Introduction

Data Systems and Pipelines: Objectives

1. Theory
   - Understand the history of data systems and how they evolved into the current landscape
   - Be able to articulate why a given class of databases should be used over another based on a use case
   - Understand how traditional data warehouses were constructed and their weaknesses
2. Practice
   - Understand how data persistence can affect performance
   - Have a basic working knowledge of SQL and how joins work in the MapReduce framework

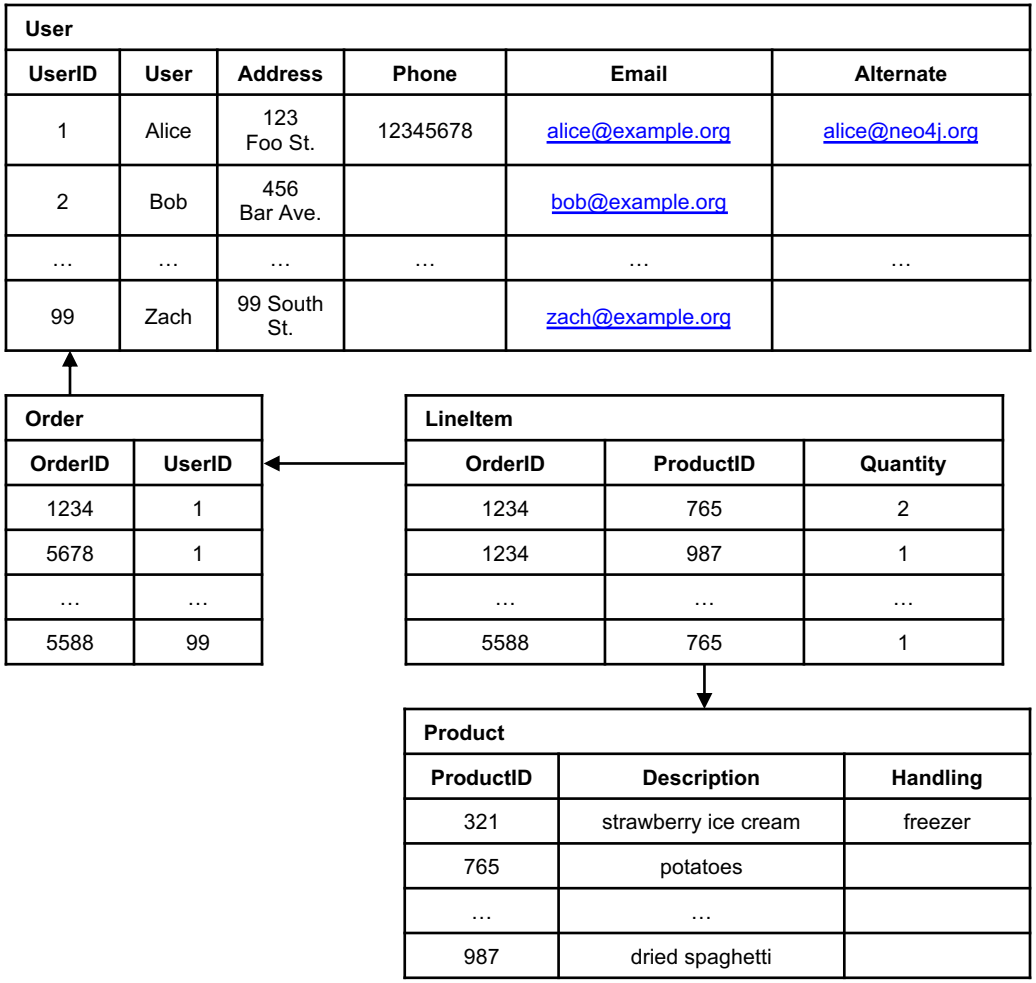
Data Systems and Pipelines: Table of Contents

The End

Types of Data

Structured Data

- Traditionally lives in a relational database
- Highly organized
- Examples
  - SSN
  - Phone
  - Address
  - User information
  - Product information
  - Order information



User

UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
…	…	…	…	…	…
99	Zach	99 South St.		zach@example.org	

Order

OrderID	UserID
1234	1
5678	1
…	…
5588	99

LineItem

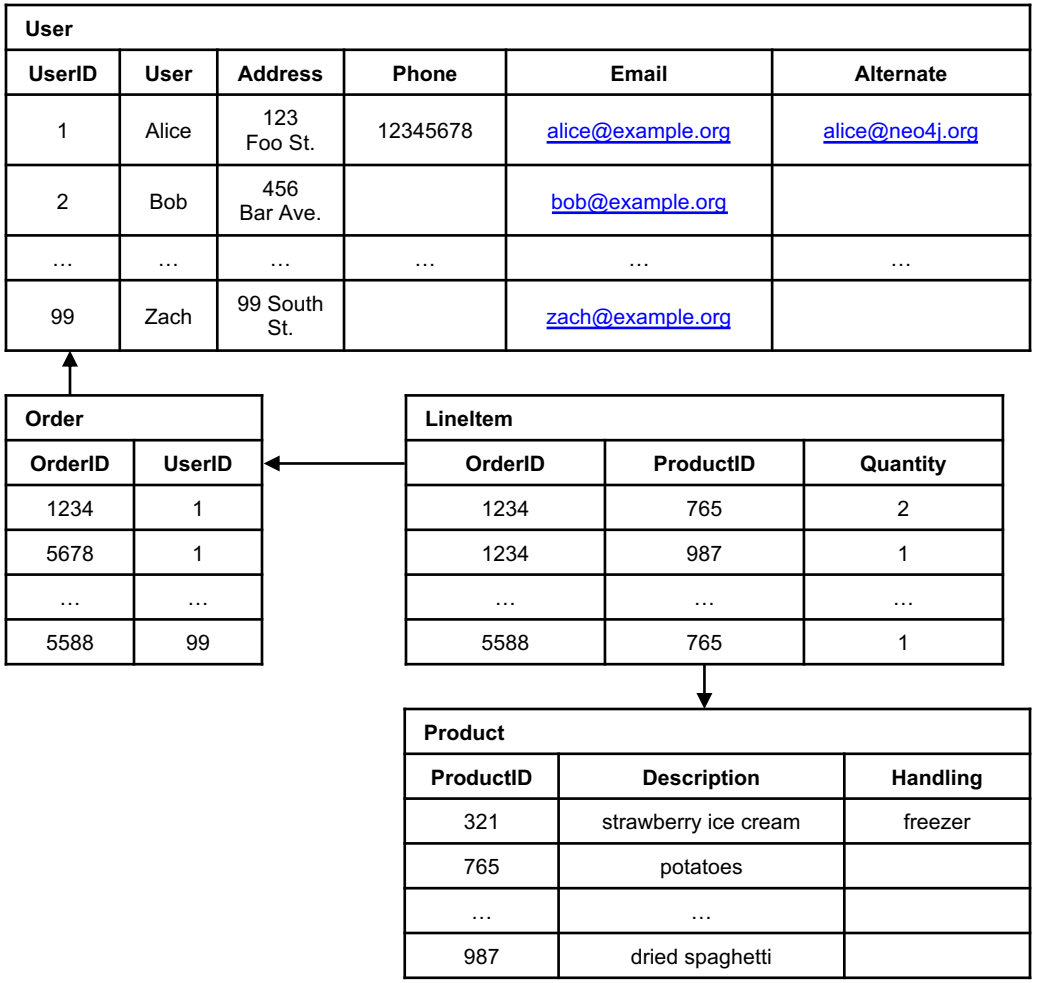
OrderID	ProductID	Quantity
1234	765	2
1234	987	1
…	…	…
5588	765	1

