Finals Review

Finals

Last year's Test2 during COVID (90 mins)

- Similar in flavor
- Differences
  1. 180 mins vs 90 mins
  2. Review of pre-midterm material (~30-35%)
     a. SQL, Sorting/Hashing, Indexing
  3. Main emphasis (60-70%)
     a. "Amazon recommender style example"
     b. Transactions -- Logging, Locking
     c. Bigschemas - FDs, closures, superkeys

- Style of questions
  - Most are similar to psets, lectures
  - "1" fun question -- you'll need to "apply" 1-2 principles to answer

CS145

Goals

Course Summary

We'll learn How To…

- **Query** over small-med-large data sets with **SQL**? [Weeks 1 and 2]
  - On relational engines, and "big data" engines
- **Scale** for "big queries"? On Clusters? [Weeks 3, 4, 5]
  - OLAP/Analytics, 1st principles of scale
- **Scale** for "big writes"? [Weeks 6, 7, 8]
  - Writes, Transactions, Logging, ACID properties
- **Design** "good" databases? [Weeks 9, 10]
  - Big Schemas, design, functional dependencies, query optimizers

Project: Query-Visualize-Learn on GB/TB scale data sets on a Cloud [sql + python]

Industry Talks: Real world talks from Google, Uber, Coinbase

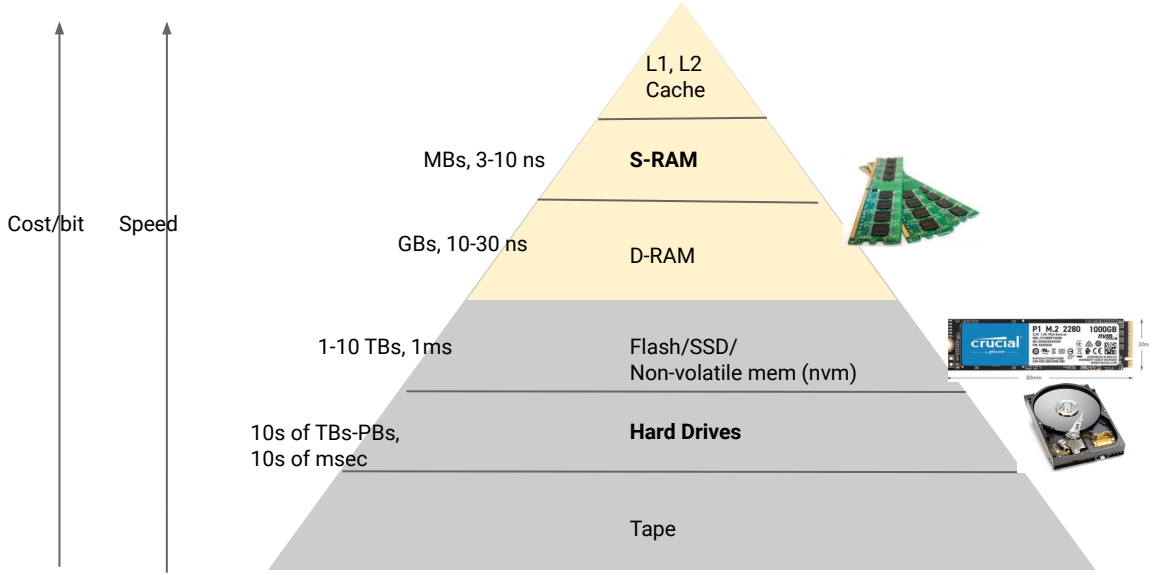
Key takeaway from class

Takeaway ⇒ How to apply CS concepts at scale to solve big problems?

- How do we scale queries by 10-100x, transactions by 10-100x
- How to work with real data sets? (bigschemas, projects, etc)

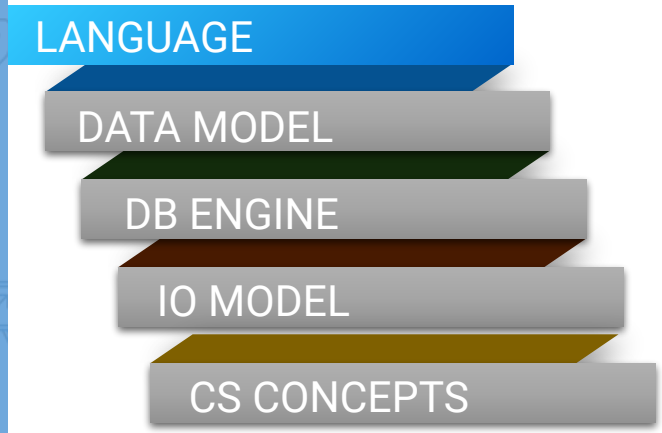
5

Takeaway [½] IO Hierarchy



⇒ Rest of cs145: Focus on simplified RAM + Disk model
(learn tools for other IO models)

Takeaway [2/2]



SQL

Relational

Non-Relational, same underlying concepts

Query Optimization,
Join algorithms, Histograms

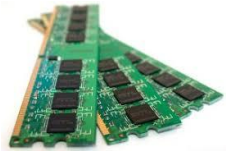
RAM, Disk, Cluster model

Hash, Sort
Indexing: B+ trees, Hash tables
Transactions: **Locking, Logging**, Conflicts, Interleaving
Normalize data

How to apply CS concepts at scale to solve big problems?
(How do we scale queries by 10-100x, transactions by 10-100x)

Big Scale Lego Blocks

Roadmap



Primary data structures/algorithms

Hashing	Sorting
HashTables (hash$_i$(key) --> location)	BucketSort, QuickSort MergeSort
HashFunctions (hash$_i$(key) --> location)	MergeSortedFiles MergeSort
HashFunctions (hash$_i$(key) --> location)	MergeSort

