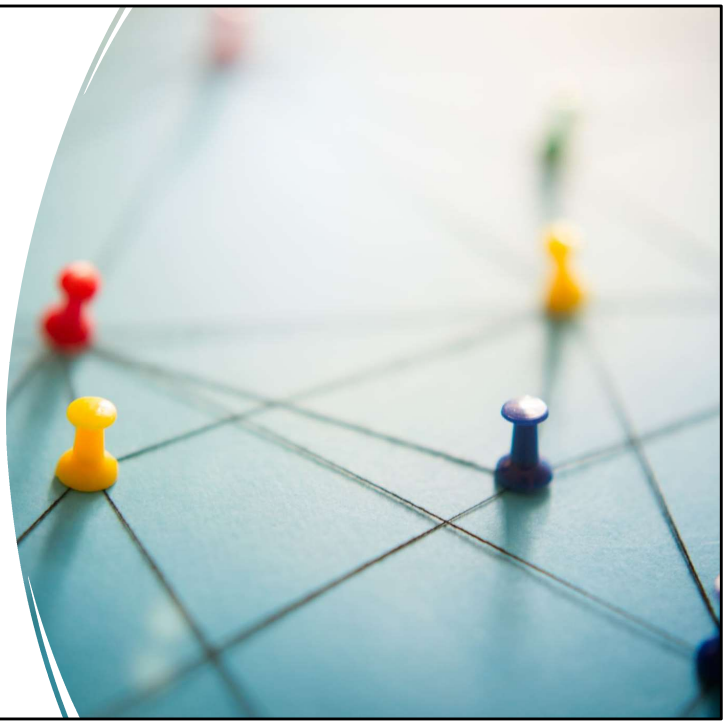
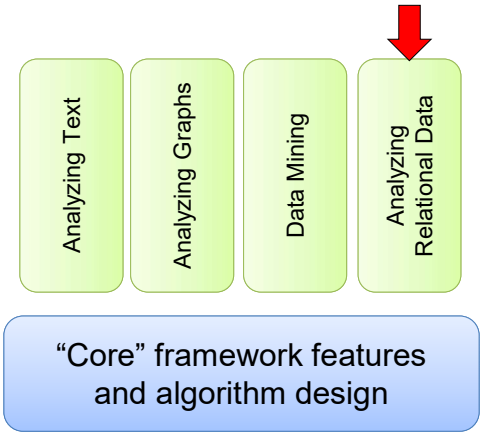
Data-Intensive Distributed Computing CS431/451/631/651

Module 7 – Analysing Relational Data

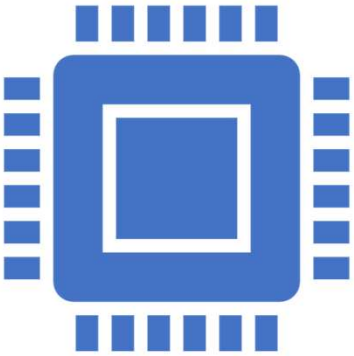
Part 1 (maybe 2) of 4ish



Structure of the Course



In the beginning…



A brief history of software and enterprise architecture

Stage 1, The Early Years

Monolithic Applications

- No external dependencies, nothing
- Just the program

Stage 2, Specialization

Front End 

- User Interface
  - Often means “webpage”

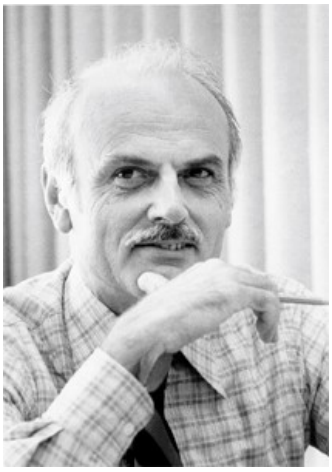
Back End 

- Business Logic
  - Or any kind of logic, really
- “Separation of Concerns”
- Often means “Database”

Just because it’s used to mean that “often” doesn’t make that the definition!
Sometimes confused with “Client-Server” but front and back end splits can happen on the same machine. No need for remote access

Edgar F. Codd

- Inventor of the relational model for DBs
- SQL was created based on his work
- Turing award winner in 1981

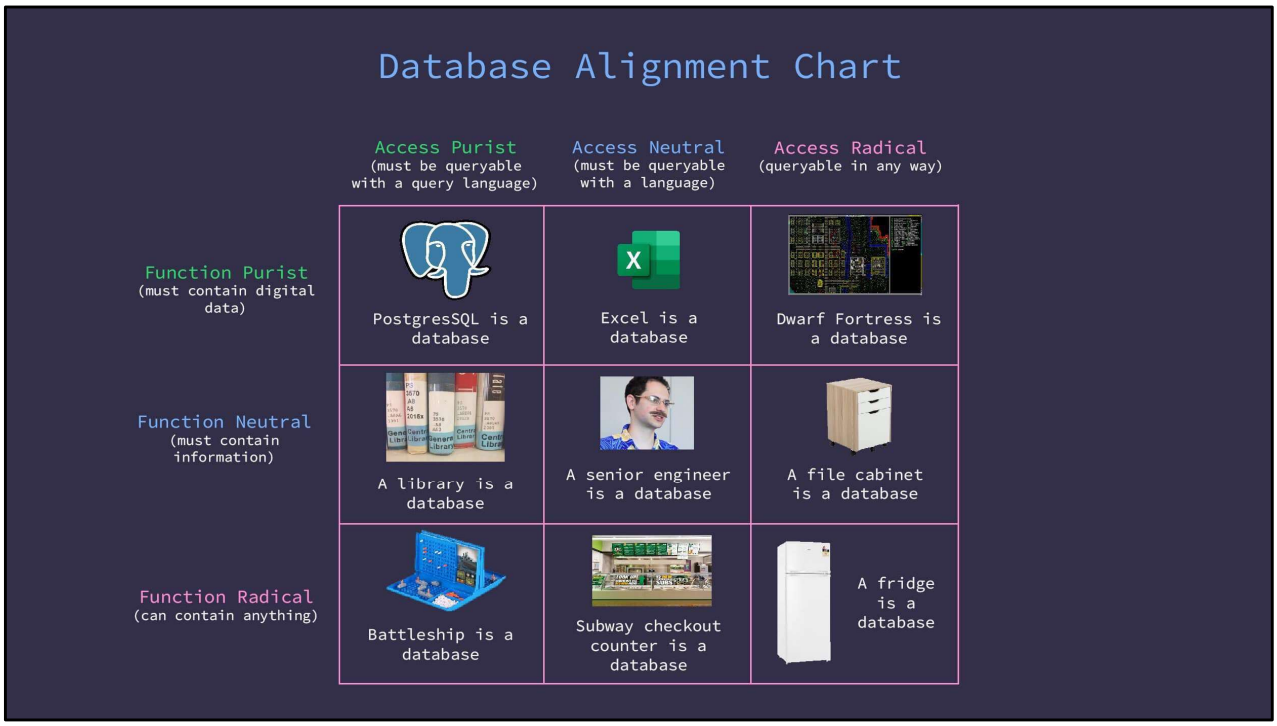


SQL History

- Codd created relational language
- IBM didn't use it, they made their own, SEQUEL
  - Structured English QUEry Language
  - It's also the sequel to SQUARE
- Larry Ellison liked it, and used it on Oracle
- SEQUEL was trademarked, so the query language is SQL – Structured Query Language
  - The SQL standard says that you **must not** pronounce it as "sequel" or you might get sued

SQLite also belongs in the middle tier…
Basically, the top ones are real RDBMS. (Though old MySQL was not great…it did not have ACID compliant transactions, and it ignored foreign key constraints. Since incorporating the InnoDB engine it's been promoted to a real database…and now Oracle owns it and call it HeatWave or whatever…I'll stick to Postgres thanks)

The Postgres Elephant doesn't mean it's connected to Hadoop. "An elephant never forgets" is the reason for both projects using an elephant. Also, they're big, Hadoop is for Big Data.



Excel is not a database.

Dwarf Fortress Steam Release - Dec 6th, also the last day of classes! Coincidence? Yes.

OK, I'm done roasting Excel. For now.

Really this is user error. If you don't want it interpreting fractions as dates, then, uhhh, don't declare the column as "Auto"?? Declare it as numerical.
Sorry…I hate jokes where the punchline is "I'm incompetent lol". It's not hard to set the clock on a microwave, either, damnit.

Although: There's an argument that "Auto" is not a smart default. It's a spreadsheet. Why is not numerical the default?

Reminder: Why Business Wants Data

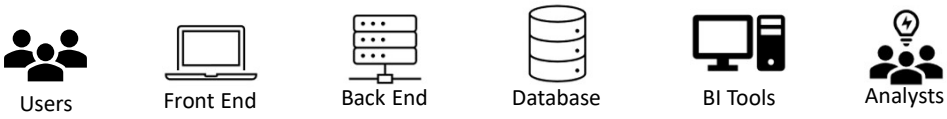
Among other things, “Business Intelligence”

> “An organization should retain data that result from carrying out its mission and exploit those data to generate insights that benefit the organization, for example, market analysis, strategic planning, decision making, etc”

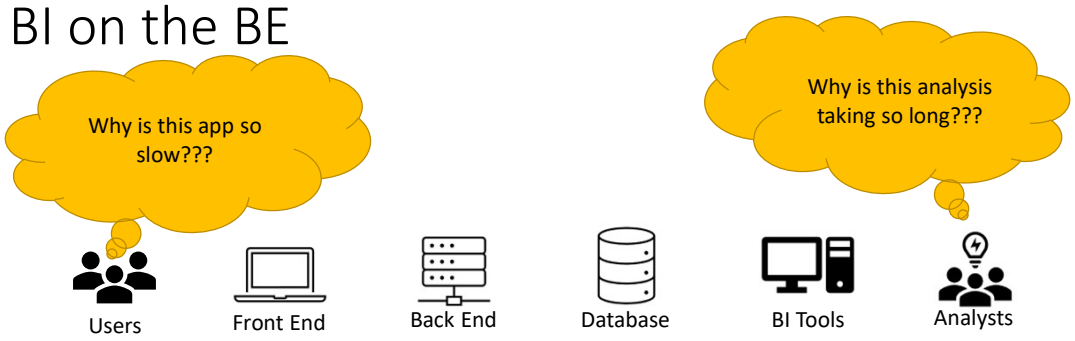
I think this is quoting Jimmy Lin? Because in HIS slides there are not quotation marks, but in Ali’s there are?

BI on the BE

Sorry…"Business Intelligence on the Back End"



BI on the BE


Why is this app so slow???
Why is this analysis taking so long???
Users
Front End
Back End
Database
BI Tools
Analysts

Database Workloads

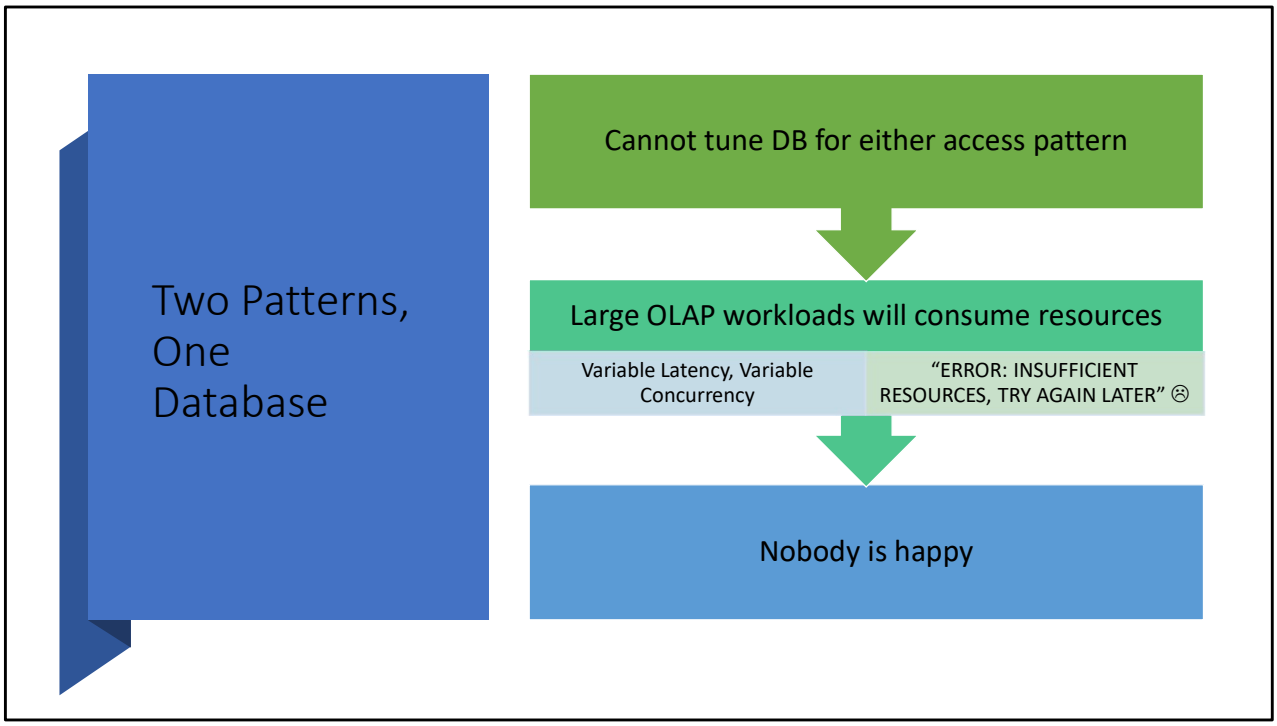
OLTP (Online Transaction Processing)

- Most Applications:
  - E-Commerce, Banking, Reddit, etc.
- User Facing: Must be fast, low latency, concurrent (many users)
- Tasks: small set of common queries
- Access: Random reads, small writes

OLAP (Online Analytical Processing)

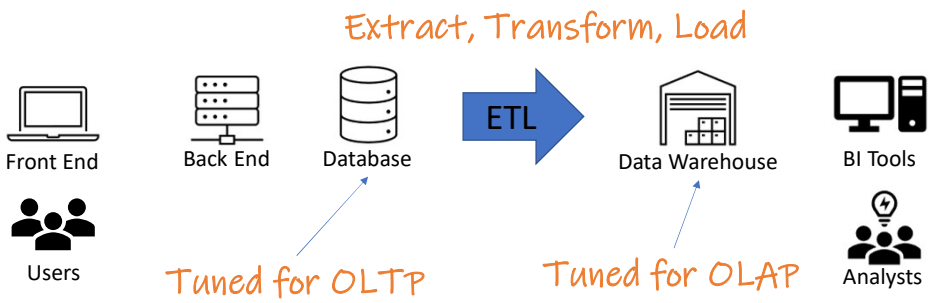
- BI and Data Mining
- Back-End: Batch workloads, low concurrency
- Tasks: Complex Analytics (Ad Hoc)
- Access: Full Table Scans, Big Data

These are another of Codd's contributions


Two Patterns, One Database
Cannot tune DB for either access pattern
Large OLAP workloads will consume resources
Variable Latency, Variable Concurrency
“ERROR: INSUFFICIENT RESOURCES, TRY AGAIN LATER” ☹
Nobody is happy

Solution: Data Warehouse!