Product

ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
…	…	
987	dried spaghetti	

Structured Data

- Pros
  - Easy to work with
- Cons
  - Not all data can be expressed in this way
  - Inflexible



User

UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
…	…	…	…	…	…
99	Zach	99 South St.		zach@example.org	

Order

OrderID	UserID
1234	1
5678	1
…	…
5588	99

LineItem

OrderID	ProductID	Quantity
1234	765	2
1234	987	1
…	…	…
5588	765	1

Product

ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
…	…	
987	dried spaghetti	

Semi-Structured Data

- Structured, but not too much
- Flexible schema
- Example
  - XML
  - JSON
- Pros
  - Fewer concerns for developers
  - Capability to nest and form hierarchical data structures allows for expression of complex relations


```
{
    "Title": "The Cuckoo's Calling",
    "Author": "Robert Galbraith",
    "Genre": "classic crime novel",
    "Detail": {
        "Publisher": "Little Brown",
        "Publication_Year": 2013,
        "ISBN-13": 9781408704004,
        "Language": "English",
        "Pages": 494
    },
    "Price": [
        {
            "type": "Hardcover",
            "price": 16.65
        },
        {
            "type": "Kidle Edition",
            "price": 7.03
        }
    ]
}
```

Semi-Structured Data

- Cons
  - No definite query language like SQL
  - Performance overhead
  - Garbage in, garbage out


```
{
    "Title": "The Cuckoo's Calling",
    "Author": "Robert Galbraith",
    "Genre": "classic crime novel",
    "Detail": {
        "Publisher": "Little Brown",
        "Publication_Year": 2013,
        "ISBN-13": 9781408704004,
        "Language": "English",
        "Pages": 494
    },
    "Price": [
        {
            "type": "Hardcover",
            "price": 16.65
        },
        {
            "type": "Kidle Edition",
            "price": 7.03
        }
    ]
}
```

Unstructured Data

- No predefined data model
- Most of the data in the world (80% or greater)
- Examples
  - Videos
  - Images
  - Free-form text (news, blogs, etc.)
  - Signal data (spectroscopy, audio, etc.)

Unstructured Data

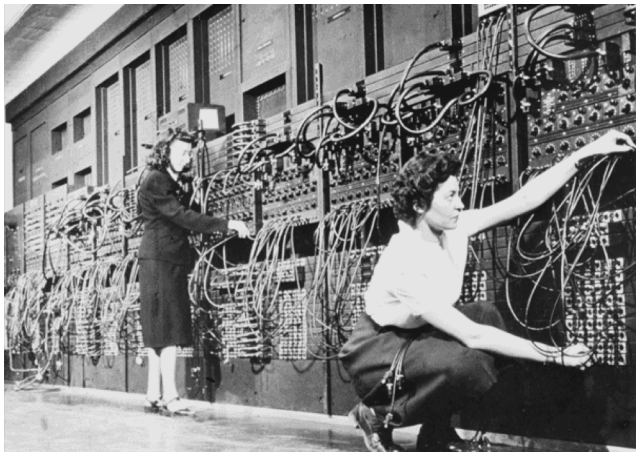
- Pros
  - A picture is worth a thousand words or a million rows of data
- Cons
  - Significant effort in processing
    - Needs to be translated into some form of structured data to extract value

The End

Evolution of Data Processing Systems

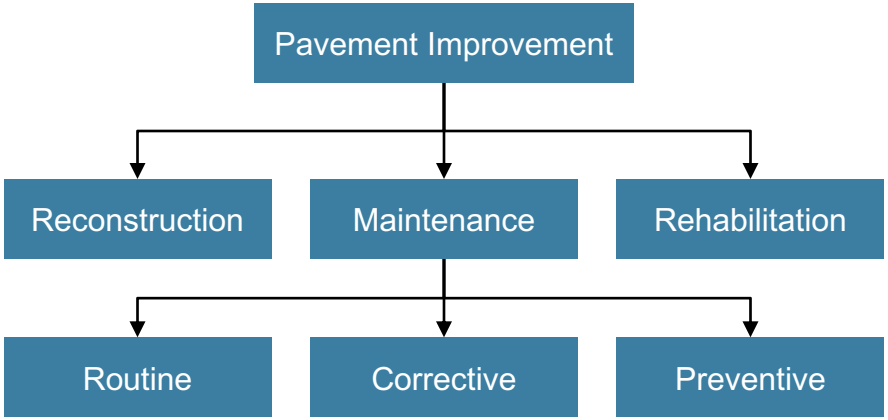
It Started With ENIAC

- World's first electrical, large-scale, general-purpose digital computer
- Women were the first developers
  - Jean Bartik
- Used mostly for physical simulations
  - Missile trajectories
  - Monte Carlo simulations for hydrogen bomb
- Lead to EDVAC
  - Binary, not decimal
  - Memory storage