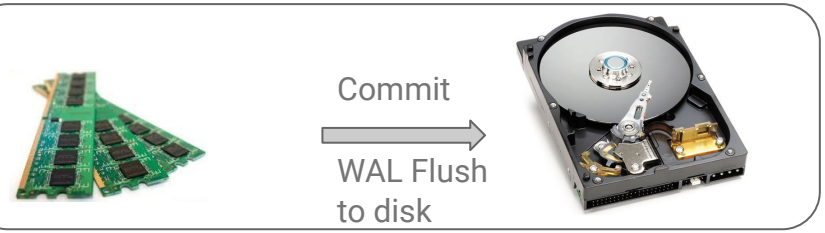
Example Visa DB

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

Visa DB

Transaction Queue

- 60000 user TXNs/sec
- Monthly 10% Interest TXN



Design#1 VisaDB

For each Transaction in Queue

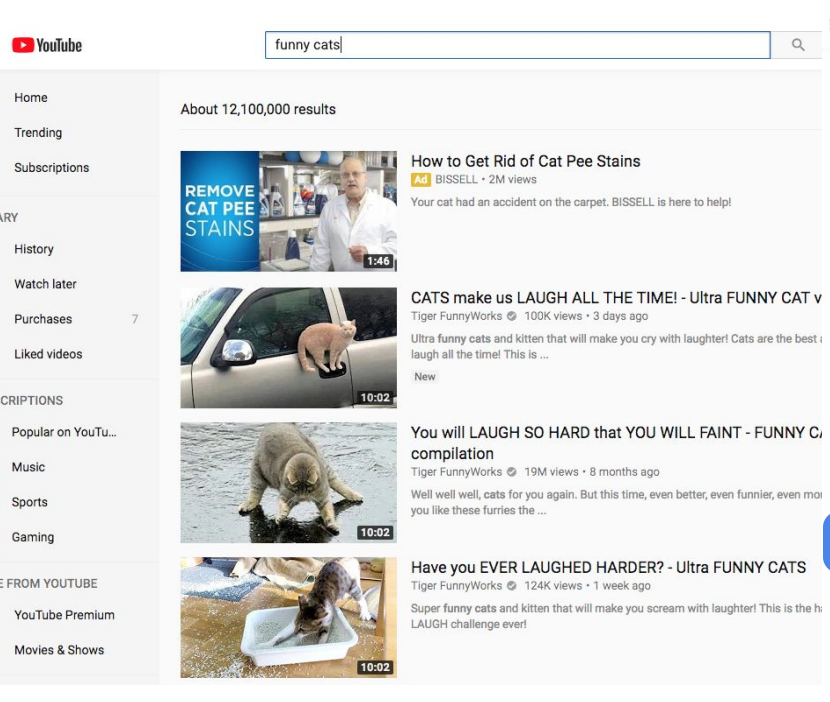
- For relevant records
  - Use **2 PL** to acquire/release locks
  - Process record
  - **WAL** Logs for updates
- Commit or Abort

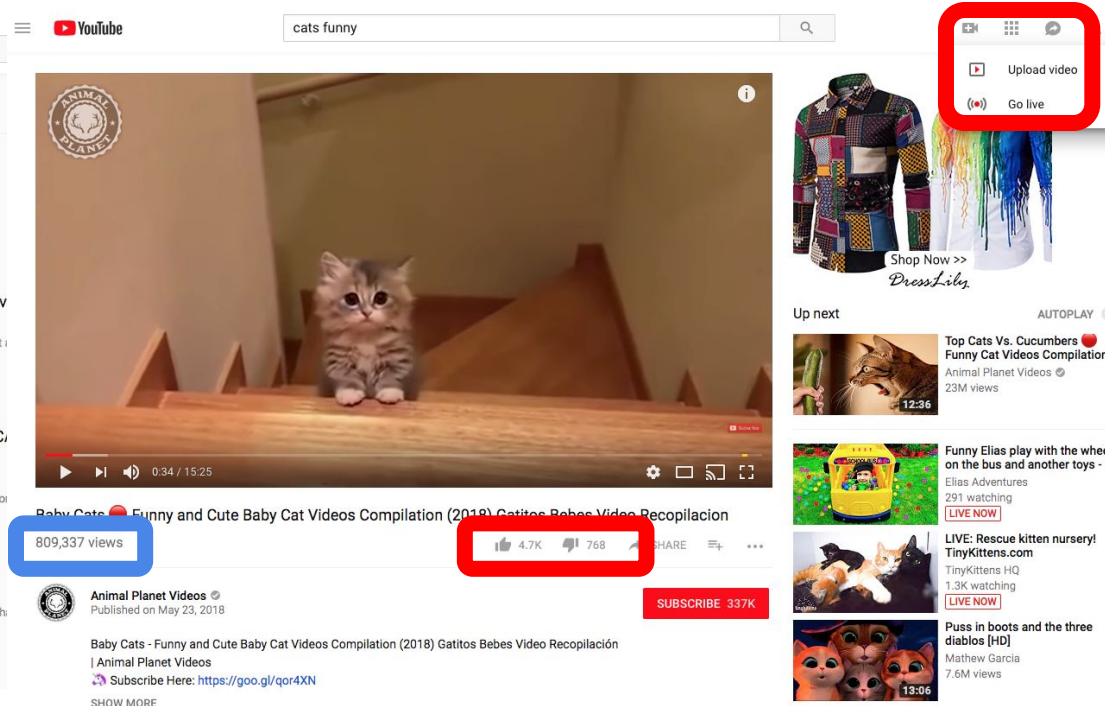
1st half

(Rapid version, Review lectures)

How to scale queries 10-100x on large data sets?

Example: Youtube DB





Key concept

Data model & SQL

Relational model (aka tables)

Simple, popular algebra (E.F. Codd et al)

Every relation has a schema

Logical Schema: describes types, names

Physical Schema: describes data layout

Virtual Schema (Views): derived tables

Data model

Organizing principle of data + operations

Schema

Describes blueprint of table (s)

SQL to express queries declaratively

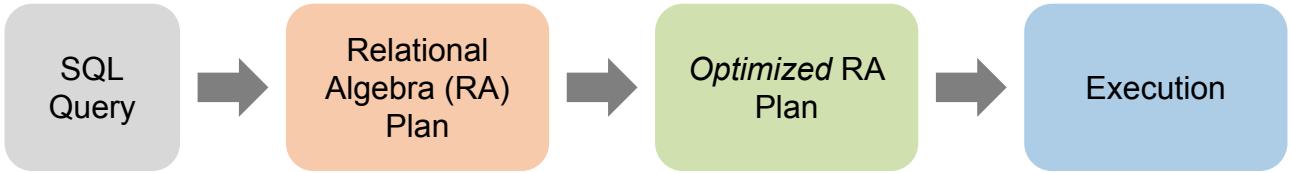
World's most successful parallel programming language

SQL

Data definition and data manipulation language

RDBMS Architecture

How does a SQL engine work ?



Declarative query (from user)

Translate to relational algebra expresson

Find logically equivalent- but more efficient- RA expression

Execute each operator of the optimized plan!

Optimization Roadmap

Build Query Plans

Analyze Plans

1. For SFW, Joins queries
   a. Sort? Hash? Count? Brute-force?
   b. Pre-build an index? B+ tree, Hash?

Cost in I/O, resources?
To query, maintain?

