you will learn about in this section

1. **A**tomicity
2. **C**onsistency
3. **I**solation
4. **D**urability

ACID: Atomicity

- TXN is all or nothing
  - C*ommits*: all the changes are made
  - A*borts*: no changes are made

ACID: Consistency

- The tables must always satisfy user-specified *integrity constraints*
    - E.g., Account number is unique, Sum of *debits* and of *credits* is 0

- How consistency is achieved:
    - Programmer writes a TXN to go from one consistent state to a consistent state
    - *System* makes sure that the TXN is atomic (e.g., if EXCEPTION, rolls back)

ACID: Isolation

- A TXN executes **concurrently** with other TXNs

- Effect of TXNs is the same as TXNs running one after another

Conceptually,

- similar to OS "sandboxes"
- E.g. TXNs can't observe each other's "partial updates"

ACID: Durability

- The effect of a TXN must **persist** after the TXN
  - And after the whole program has terminated
  - And even if there are power failures, crashes, etc.
- ⇒ Write data to durable IO (e.g., disk)

ACID Summary

- Atomic
  - State shows either all the effects of TXN, or none of them
- Consistent
  - TXN moves from a state where integrity holds, to another where integrity holds
- Isolated
  - Effect of TXNs is the same as TXNs running one after another
- Durable
  - Once a TXN has committed, its effects remain in the database

A Note: ACID is one popular option!

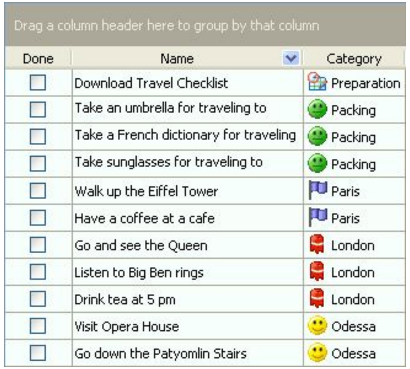
- Many debates over ACID, both **historically** and **currently**
- Some “NoSQL” DBMSs relax ACID
- In turn, now “NewSQL” reintroduces ACID compliance to NoSQL-style DBMSs…

⇒ Usually, depends on what consistency and performance your application needs

ACID is an extremely important & successful paradigm, but still debated!

4. Atomicity & Durability via Logging

Conceptual Idea:
Trip to Europe



Drag a column header here to group by that column

Done	Name	Category
☐	Download Travel Checklist	Preparation
☐	Take an umbrella for traveling to	Packing
☐	Take a French dictionary for traveling	Packing
☐	Take sunglasses for traveling to	Packing
☐	Walk up the Eiffel Tower	Paris
☐	Have a coffee at a cafe	Paris
☐	Go and see the Queen	London
☐	Listen to Big Ben rings	London
☐	Drink tea at 5 pm	London
☐	Visit Opera House	Odessa
☐	Go down the Patyomlin Stairs	Odessa

1. Make TODO list. Buy tickets

2. Actual Visit

(Much longer than buying tickets)

Big Idea

LOGS!

(aka TODO/ ledger)

Recall (on disks)

- Sequential reads FASTER than random reads
- Sequential writes (aka “appends”) FASTER than random writes

Big Idea: LOGs (or log files or ledger)

- Any value that changes? Append to LOG!
  - LOG is a compact “todo” list of data updates
- Intuition:
  - Data pages: (a) Update in RAM (fast) (b) Update on disk later (slow)
  - LOGs: (c) Append “todo” in LOGs and (d) control when you Flush LOGs to disk

Many kinds of LOGs. We’ll study a few key ones!

What you will learn about in this section

1. How to make/use LOGs?

2. How to make it fast? (Mess with memory and disk)

Basic Idea: (Physical) Logging

Idea:

- Log consists of an ordered list of Update Records
- Log record contains UNDO information for every update!

<TransactionID, &reference, old value, new value>

(e.g., key)

What DB does?