Data Warehousing



The data warehouse is also a database. Tuned for mass storage, and OLAP queries (typically full table scans, rarely the same query twice)

Extract – Pull data from database
Transform – Put it into a different schema that's more suited for OLAP / BI / Datamining
Load – Put it into the data warehouse.

The ETL process is also called "Data ingestion" sometimes. Gross.

ETL

- Extract – Self Explanatory
- Transform
  - Clean and validate data
  - Schema Conversion
  - Field Transformation?
- Load – Self Explanatory

You don't want bad data slipping in

I'm frankly embarrassed it's taken so long for a Data reference!

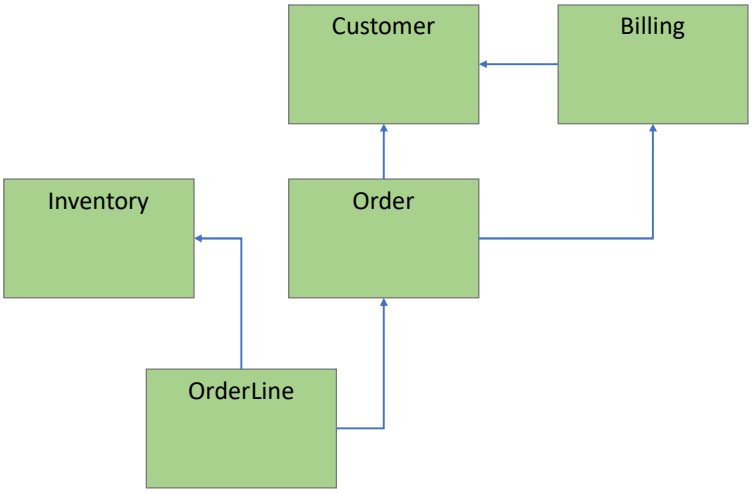
If the OLTP database has constraints, how can bad data slip in?
Well, the Analysists might have different constraints, different definitions of "validated" and "clean".
Plus, the OLTP might not do much validation, in the name of speed.

ETL – When to perform

- You might be able to do a rolling ETL where updates from the OLTP database are immediately sent to the warehouse
- You might also just have a batch transfer that happens on a daily basis
  - Since the BI queries are slow, stale data is OK

Typical OLTP Schema

- Captures relation between items



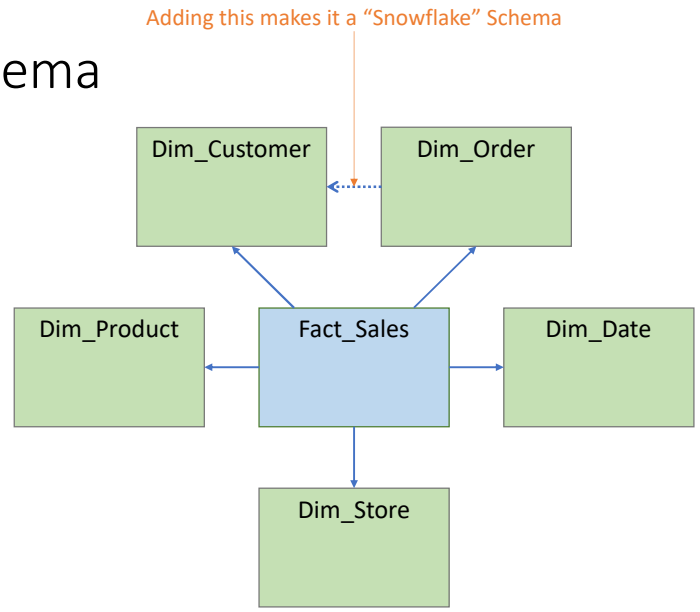
Arrow Direction:

Foreign Key points to Primary Key

Typical OLAP Schema

"Star" Schema

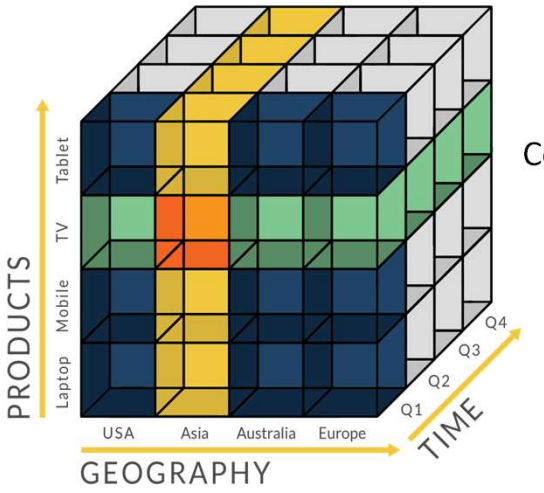
- Center – Fact Table
- Points – Dimension Tables



Fact_Sales is the primary table. It contains all elements that do not relate to the other rows.
Any values that DO relate / correlate with other rows will "dimension" attributes – foreign keys to a dimension table.

In this example – When you sell a Widget, that goes into Fact_Sales.
Things that are unique to the sale will go in this table. (E.g. the number of items sold)
The fact that it was a widget will be a foreign key into the product Dimension table.
Which store it was bought at will be a foreign key into the store Dimension table.
Same with which customer, which order this sale was a part of, what day it was sold on. It's all up there, on the slide.

OLAP (Hyper) Cubes



Common operations

- slice and dice
- roll up/drill down
- pivot

Many OLAP queries are about getting chunks (whether an individual cell, a row/column, a plane, etc) of an n-dimensional hypercube!

Why Star?

BI and Data Mining queries are *ad hoc*

The schema cannot be designed around *ad hoc* queries

Why?

Sorry, was that not obvious? You don't know what they are yet!

"There are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are unknown unknowns – the ones we don't know we don't know…" – Donald Rumsfeld

At the time (Iraq War) this quote got made fun of a lot. He was absolutely ROASTED. Mostly by Stewart and Colbert. Actually no, it was near universal. Out of context, it's a pretty good quote though. It's true! It's also pretty ancient wisdom

He was roasted because this was an answer to the question "What evidence do you have that Iraq is supplying WMD to terrorists?"
That's not an answer at all. If it means anything, it means "none whatsoever". Anyway…it's relevant because "known unknowns" and "unknown unknowns" is relevant…here, let's see if we can't use a better quote…

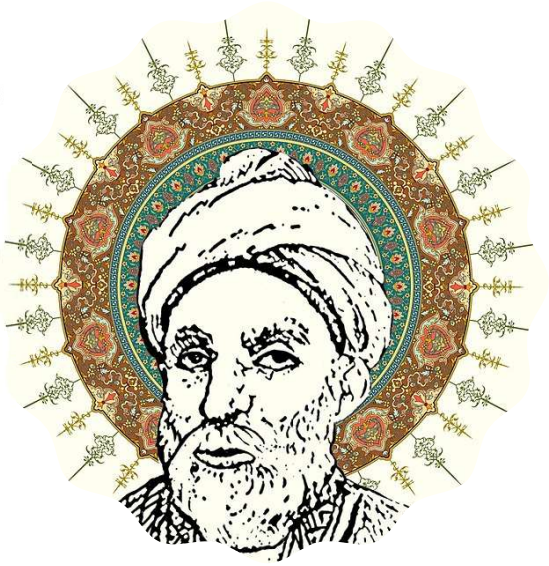
Also, like, this slide goes back to Jimmy's first offering of this course. I dunno, it feels rude to remove it.

One who knows and knows that he knows
His horse of wisdom will reach the skies

One who doesn't know, but knows that he doesn't know
His limping mule will eventually get him home

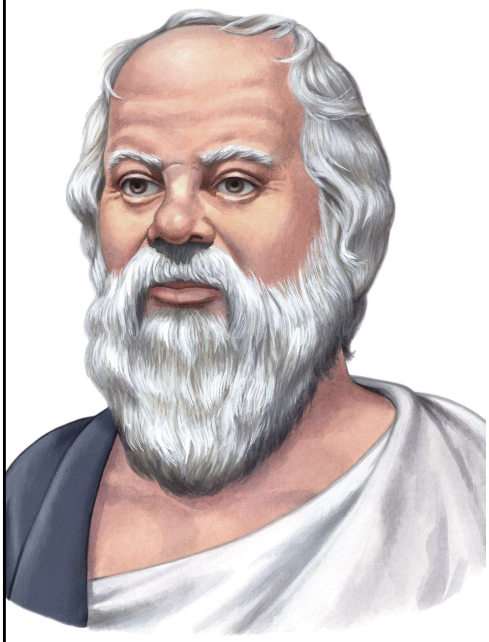
One who doesn't know and doesn't know that he doesn't know
He will be eternally lost in his hopeless ignorance!

Ibn Yamin (1286-1368)



Ahh, that's better, Persian Poet Ibn Yamin had basically the same thing to say.

This slide goes back to Ali's time running the course.



"I know only one thing: that I know nothing"

-Socrates

And this is MY addition.
If we switch instructors too many times this lecture is going to be all philosophy…this isn't as relevant, maybe I messed up trying to be cool...

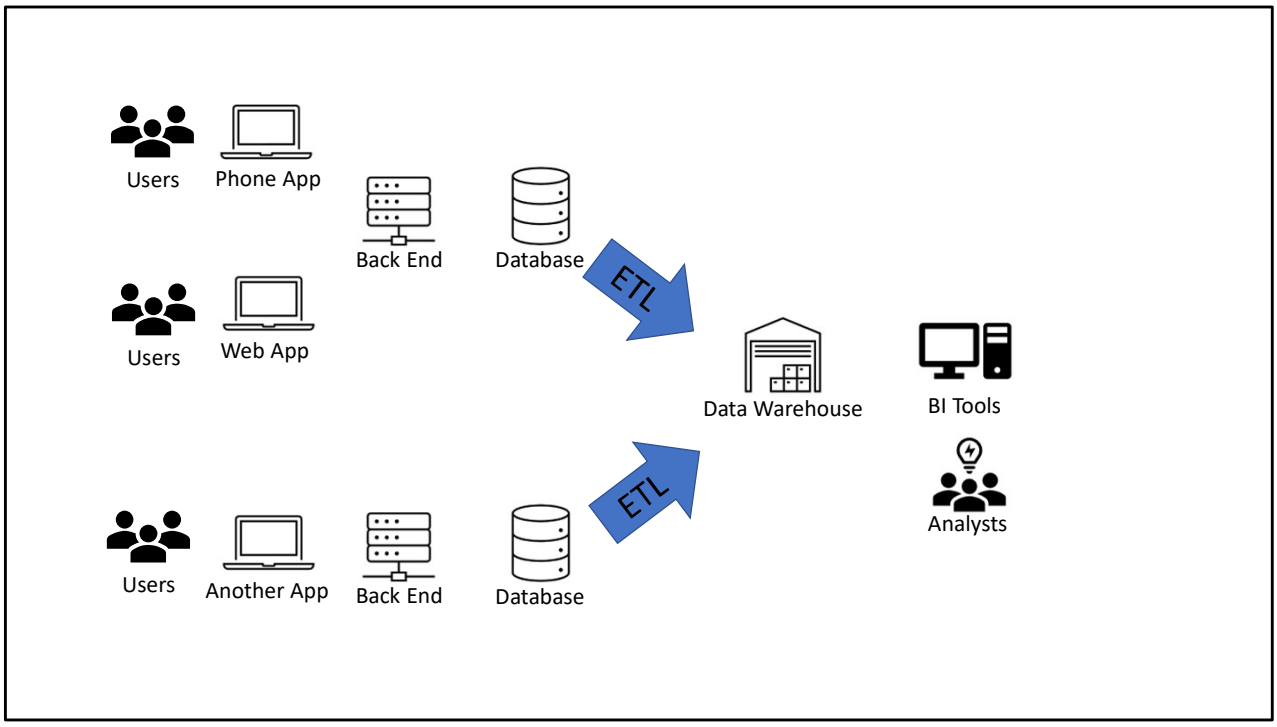
I think there's still a good takeaway though.
Accept that some things you will not know. You can't plan for the unknown, so plan to be flexible.

OK, very philosophical but…why a star?

It supports common OLAP queries efficiently. That's all.

- Known Known – Something you already know, don't need a query
- Known Unknown – Something you don't know, but know you want to know. You can write the query now
- Unknown Unknown – Something you don't even know you want to know. You cannot write the query

Alright, slide count 4 for the simple idea "You cannot tune a database for a query that doesn't yet exist"
You can, however, tune it for a fairly broad class of queries
The ol' slice'n'dice / pivot table presented a few slides back with the hypercube!

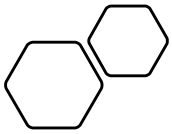


The data warehouse can pull from multiple backend databases. That's why it's a warehouse. Fill it to capacity!

Problem: Data generation rate exceeds data ingestion rate. Or "ACID" reflux.

(It's a database joke. A good database requires ACID = "atomic, consistent, isolated, durable" . I think this is in and of itself also a pun? Acid is the opposite of base)

TL;DR?



- Facebook was generating a LOT of data
- Oracle could not keep up
  - Oracle said “Build a big cluster and then pay us $+inf.0 and it’ll scale”

I may have rounded the dollar amount up a bit, but not by a lot.

What Changed?

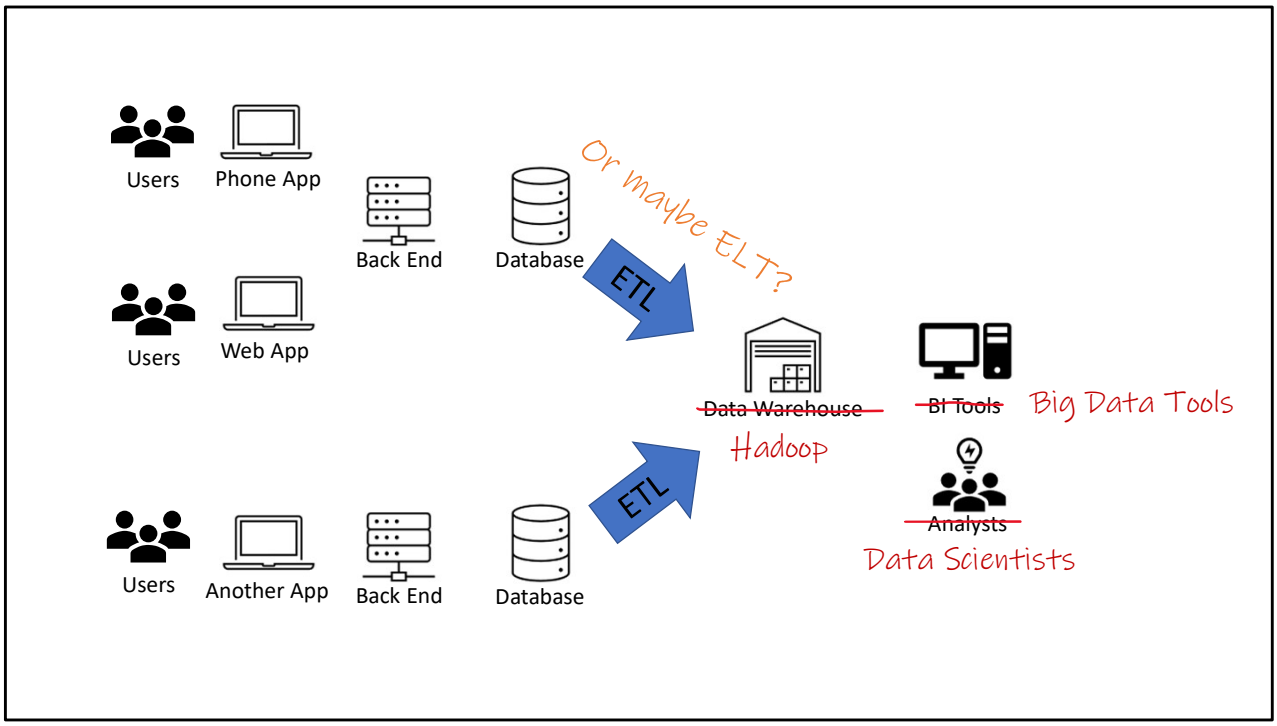
We covered this in Lecture 1!