2. What statistics can I keep to optimize?
   a. E.g. Selectivity of columns, values

IO Aware algorithms

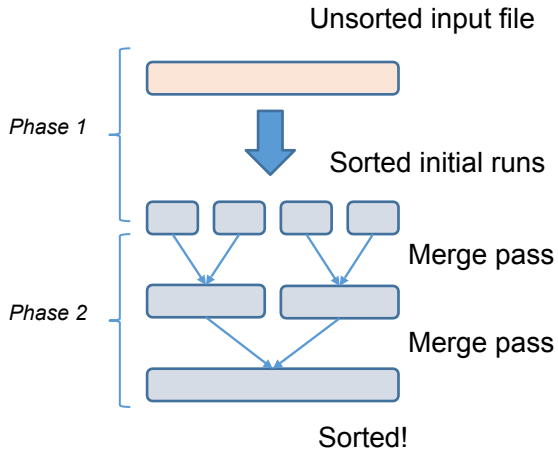
A class of algorithms which try to minimize IO, and *effectively ignore cost of operations in main memory*

External Merge Sort Algorithm

Goal: Sort a file that is much bigger than the buffer

Key idea:

- *Phase 1:* Split file into smaller chunks ("initial runs") which can be sorted in memory
- *Phase 2:* Keep merging (do "passes") using external merge algorithm until one sorted file!



Join Algorithms: Overview

For $R \bowtie S \text{ on } A$

- NLJ: An example of a *non*-IO aware join algorithm
- BNLJ: Big gains just by being IO aware & reading in chunks of pages!

Quadratic in P(R), P(S)

*I.e. O(P(R)*P(S))*

- SMJ: Sort R and S, then scan over to join!
- HPJ: Partition R and S into buckets using a hash function, then join the (much smaller) matching buckets

Given sufficient buffer space, linear in P(R), P(S)
I.e. ~O(P(R)+P(S))

By only supporting equijoins & taking advantage of this structure!

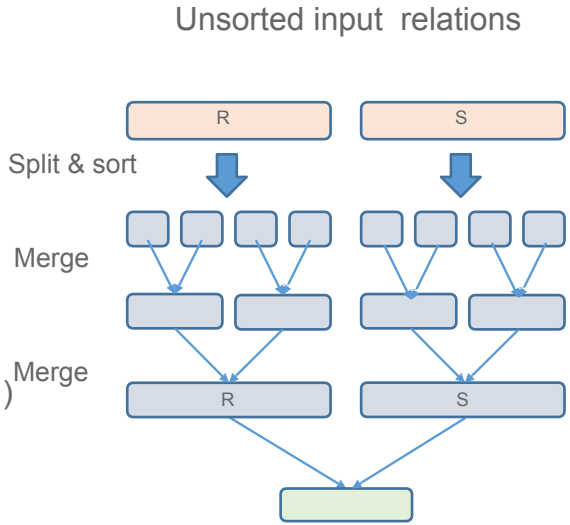
Sort Merge Join (SMJ)

Goal: Execute R ⋈ S on A

Key Idea: We can sort R and S, then just scan over them!

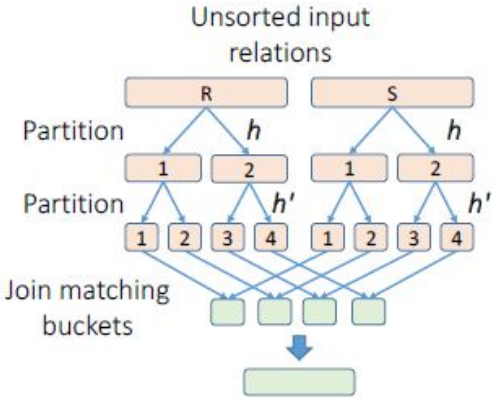
IO Cost:

- *Sort phase:* Sort(R) + Sort(S) (~2 (P(R) + P(S))
- *Merge / join phase:* ~ P(R) + P(S) + OUT



Hash Join

- ***Goal:*** Execute R ⋈ S on A
- ***Key Idea:*** We can partition R and S into buckets by hashing the join attribute-then just join the pairs of (small) matching buckets!
- ***IO Cost:***
  - *Hash Partition phase:* 2(P(R) + P(S)) each pass
  - *Partition Join phase: Depends on size of the buckets… can be* ~ P(R) + P(S) + OUT if they are small enough!
    - ***Can be worse due to skew!***



Systems Design Example:

Product Search & CoOccur

Billion products

User searches for “coffee machine”

Nespresso Vertuo Coffee and Espresso Machine Bundle with Aeroccino Milk Frother by Breville, Red

by Breville

980 customer reviews | 259 answered questions

Amazon's Choice for "nespresso machine red"

List Price: $249.95

Price: $189.96 prime | FREE One-Day

You Save: $59.99 (24%)

Your cost could be $179.96. Eligible customers get a $10 bonus when reloading $100.

Free Amazon product support included

Style Name: Nespresso by Breville

Nespresso | Nespresso by Breville

Color: Red

Product recommendations

Customers who viewed this item also viewed these products

Dualit Food XL1500 Processor

$560

Add to cart

Kenwood kMix Manual Espresso Machine

$250

Select options

Weber One Touch Gold Premium Charcoal Grill-57cm

$225

Add to cart

NoMU Salt Pepper and Spice Grinders

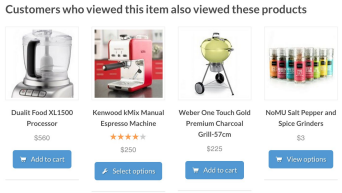
$3

View options

Systems Design Example:

Product Search & CoOccur

Counting popular product-pairs



Story: Amazon/Walmart/Alibaba (AWA) want to sell products

1. AWA wants fast user searches for product
2. AWA shows 'related products' for all products so users can explore
   - Using collaborative filtering ('wisdom of crowds') from historical website logs.
   - Each time a user views a set of products, those products are related (co-occur)

