# CSE 344
# Final Review

August 16th

# Final

- In class on Friday
- One sheet of notes, front and back
  - cost formulas also provided
- Practice exam on web site
- Good luck!

# Primary Topics

- Parallel DBs
  - parallel join algorithms
  - parallel query optimization
  - MapReduce
- DB Design
  - E/R diagrams
  - constraints
  - normalization

# Primary Topics

- Transactions
  - ACID properties
  - serial vs serializable vs conflict serializable
  - (strict) 2PL locking
- First-half material also fair game!

# Review…

# Motivation

- important uses of databases
  - big data
    - almost every company has too much for 1 machine
  - very important data
    - critical that the data is not lost or corrupted
  - complex queries
    - hard to answer them efficiently
- modern DBs solve these problems

# What is a database?

- collection of related data
- provides languages to describe, query, and update data
  - checks that new data satisfies constraints
  - allows you to change the schema
    - very common & hard problem
  - easy languages for querying data
    - also efficient implementation
  - provides high levels of reliability
  - hides many (physical) details

# SQL (everywhere)

- best language we have
- easy for non-programmers to learn
- can express almost any query you have
- works well for parallel DBs just as well

# Relational Algebra

- all of SQL simplifies to just a few operations
  - union, intersect, select, projection, join, & aggregation
- not as useful for users
  - harder than writing SQL
  - have to say how to implement query
- very useful for DB implementers
  - similar to intermediate language of compiler
- implementers call RA “query plans”

# Datalog

- very different way of writing relation query
- closer to logic
  - explanation for why it can answer any question
  - (connections to AI)
- only slightly more expressive than SQL / RA
  - only adds recursion!
  - w/out recursion can convert back and forth
- need to write “safe” rules
  - unsafe rules generate infinite results

# NoSQL

- relational data is not natural
  - lists are natural to users but not 1NF
  - JSON is more natural
- early systems: key-val pairs & extensible records
  - huge scale for OLTP workloads
  - BUT reduced functionality
    - limited data model
    - no joins (ouch!)
- modern systems have JSON + full functionality

# SQL++

- supports querying non-1NF data
- all you need is un-nesting
  - e.g., “world x, x.rivers y”
  - pretty easy to add to other systems
- also allows working with lists directly
  - both input and output
  - remove restriction on subquery location
    - can have >1 rows in result in SELECT clause
  - more convenient for users

# Internals

- logical plans: RA
- physical plans: choice of op implementations
  - e.g., join algorithms
- pipelining
  - allows tuples to go to the user more quickly
    - don't need to wait for all tuples to be ready
  - no need to store intermediate results
    - no disk cost for selection & projection

# Internals cont.

- indexing
  - clustered vs unclustered
  - hash vs B+ tree vs other
- disks are unbelievably slow
  - hard disks are mechanical devices
  - reading 1-2% is as slow as reading whole file
  - for speed: store in memory of many machines
    - becoming increasingly common

# Parallel DBs

- shared memory & shared disk work with smaller amounts of data
  - BUT modern systems are shared nothing
- workloads: OLAP vs OLTP
  - OLAP is big read-only queries
  - OLTP is many read/write queries, each accessing only a small amount of data
  - can’t support both at scale!
    - best solution is to execute OLAP on old data
      - (multi-version)

# Parallel DBs cont.

- *easy* solutions for OLAP & OLTP are different
  - partitioning & replication
- we want both!
  - need to use partitioning (for OLTP)
  - then figure out how to execute OLAP queries on partitioned data…

# Parallel query plans

- can still use cost-based optimization
  - could be disk cost or network cost
  - (only network if no disk involved)
- only new operation is reshuffle!
  - all other work on a single machine
    - can use in-memory operations at no cost
  - cost will generally be worse with more reshuffling
  - where have I seen reshuffle before...

# Parallel query plans cont.

- map reduce
  - steps: input > map > shuffle > reduce > output
  - you provide map and reduce parts
  - framework provides (re)shuffle
  - all steps use (key,value) pairs as data format
    - framework only looks at key
    - value is opaque
  - framework handles many low-level details
    - restarts workers that fail
    - reassigns finished workers to *straggling* jobs

# DB Design

- Getting the data right is half(?) the problem
- E/R is a great way to communicate design
  - higher level than SQL, a picture!
  - some new complexities
    - multi-way relationships
    - subclasses
    - weak entities

# DB Design cont.

- People really do make schema design errors
- Normalization is a great way to find them!
  - BCNF is the standard
  - 4NF might also be useful
- Functional dependencies are *constraints*

# DB Design cont.

- Constraints are really important
  - how can you ensure they are always satisfied without identifying them?
- DB automatically checks many constraints
  - even auto-fixes broke FK constraints
    - cascade, set null, etc.
- Rest are up to you
  - only need to check before committing

# DB Performance

- Main choices that affect performance
  - indexes
  - materialized views
- Indexing
  - includes choice of clustering
    - e.g., if multiple keys, which is primary?
    - primary key becomes clustering
  - indexes improve queries but slow updates
    - also create more lock contention
    - queries are most important though...

# DB Performance cont.

- Views are tables computed from others
  - you give the query used to compute them
  - can then refer to the view by name without giving the query
  - ways to implement:
    - re-compute on demand
      - just substitute the query
    - store and updated
      - called materialized views

# Transactions

- ACID properties let us write correct apps
  - (you saw this in HW8)
  - other models are difficult
- consistency and durability are easy
  - consistency: check constraints before commit
  - durability; write to (multiple) disks
    - ideally, geographically-separated disks
- atomicity and isolation are harder
  - locking or MVCC provide these

# Locking

- serial schedules are isolated by definition
- serializable schedules are more general
  - just as good: identical behavior
  - allows parallelism
- conflict serializability
  - special type of serializable schedules
  - can prove serializability by simple swaps

# Locking cont.

- 2PL ensures conflict serializability
- strict 2PL gives atomicity & isolation
  - trouble for 2PL is rollback (atomicity)
- phantom tuples are still an issue
  - easiest solution: lock part of the index
  - can also use predicate locks (hard)
- ways to fine tune performance
  - lock modes
  - lock granularity
  - admission control sometimes actually helps!

# Locking

- tradeoff between correctness & performance
  - saw how to get ACID but at substantial cost
- DBs default to non-ACID
  - SQL Server allows phantoms
- in that case, correctness is up to you
  - need to think through whether phantoms will break any of your apps
  - this is very hard!
    - especially as the code is changing
    - especially with many programmers

# Final notes

- Study
  - transactions: CS, locking
  - DB design: E/R, BCNF
  - parallel DBs: network cost estimation
    - see Wednesday lecture
- Briefly review other materials