- Owns the log “service” for all applications/transactions.
- Appends to log. **Flush** when necessary — force writes to disk

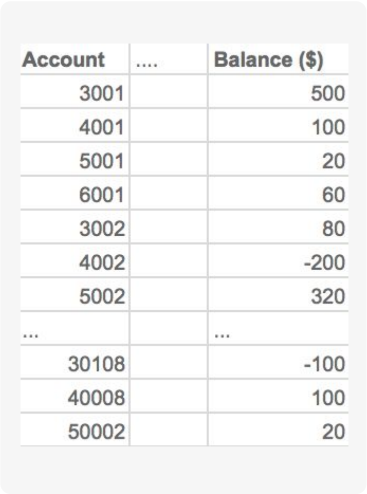
This is sufficient to UNDO any transaction!

Example

Monthly bank interest transaction

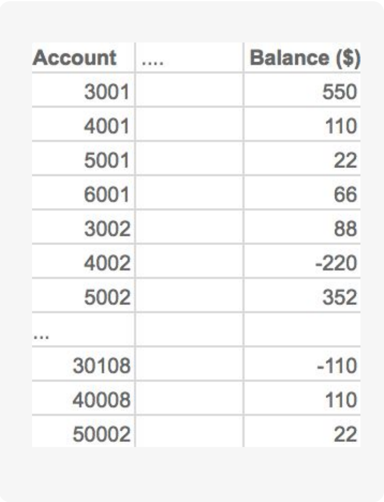
Full run

Money



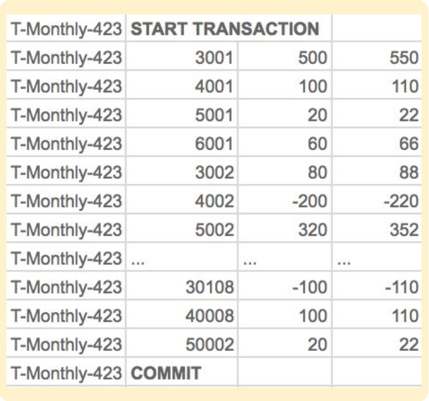
Account	....	Balance ($)
3001		500
4001		100
5001		20
6001		60
3002		80
4002		-200
5002		320
...		...
30108		-100
40008		100
50002		20

Money (@4:29 am day+1)



Account	....	Balance ($)
3001		550
4001		110
5001		22
6001		66
3002		88
4002		-220
5002		352
...		
30108		-110
40008		110
50002		22

WA Log (@4:29 am day+1)



T-Monthly-423	**START TRANSACTION**		
T-Monthly-423	3001	500	550
T-Monthly-423	4001	100	110
T-Monthly-423	5001	20	22
T-Monthly-423	6001	60	66
T-Monthly-423	3002	80	88
T-Monthly-423	4002	-200	-220
T-Monthly-423	5002	320	352
T-Monthly-423	...	...	...
T-Monthly-423	30108	-100	-110
T-Monthly-423	40008	100	110
T-Monthly-423	50002	20	22
T-Monthly-423	**COMMIT**		

Update Records

Commit Record

'T-Monthly-423'

Monthly Interest 10%
4:28 am Starts run on 100M bank accounts
Takes 24 hours to run

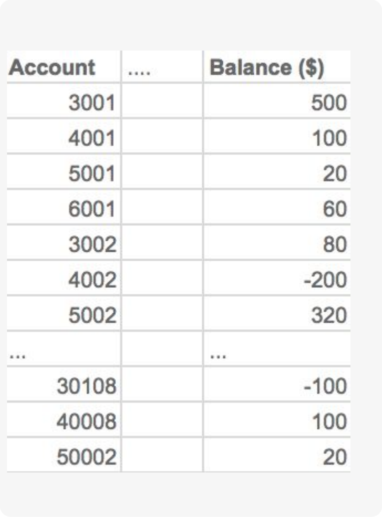
```
START TRANSACTION
        UPDATE Money
        SET Amt = Amt * 1.10
COMMIT
```

