EPBI 414

Unit 4
Data Warehousing & DB Optimization

The Recap - Unit 3

- The relational data model
  - Predicates and propositions
  - First three normal forms
  - Database normalization

- RDBMSs and their functions

- Design of complex CRFs using relational databases

Unit 4 Overview

- Database performance and indexes
- Database transactions and ACID
- Distribution and Brewer's CAP theorem
- Denormalized data storage
    - Data warehousing, star schema, fact / dimension
- NoSQL and the future of the RDBMS

Improving speed - indexes

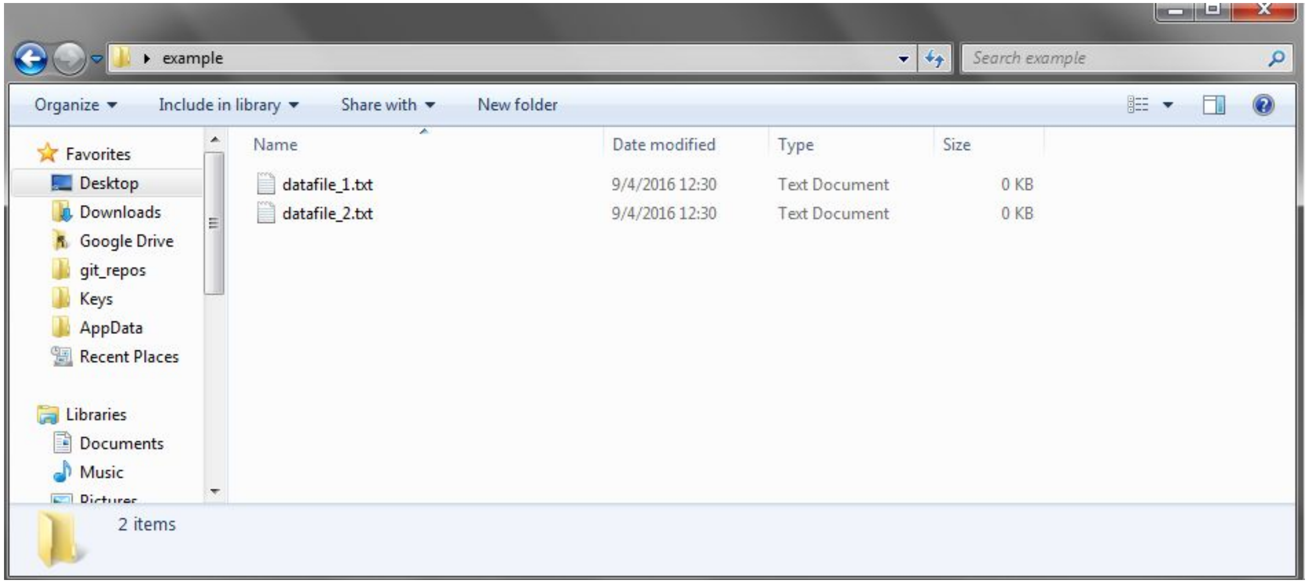
- Database indexes are ways that you can improve the performance of your database

- Indexes depend heavily on the RDBMS in which they are implemented

- You need a theoretical understanding
  - Partner with a DBA for implementation of the index

How is data stored?

This is how we're used to thinking of data on the computer...



Physical data storage

- The reality is that ultimately, data must be stored physically
    - The computer is a ultimately a physical machine
    - Has implications for storage, retrieval, and manipulation
- Storage is called ***non-volatile*** if it remains stored even when the system is not powered
    - In contrast to ***volatile*** storage, which loses its state when the power is cut

Data storage options

- Two common types of non-volatile physical data storage:
  - Magnetic hard drive (historically common)
  - Solid state drive (newer, more expensive)
- On a magnetic hard drive, data is stored on ***platters***
  - A ***head*** moves over the area where the data is stored to retrieve it
  - This involves ***physical motion***, which takes time

History: sectors and blocks

- Magnetic hard drives have discrete physical locations for data storage, called ***sectors***
  - For many years, a sector could generally store 512 bytes - recent change to 4096 bytes
- As a disk gets bigger, it has more sectors
  - Can be hard for the operating system to track
- By grouping ***sectors*** into ***blocks***, the OS could manage more storage

Why does this matter?