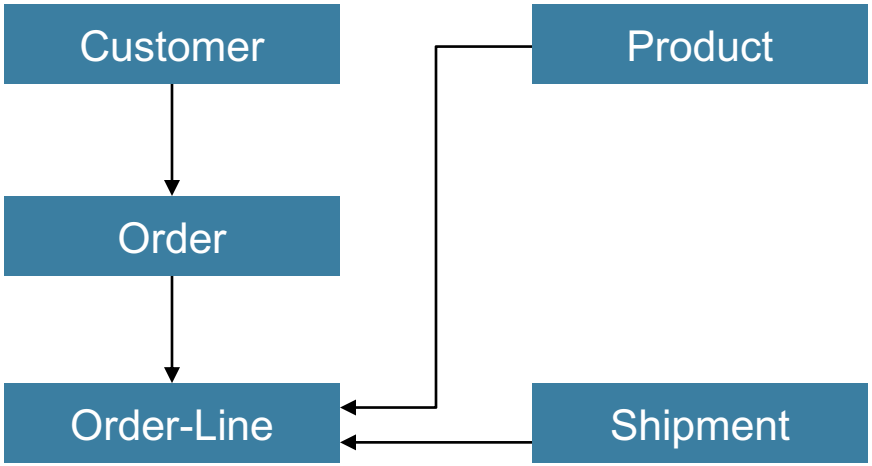
Hierarchical Model

- Represent data as a tree
- Developed by IBM in 1960s
- Inflexible model
  - One-to-many relations
- Very performant
  - Still in use today in mainframes
  - EHR EPIC
  - Banking



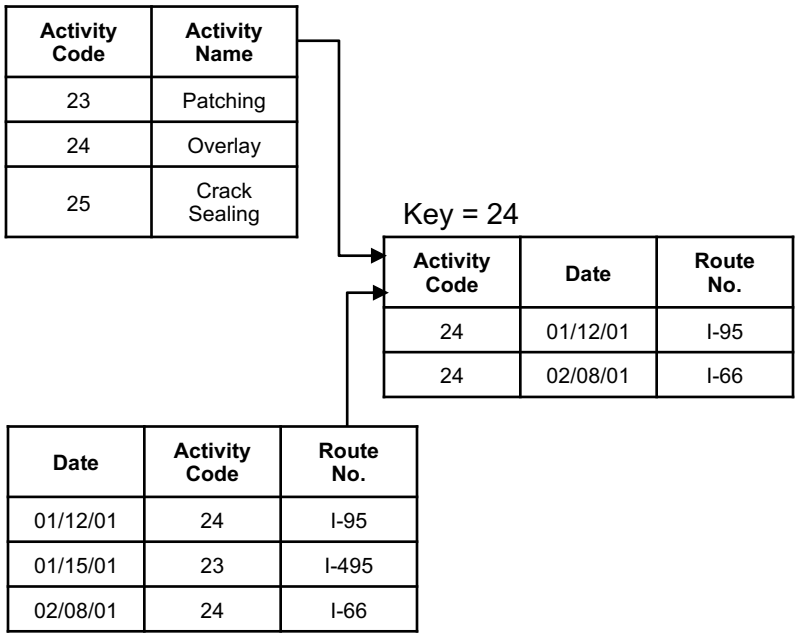
Network Model

- Generalizes hierarchical model
  - Allows multiple parents
  - Creates a generalized graph structure
- Did not have much favor, beat out by the relational model



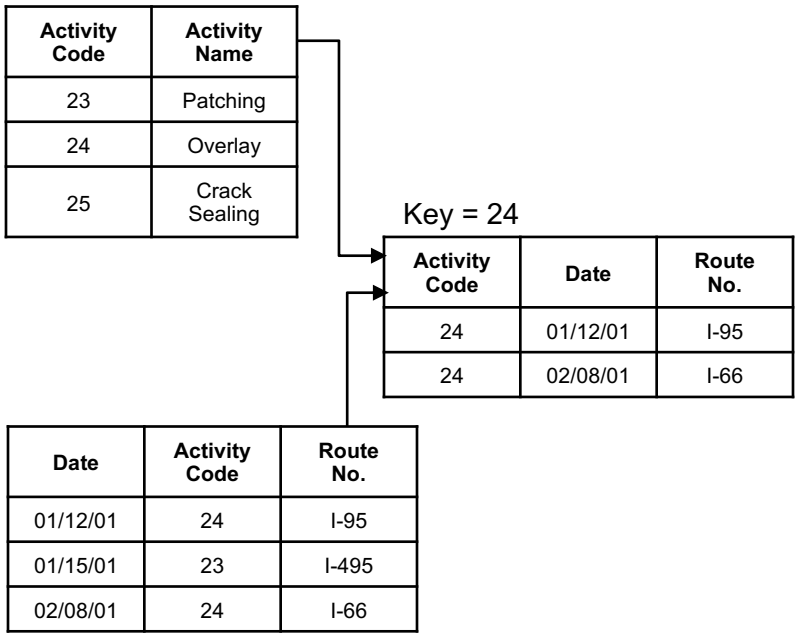
Relational Model

- Data represented as sets of tuples (tables) with relations between them
- Expressive query language (SQL)
  - Turing complete in systems with Common Table Expressions (CTEs)



Relational Model

- Created by Codd in 1969
  - Worked at IBM
- Oracle released the first relational database product in 1979
- Most common structured data storage



Distributed Systems

- Vertical scaling only goes so far
- Need to scale-out horizontally
- What constraints do we have?
  - Fault tolerance
  - Latency
- Things to consider
  - Consistency
  - Availability
  - Partition tolerance
- Effectively, this led to a split for distributed transactional and analytical systems

NoSQL

- Traditional RDBMS systems typically focus on consistency and availability guarantees
- Due to how these systems handled partition tolerance companies developed other systems
- Vertical vs. horizontal scaling