Example

Monthly bank interest transaction

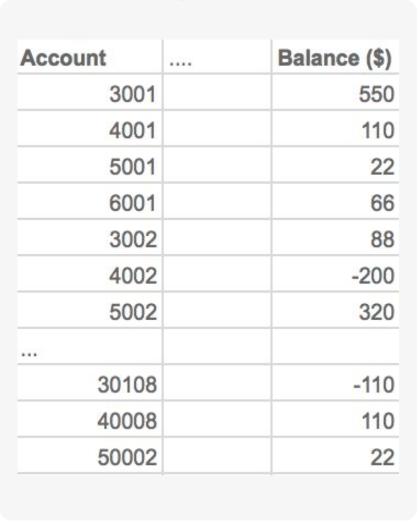
With crash

Money



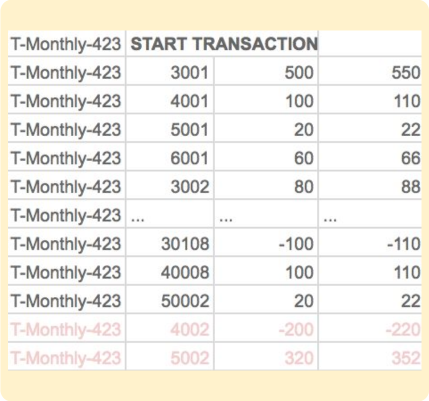
Account	....	Balance ($)
3001		500
4001		100
5001		20
6001		60
3002		80
4002		-200
5002		320
...		...
30108		-100
40008		100
50002		20

Money (@10:45 am)



Account	....	Balance ($)	
3001		550	??
4001		110	
5001		22	
6001		66	
3002		88	
4002		-200	??
5002		320	??
...			
30108		-110	
40008		110	
50002		22	??

WA Log (@10:29 am)



T-Monthly-423	START TRANSACTION		
T-Monthly-423	3001	500	550
T-Monthly-423	4001	100	110
T-Monthly-423	5001	20	22
T-Monthly-423	6001	60	66
T-Monthly-423	3002	80	88
T-Monthly-423	...	...	...
T-Monthly-423	30108	-100	-110
T-Monthly-423	40008	100	110
T-Monthly-423	50002	20	22
T-Monthly-423	4002	-200	-220
T-Monthly-423	5002	320	352

TXN 'T-Monthly-423'

Monthly Interest 10%
4:28 am Starts run on 100M bank accounts
Takes 24 hours to run
Network outage at 10:29 am,
System access at 10:45 am

Did T-Monthly-423 complete?
Which tuples are bad?

Case1: T-Monthly-423 was crashed
Case2: T-Monthly-423 completed. 4002 deposited 20$ at 10:45 am

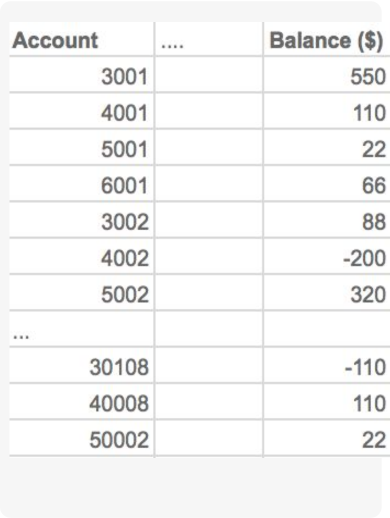
Can you infer from RED log records?

Example

Monthly bank interest transaction

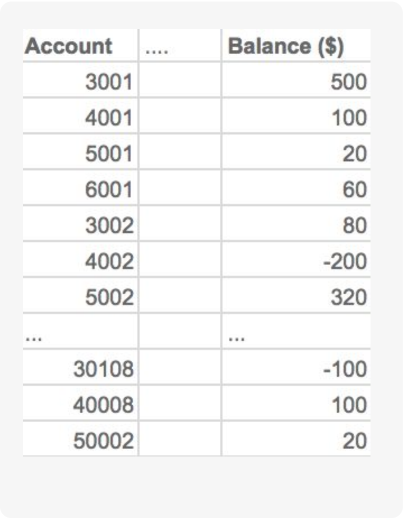
Recovery

Money (@10:45 am)