- Data must be ***read*** and ***written*** at the block level
  - Not strictly true of SSDs, but SSDs emulate block devices for compatibility (beyond our scope)

- Data larger than a single block (most data) gets spread over multiple blocks
  - Each block contains data pointing to the next block with related data

  - Blocks might not be physically contiguous!

But WHY does it matter?

- When data spans ***multiple blocks***, you have to look in more than one block
    - That means more physical motion - which means more time

- Given a unique, unordered field spanning multiple blocks, to find any specific value, you will need to look at half of them (on average)
    - Look up ***negative hypergeometric*** distribution

Even more block accesses

- If the field isn't unique, you need to look at all the blocks
    - Otherwise, you don't know if you have them all
- You can do searches much more quickly if fields are sorted
    - This allows for a ***binary search***, which is faster
    - Gets you into the analysis of algorithms and big-O notation (also out of scope)

The conclusion

- Keeping data sorted is useful to searching quickly
- Storing certain data in a sorted format at all times could make it faster to look up records
- These conclusions lead us into...***indexes***

What is an index?

- Consider the phone book
    - Optimized to retrieve people by first and last name
    - If you know these two things, you can easily locate the phone number
- How does the phone book work? It ***points*** you to a specific area for each person
- An index combines ***keys*** and ***pointers***

How does it work?

- An index stores sorted values (or ***keys***), and then a ***pointer*** to the record associated with that value
  - Pointer gives the disk location of the data, making access faster
- The index is a separate structure from the column
  - Only contains some values, to save space and be more efficient

So, what's the catch?

- The catch is: indexes make it easier to ***find data***, at a cost of making it harder to ***store data***

- Specifically, for the index to work, the RDBMS must perform additional ***write operations*** whenever the DB is updated
    - This is the downside of indexes

Why more writes?

- Think about our phone book again
- Imagine that the phone book could be altered in real-time, by having new records inserted, or people's names being changed
- The names and locations would need to be updated when the phone book changes
    - Hence, the ***index*** needs to be updated

Not a silver bullet

- Indexes can improve database performance
  - Not always relevant to your work
- No index can fully replace logically optimizing your work
- For your work, you should consider the ***way you access data*** when solving problems

Using indexes

- Generally speaking, you should defer to a DBA or architect
  - They will know the best way to implement an index in your chosen RDBMS
- However, you need to work ***with*** them to make the index work for you
- Chances are, they won't have the subject-matter knowledge you do

Choosing your index

- You should choose your index based on the ***most common type of search*** you do

- If you commonly look up a patient by hospital and then by MRN...
  - Then your index should start with the hospital, not the MRN

- Why?

Using an index right

Hospital	MRN
Clarkeview	103
Clarkeview	221
Clarkeview	310
Clarkeview	399
Harrisburg	123
Harrisburg	144
Harrisburg	151
Northwood	452
Northwood	488

Looking up a patient at Clarkeview with MRN 310...

City	MRN	Chunk
Clarkeview	100 - 199	1
	200 - 299	2
	300 - 399	3
Harrisburg	100 - 199	4
Northwood	400 - 499	5

The index fails us

Hospital	MRN
Clarkeview	103
Clarkeview	221
Clarkeview	310
Clarkeview	399
Harrisburg	123
Harrisburg	144
Harrisburg	151
Northwood	452
Northwood	488

Now, let's say we want to find the patient with MRN 144...

City	MRN	Chunk
Clarkeview	100 - 199	1
	200 - 299	2
	300 - 399	3
Harrisburg	100 - 199	4
Northwood	400 - 499	5

Other notes about indexes

- There is very little value in creating an index on data primarily used for outputs
  - Neither sorting or searching on output
- The same features that make indexes useful for searching help with sorting
  - By default, the sets that make up a relational data model are not ordered
- Indexes get more common as size increases

Database transactions

- Databases are particularly concerned with the problem of simultaneous operations
- A database may have multiple people reading and writing at the same time
- A reliable system needs to handle this fact, which requires specific principles

What is a transaction?