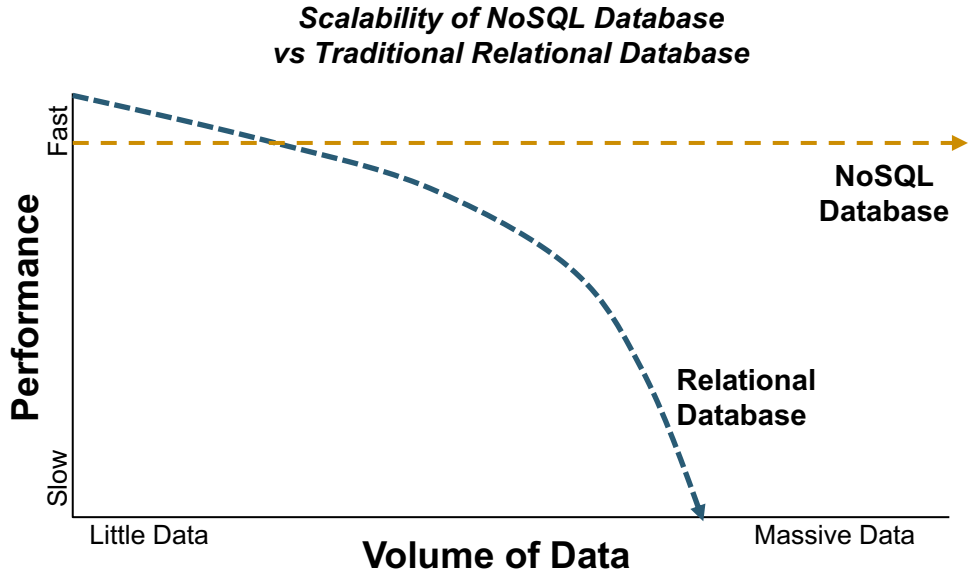


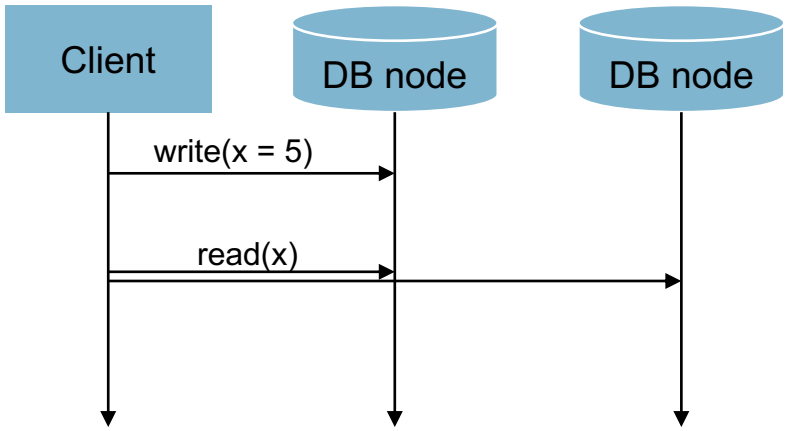
Image credit: DataJobs.com

Motivation: CAP

- Why does the performance of a traditional RDBMS fall off as we increase the volume of data we throw at it?
- Let's discuss.

Consistency

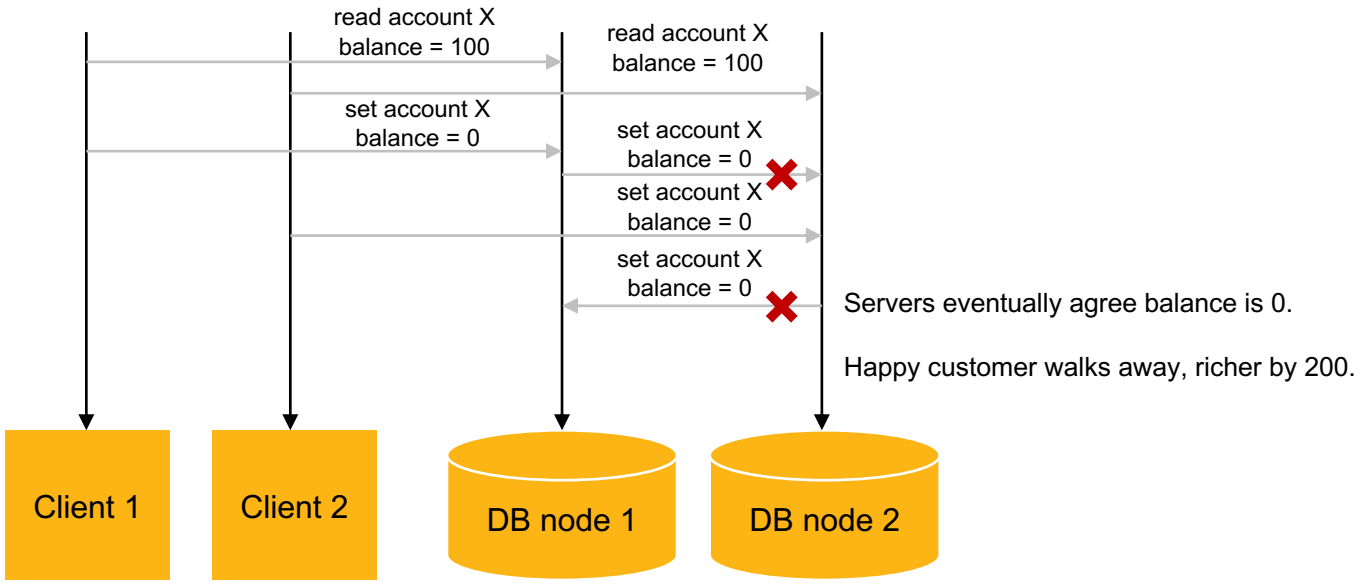
- Every read receives the most recent write or an error.
- All members of the system return the same result, regardless of location.