- Disks are cheap now – Cheap to just save everything and worry about importance later
- Data Mining – something that doesn't appear valuable might be
  - Businesses now see the value of analytics
- Social Media – People are generating their own data now

What queries does Facebook do?

OLTP	OLAP
Update Profile	Feed Rankings (“The Algorithm”)
Add Friend	Friend recommendations
Like, comment	Demographic Analysis
…	Engagement (“clickstream”)
(Basically everything a user can do)	(All the non-social-media BI stuff too)

Some of the OLAP stuff ends up user facing…in a way. Finding friends for you is a long and difficult process (BURN) but runs in the background. Every so often (whenever the batch finishes) the front end can be updated with new suggestions.



What if it wasn't a giant Oracle database, but Hadoop? What if indeed.

A bunch of analysts would be out a job, for one thing. Unless…

ELT? Extract, Load (onto HDFS), Transform (with a MapReduce job to clean the data up, etc.)

(Not an) Actual Conversation Between Oracle and Facebook, c. 2009 (colourized)

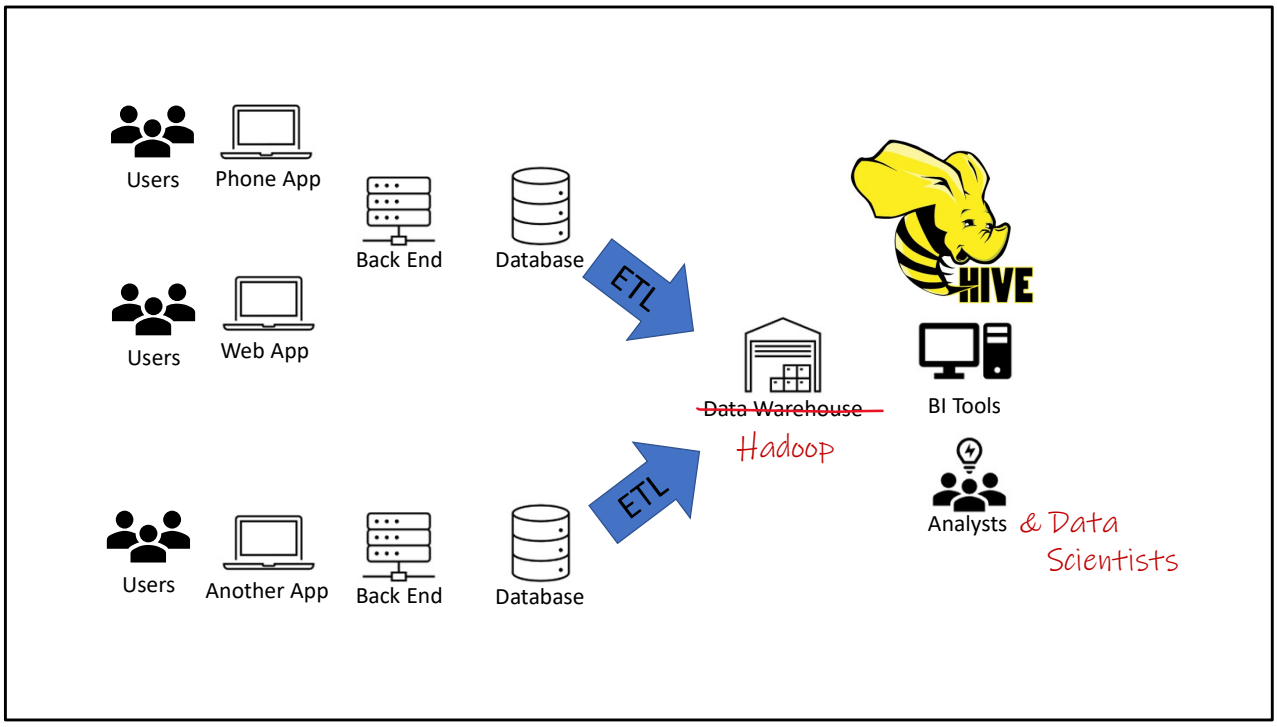
"You can't use Hadoop as a data warehouse! Your analysists don't know Java!"

"But they know SQL, so we'll just run SQL on Hadoop"

"That will never work! Huge mistake!"

Oracle now owns Java so you're still using an Oracle product…technically.

Very conveniently, Oracle's colour is red, and Facebook's is blue. Easy contrasting colours!



Why not just use a database, though?

Databases Are Great…Unless They Aren't

- Mature Software
- SQL is powerful

But…

- Databases do not scale well
- Oracle and IBM want a LOT of money. SO MUCH MONEY.

Microsoft does too, but are an "also ran" in the database world.
Walmart does all their cloud and database stuff through Microsoft so that they don't need to pay Amazon, a competitor, for AWS. Or Oracle. They're not a competitor, but nobody likes paying Oracle.

Databases are great…

- If your data has structure
  - And you know that structure
- Your data are clean
- You know what queries you'll run ahead of time
  - The Star Schema / Snowflake Schema isn't a silver bullet here

Databases are not great…

- If your data has little structure
  - Or you don't know the structure
- If your data has a lot of noise / mess
- If you're going data mining / ML
  - (You don't know what you're looking for)

Again, Known Unknowns vs Unknown Unknowns

What does a BI Analyst Do?

Generate Reports

Create Monitoring Dashboards

Which of these are
Known Unknowns?
Unknown Unknowns?

Ad hoc analyses

- Descriptive – extract a description of the data
- Predictive – extract a model of the data that will predict future data

Advantages of Hadoop Dataflow over RDBMS

No Schemas

“Most” OLAP is full-table scans.

Flexible – Can write imperative code

HDFS is designed for rapid ingestion

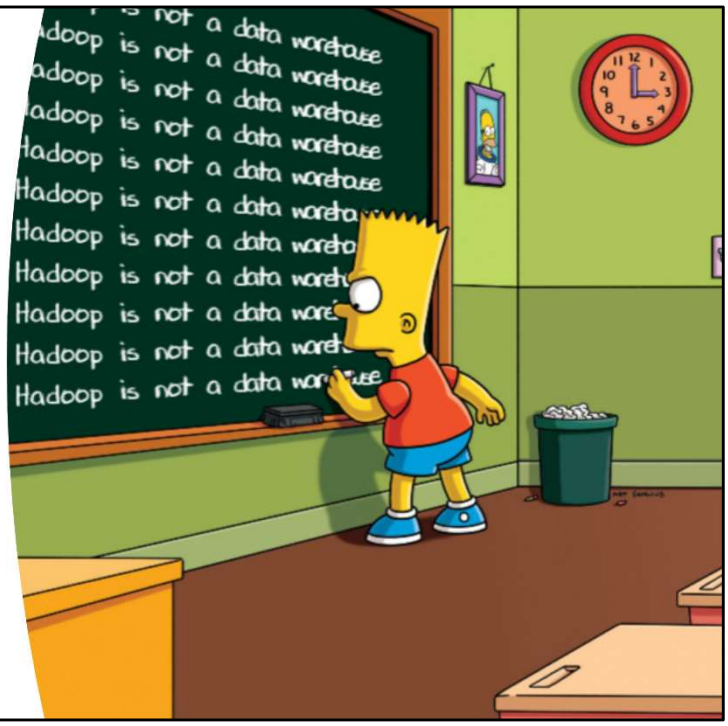
No Schemas, No Kings, No Lords! Freedom!

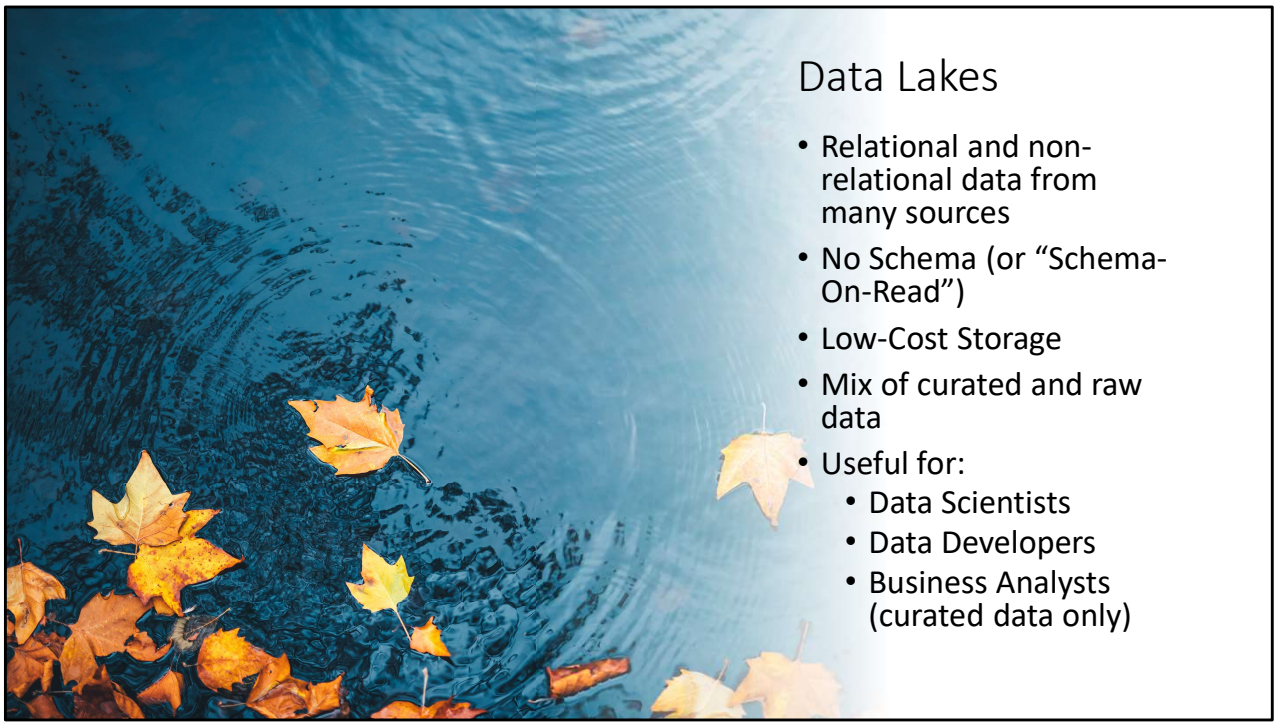
Ingestion again…

Database people insist: Hadoop, even with Hive, is not a data warehouse!

What is it then?

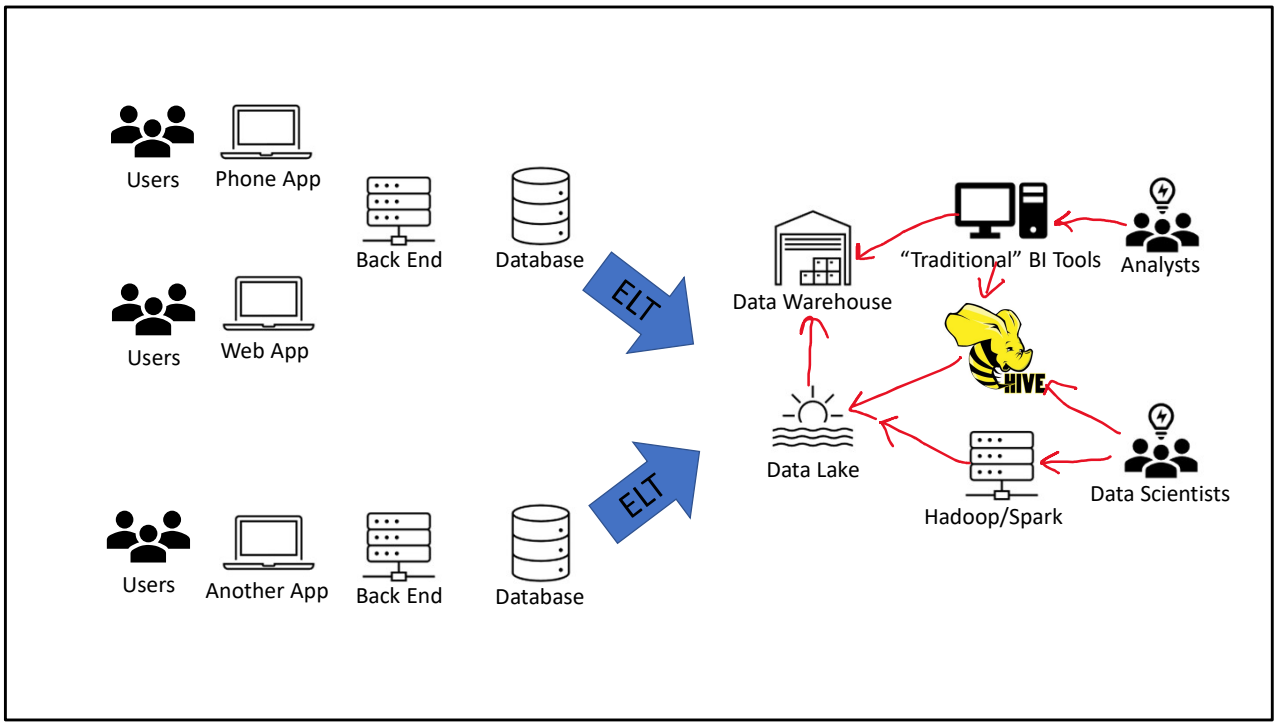
A data lake! Apparently.





Data Lakes

- Relational and non-relational data from many sources
- No Schema (or “Schema-On-Read”)
- Low-Cost Storage
- Mix of curated and raw data
- Useful for:
  - Data Scientists
  - Data Developers
  - Business Analysts (curated data only)



This is getting complicated!

So, Analysts really want curated data. They can use HIVE, since it's just SQL like their other tools. But still prefer a data warehouse approach! So you need both? Maybe. This setup is sometimes called a "data lakehouse". Again, apparently.

If the data warehouse is small and dedicated to one team of analysts for a particular aspect of the business, it's called a datamart.
So you can have your datalake getting ELT'd, then to another TL to send some clean data into various department's datamart. This is giving me a headache.