- A ***database transaction*** is a single logical operation
  - This could be multiple changes in data, or even multiple commands
  - These are kept together into a single unit, generally because they are related
  - Imagine a system updating the date, then updating everyone's age based on it
    - One transaction, two actions

Transaction requirements

- To ensure reliability, a database transaction must meet the four ACID principles:
  - ***Atomicity***
  - ***Consistency***
  - ***Isolation***
  - ***Durability***

- A database system is ACID-compliant when the transactions fulfill all four of these

A - Atomicity

- An ***atomic transaction*** is a singular unit
- If any part of that transaction fails, the whole transaction fails
  - All or nothing
  - Failure can be data, power failure, disk loss, etc
- Often represented in RDBMSs by the concepts of rollback and commit

Failures in atomicity

- Imagine a database storing grades
- Each time an assignment is graded, the overall grade is recomputed
- If the new assignment is graded, but the overall grade computation fails, then atomicity has been violated

C - Consistency

- A ***consistent transaction*** manipulates the database from one valid state to another
  - Does not violate any of the relational rules
- A proper system will require that transactions comply with the system
  - The system will require that the data be stored consistently
  - Stops programmers from breaking things on accident - let the DB do the thinking

Failures in consistency

- You have defined a field called `gender`, which contains values in the set `{Male, Female, Other}`

- A developer performs an update which changes someone's `gender` to `Transgender`
    - This creates an inconsistent state

I - Isolation

- An ***isolated transaction*** is indifferent to the concurrency of execution

- ***Transaction isolation*** means that two transactions executed simultaneously have the same result as if they were executed in sequence
  - In other words, the transactions do not ***collide***

Failures in isolation

- A calculation is done where deposits are added to a total, and then the deposit fee is subtracted

- In the period between when the deposit is added, and the fee is subtracted, another process retrieves the total
    - This total is incorrect, and is an isolation failure

D - Durability

- A ***durable transaction*** is permanently stored when it is committed

- This means that even if the database crashes right after execution, the results are stored
    - This means data must be written to permanent storage

    - Can be difficult in distributed systems (more later)

Failures of durability

- You are working on a remote server, and send a series of commands
  - The commands execute, giving you an acknowledgement
- A few moments later, you lose your server connection
- Upon reconnection, your data is missing
  - Was not written to storage quickly enough

Implications of ACID

- ACID is what makes RDBMSs valuable

- Implementing the various conditions of ACID is a big part of what an RDBMS does
  - Different RDBMSs have differing approaches to the various features

  - DB architecture is a deep field with a lot of options

  - What works for a small project may not scale at all

Break Time

Distributed computing

- As projects grow larger, it becomes dangerous to place everything on a single machine
- As scale grows, it becomes impossible to support operations on a single computer
- Distributed computing has some challenges