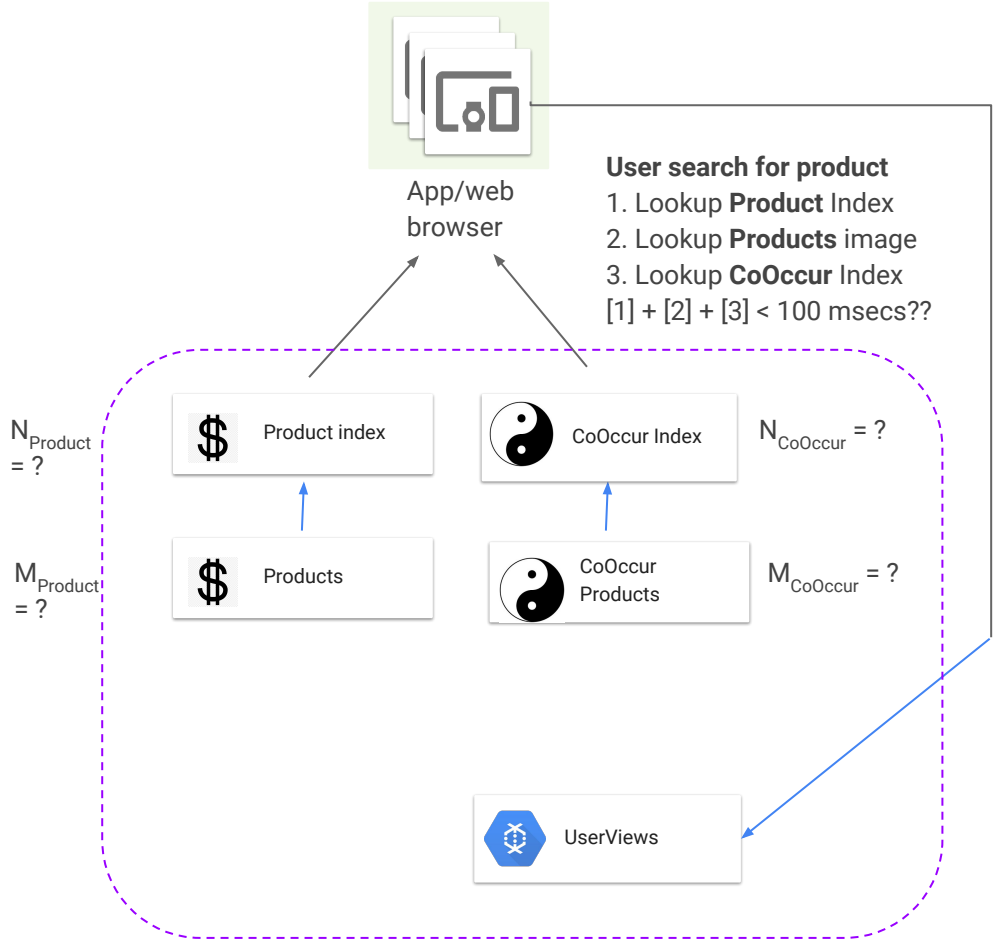
⇒ Goal: compute product pairs and their co-occur count, across all users

Data input:

- AWA has 1 billion products. Each product record is ~1MB (descriptions, images, etc.).
- AWA has 10 billion UserViews each week, from 1 billion users. Stored in UserViews, each row has <userID, productID, viewID, viewTime>.

Data Systems Design Example:

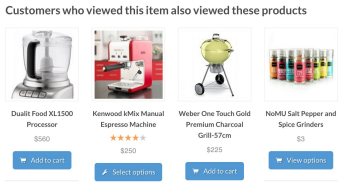
Product Search& CoOccur



Systems Design Example:

Product CoOccur

Counting popular product-pairs



Your mission: Design an efficient system to compute co-occur counts on *Sundays* from weekly logs and produce a CoOccurCount table <productID, productID, count>

1. AWA's data quality magicians recommend
   - (a) keep only top billion popular pairs, and (b) drop pairs with co-occur counts less than million.
   - (c) Also, assume users view ten products on average each week (*User is interested in ~10 products/week, not 1000s*).

2. For simplicity, SortedUserViews is stored sorted by <userID, productID>.
   - You can sequentially scan the log and produce co-occurring product pairs for each user. In other words, output (p_i, p_j) if a user viewed products p_i and p_j.
   - This "stream" of tuples (TempCoOccur) may then be (a) stored on disk or (b) discarded after updating any data structures.

Systems Design Example:

Product CoOccur

Plans?

~~Plan#1: With 1 machine, use RAM to count (Cost = 25B$ or > 100 milllion years).~~

Plan#2: With 1 machine

Plan 2

1. Scan SortedUserViews. For each user, **append** <p_i, p_j> to a file TempCoOccurLog if the user has viewed p_i and p_j. (i.e., produce per-user co-occur product pair. *Append to log ⇒ No seek...*)
2. Externally sort TempCoOccurLog on disk, so identical product pairs are adjacent to each other in the sorted file
3. Scan sorted TempCoOccurLog. With a single pass, you can count co-occur pairs. Drop co-occur pairs with < 1 million.

Nespresso	Iphone
...	
...	
Nespresso	Iphone
...	
...	
...	
Nespresso	Iphone

TempCoOccurLog
(After Step 1)

.....	
Nespresso	Iphone
Nespresso	Iphone
Nespresso	Iphone
.....	

Sorted TempCoOccurLog
(After Step 2)

Nespresso	Iphone	3

Count sorted TempCoOccurLog
(After Step 3)

Systems Design Example:

Product CoOccur

Pre-design

	Size	Why?
ProductId	4 bytes	1 Billion products ⇒ Need at least 30 bits (2^30 ~= 1 Billion) to represent each product uniquely. So use 4 bytes.
UserID	4 bytes	"
ViewID	8 bytes	10 Billion product views.
Product	1 PB	1 Billion products of 1 MB each
SortedUserViews	240 GB (4 M pages)	Each record is <userID, productID, viewID, viewtime>. Assume: we use 8 bytes for viewTime. So that's 24 bytes per record. 10 Billion*24 bytes = 240 GBs.
CoOccur (for top 1 Billion)	12 GB	The output should be <productID, productID, count> for the co-occur counts. That is, 12 bytes per record (4 + 4 + 4 for the two productIDs and 4 bytes for count). To keep top billion product pairs (as recommended by AWA data quality), you need 1 billion * 12 bytes = 12 GBs.
TempCoOccurLog (assume: ~10 product views/user)	800 GB (12.5 M pages)	# product pairs produced: 1 billion users * 10^2 = 100 billion Size @8 bytes/record (productID, productID) = 800 GBs

Systems Design Example:

Product CoOccur

Plan #2

Steps	Cost (IO)	Why?
Scan SortedUserViews Append <p_i, p_j> to TempCoOccurLog	4 M 12.5M	240GB (4 M pages) 800 GB (12.5M pages)
Externally sort TempCoOccurLog on disk (Assume sort cost is ~2N, where N is number of pages for table and B is number of buffers, and B ~~ N)	25M	IO cost is (appx) = 2*N = 2*12.5M
Scan TempCoOccurLog (sorted) and keep counts in CoOccur	12.5M	800 GB

Total IO cost = (4M+ 12.5M +25M + 12.5M) = 54M

Recall: Scan at 100 MBps, then time (secs) [assume, files are stored sequentially]

= (54M * 64 KB) /100 MBps = ~34.5K secs

2nd half Highlights

(Rapid version, Review lectures)

How to scale transactions by 10-100x for large data?