Also: Spark has SparkSQL, so you can write SQL with Spark instead of using Hive (which generates MapReduce tasks, with all the benefits and drawbacks that entails)

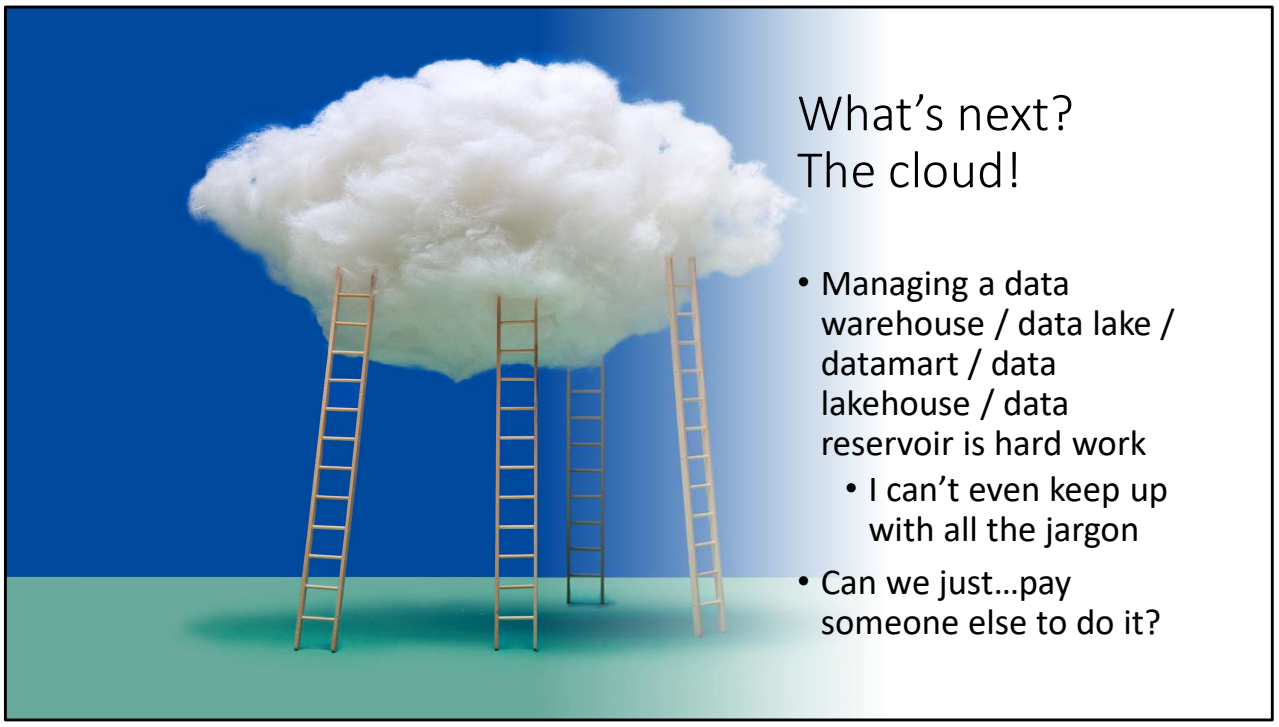
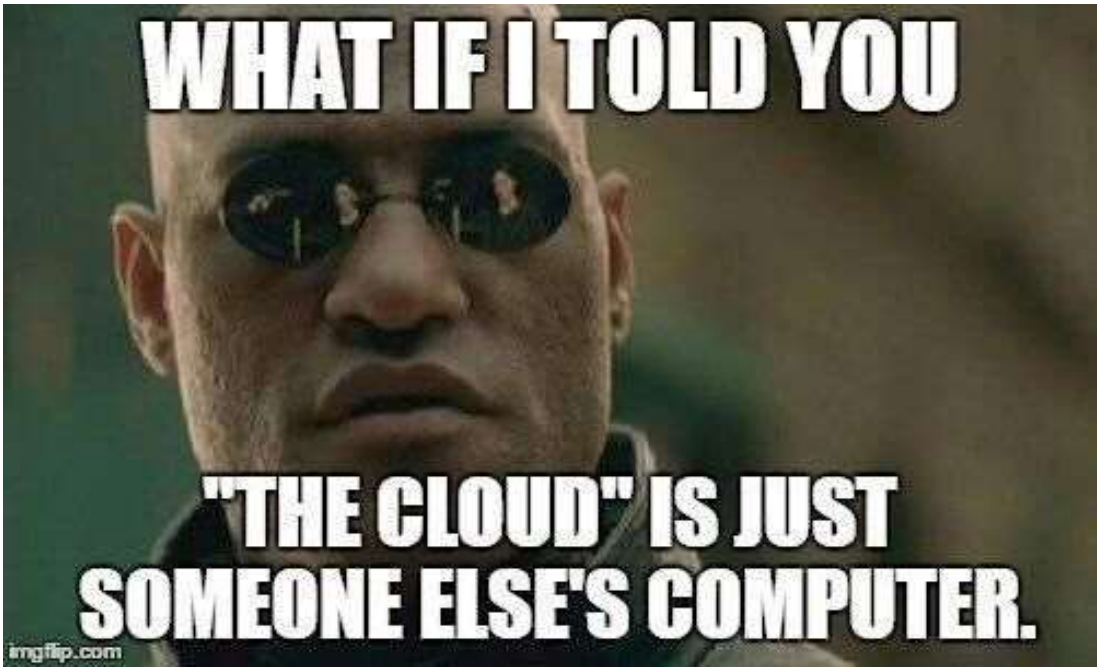
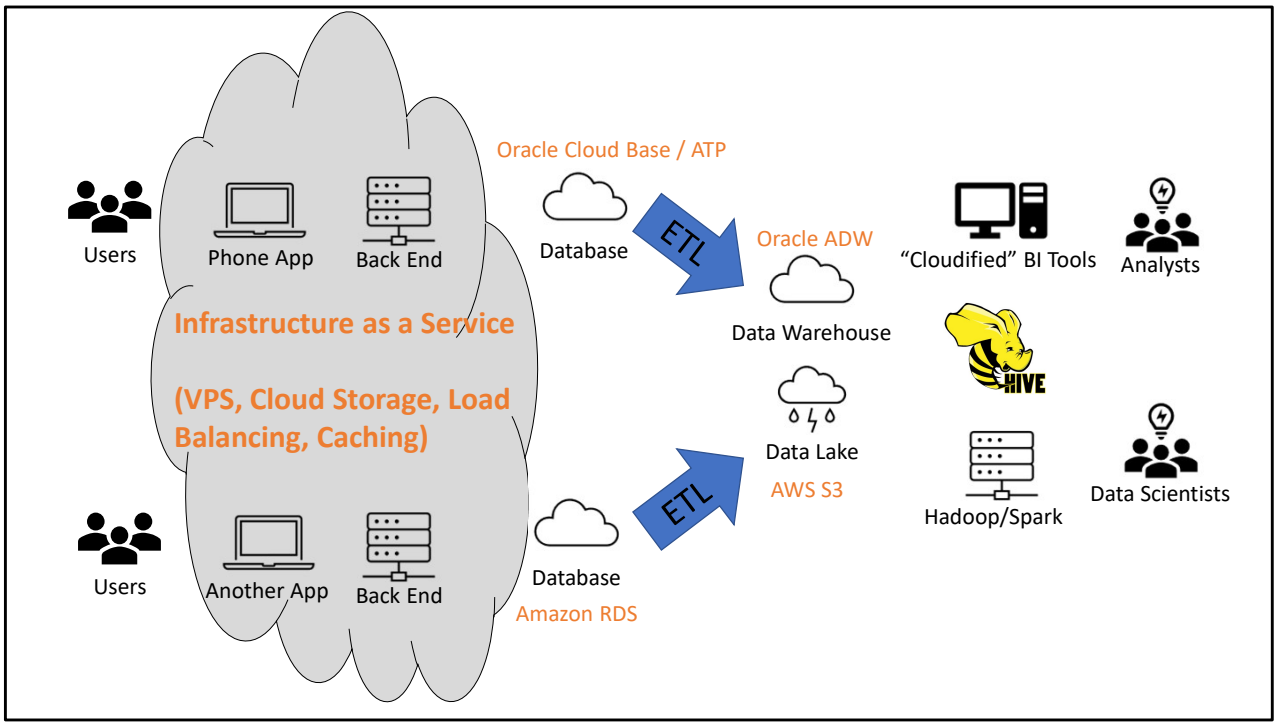


Image – Migrating to the cloud

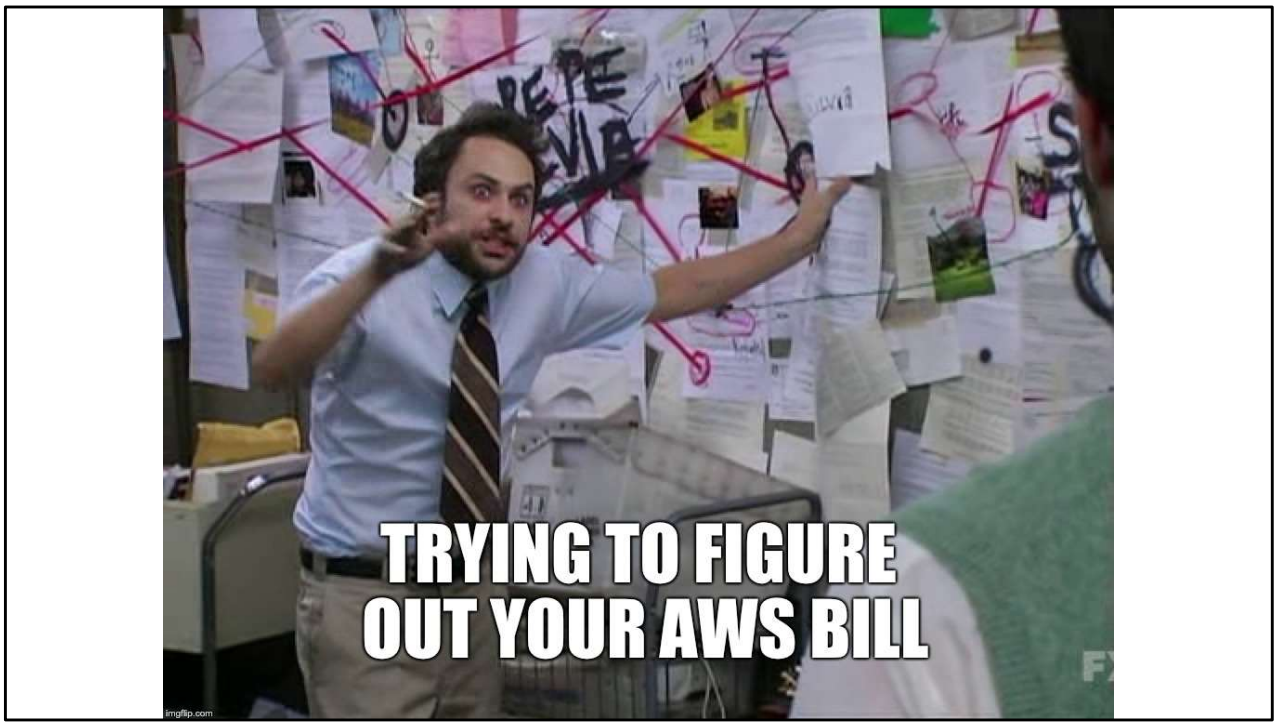
In class I said I didn't make any of those jargon terms up. I actually did make up "data lakehouse" but then googled it and it's also an industry term.



"Put it on the cloud" means "rent a server and pay someone to maintain it"



I'm getting a lot of milage out of this one slide!



The cloud isn't free.

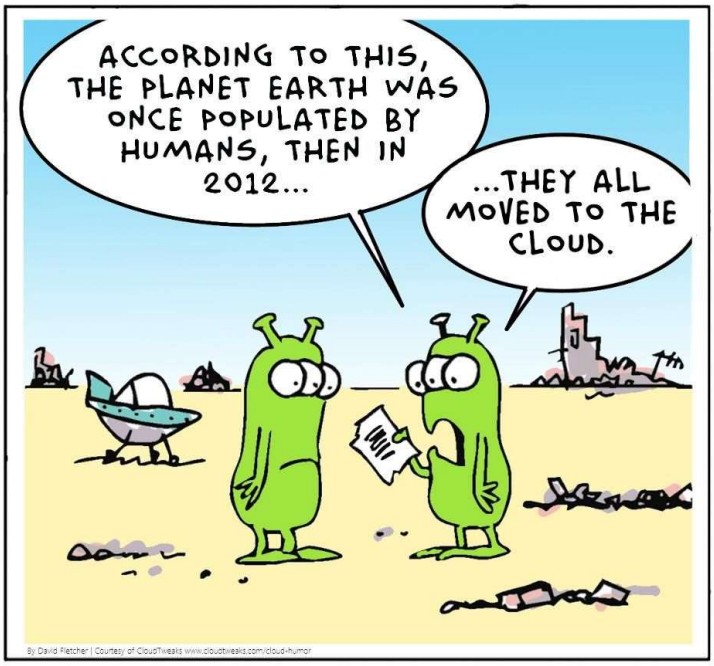
OK, well, my Oracle Cloud VPS is free. They're like drug dealers, they want to get you hooked. Still, 4 ARM cores and 24GB RAM? Score. This isn't an ad. If it was, I'd tell you my referrer ID.

Topical: This is Elon Musk right now trying to figure out why Twitter's AWS bill is so high

What's left?

All we need is to put those Analysists and Data Scientists in the Cloud and the migration will be complete!

Please do not try to upload yourself to the cloud, the tech is not there yet.



So you don't have to zoom – Courtesy of CloudTweaks: www.cloudtweaks.com/cloud-humor

How all this works…

SQL query interface

Execution Layer

HDFS | Other Data Sources

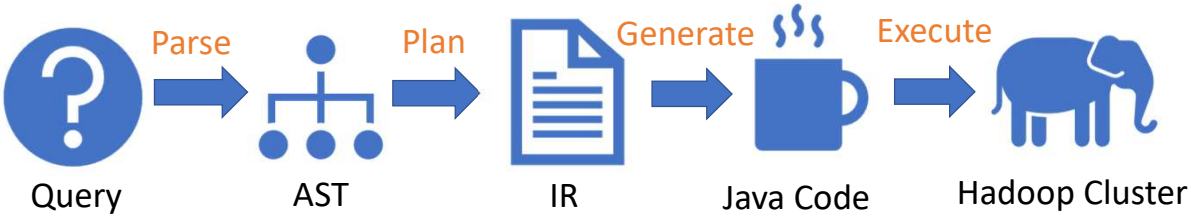
Hive Example!

```
SELECT a.word, a.freq, b.freq FROM
    "shakespeare_wc" a JOIN
    "bible_wc" b ON (a.word = b.word)
    WHERE (a.freq >= 1) AND
          (b.freq >= 1)
    ORDER BY a.freq DESC
    LIMIT 10;
```

the	25848	62394
I	23031	8854
and	19671	38985
to	18038	13526
of	16700	34654
a	14170	8057
you	12702	2720
my	11297	4135
in	10797	12445
is	8882	6884

What's this doing? "What are the 10 most frequent words in Shakespeare's sonnets that also occur in the bible, and what are their counts in both the sonnets and the bible?"