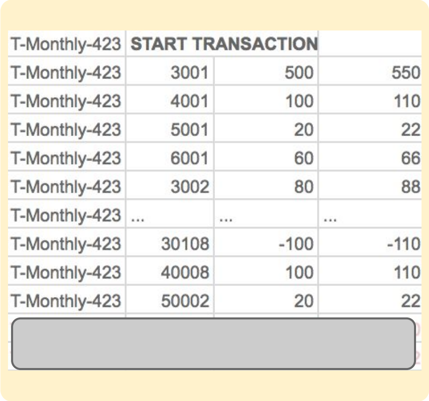
Account	....	Balance ($)
3001		550
4001		110
5001		22
6001		66
3002		88
4002		-200
5002		320
...		
30108		-110
40008		110
50002		22

Money (after recovery)



Account	....	Balance ($)
3001		500
4001		100
5001		20
6001		60
3002		80
4002		-200
5002		320
...		...
30108		-100
40008		100
50002		20

WA Log (@10:29 am)



T-Monthly-423	**START TRANSACTION**		
T-Monthly-423	3001	500	550
T-Monthly-423	4001	100	110
T-Monthly-423	5001	20	22
T-Monthly-423	6001	60	66
T-Monthly-423	3002	80	88
T-Monthly-423	...	...	...
T-Monthly-423	30108	-100	-110
T-Monthly-423	40008	100	110
T-Monthly-423	50002	20	22

System recovery (after 10:45 am)

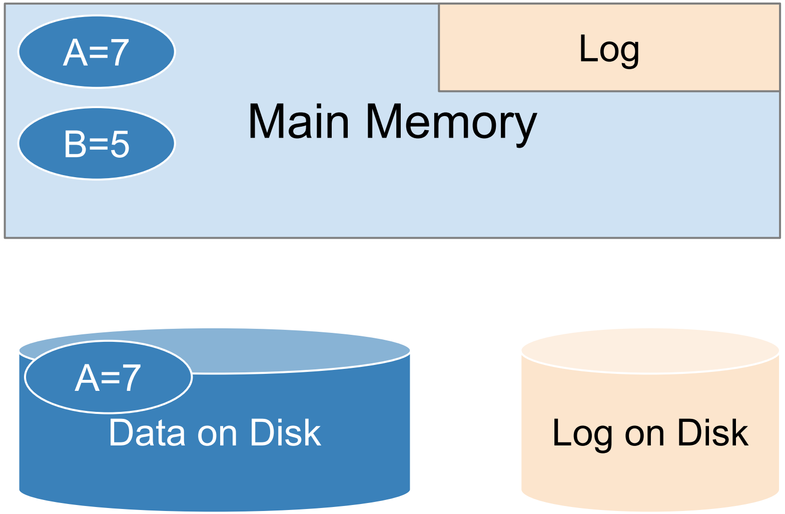
1. Rollback uncommitted transactions
   - Restore old values from WAL Log (if any)
   - Notify developers about aborted TXN
2. Redo Recent transactions (w/ new values)
3. Back in business; Redo (any pending) transactions

What you will learn about in this section

1. How to make/use LOGs?

2. ⇒ How to make it fast? (Mess with memory and disk)