Consistency: these two reads are guaranteed to return the same value (which need not be 5)

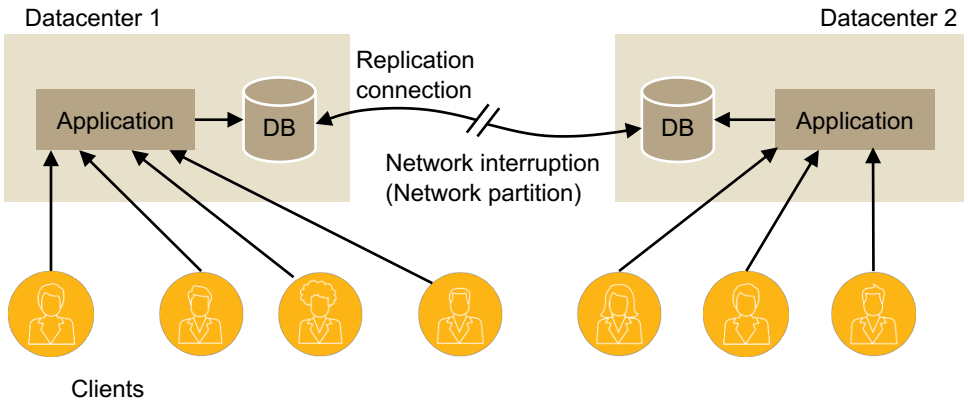
Availability

Every request receives a (nonerror) response—without the guarantee that it contains the most recent write.



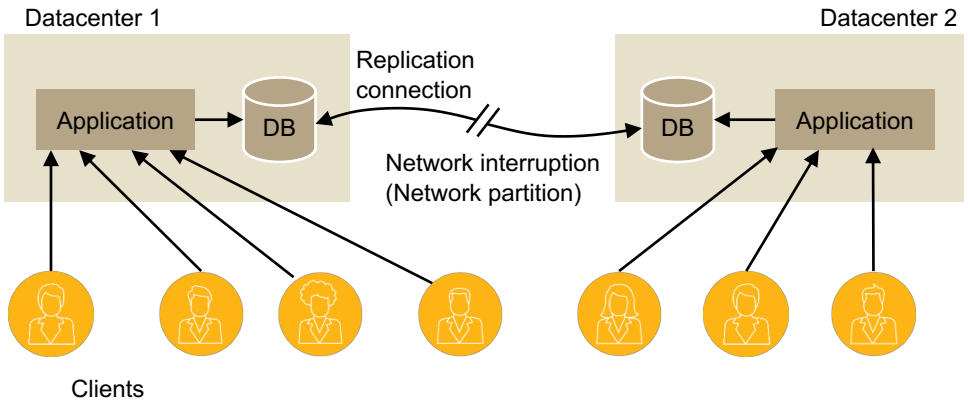
Partitioning

- The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
- Required for a system to be a source of truth since all networks require this property
- How do we solve this?



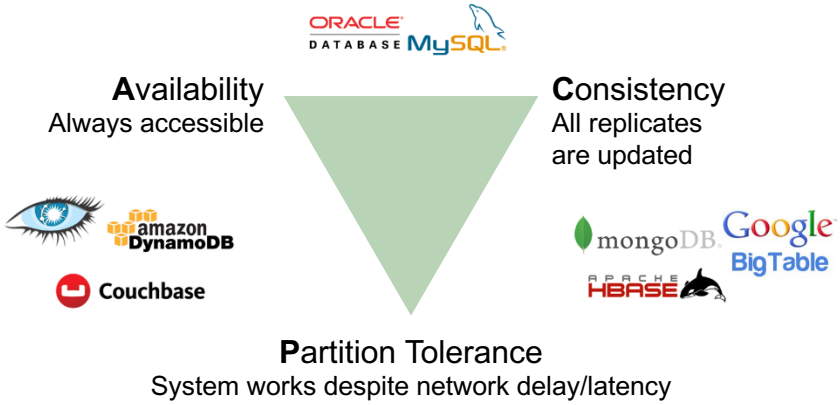
RDBMS Partitioning Implications

- We have two ways to solve this problem in traditional systems.
  - Read/write to both databases without syncing between them
    - Loses consistency
  - Read/writes only happen to one side (leader)
    - Reroute clients to leader in case of failure
- This leads to inherent scalability constraints on traditional databases.



CAP Theorem

- Abuse of naming conventions
- Focused on the primary concerns that the products aim to fit
- CA
  - Traditional RDBMS
- CP
  - Handle analytical use cases where availability isn't the primary concern
- AP
  - Highly scalable but choosing eventual consistency to ensure availability of the system



ACID vs. BASE

- **A**tomicity
  - Small transactional units either pass or fail; if any failure, then the transaction fails
- **C**onsistency
  - Data must follow all rules in the database such as constraints, cascades, and triggers, and prevents corruption
  - Different than CAP theorem
- **I**solation
  - Atomic transactions are processed concurrently, and each atomic transaction is independent of the other
  - Leaves database in same state
- **D**urability
  - Transactions that have been committed will stay in the database state even in failure scenarios

- **Ba**sically available
  - Always returns a response
- **S**oft state
  - System can change over time, even with no activity (due to eventual consistency)
- **E**ventual consistency
  - Without input, the data will spread to every node eventually—thus becoming consistent eventually