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
```

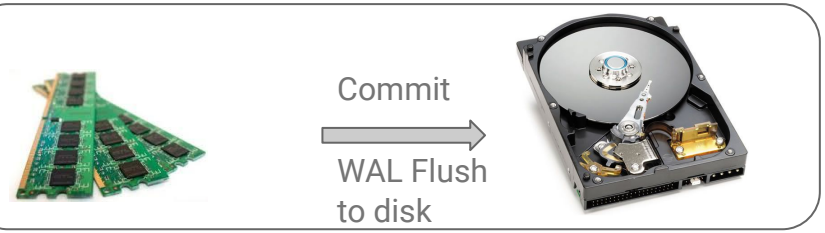
Example Visa DB

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

Visa DB

Transaction Queue

- 60000 user TXNs/sec
- Monthly 10% Interest TXN



Design#1 VisaDB

For each Transaction in Queue

- For relevant records
  - Use **2 PL** to acquire/release locks
  - Process record
  - **WAL** Logs for updates
- Commit or Abort

Example Problem 1

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	??

‘T-Monthly-423’

Monthly Interest 10%
4:28 am Starts run on 10M 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

Transactions

Summary

"Need to Master Extreme Transactions" (Forbes (Insights))

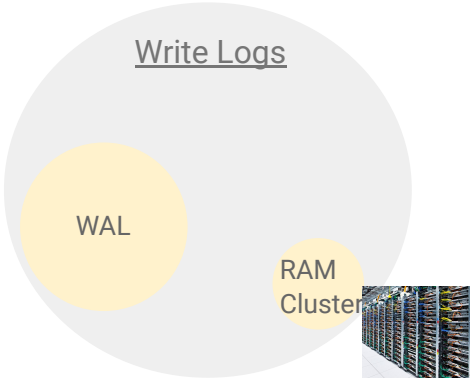
Why study Transactions?

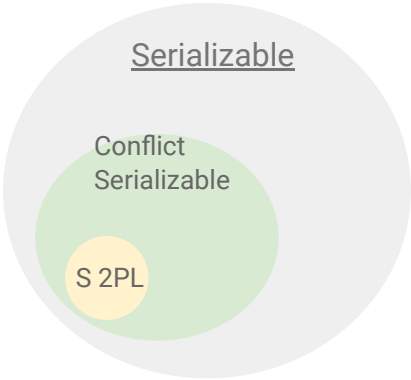
Good programming model for parallel applications on shared data !

- Atomic
- Consistent
- Isolation
- Durable

Design choices?

- Write update Logs (e.g., WAL logs)
- Serial? Parallel, interleaved and serializable?





Transaction Properties: ACID

- 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 (ie looks like batch mode)
- Durable
  - Once a txn has committed, its effects remain in the database

Write-Ahead Logging (WAL)

Algorithm: WAL

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

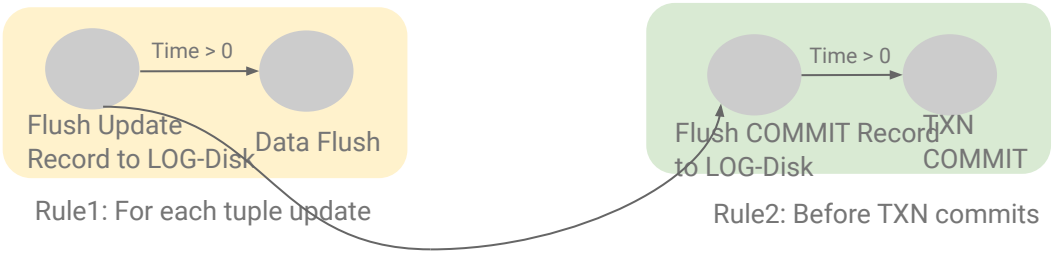
Follow two Flush rules for LOG

- 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**

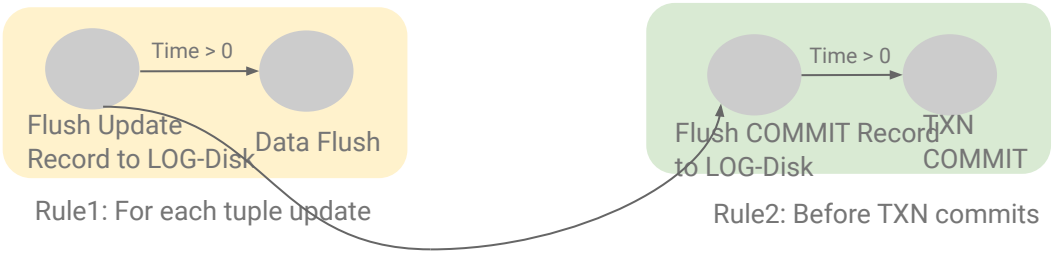
→ **Atomicity**

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



Example WAL (+S2PL) scenarios

TXN commit before COMMIT Record on disk?	No
Data page flushed before its Update Record flushed?	No
TXN commit before modified data page flushed?	Yes, often. Especially for large transactions. For TXN Commit, should have Flushed... - All Update Records to Log - COMMIT record to Log
TXN updated “Bob” and committed. TXN2 needs committed data record for ‘Bob’; record still in RAM, not flushed to disk	TXN2 requests/gets LOCK to get latest from memory.
TXN3 updated “John” but not yet committed. TXN4 needs updated record for “Tom”	TXN4 requests LOCK. Waits for TXN3.



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

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		

‘T-Monthly-423’

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

```
START TRANSACTION
        UPDATE Money
        SET Balance = Balance * 1.1
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	??

WAL 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

‘T-Monthly-423’

Monthly Interest 10%
4:28 am Starts run on 10M 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

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

WAL 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

System recovery (after 10:45 am)

1 Rollback uncommitted transactions
- Restore old values from WALlog (if any)
- Notify developers about aborted txn

1.1 Redo Recent committed transactions (w/ new values)
2 Back in business
3 Redo (any pending) transactions

(Sometimes swap 2 and 3, as a tradeoff)

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!!!

Example- consider two TXNs:

```
T1: START TRANSACTION
        UPDATE Accounts
        SET Amt = Amt + 100
        WHERE Name = 'A'

        UPDATE Accounts
        SET Amt = Amt - 100
        WHERE Name = 'B'
COMMIT
```

T1 transfers $100 from B's account to A's account

```
T2: START TRANSACTION
        UPDATE Accounts
        SET Amt = Amt * 1.06
COMMIT
```

T2 credits both accounts with a 6% interest payment

Note:

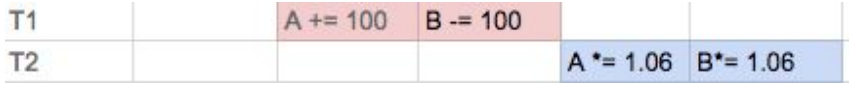
1. DB does not care if T1 —> T2 or T2 —> T1 (which TXN executes first)
2. If developer does, what can they do? (Put T1 and T2 inside 1 TXN)

Scheduling Definitions

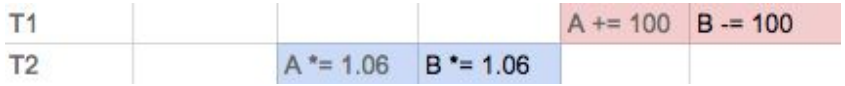
- A **serial schedule** is one that does not interleave the actions of different transactions
- A and B are **equivalent schedules** if, ***for any database state***, the effect on DB of executing A **is identical to** the effect of executing B
- *A* **serializable schedule** is a schedule that is equivalent to ***some*** serial execution of the transactions.