A picture of logging



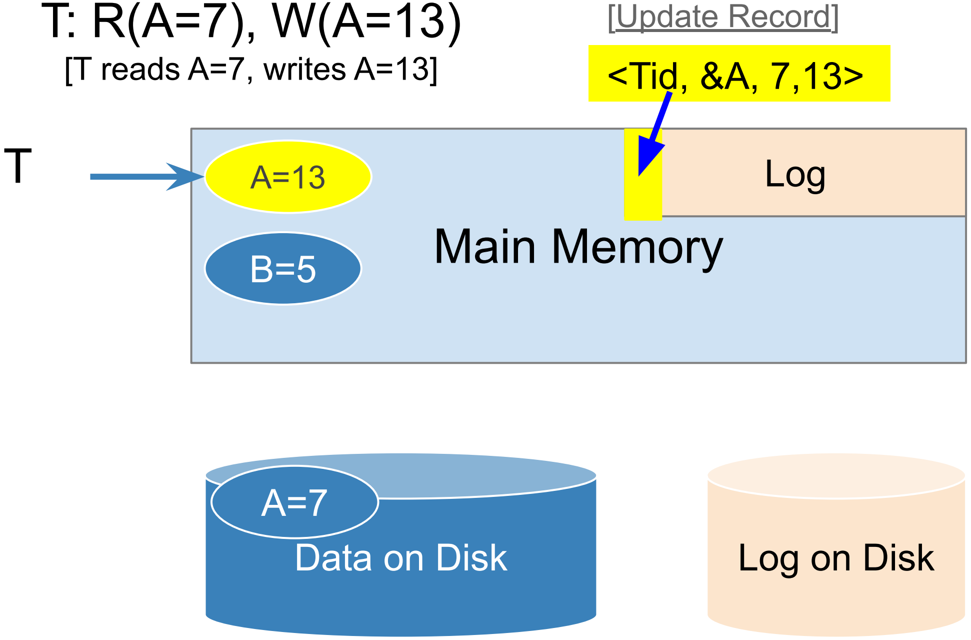
Log is a file (like any data table)
1. Pages updated in RAM
2. Flushed as DB blocks on disk (sequential I/O)

“**<u>Flushing</u>** to disk” = writing to disk from main memory

A picture of logging

T: R(A=7), W(A=13)

[T reads A=7, writes A=13]



Operation recorded in update log in main memory!

Why do we need logging for atomicity?

- Could we just write TXN updates to disk **only** once whole TXN complete?
  - Then, if abort / crash and TXN not complete, it has no effect- atomicity!
  - *With unlimited memory and time, this could work…*

- ⇒ We **need to log partial results of TXNs** because of:
  - Memory constraints (e.g. , billions of updates)
  - Time constraints (what if one TXN takes very long?)

We need to write partial results to disk!
…And so we need a **LOG** to (maybe) ***undo*** these partial results!

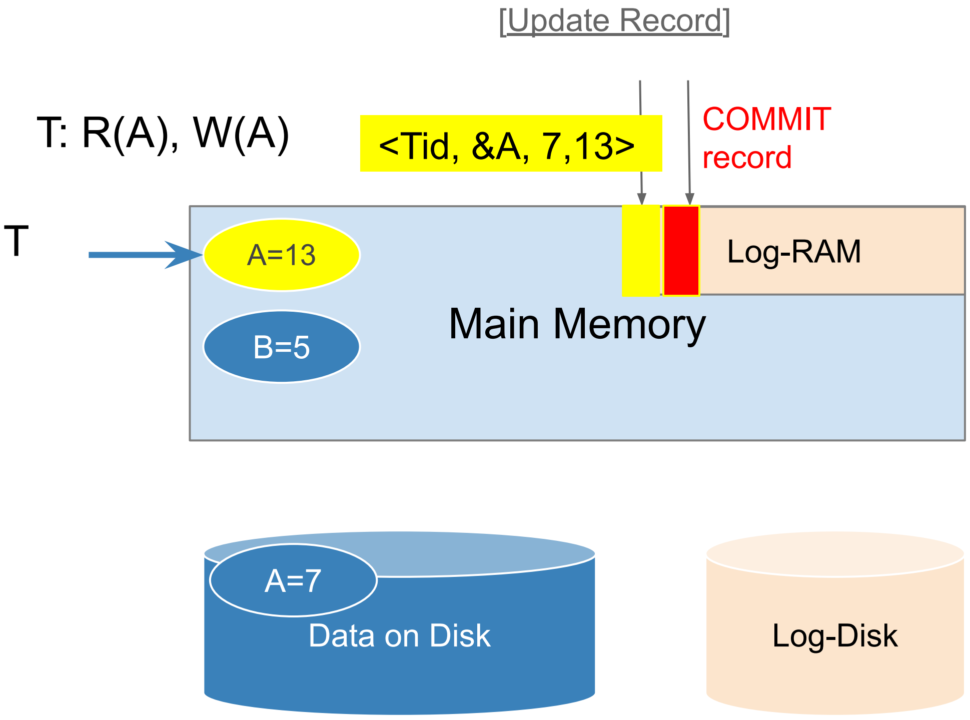
What is the correct way to LOG to disk?