Eventual Consistency

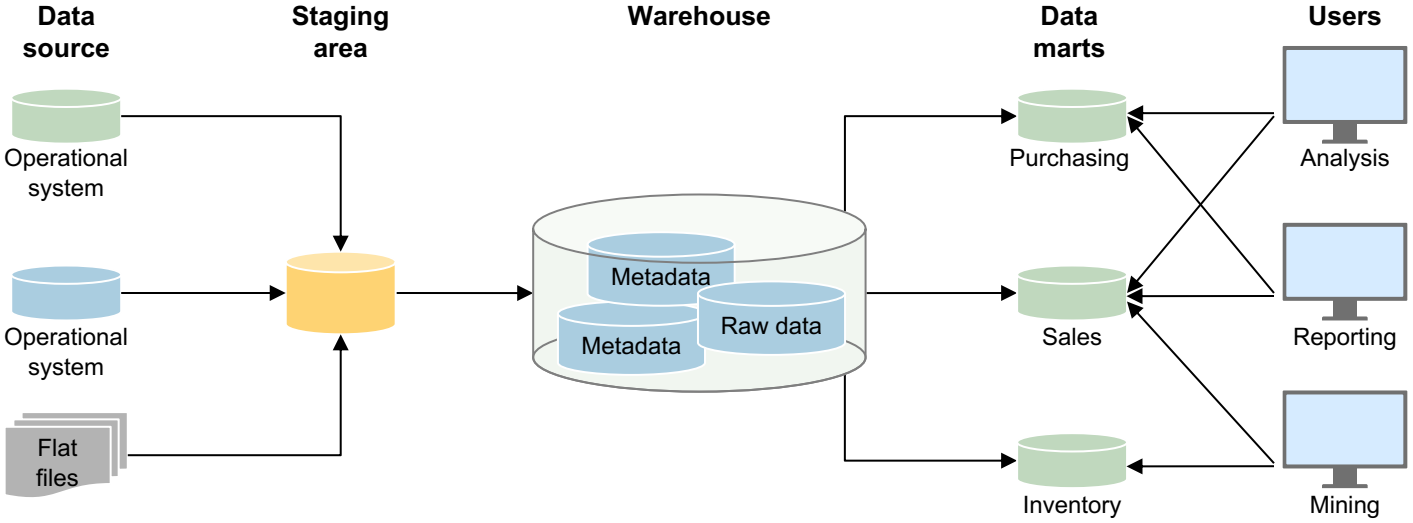
- Many practical systems have approximately 200 ms to the most recent write.
- Amazon's DynamoDB was one of the first studies of eventual consistency effects on CAP (2007).

The End

Analytical Data Systems

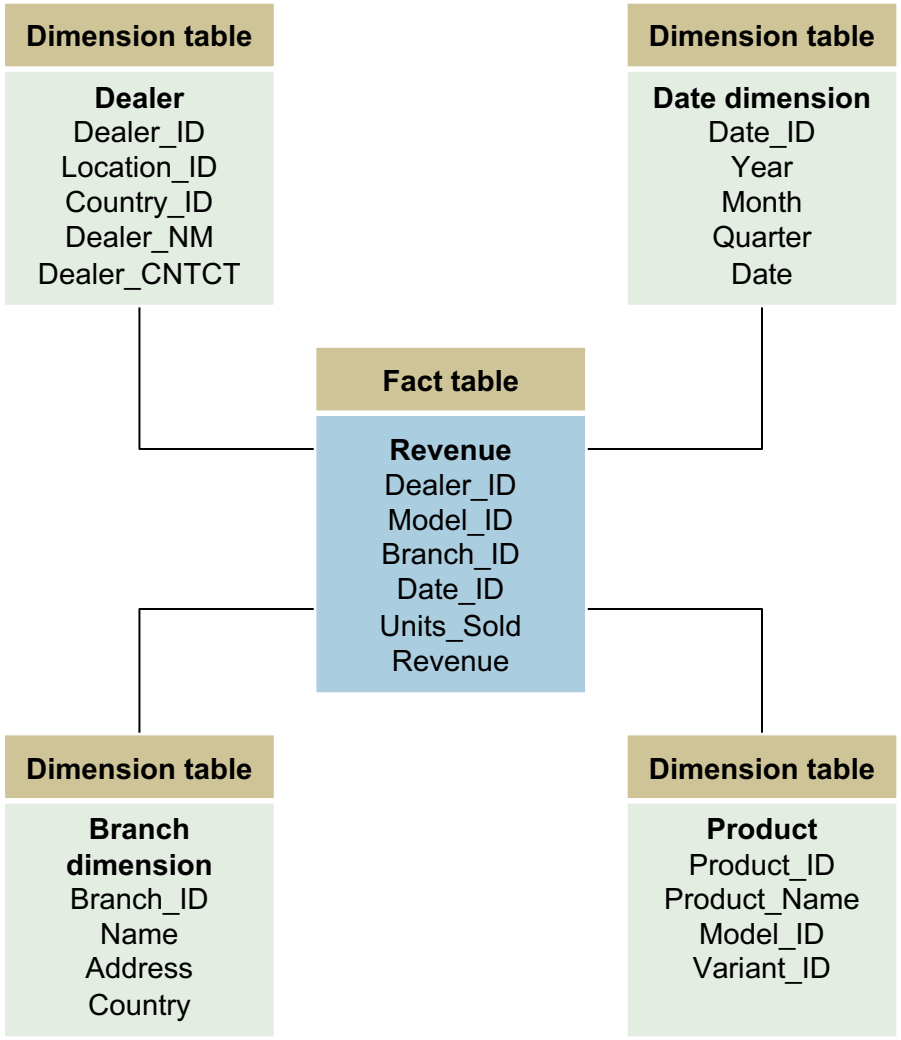
Data Warehousing: Idea

- Companies have several transactional applications that drive their business.
- Bringing it together can enable the company to answer questions across their organization.
- Performance optimization can be done prior to exposing data to end users, leading to order of magnitude performance increases.
- It can be complex to manage.