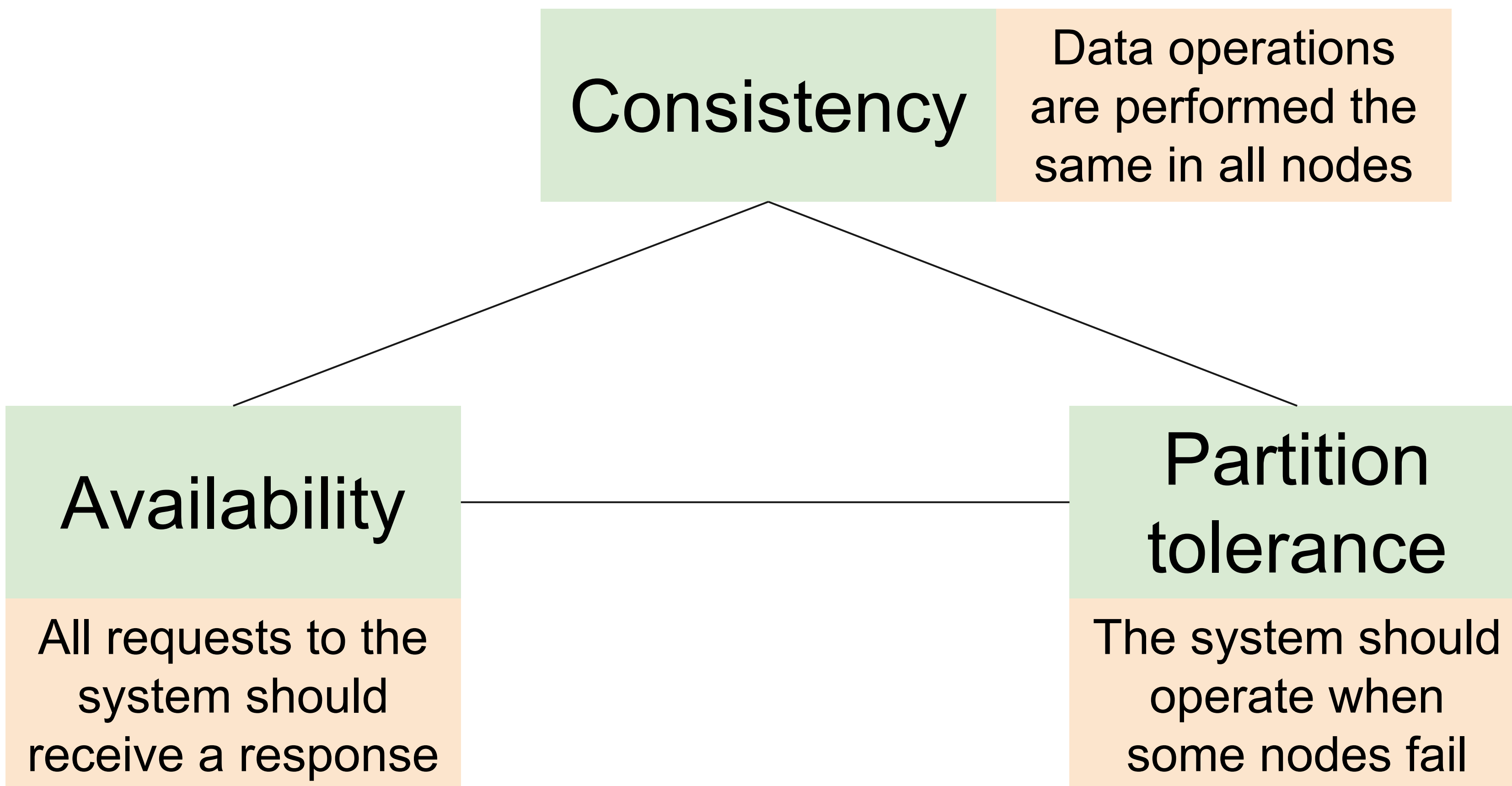
Brewer's theorem

- Sometimes known as ***CAP theorem***, based on the three principles
- Created by theoretical computer scientist Eric Brewer sometime in 1998
- Relates to data sharing across networked systems

The CAP principles

- The ***CAP*** principles stand for:
  - Consistency
  - Availability
  - Partition tolerance
- Brewer's theorem states that any distributed system can only possess two of the three principles
- This has important implications for DBs