- We’ll look at the *Write-Ahead Logging (WAL)* protocol
- We’ll see why it works by looking at other protocols which are incorrect!

Remember: Key idea is to ensure durability *while* maintaining our ability to “undo”!

Write-Ahead Logging (WAL) TXN Commit Protocol

Write-ahead Logging (WAL) Commit Protocol

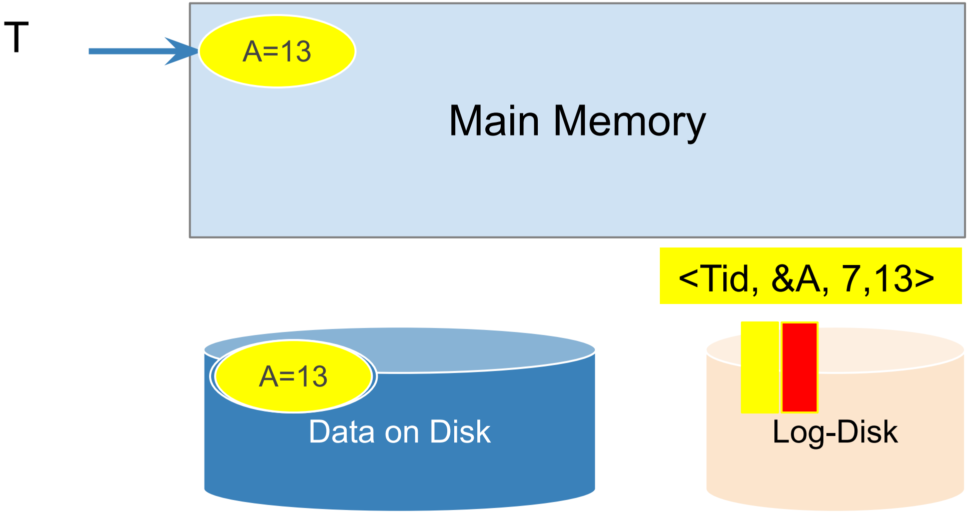


Commit ***after*** **we've written log to disk but *before* we've written data to disk**…

OK, Commit!

Write-ahead Logging (WAL) Commit Protocol

T: R(A), W(A)



Commit ***after*** **we've written log to disk but *before* we've written data to disk**… this is WAL!

OK, Commit!

If we crash now, is T durable?

USE THE LOG!

Write-Ahead Logging (WAL)

Algorithm: WAL

For each tuple update, write Update Record into LOG-RAM

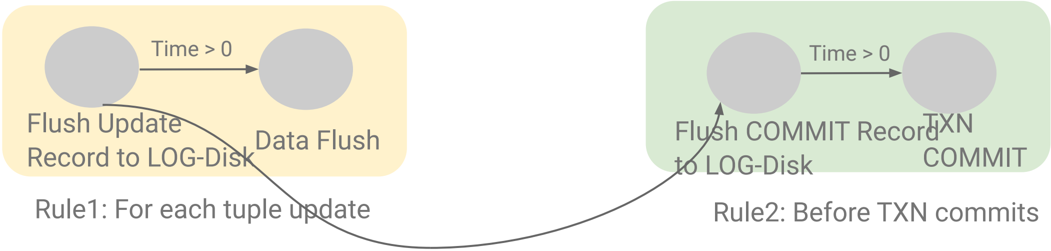
Follow two Flush rules for LOG

- Rule1: Flush Update Record *into LOG-Disk before* corresponding data page goes to storage
- Rule2: Before TXN commits,
  - Flush all Update Records to LOG-Disk
  - Flush COMMIT Record to LOG-Disk