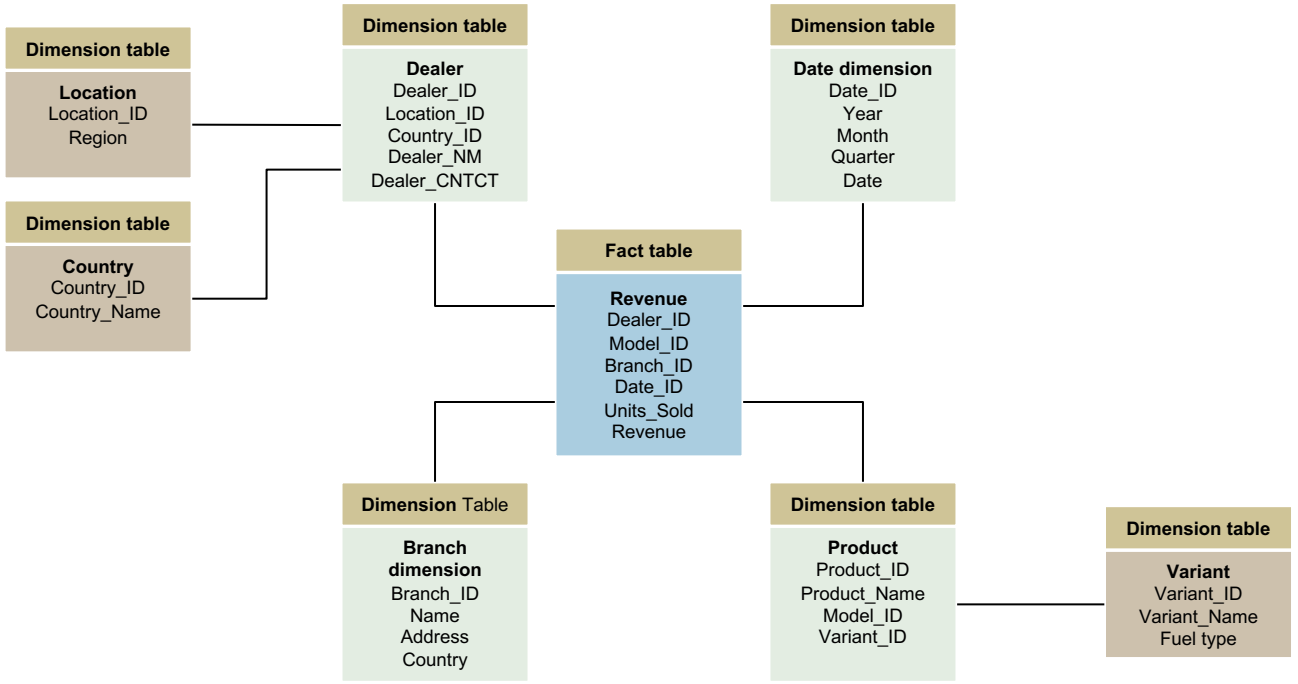
Data Warehousing: Star Schema

- Fact table
  - Contains reference for metadata at the transactional or aggregate level
  - Contains the factual details about the transaction/aggregate and provides reference to all other metadata for joins as needed
  - Can be taken as a snapshot
  - Extremely large
- Dimensional table
  - Descriptive data such as product information
  - Lowest level of detail possible
- Why?



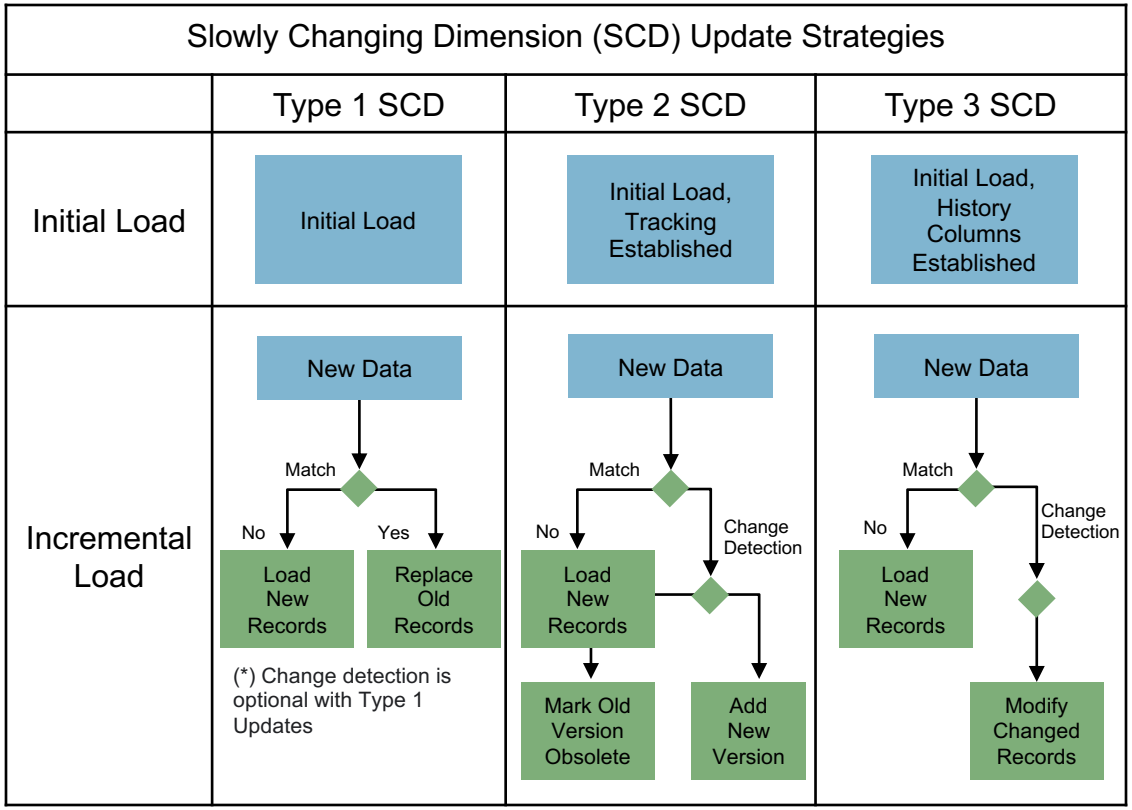
Data Warehousing: Snowflake Schema

- Star schema is a special case
  - “Normalized” star schema
- Less storage required
- More complex queries
- Easier to validate dimension tables