The CAP triangle



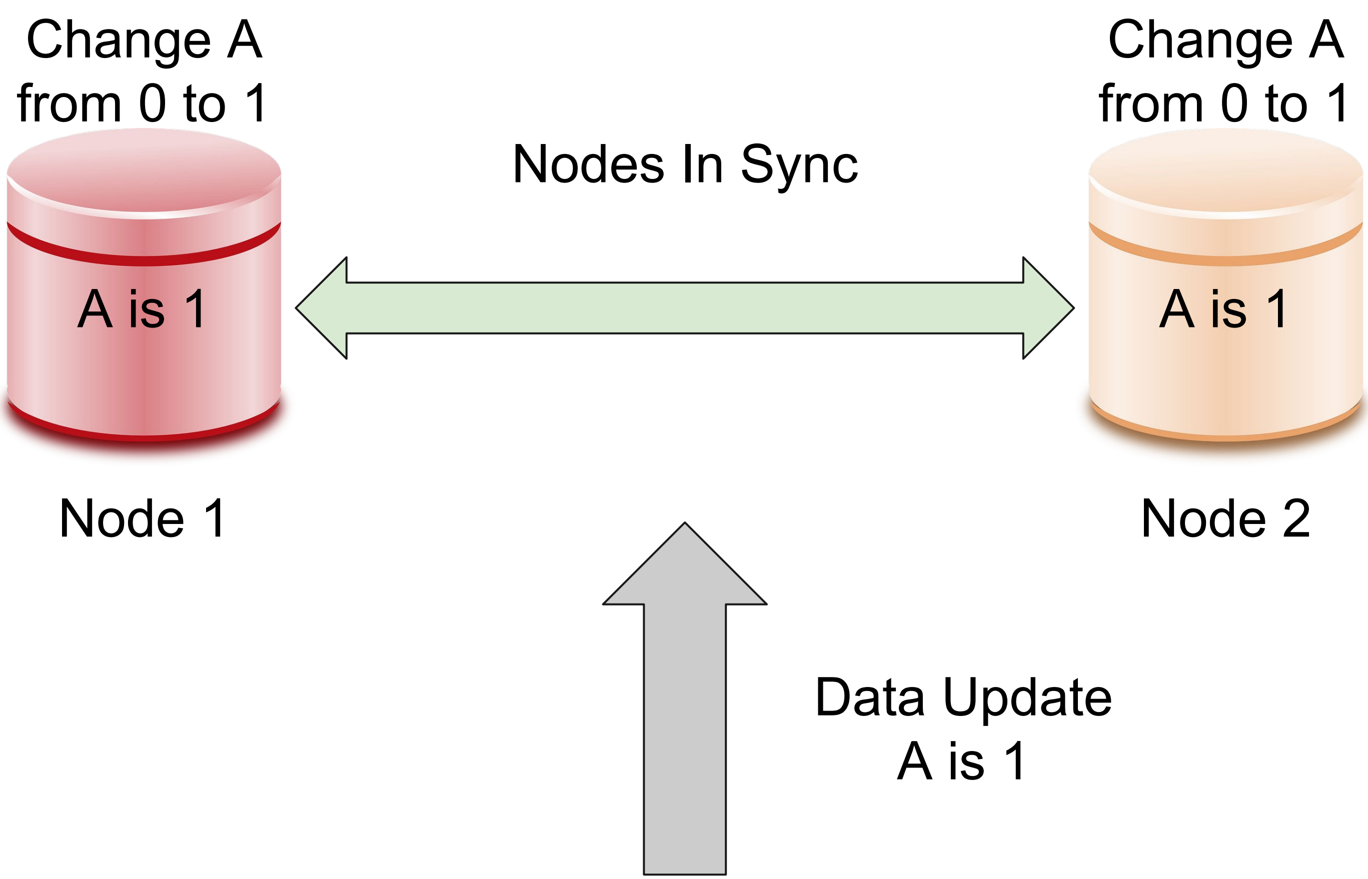
Why not all 3?