→ **Durability**

→ **Atomicity**

Transaction is committed *once COMMIT record is on stable storage*

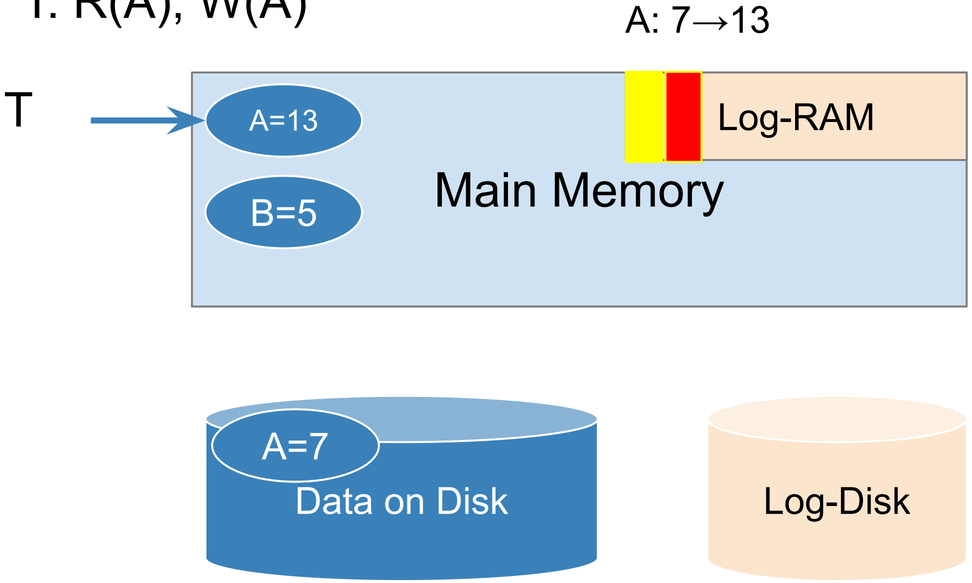

Time > 0
Flush Update Record to LOG-Disk
Data Flush
Time > 0
Flush COMMIT Record to LOG-Disk
TXN COMMIT

Rule1: For each tuple update

Rule2: Before TXN commits

Incorrect Commit Protocol #1

T: R(A), W(A)



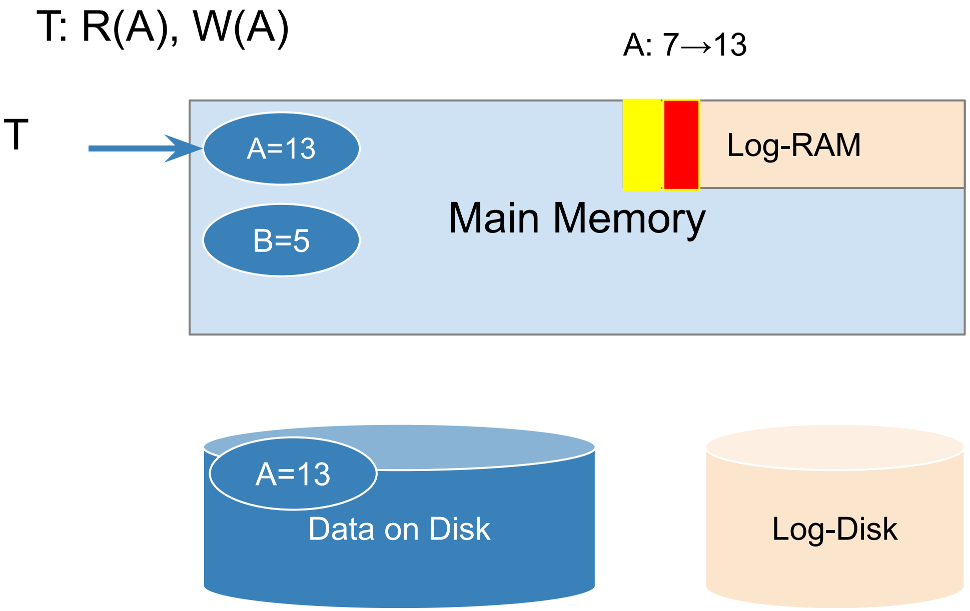
Let's try committing *before* we've written either data or LOG to disk…

OK, Commit!