The word "**some"** makes this def powerful and tricky!

Serial Schedules

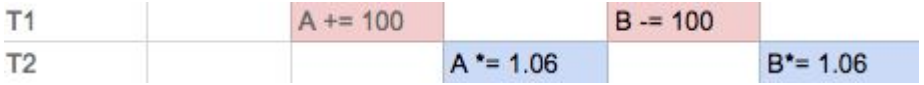


S1

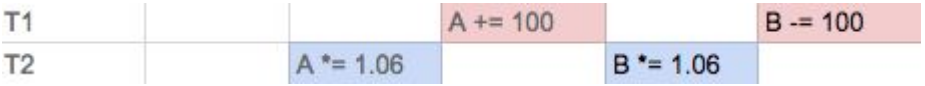


S2

Interleaved Schedules

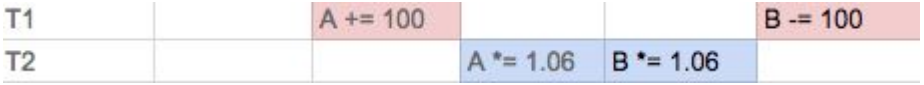


S3



S4

S5

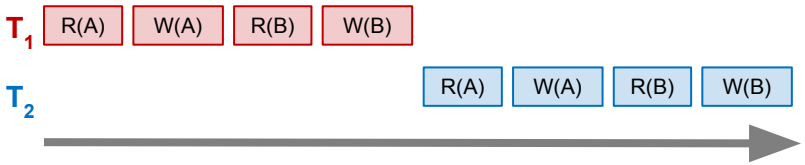


S6

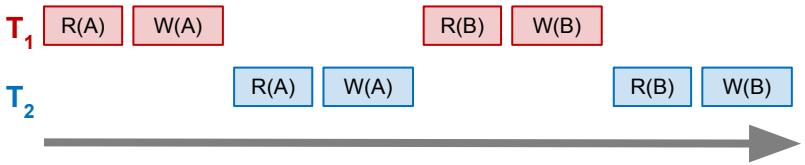
Serial Schedules	S1, S2
Serializable Schedules	S3, S4 (And S1, S2)
Equivalent Schedules	<S1, S3> <S2, S4>
Non-serializable (Bad) Schedules	S5, S6

General DBMS model: Concurrency as Interleaving TXNs

Serial Schedule



Interleaved Schedule



Each action in the TXNs *reads a value from global memory* and then *writes one back to it*

For our purposes, having TXNs occur concurrently means **interleaving their component actions (R/W)**

We call the particular order of interleaving a **schedule**

Conflict Types

Two actions **conflict** if they are part of different TXNs, involve the same variable, and at least one of them is a write

Thus, there are three types of conflicts:

- Read-Write conflicts (RW)
- Write-Read conflicts (WR)
- Write-Write conflicts (WW)

Why no "RR Conflict"?

Note: **conflicts** happen often in many real world transactions. (E.g., two people trying to book an airline ticket)

Conflict Serializability

Two schedules are **conflict equivalent** if:

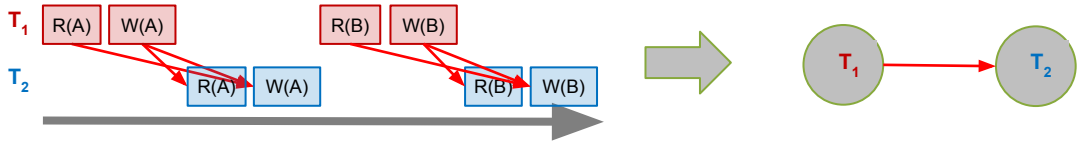
- They involve *the same actions of the same TXNs*
- Every *pair of conflicting actions* of two TXNs are *ordered in the same way*

Schedule S is **conflict serializable** if S is *conflict equivalent* to some serial schedule

Conflict serializable ⇒ serializable
So if we have conflict serializable, we have consistency & isolation!

The Conflict Graph

- Let's now consider looking at conflicts **at the TXN level**
- Consider a graph where the **nodes are TXNs**, and there is an edge from $T_i \rightarrow T_j$ **if any actions in T_i <u>precede and conflict with</u> any actions in T_j**

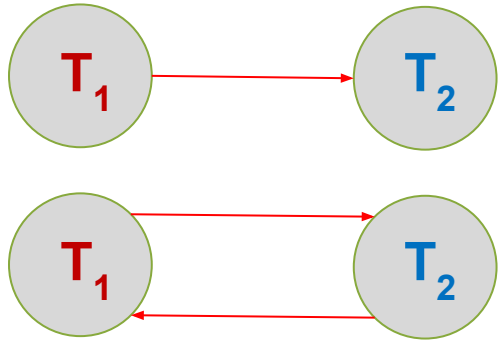


What can we say about “good” vs. “bad” conflict graphs?

Serial Schedule:

Simple!

Interleaved Schedules:



Theorem: Schedule is **conflict serializable** if and only if its conflict graph is **acyclic**

Example with 5 Transactions

Schedule S1

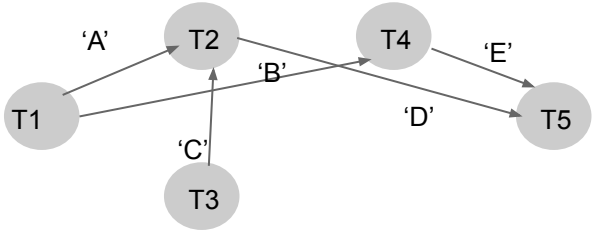
w1(A)	r2(A)	w1(B)	w3(C)	r2(C)	r4(B)	w2(D)	w4(E)	r5(D)	w5(E)

Good or Bad schedule?
Conflict serializable?