- A partition-tolerant system cannot satisfy both consistency and availability

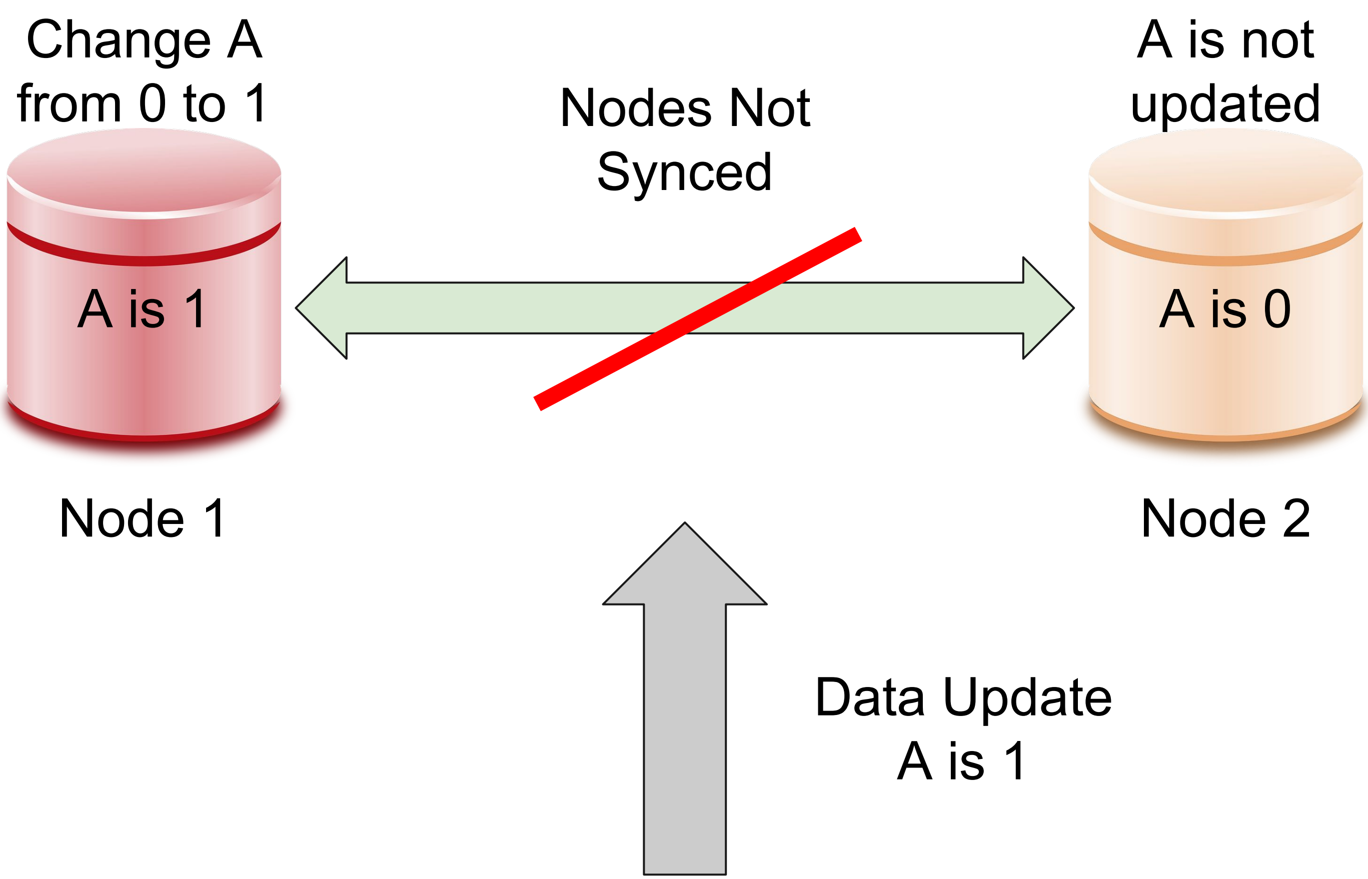
- Why is this? (simple version)
    - Partition tolerance requires that the system will work even if some nodes cannot be reached

    - If you cannot reach all the nodes, you must either wait (to ensure consistency) or answer (to ensure availability)

Data is updated



Update with network failure



Notes on CAP

- CAP is not the only thing that matters
  - NoSQL
- CAP is applicable to ***distributed systems***
  - This makes it particularly important to large-scale web applications
- You are unlikely to design anything that requires you to think about this
  - But you very well may query systems that take this into account

Denormalization

- We just spent an entire unit talking about the greatness of the 3NF

- So, why would we want to not use the 3NF?
    - 3NF can add complexity in certain contexts

    - 3NF can make extraction for reporting more difficult

- Sometimes,a denormalized structure makes sense

Data warehousing

- Denormalized data structures are often encountered in the context of ***data warehouses***

- Data warehousing is its own field, with various theories and advocates
  - Won't cover the entire field in this course

- Data warehouses are most common in business environments (BI)

Very basic terminology

- A ***data warehouse*** is a system which aggregates and stores data from systems across the entire enterprise

- A ***data mart*** is generally a smaller system, often focused on a specific domain, and integrating only some inputs

Data warehousing pioneers

- The two fathers of data warehousing are ***Ralph Kimball*** and ***Bill Inmon***
- Each has spent a considerable amount of their career studying this subject
    - We are greatly simplifying their positions
- Generally, these considerations won't be directly relevant to your career

Kimball v. Inmon

Ralph Kimball

- Bottom-up
- Data warehouse is denormalized
- Cheaper up front, more expensive over time

Bill Inmon

- Top-down
- Data warehouse is normalized
- Expensive up front, lower costs going forward

Kimball v. Inmon, part 2

Ralph Kimball

- Over time, departments tend to build systems to support their operations. Eventually, these conglomerate into a data warehouse.

Bill Inmon

- At the start, a system is built to integrate data from numerous sources into a single source of truth. This is the data warehouse.