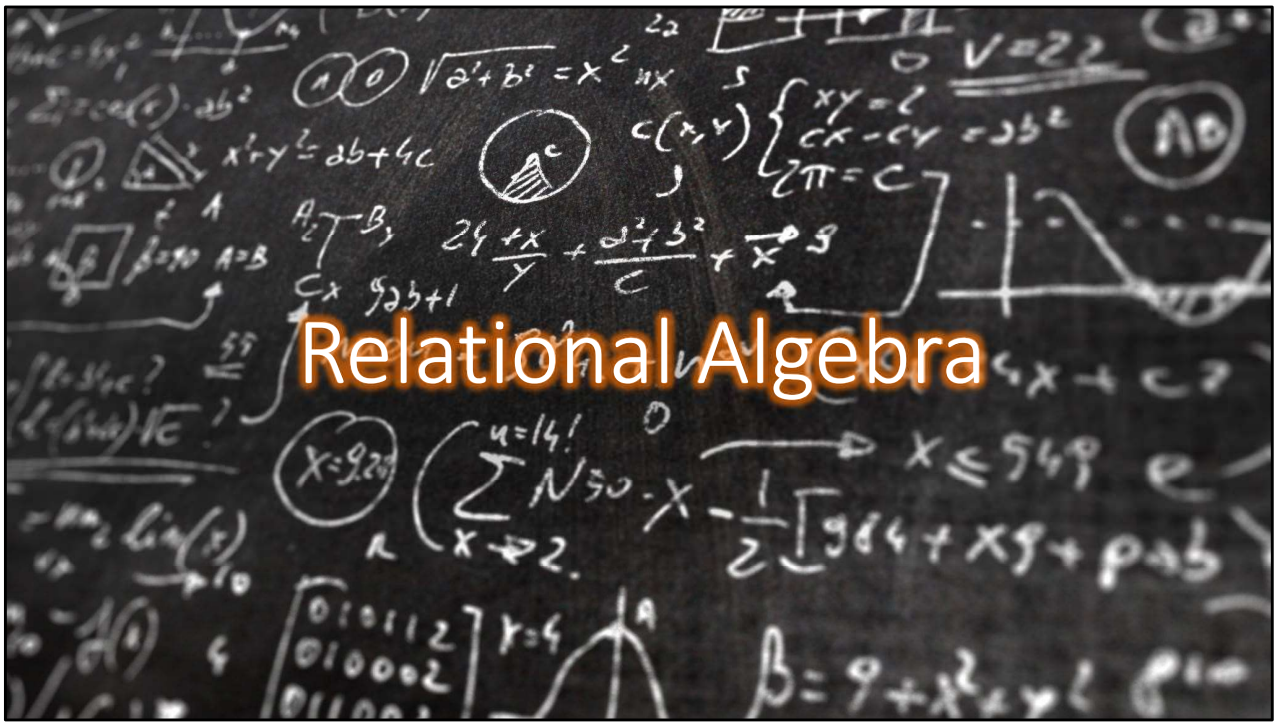
Behind the Scenes



PRO-TIP: If you remove all the labels, this slide is complete nonsense!

Step 1 – Parse the query -- create an abstract syntax tree
Step 2 – Plan the job – decide how to most efficiently run the query, create an intermediate representation
Step 3 – Code generation – Emit Java code equivalent to the IR
Step 4 – Execution -- Compile Code and Submit to Cluster

An RDBMS is going to do the same steps 1 and 2. It's just that instead of then turning the plan into java code, it just executes is directly


Relational Algebra

Relational Algebra

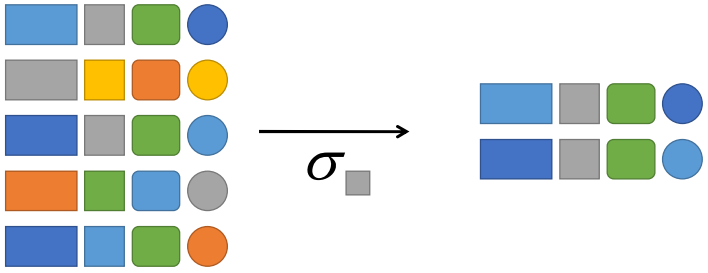
Primitives

- Projection (π)
- Selection (σ)
- Cartesian product ($\times$)
- Set union ($\cup$)
- Set difference ($-$)
- Rename (ρ)

Other Operations

- Join ($\bowtie$)
- Group by… aggregation
- …

Selection



Select all rows with a gray square.

This is straight up just the “SELECT … WHERE” from SQL

How to SELECT in MapReduce

Easy!

Map-side filter

No reducer task

Performance?

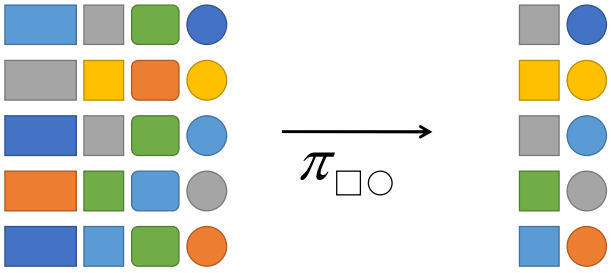
As fast as HDFS can load the tuples?

Not quite – also must parse them. Text parsing is slow.

-- Format Matters

Any operators that are map-side only can be pipelined!

Projection



This is the other half of the “SELECT (col1, col2, …) FROM” SQL syntax

“Project each row into lower dimensional space, keeping only the squares and circles”

How to PROJECT in MapReduce

Easy!

Map-side tuple transformation

No reducer task

Performance?

Same as with selection

Fiddly Details

Need to remember column mappings.

E.g. column 4 is now column 1 in the projection

Any operators that are map-side only can be pipelined!

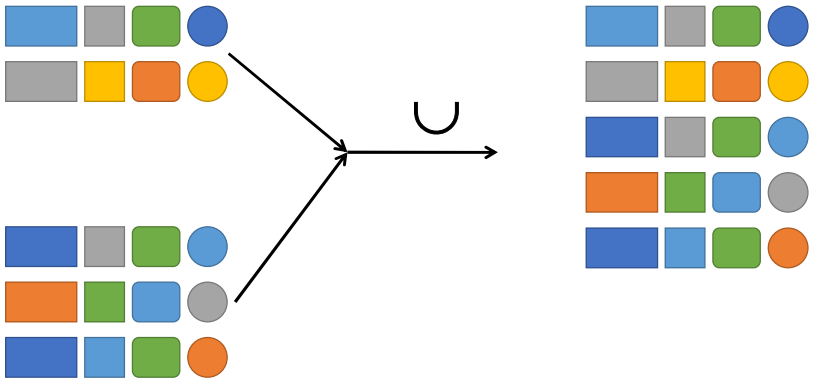
Rename (ρ)

SELECT (userID u) FROM users;

Renaming userID as u doesn't matter at the MapReduce level

- MapReduce just sees tuples
- It doesn't care what column 1 is called
  - 1 is 1

Union



This is also part of SQL. I don't recall ever using it, but it's there!

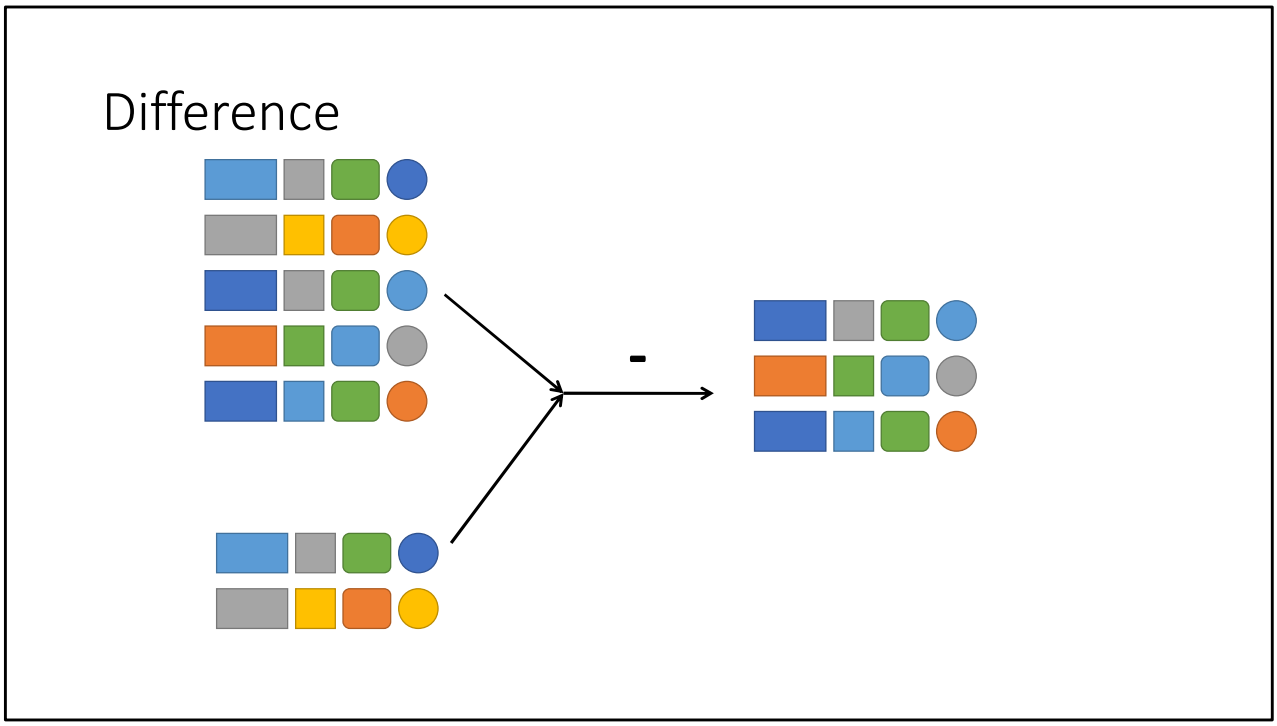
(SELECT ….) UNION (SELECT ….)

How to UNION in MapReduce

Hadoop MapReduce has a MultipleInputFile class
- It's LITERALLY UNION

1 Mapper class per input file
- Can be pipelined with other map-side operations

Difference



This is also part of SQL. I

(SELECT ….) MINUS (SELECT ….)

I did not know this…it's not common…to me.

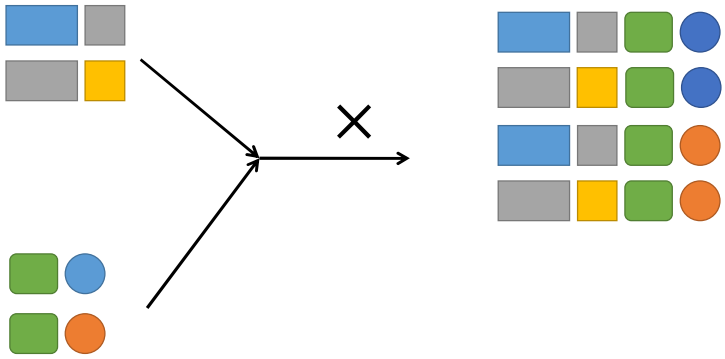
How to SUBTRACT in MapReduce

Cannot be done on the map side alone

- MultipleInputFiles again – Each mapper has:
  - Key: An Entire Tuple, plus "Which Mapper Sent Me"
  - Value: Not used
  - Sort "RHS" tuples before equal "LHS" tuples
- Reducer
  - Remember last RHS tuple
  - Emit LHS tuples only if they do not equal the last RHS

This is a modification of merge from merge sort. Neat!

Cartesian Product



This is also part of a SELECT statement. It's not actually a CROSS operator, just write

SELECT table1.thing, table2.thing FROM table1, table2

SQL calls this a type of join (cross join)

How to cross in MapReduce

- Don't

I wasn't going to!

- Good, don't

Wow…that family guy episode aired in 1999…were you folk even born then? Does anybody get it? Anybody? Bueller?

Anyway, when I say don't, I mean…BIG x BIG = HUUUUUUUUUUUUGE

In Hadoop, 1 million records is a small dataset.
And yet, a self-join will result in a trillion rows. Have mercy!

Cross Product is an operation of last resort. Only use if it literally no other approach will work

How to cross in MapReduce

If you MUST

- Job setup looks at the splits (partitions) and performs a cross
  - File1 = HDFS blocks [1], [2], [3]
  - File2 = HDFS blocks [a], [b], [c]
  - Cross = [1a] [1b] [1c] [2a] [2b] [2c] [3a] [3b] [3c]
- Custom Reader gets given two files
  - Reader given [3] and [a] will:
    - Read record 1 from [3], cross it with all records of [a]
    - Repeat
  - Mapper does nothing (Can be pipelined)

The Mapper does nothing because the custom Reader class is doing all of the crossing, so the Mapper just reads tuples and emits them again.
Really really hopefully you're pipelining with a Select to filter tuples.

This is expensive, but something that's in both PIG and HIVE.

(Spark RDDs also have a .cartesian operator, the same rule applies: Don't)

How to AGGREGATE in MapReduce

The Reduce in MapReduce is often called the "Aggregation Phase"

Every SQL aggregation function is done on the reduce side

We've done most of them on the assignments.

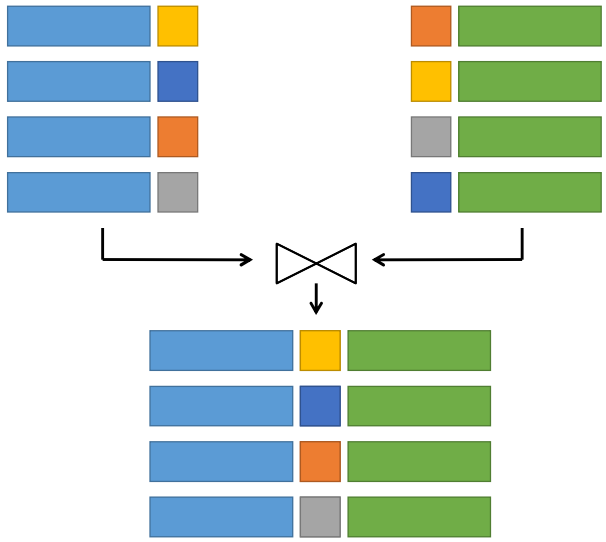
COUNT, SUM – Frequencies

MIN, MAX – OK, but we could have! (431 uses MAX for PageRank convergence)

AVG – That was an example in the slides already

This stock photo goes all the way back to the first offering of the course. A piece of history.