Step1
Find conflicts
(RW, WW, WR)

T1	w1(A)		w1(B)							
T2		r2(A)			r2(C)		w2(D)			
T3				w3(C)						
T4						r4(B)		w4(E)		
T5									r5(D)	w5(E)

Step2
Build Conflict graph
Acyclic? Topo Sort



Acyclic
⇒ Conflict serializable!
⇒ Serializable

Step3
Example serial schedules
Conflict Equiv to S1

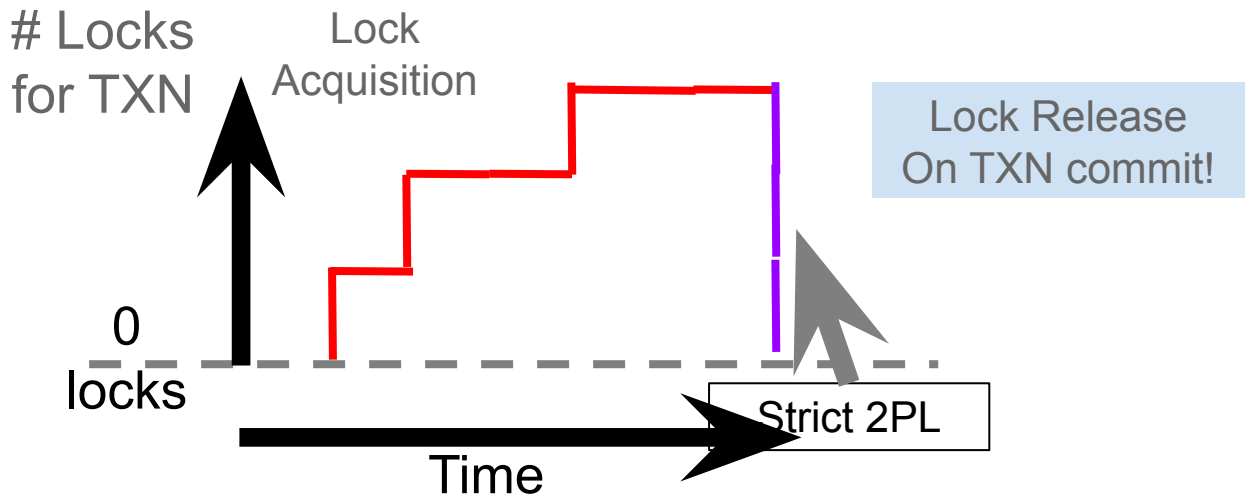
SerialSched (SS1)

T3	T1	T4	T2	T5					
w3(C)	w1(A)	w1(B)	r4(B)	w4(E)	r2(A)	r2(A)	w2(D)	r5(D)	w5(E)

SerialSched (SS2)

T1	T3	T2	T4	T5					
w1(A)	w1(B)	w3(C)	r2(A)	r2(A)	w2(D)	r4(B)	w4(E)	r5(D)	w5(E)

Strict 2-Phase Locking (S2PL)



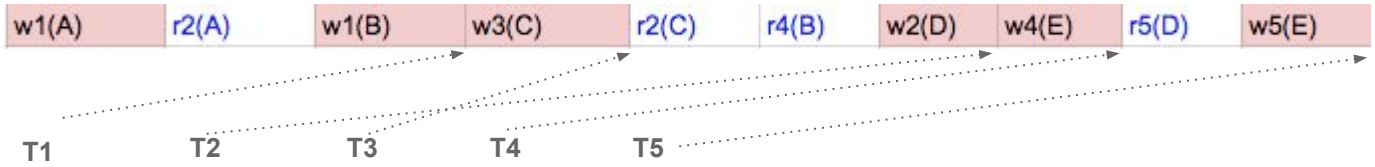
2-Phase Locking: A transaction can not request additional locks once it releases any locks. [Phase1: “growing phase” to get more locks. Phase2: “shrinking phase”]

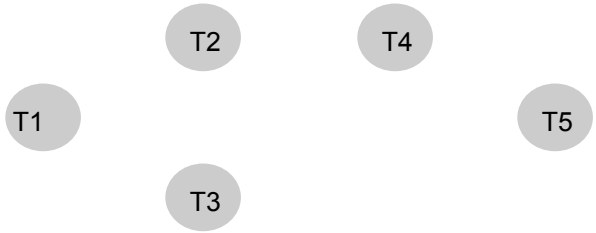
Strict 2-PL: Release locks only at COMMIT (COMMIT Record flushed) or ABORT

Example with 5 Transactions (2PL)

Schedule S1

Execute with S2PL



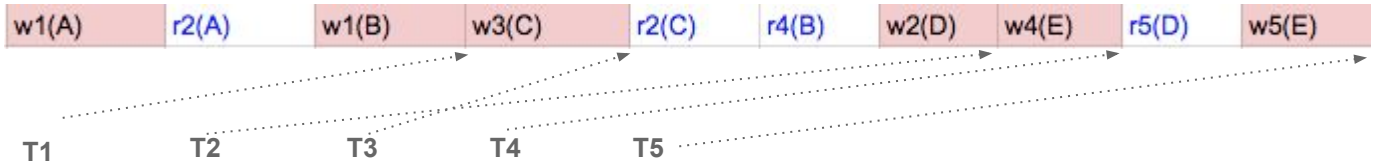


Waits- For Graph

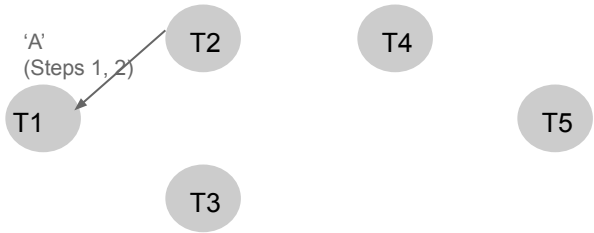
Example with 5 Transactions (2PL)

Schedule S1

Execute with S2PL



	T1	T2	T3	T4	T5
Step 0	X (A) w1(A)				
Step 1		Req S(A)			
Step 2	X (B) w1(B) Unl B, A				
Step 3		Get S(A) r2(A)			
Step 4			X (C) w3(C) Unl C		
Step 5		S(C) r2(C)			
Step 6				S(B) r4(B)	
Step 7		X(D) w2(D) Unl A, C, D			
Step 8				X(E) w4(E) Unl B,E	
Step 9					S (D) r5(D)
Step 10					X (E) w5(E) Unl D, E



Waits- For Graph

Example Visa DB -- Need Higher Performance?