A final note on founders

- Because of their differing approaches, Kimball and Inman tend to have different definitions to terms

- We will focus a bit more on Kimball, because we will talk about denormalized data storage
    - Tends to be more prevalent in Kimball-style systems

- There is a lot more to this subject than we can cover

Reporting vs. operations

- Data systems that support operations tend to be focused on ***consistency*** of the data
    - Think "data entry for a clinical trial" - you care about the data being right

- But there is a subset of users who want to use data - mostly historical data - to derive ***insights***
    - "Business intelligence", "Reporting", et cetera

Priorities can differ

Operations

- Consistency of data is key
- Systems need to support the application

Reporting

- Ease of access is important
- Speed is good
- Systems need to support end-users
  - Some aren't very technical

Why not query ops data?

- In a properly normalized (3NF) database, data is spread across multiple tables ***by design***

- This is excellent for consistency, but it means that simple questions could require many tables (and many joins)

- The gain is perhaps not worth the pain

The star schema

- The star schema is a common method for organizing data for reporting

- Star schemas simplify queries and improve performance
  - Not required to be denormalized, but often this is the case

- The key conceptual feature to star schema is the distinction of ***fact*** and ***dimension*** tables

Facts and dimensions

- ***Facts*** are measurements of a specific event
  - Strictly speaking, facts should be numeric, but you can have non-numeric facts

- ***Dimensions*** are factors which describe the entities involved in the event

- The simplest star schema has a single fact table and multiple dimension tables
  - Hence the name, star schema

Facts and dimensions

Date	Day	Month	Year	Quarter
1	1	1	2010	Q1
2	2	1	2010	Q1
3	3	1	2010	Q1
...	...	...	...	...

Clinic	Name	City	State
1	Philips Hospital	Kensfield	WY
2	St. Mary's	Grand Rapids	MI
3	Athena Health	Youngstown	OH
...	...	...	...

Procedure	Procedure Name
1	Hemoglobin A1C
2	Blood pressure
3	Cholesterol
...	...

Date	Clinic	Procedure	Number Performed
1	1	1	13
1	1	2	21
1	1	3	18
...	...	...	...

Denormalization

- Star schema is often found in denormalized fashion
  - System design is driven by queries most commonly done
  - Goal is generally to minimize the number of joins required
- Generally, the dimensions are denormalized, not the facts
  - The fact tables might have computed values

Denormalized dimensions

Date	Employee	Procedure	Number Performed
1	1	1	13
1	2	1	21
1	3	1	18
...	...	...	...

Employee	Last Name	First Name	Department	Department Classification
1	Smith	Jennifer	Pediatrics	Primary
2	Douse	Sarah	Pediatrics	Primary
3	Ulmer	Carl	Orthopedics	Secondary
...	...	...	...	...

Precalculation of facts

- There are often a ***lot*** of rows in a fact table
  - Easily in excess of a million
- Storing the results of simple computation can make sense at that scale
  - Normally, you wouldn't store the results of computation
- Storing calculations violates the third normal form

Consistency concerns

- When data is put into a denormalized format, outside processes need to enforce consistency

- The process of moving data into and through warehouses is known as ETL:
    - Extract
    - Transform
    - Load

When ETL Fails

- Getting ETL right is a complex process
  - Beyond our course
- But as a consumer of data, you might encounter ETL failures
  - Unlike 3NF, these can be permitted to propagate
- Consider consistency checks and historical trends to look for possible issues

Star Schema "Gotchas"

- Star schema is explicitly designed for ***aggregation***
  - When you are using it, be sure to understand the level of ***granularity*** (what a single row shows)
- Be sure you know which table is fact and which is dimension
  - Hopefully they are labeled! `fact_` and `dim_`
- Be careful when merging (big data)

NoSQL

- RDBMSs are perennially declared to be ***dying / dead / et cetera***
  - Rumors of death are likely overstated

- However, the problems discussed in this unit have been driving innovation

- These new databases are commonly referred to as NoSQL
  - Something of a misnomer

Why NoSQL?

- NoSQL databases are generally used to solve the problems of CAP
- This also often results in problems in ACID compliance
    - Sometimes called BASE systems: **B**asically **A**vailable, **S**oft state, **E**ventual consistency
- Large-scale systems can often make do with this

NoSQL and you

- It is unlikely that you will use or pull data out of NoSQL systems for regular research
- If you work with application or web developers, they might use NoSQL data storage
- NoSQL continues to evolve, so it is worth at least checking once in awhile