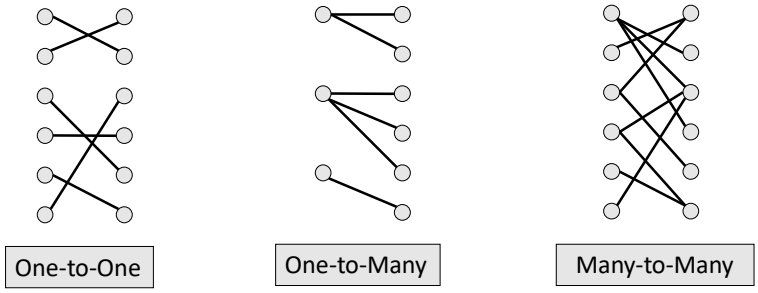
(Inner) Join



Efficient joins is one of the harder parts of query / schema design

Someone used to “database = a bunch of values” find this part the hardest to grok

Types of Relationships



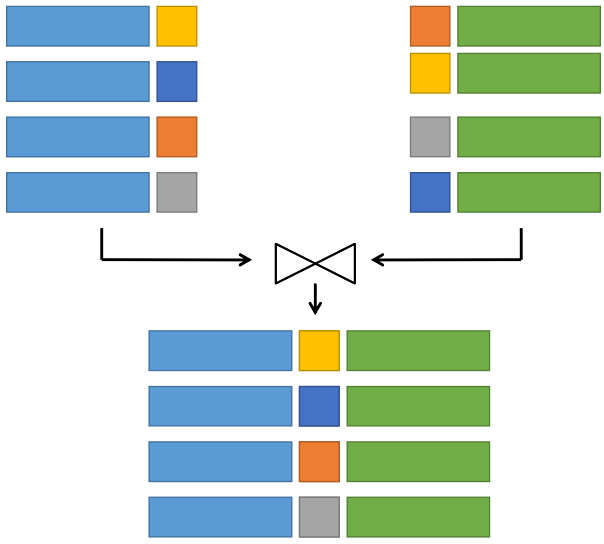
One-to-One | One-to-Many | Many-to-Many

One-to-One Joins

- The easiest there is!

No crossing needed, memory requirements are minimal

Basically modified merge sort



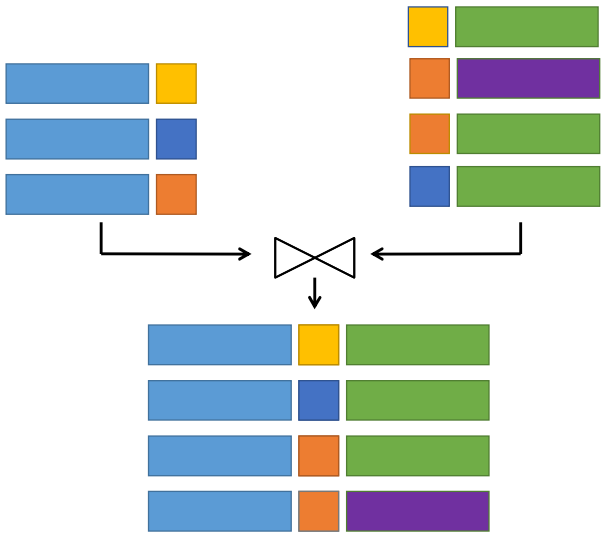
One-to-Many Joins

Regardless of exact technique:

For the “One” side –

- Hold Tuple in memory
- Cross with all Tuples from the “Many” side with matching join key

One pass, minimal memory usage



Many-to-Many Joins

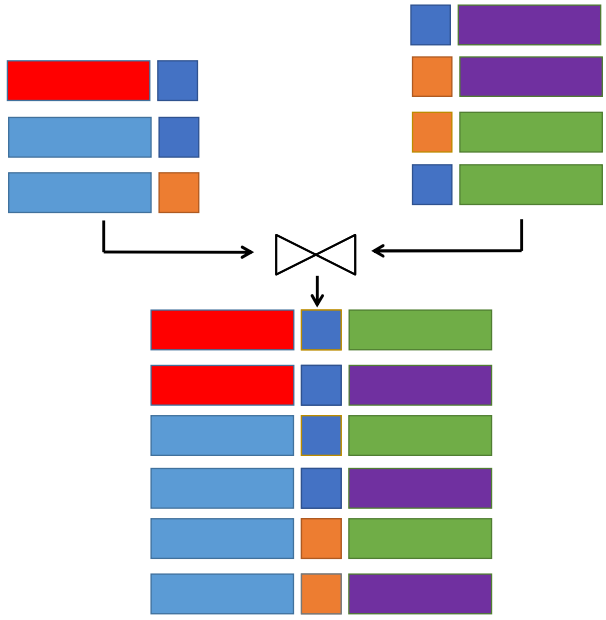
- The worst

Need to hold ALL Tuples with a given key in memory

- For both sets

Cross one set against the other

- Ugh



It's actually only bad when both sets are large and the many is also large

How to (inner) Join on MapReduce

- There are 3 options!
  - Hash Join (aka Broadcast Join, Side-Loaded Join, or Replicated Join)
  - Map-Side Join (like the map-side cross…a bit)
  - Reduce-Side Join (aka Shuffle Join or Repartition Join)

This is also ordered by “goodness”

Use a Hash Join if you can
Use a Map-Side join if you can
You always have Reduce-Side join, it can always be done

Hash Join

If one set is so small it easily fits into memory of a single node then:

- Load it into every mapper as a hash table
- Each Mapper joins its split against the hash table
- Map-Side only, can be pipelined
- Fast
- **IF**

You know the story of the "if" reply, right?
So the legend goes:

King Philip II of Macedon had conquered most of the Greek city states, and sent Sparta a message
"Should I come to Sparta as a friend or a foe?" (asking them to surrender, in other words)

The Spartan reply was one word. "No"

This angered Philip, who said "If I take Laconia, I shall turn you all out" (banish them)

The Spartan reply was "If"

(So Philip did invade, devastate Laconia, and do what he promised, eject the Spartans. Still, emotionally speaking Philip was equally devastated)

A witty one-word reply is called a "Laconic response". The fact that being Laconic lead to bad outcomes is also something important to remember.
Don't taunt people, basically.

“What are you going to do, invade me?”
-- State invaded

Map Side Join

Remember what we did for a cross join? Sure ya do.

This is a bad idea. Unless…

What if both files are sorted by key and split by the same partitioner?

Oh, in that case – the mappers can join row by row

Map Side Join

Use it if:

- Both tables (or subqueries) are sorted and partitioned by the join key
- When is this reasonable to expect?

Reduce-Side Join

Mappers – MultipleInputFiles class

Basic Idea:

Each mapper emits its stuff, key = join key

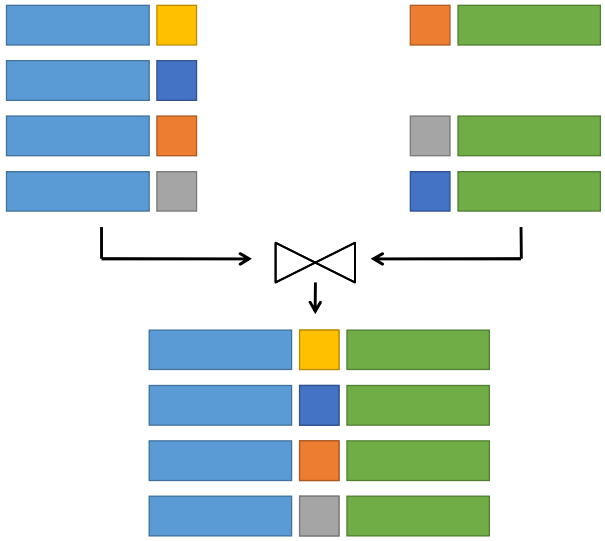
Reducer gets keys in sorted order, easy to join them together

OR IS IT?

Reduce-Side, 1-to-1 join

Reducer always gets 1 key per subquery. Join is straightforward

Need to tag each tuple with its source, but the sort order doesn't matter.



Reduce-Side, 1-to-many join

Oh, I know this!

Reducer holds the tuple from the 1-side in memory, then joins with all Of the many-side tuples with the same key

Wait: How did the one side arrive first???

Remember: Secondary Sorting Pattern

Join Key -> Map Reduce Key ✗

(Join Key, Origin) -> Map Reduce Key ✓

Just make sure the partitioner only cares about the join key, and the sort order puts the “one” side first

Aside: Tertiary Sorting

If your query is like:

SELECT a.key, a.thing, b.other from a JOIN b ON key ORDER BY a.key, a.thing

If you're doing a reduce-side join anyway…might as well have the keys by

(a.key, origin, a.thing_or_nothing)

Now the values arrive sorted by the sort column, too!

Of course if you're JUST sorting by a.thing this won't work. Sorting by key then by thing is not the same!

This might not be a very common situations. Still, it can help you to avoid an extra pass in some cases, even if they're rare.

Reduce Side, Many-to-Many

Well, now you have to hold ALL tuples with one key from one set, and cross against the other set.

☹

That's life.

Many-to-Many joins are expensive for large values of "many" because the result is large.

Outer Joins?

The logic is the same, except you now have to handle the empty case instead of skipping the unpaired tuple.


Oh, yeah. It's all coming together.

Putting Everything Together

```
SELECT big1.fx, big2.fy, small.fz
FROM big1
JOIN big2 ON big1.id1 = big2.id1
JOIN small ON big1.id2 = small.id2
WHERE big1.fx = 2015 AND
    big2.f1 < 40 AND
    big2.f2 > 2;
```

Build logical plan

Optimize logical plan

Select physical plan

Note: generic SQL-on-Hadoop implementation; not exactly what Hive does, but pretty close.

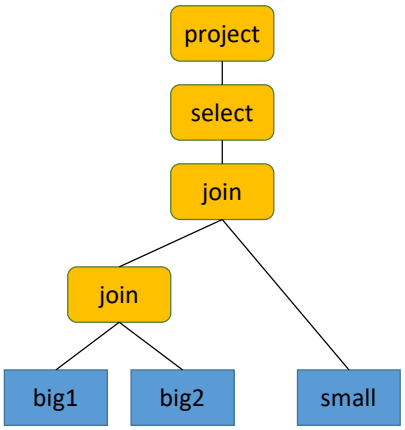
Putting Everything Together

```
SELECT big1.fx, big2.fy, small.fz
FROM big1
JOIN big2 ON big1.id1 = big2.id1
JOIN small ON big1.id2 = small.id2
WHERE big1.fx = 2015 AND
    big2.f1 < 40 AND
    big2.f2 > 2;
```

Build logical plan

Optimize logical plan

Select physical plan



87

Right: The concrete syntax tree, aka the “logical plan”

Big1 and Big2 are joined, and this is then joined with small. It is then filtered (select operator) based on big1.fx, big2.f1, big2.f2, and finally, projected (limited to 3 columns: fx, fy, fz)

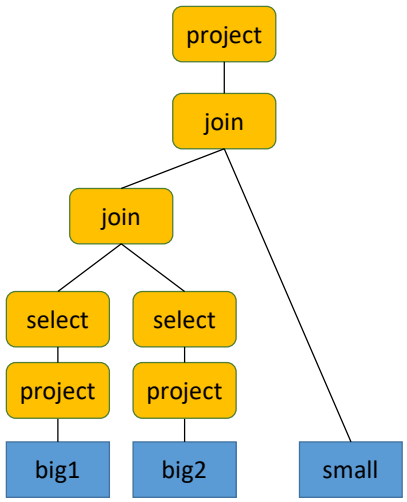
Putting Everything Together

```
SELECT big1.fx, big2.fy, small.fz
FROM big1
JOIN big2 ON big1.id1 = big2.id1
JOIN small ON big1.id2 = small.id2
WHERE big1.fx = 2015 AND
    big2.f1 < 40 AND
    big2.f2 > 2;
```

Build logical plan

Optimize logical plan

Select physical plan



88

Optimizations:

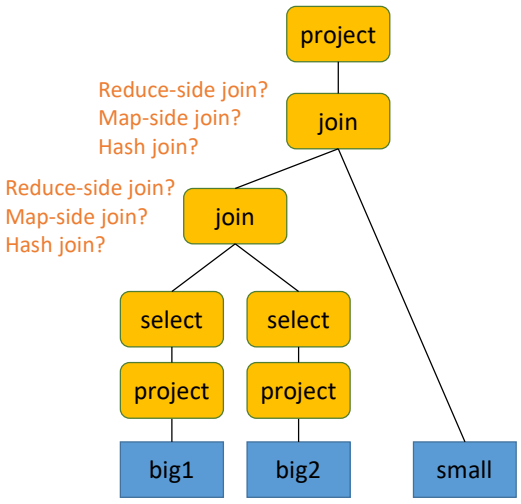
Big1 should be projected down to: fx, id1, id2. fx because its needed for the final projection, id1 and id2 for the joins
Big2 should be projected down to: fy, id1, f1, f2. fy because it's needed for the final projection, id1 for the join, f1 and f2 for the "WHERE" selection

Big1 and Big2 should then be filtered by the "WHERE" clauses.

Then, there are fewer tuples to join. We no longer need a post-join filter as none of the "WHERE" clauses involve comparisons that are only valid on the joined tables. The final projection is still needed to remove the columns used as join keys and select criteria

Putting Everything Together

```
SELECT big1.fx, big2.fy, small.fz
FROM big1
JOIN big2 ON big1.id1 = big2.id1
JOIN small ON big1.id2 = small.id2
WHERE big1.fx = 2015 AND
    big2.f1 < 40 AND
    big2.f2 > 2;
```



Build logical plan

Optimize logical plan

Select physical plan

89

There are 3 types of joins that can be used.
Assumption: small is small, so a hash join is suitable.

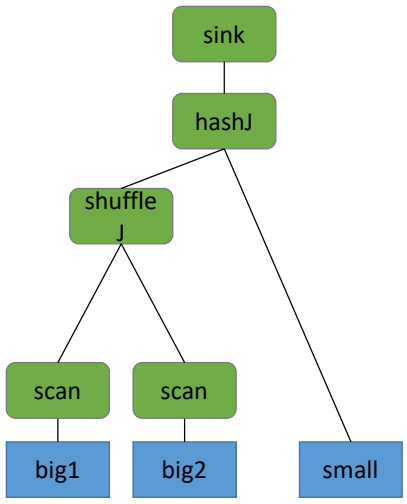
Putting Everything Together

```
SELECT big1.fx, big2.fy, small.fz
FROM big1
JOIN big2 ON big1.id1 = big2.id1
JOIN small ON big1.id2 = small.id2
WHERE big1.fx = 2015 AND
    big2.f1 < 40 AND
    big2.f2 > 2;
```

Build logical plan

Optimize logical plan

Select physical plan



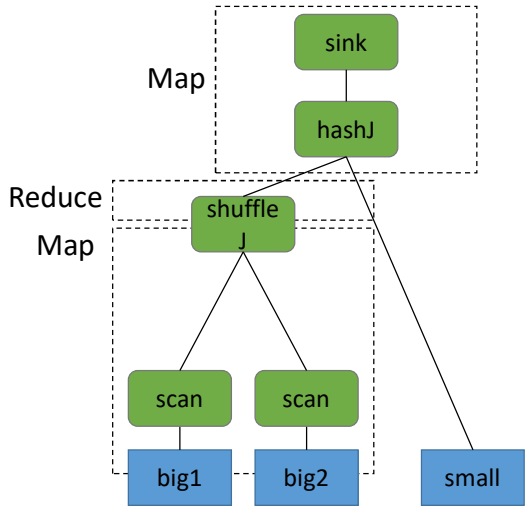
Big1 JOIN Big2 is a reduce-side (or "shuffle") join. They're big so hash is not suitable. We can't assume they're co-partitioned so map-side is also not suitable.

Scan means "a full table scan" – this pipelines both the selection and projection.