If we crash now, is T durable?

Lost T's update!

Incorrect Commit Protocol #2



Let's try committing *after* we've written data but *before* we've written LOG to disk…

OK, Commit!

If we crash now, is T durable? Yes! Except…

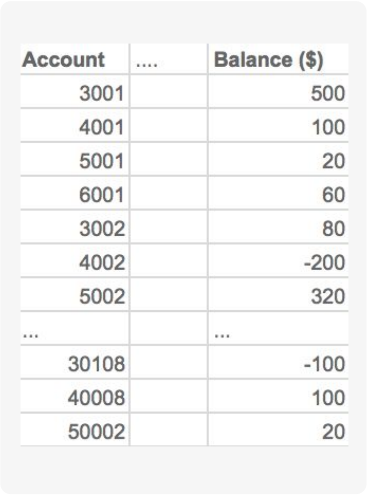
How do we know whether T was committed??

Example

Monthly bank interest transaction

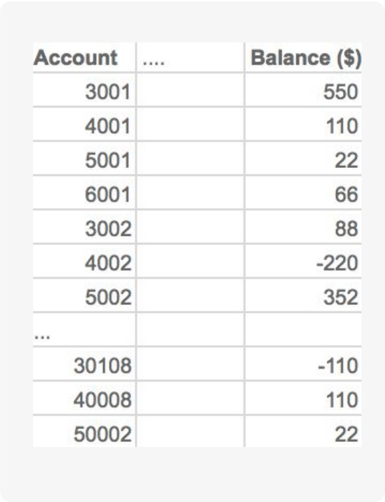
Performance

Money



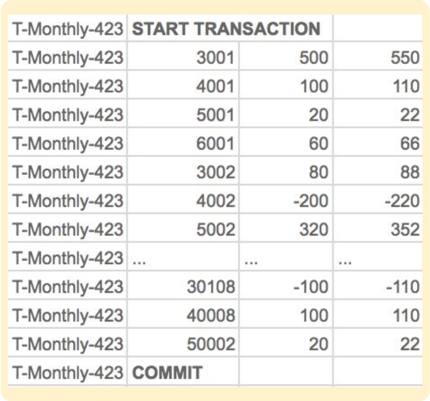
Account	....	Balance ($)
3001		500
4001		100
5001		20
6001		60
3002		80
4002		-200
5002		320
...		...
30108		-100
40008		100
50002		20

Money (@4:29 am day+1)



Account	....	Balance ($)
3001		550
4001		110
5001		22
6001		66
3002		88
4002		-220
5002		352
...		
30108		-110
40008		110
50002		22

WAL (@4:29 am day+1)

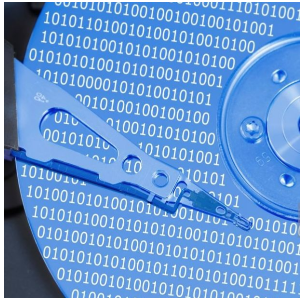


T-Monthly-423	**START TRANSACTION**		
T-Monthly-423	3001	500	550
T-Monthly-423	4001	100	110
T-Monthly-423	5001	20	22
T-Monthly-423	6001	60	66
T-Monthly-423	3002	80	88
T-Monthly-423	4002	-200	-220
T-Monthly-423	5002	320	352
T-Monthly-423	...	...	...
T-Monthly-423	30108	-100	-110
T-Monthly-423	40008	100	110
T-Monthly-423	50002	20	22
T-Monthly-423	**COMMIT**		

Cost to update all data

100M bank accounts → 100M seeks? (worst case)

(@10 msec/seek, that's 1 Million secs)



Cost to Append to log

+ 1 seek to get 'end of log'
+ write 100M log entries sequentially (fast!!! < 10 sec)

[Lazily update data on disk later, when convenient.]

Speedup for TXN Commit

1 Million secs vs 10 sec!!!

Logging Summary

- If DB says TX commits, TX effect remains after database crash
- DB can undo actions and help us with atomicity
- This is only half the story…