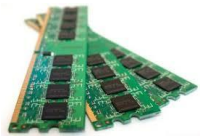
Transaction Queue

- 60000 TXNs/sec
- Monthly Interest TXN

‘T-Monthly-423’

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

Visa DB



Commit

WAL Flush to disk

Design#2 VisaDB

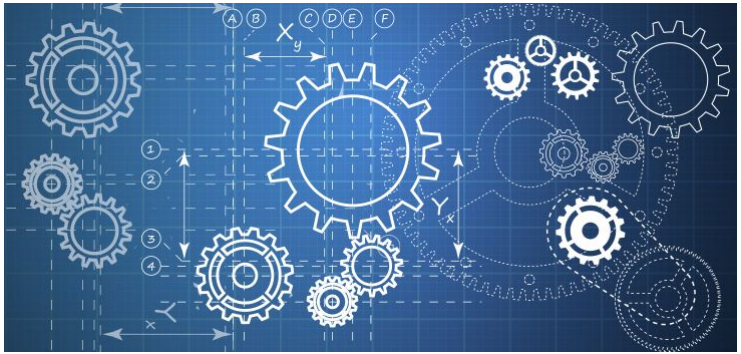
For each Transaction in Queue

- For relevant records
  - Use ~~2 PL~~ to acquire/release locks
  - Process record
  - ~~WAL~~ Logs for updates
- Commit or Abort

Replace with more sophisticated algorithms (cs245/cs345)

Design Theory

- Design theory is about how to represent your data to avoid ***anomalies***.
- Simple algorithms for “best practices”

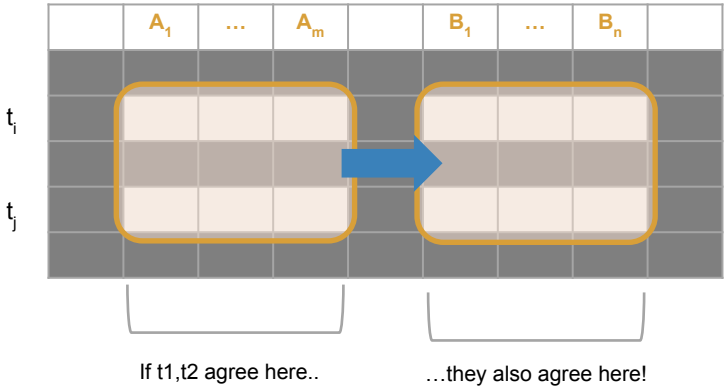


Relational Schema Design

High-level idea

1. Start with some relational *schema*
2. Find out its *functional dependencies (FDs)*
3. Use these to *design a better schema*
   One which minimizes the possibility of anomalies

A Picture Of FDs



Defn (again):
Given attribute sets **A={A_1,…,A_m}** and **B = {B_1,…B_n}** in **R,**

The ***functional dependency*** **A→ B on R** holds if for ***any*** t_i,t_j in R:

if $t_i[A_1] = t_j[A_1]$ AND $t_i[A_2]=t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

then $t_i[B_1] = t_j[B_1]$ AND $t_i[B_2]=t_j[B_2]$ AND … AND $t_i[B_n] = t_j[B_n]$

Finding Functional Dependencies

Given a set of FDs, F = {f_1,…f_n}, does an FD g hold?

Inference problem: How do we decide?

Answer: Three simple rules called
Armstrong's Rules.

1. **Split/Combine**
2. **Reduction**
3. **Transitivity**

Finding Functional Dependencies

Example:

Products

Name	Color	Category	Dep	Price
Gizmo	Green	Gadget	Toys	49
Widget	Black	Gadget	Toys	59
Gizmo	Green	Whatsit	Garden	99

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Department}
3. {Color, Category} → {Price}

Which / how many other FDs hold?

Finding Functional Dependencies

Example:

Inferred FDs:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	Trivial
5. {Name, Category} -> {Color}	Transitive (4 -> 1)
6. {Name, Category} -> {Category}	Trivial
7. {Name, Category} -> {Color, Category}	Split/Combine (5 + 6)
8. {Name, Category} -> {Price}	Transitive (7 -> 3)

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Dept.}
3. {Color, Category} → {Price}

What's an algorithmic way to do this?

Keys and Superkeys

A **superkey** is a set of attributes $\mathbf{A_1, \ldots, A_n}$ s.t.
for *any other* attribute **B** in R,
we have $\mathbf{\{A_1, \ldots, A_n\} \rightarrow B}$

I.e. all attributes are *functionally determined* by a superkey

A **key** is a *minimal* superkey

Meaning that no subset of a key is also a superkey

Superkey Algorithm:
For each set of attributes X

1. Compute X^+
2. If X^+ = set of all attributes then X is a **superkey**
3. If X is minimal, then it is a **key**

Conceptual Design

For a “mega” table

- Search for “bad” dependencies
- If any, *keep decomposing (lossless) the table into sub-tables* until no more bad dependencies
- When done, the database schema is *normalized*

Recall: there are several normal forms…

CS145

Goals

Course Summary

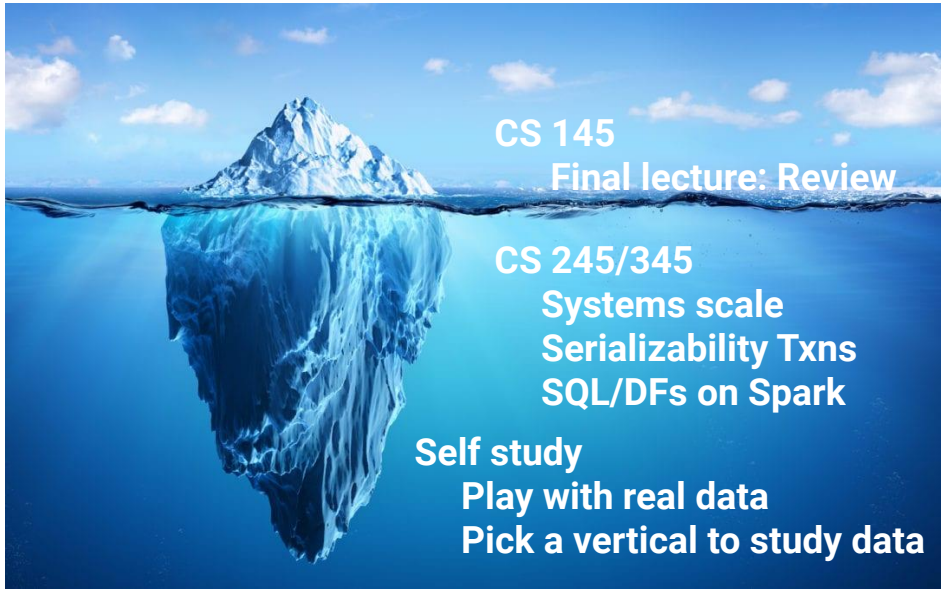
We'll learn How To…

- **Query** over small-med-large data sets with **SQL**? [Weeks 1 and 2]
  - On relational engines, and "big data" engines
- **Scale** for "big queries"? On Clusters? [Weeks 3, 4, 5]
  - OLAP/Analytics, 1st principles of scale
- **Scale** for "big writes"? [Weeks 6, 7, 8]
  - Writes, Transactions, Logging, ACID properties
- **Design** "good" databases? [Weeks 9, 10]
  - Big Schemas, design, functional dependencies, query optimizers

Project: Query-Visualize-Learn on GB/TB scale data sets on a Cloud [sql + python]

Course Summary

Next Steps



THANK YOU!