"Sink" means "write the output" – this is pipelined with the projection

Putting Everything Together

```
SELECT big1.fx, big2.fy, small.fz
FROM big1
JOIN big2 ON big1.id1 = big2.id1
JOIN small ON big1.id2 = small.id2
WHERE big1.fx = 2015 AND
    big2.f1 < 40 AND
    big2.f2 > 2;
```



Build logical plan

Optimize logical plan

Select physical plan

91

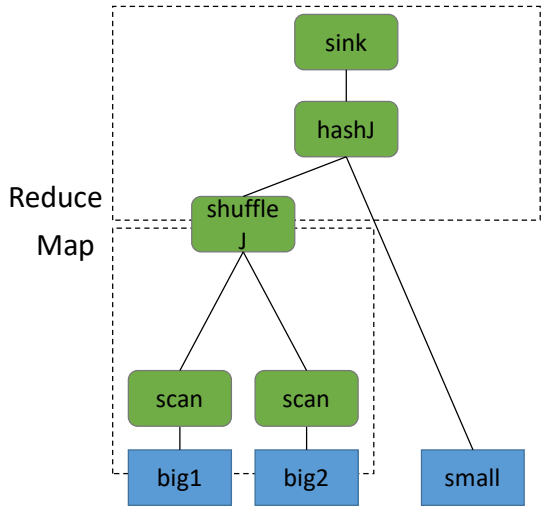
Two MapReduce Jobs.

Job1 – Mapper does the selection + projection scan, and keys each tuple with (id1, source).
Reducer does the join, and writes to an intermediate file.

Job2 – Loads intermediate file. Mapper does the hash join with small1, and writes the final output (no reducer phase)

Putting Everything Together

```
SELECT big1.fx, big2.fy, small.fz
FROM big1
JOIN big2 ON big1.id1 = big2.id1
JOIN small ON big1.id2 = small.id2
WHERE big1.fx = 2015 AND
    big2.f1 < 40 AND
    big2.f2 > 2;
```



Build logical plan

Optimize logical plan

Select physical plan

92

That previous plan SUCKS.
A hash join can be done on either side. Any of the "map-side" operators can also be pipelined into the reduce phase, too.

Just one job.

Mapper: selection and projection on big1, big2, keys are (id1, source). (How source is tagged will depend on the arity of the relationship)

Reducer: three-way join. Do the cross to join based on id1 key, then cross these tuples with the small hash table). Then do the final column projection and write the results to HDFS.

Schema?

How does it know the format of the tables? How does it know the column names?

Metadata

Somewhere on HDFS is a file that says “big1 means this HDFS file with this schema”

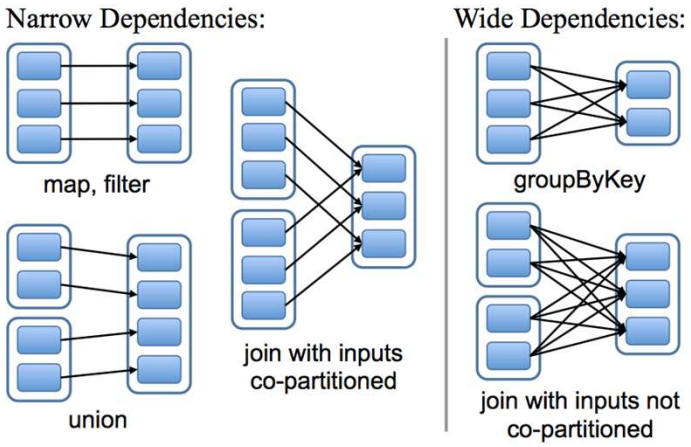
Spark?

Spark does the planning already, and writing Spark is pretty close to writing imperative queries:

Most Relational operators are RDD transformations:
Select / Project: (Base RDD or .filter) / .map
Join : .join
Union : .union
Cartesian Product: .cartesian
Count: .count
Sum : .sum , .reduce(_+_)

THAT'S A BUTT

Spark SQL: Physical Execution



Spark's join will do map-side or reduce-side automatically, depending on whether it's a narrow or a wide dependency.

If you want a hash join, use a broadcast variable (i.e. what you did in A2)

Or… Spark SQL

Spark DataFrames are like Tuple RDDs but with named columns
DataFrame API operators are named like the SQL operators

You can use these directly, or use sqlContext.sql(queryString)

Spark SQL can read your HIVE metadata for easy migration!

Two Options

SELECT dept.name, count(dept.name) FROM employees JOIN dept ON (employees.did = dept.id) WHERE employees.title = “intern”
GROUP BY dept.id

```
employees.join(dept, employees(“did”) === dept(“id”)).
        where(employees(“title”) === “intern”).
        groupBy(dept(“id”), dept(“name”)).agg(count(dept(“name”)))
```

What on Earth is all that noise?

In Scala at least, a dataframe, when indexed, returns a column. This, like an RDD or DataFrame, is “lazy”

column === column returns a selector function that matches column-to-column for equality.
Count(column)

Spark Dataframes

A Dataframe is like a Tuple RDD except:

- Different operators
- Each element of the tuple is called a "column"
- Each column has a name!

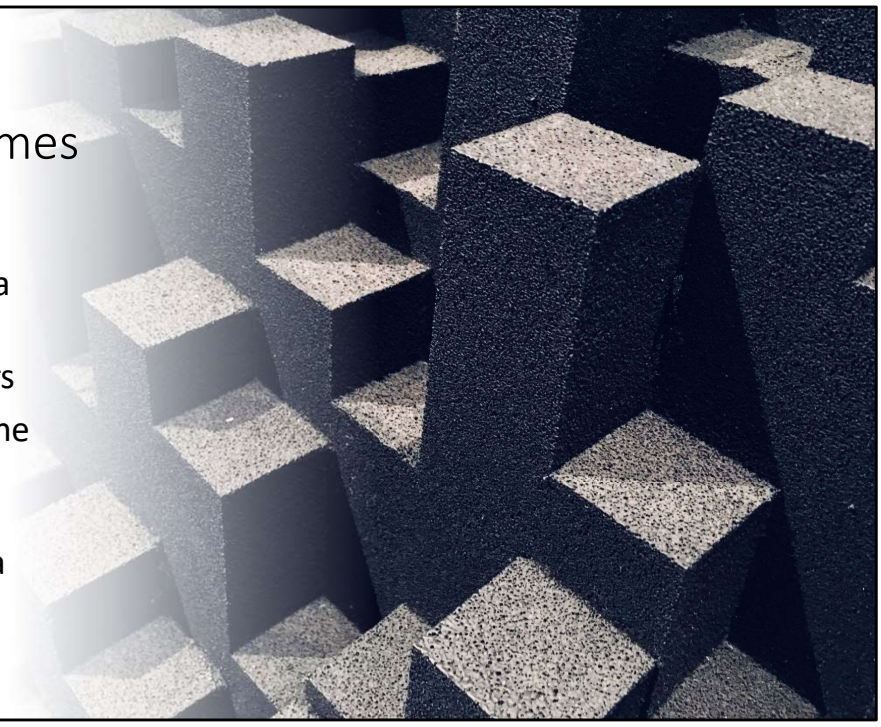


Image: Microsoft Stock Art. "Columns"

Creating a Dataframe (451 Edition)

```
val myRDD =
sc.textFile("marks.csv").map(_.split).
  map(attrs =>(attr[0].toInt, attr[1],
               attr[2].toFloat))

val myDF = myRDD.toDF("studentid", "component",
                      "mark")
```

This will infer the schema based on the data types in the RDD, and will set all columns to allow nulls.
If you don't give names to the columns, they'll default to _1, _2, etc.

Creating a Dataframe (431 Edition)

```
myRDD = sc.textFile("marks.csv").map(lambda
line: line.split()).
  map(lambda attrs:(int(attr[0]), attr[1],
                    float(attr[2])))

myDF = myRDD.toDF("studentid", "component",
                  "mark")
```

This will infer the schema based on the data types in the RDD, and will set all columns to allow nulls.
If you don't give names to the columns, they'll default to _1, _2, etc.

Specify a Schema

```
val schema = StructType(Array(
        StructField("username", StringType, false), // not nullable
        StructField("title", StringType, false),
        StructField("salary", IntType, true)))

schema = StructType([\
        StructField('username', StringType(), False),\
        StructField('title', StringType(), False),\
        StructField('salary', IntType(), True)])
```

Top is Scala, bottom is Python

You can pass this in to the toDF function instead of just passing in the column names. Now it knows the Schema.

Loading Directly

```
peopleDF =
spark.read.format("json").load("people.json")

Or

people2 =
spark.read.format(“csv”)
.option(“inferSchema”, “true”)
.option(“header”, “true”).load(“people.csv”)
```

(Other than the missing val, this is both Python and Scala)

“spark” here is a “SparkSession” object. It’s called “spark” by default in spark-shell / pyspark. You create it in a similar way.

The JSON file should contain an array of objects. Their attributes are used as the columns. E.g. you’d have
[{“name” : “Bob”, “department” : “IT”, …}, …]

JSON is typed (to an extent) so the types should all be correct. If you want control over nullability you should still provide a schema

You can chain .options if you want to change settings. In the CSV example:
inferScheme means “guess the column type”
Header means “the first row contains the column names).

How to Query a Dataset

- You first register it

```
people.createOrReplaceTempView
("people")
```

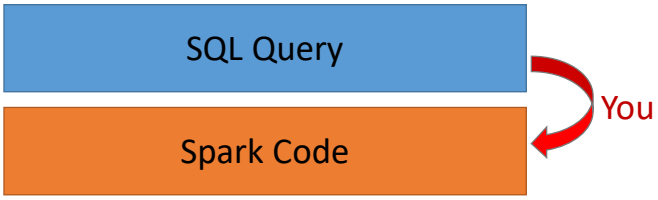
- Then you query it

```
teens = spark.sql("SELECT
    name, age FROM people
    WHERE age BETWEEN 13 AND
    19")
```

Spark can also have tables registered permanently, either through it's API or using an existing HIVE install.

So we're doing this on the SQL Assignment?

- Yes, though what "this" means depends on which course
  - CS451: "this" means "translating a query to Spark RDD transformations"



  - CS431: "this" means "Using Spark Dataframes"

MapReduce: A Major Step Backwards?

MapReduce is a step backward in database access

- Schemas are good
- Separation of the schema from the application is good
- High-level access languages are good

MapReduce is poor implementation

- Brute force and only brute force (no indexes, for example)

MapReduce is not novel

MapReduce is missing features

- Bulk loader, indexing, updates, transactions…

MapReduce is incompatible with DBMS tools

Source: Blog post by DeWitt and Stonebraker

See the forum row in the assigned readings ;)

Benchmarking Hadoop vs Vertica

- Vertica is an Analytical Database Management System
- Designed for Big Data and Clustering
- Fast
- Stonebraker (you might recognize the name from the blog post)

Hadoop vs. Databases: Grep

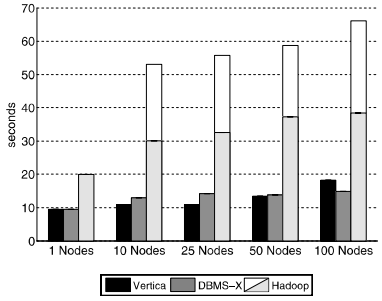


Figure 4: Grep Task Results – 535MB/node Data Set

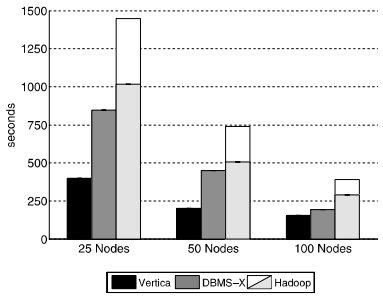


Figure 5: Grep Task Results – 1TB/cluster Data Set

SELECT * FROM Data WHERE field LIKE '%XYZ%';

107

Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

The upper segments of each Hadoop bar in the graphs represent the execution time of the additional MR job to combine the output into a single file.

Hadoop vs. Databases: Select

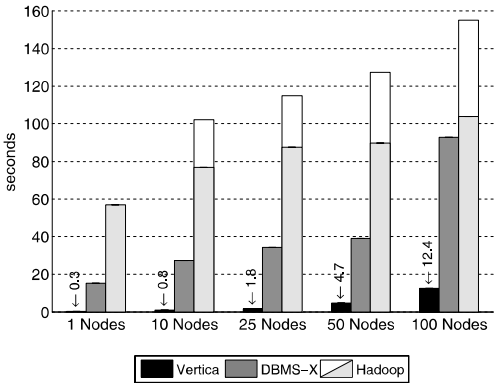


Figure 6: Selection Task Results

SELECT pageURL, pageRank
FROM Rankings WHERE pageRank > X;

108

Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

Hadoop vs. Databases: Aggregation

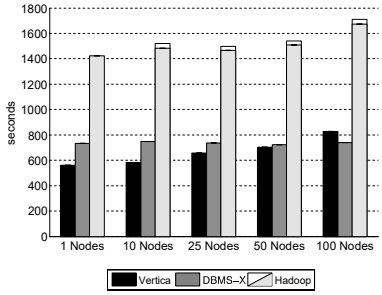


Figure 7: Aggregation Task Results (2.5 million Groups)

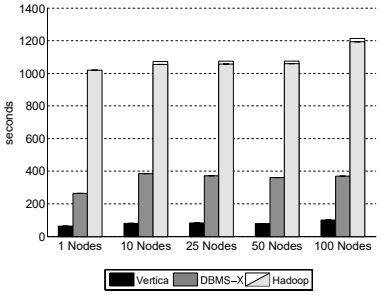


Figure 8: Aggregation Task Results (2,000 Groups)

```
SELECT sourceIP, SUM(adRevenue)
FROM UserVisits GROUP BY sourceIP;
```


Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

Hadoop vs. Databases: Join

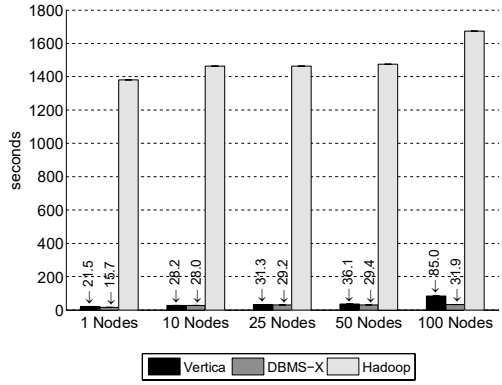


Figure 9: Join Task Results

```
SELECT INTO Temp sourceIP, AVG(pageRank) as avgPageRank, SUM(adRevenue) as totalRevenue
FROM Rankings AS R, UserVisits AS UV
WHERE R.pageURL = UV.destURL AND UV.visitDate BETWEEN Date('2000-01-15') AND Date('2000-01-22') GROUP BY UV.sourceIP;

SELECT sourceIP, totalRevenue, avgPageRank FROM Temp ORDER BY totalRevenue DESC LIMIT 1;
```

110

Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

Wasn't this supposed to be fast?

Well, yes, at some things. Like full table scans.

Most of these look like the fabled "full table scan" though. What gives?

It turns out, parsing text SUCKS

Aside: Dan's Tale of Text being Bad

- My thesis involved loading a database of thousands of proteins

```
ATOM   2384  N   ASN    12     -25.781 -17.512  27.342  1.00 53.51
ATOM   2385  CA  ASN    12     -26.388 -17.512  28.686  1.00 53.76
...
```

Loading took ~10 minutes (2007 era)

Convert to binary. Each row is

2 bytes (AA#) 1 byte (AA type) 1 byte (pad) 3 * 4 bytes (coordinates)

Loading takes ~5 seconds

Really? Yes, really.
Using a string stream it's even worse, taking an HOUR to load. C++ sucks, C rules. atoi and atof are dramatically faster than istringstream::operator>>
But nothing is faster than just grabbing bytes as you find them.

(It "probably" should have at least use htoni and ntohi to avoid endian issues for a file built on one host and read from another with different endian…but I was a grad student, give me a break)

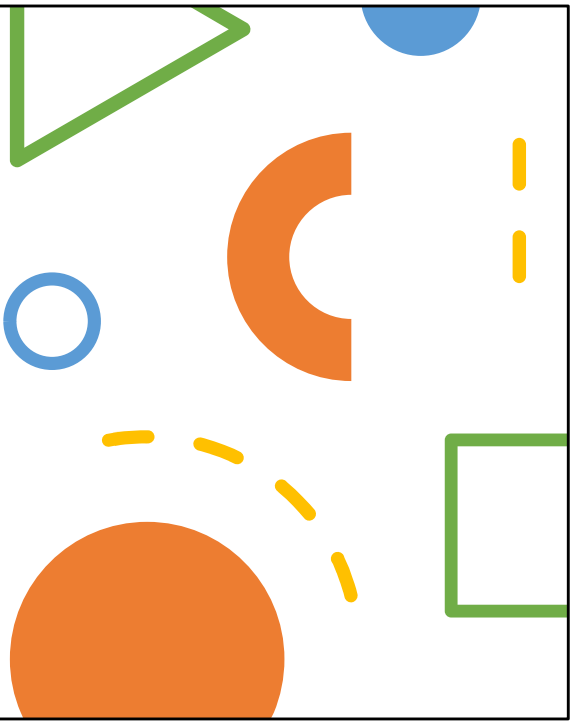
Hadoop is slow because strings are slow?

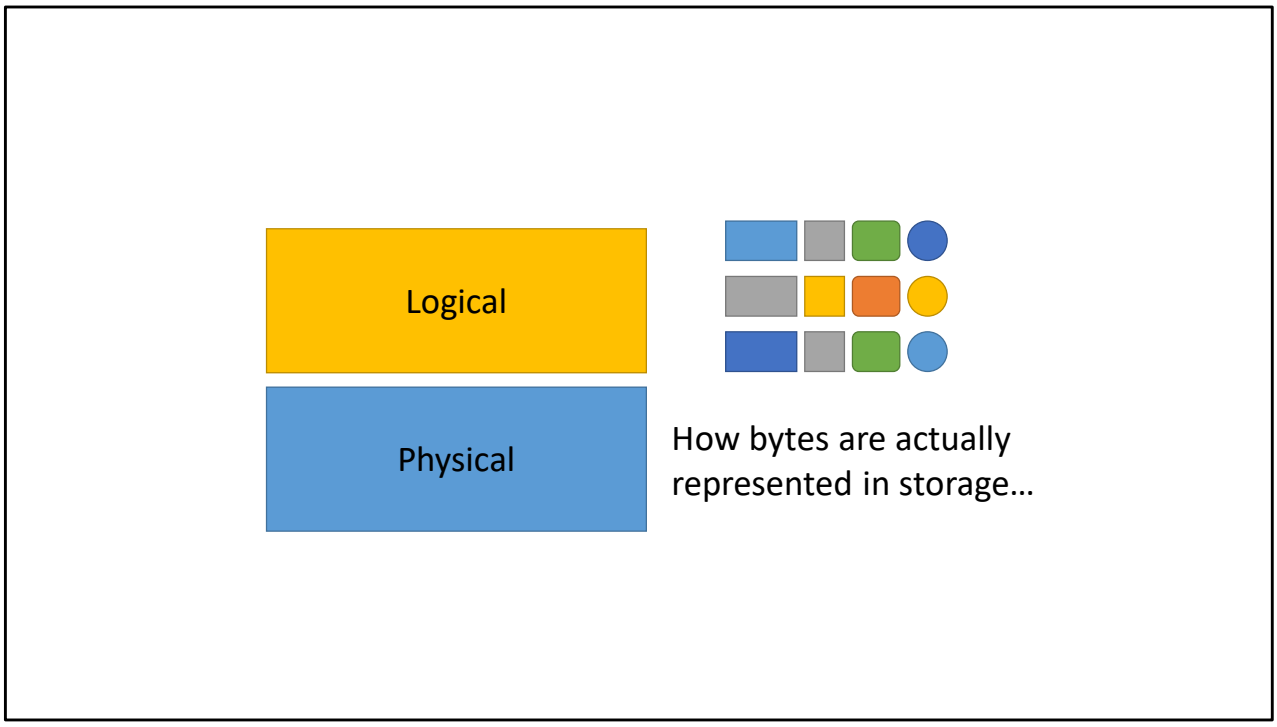
Yeah, basically.

Loading lines, splitting on whitespace, parsing integers, these are all very slow operations.

Solution?

- Use a binary format
- This needs a schema
- A schema separates logical and physical views
- Abstraction is a wonderful thing





Step 1. Figure out how a rectangle, square, rounded-rectangle, and circle are represented as bytes.
Step 2. Figure out how rows and columns are arranged

Row vs. Column Stores

Each column could be its own file, too.

We have to do one or the other, or something else along the same lines.
A file is 1 dimensional. You need SOME way of projecting high dimensional data into 1D sequence of bytes

Row vs. Column Stores

Row stores

Easier to modify a record: in-place updates

Might read unnecessary data when processing

Column stores

Only read necessary data when processing

Tuple writes require multiple operations

Tuple updates are complex

Advantages of Column Stores

Inherent advantages:

Better compression

Read efficiency

Works well with:

Vectorized Execution

Compiled Queries

These are well-known in traditional databases…

Vertica (from the “Hadoop is bad” slides) uses column stores. Many RDMS do as well.

Compression?

- Each column within the column store is grouped together
  - GZIP streams will do well at compressing Why?
  - Can go back to Vint, Simple9, etc encodings for integer columns.

Why? Repetition. Repetitive columns are grouped together.

Whether Vint gains you anything will depend on the distribution

Columns Stores: RLE

Column store

Run-length encoding example:


is a foreign key, relatively small cardinality
(even better, boolean)

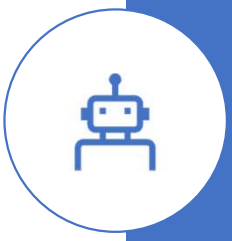
In reality:


…

Encode:

Read Efficiency?

Binary is faster: See my earlier slide about loading protein databank files



Column stores are fast:

- Only need the relevant columns.

```
val size = 100000000

var col = new Array[Int](size)          // List of random ints
var selected = new Array[Boolean](size)  // Matches a predicate?
```

```
for (i <- 0 until size) {
  selected(i) = col(i) > 0
}
```

```
for (i <- 0 until size by 8) {
  selected(i) = col(i) > 0
  selected(i+1) = col(i+1) > 0
  selected(i+2) = col(i+2) > 0
  selected(i+3) = col(i+3) > 0
  selected(i+4) = col(i+4) > 0
  selected(i+5) = col(i+5) > 0
  selected(i+6) = col(i+6) > 0
  selected(i+7) = col(i+7) > 0
}
```

Which is faster?
Why?

On Jimmy's laptop: 409ms (avg over 10 trials)

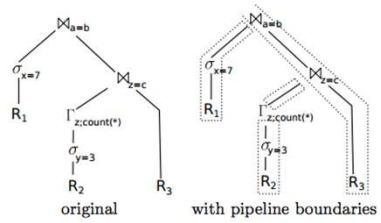
On Jimmy's laptop: 174ms (avg over 10 trials)

122

I get nearly the same numbers on my linux server at home. This is called "loop unrolling". Compilers will often do this optimization automatically.
Scala's JIT compiler doesn't have time for optimizations though.

Compiled Queries

```
select   *
from     R1,R3,
         (select   R2.z,count(*)
         from      R2
         where     R2.y=3
         group by R2.z) R2
where    R1.x=7 and R1.a=R3.b and R2.z=R3.c
```




```
initialize memory of ⋈a=b, ⋈c=z, and Γz
for each tuple t in R1
   if t.x = 7
      materialize t in hash table of ⋈a=b
for each tuple t in R2
   if t.y = 3
      aggregate t in hash table of Γz
for each tuple t in Γz
   materialize t in hash table of ⋈z=c
for each tuple t3 in R3
   for each match t2 in ⋈z=c[t3.c]
      for each match t1 in ⋈a=b[t3.b]
         output t1 ∘ t2 ∘ t3
```

Source: Neumann (2011) Efficiently Compiling Efficient Query Plans for Modern Hardware. VLDB.

Compiled Queries

Example LLVM query template

```
define internal void @scanConsumer(%8* %executionState, %Fragment_R2* %data) {
body:
  ...
  %columnPtr = getelementptr inbounds %Fragment_R2* %data, i32 0, i32 0
  %column = load i32** %columnPtr, align 8                                    1. locate tuples in memory
  %columnPtr2 = getelementptr inbounds %Fragment_R2* %data, i32 0, i32 1
  %column2 = load i32** %columnPtr2, align 8
  ... (loop over tuples, currently at %id, contains label %cont17)            2. loop over all tuples
  %yPtr = getelementptr i32* %column, i64 %id
  %y = load i32* %yPtr, align 4                                               3. filter y = 3
  %cond = icmp eq i32 %y, 3
  br i1 %cond, label %then, label %cont17
then:
  %zPtr = getelementptr i32* %column2, i64 %id                                4. hash z
  %z = load i32* %zPtr, align 4
  %hash = urem i32 %z, %hashTableSize
  %hashSlot = getelementptr %"HashGroupify::Entry"** %hashTable, i32 %hash
  %hashIter = load %"HashGroupify::Entry"** %hashSlot, align 8
  %cond2 = icmp eq %"HashGroupify::Entry"* %hashIter, null                    5. lookup in hash table (C++ data structure)
  br i1 %cond, label %loop20, label %else26
  ... (check if the group already exists, starts with label %loop20)
else26:
  %cond3 = icmp le i32 %spaceRemaining, i32 8                                 6. not found, check space
  br i1 %cond, label %then28, label %else47
  ... (create a new group, starts with label %then28)
else47:
  %ptr = call i8* @_ZN12HashGroupify15storeInputTupleEmj                      7. full, call C++ to allocate mem or spill
          (%"HashGroupify"* %1, i32 hash, i32 8)
  ... (more loop logic)
}
```

Source: Neumann (2011) Efficiently Compiling Efficient Query Plans for Modern Hardware. VLDB.

I see…

Databases are faster because they use binary column stores, not plain text files.

It's too bad Hadoop can't do that!

Wait, hold on, why can't Hadoop do that?

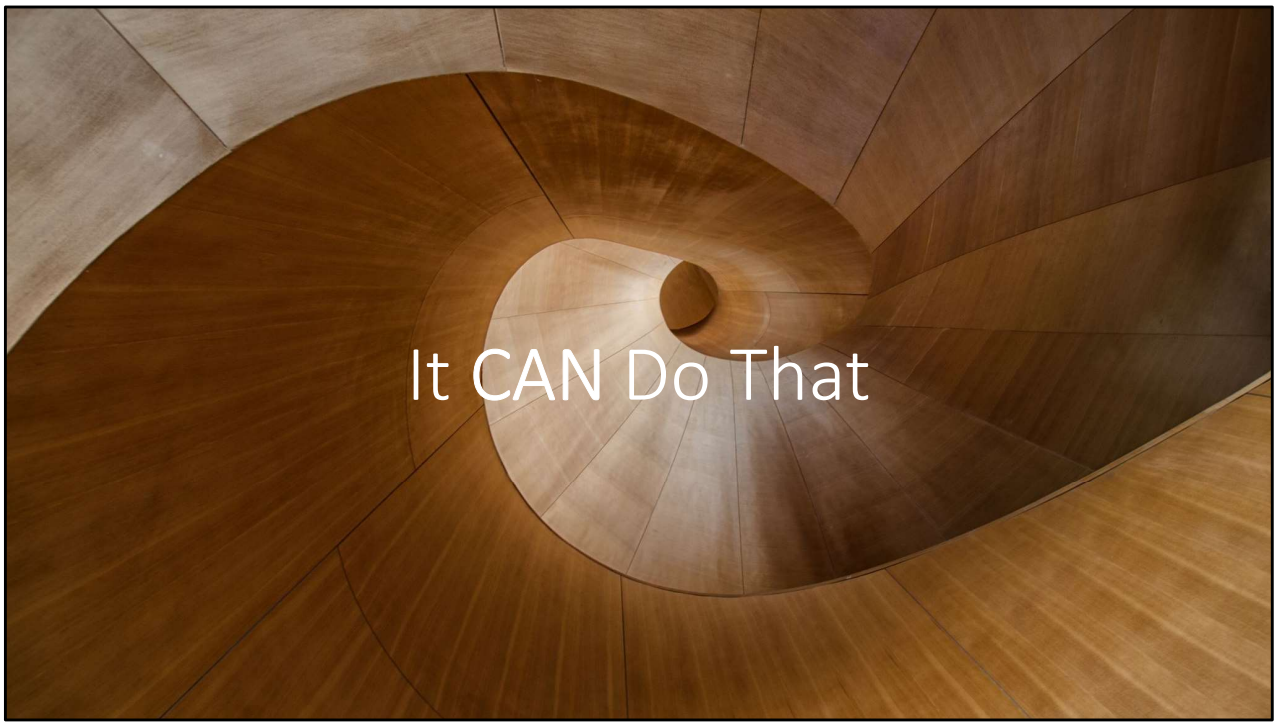


Image: Parquet flooring. You'll see why in a second…

Apache Parquet

A columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.

127

par-KAY – like the flooring. See it in the icon? Like a school gym.

Parquet in MapReduce

- Read: ParquetInputFormat
- Write: ParquetOutputFormat

But…the ParquetInputFormat returns Group values. You cannot select the elements of a Group, they're private. You have to convert to string, then split

Pointless? (I think so)

Parquet in Spark SQL

```
myDF = spark.read.parquet("path/to/parquet/file.parquet")

…

otherDF.write.parquet("path/to/output/file.parquet")
```

Advantages of Column Stores

Inherent advantages:

Better compression

Read efficiency

Works well with:

Vectorized Execution

Compiled Queries

Parquet gives us the top two things. Can we do the bottom two?

Vectorized Operators?

```
hive.vectorized.execution.enabled = true
```

```
spark.sql.parquet.enableVectorizedReader = true
```

(For Spark, the default is already true)

Compiled Queries?

HIVE (and PIG) are already compiled to Java code

Spark?

If you write an SQL query, yes, the plan will be compiled and optimized, though only at runtime.

If you write using dataframe operators? Yes. Scala and Python can be compiled

Compiling Spark SQL

SELECT x, y FROM z WHERE x * (1 – y)/100 < 434;

Interpret the predicate: SLOW!
x * (1 – y)/100 < 434; => LessThan(Times(row("x"), Minus(…))

Compile: FAST! (to run the query, at least…)
Feed AST of expression into Scala compiler:

row => row("x") * (1 – row("y")) / 100 < 434