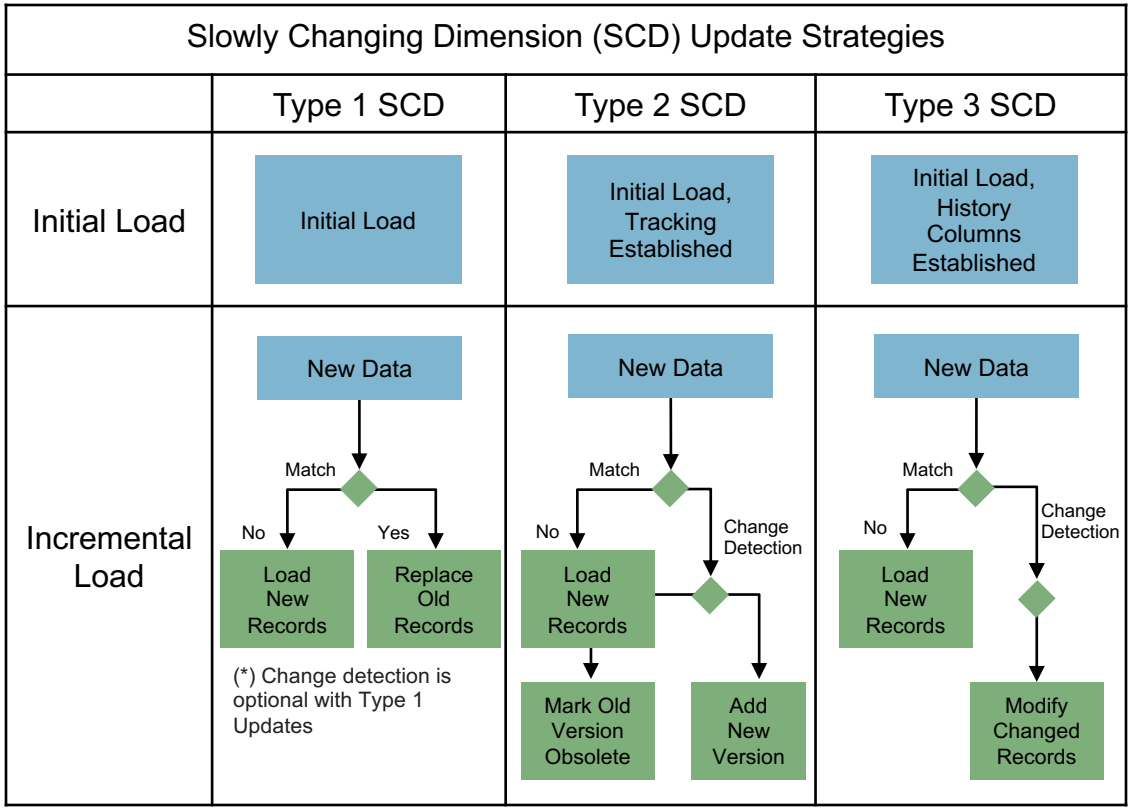
Data Warehousing: Slowly Changing Dimensions

- A slowly changing dimension (SCD) is a dimension that stores and manages both current and historical data over time in a data warehouse
- One of the most critical ETL tasks in tracking the history of dimension records

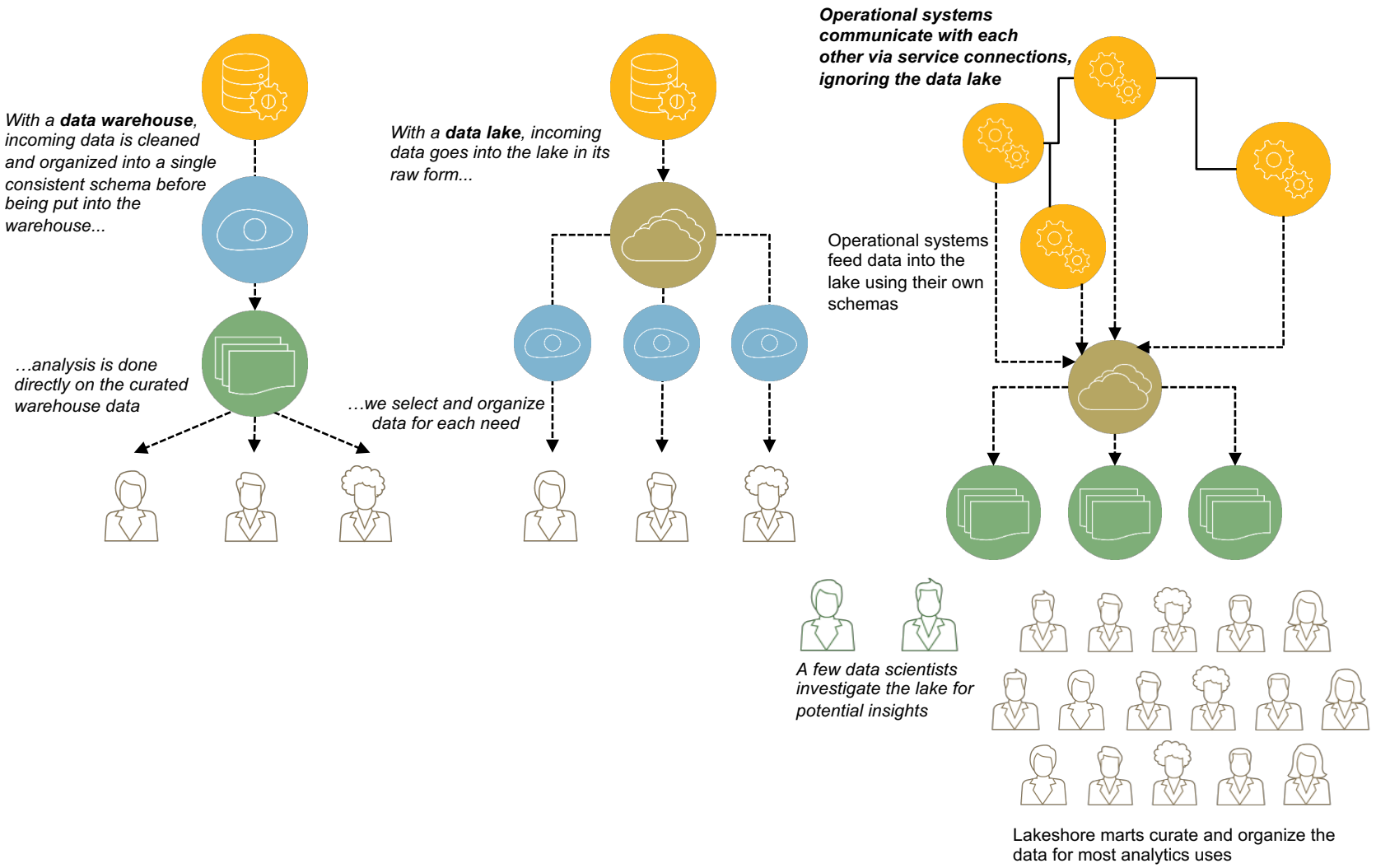


Data Warehousing: Slowly Changing Dimensions

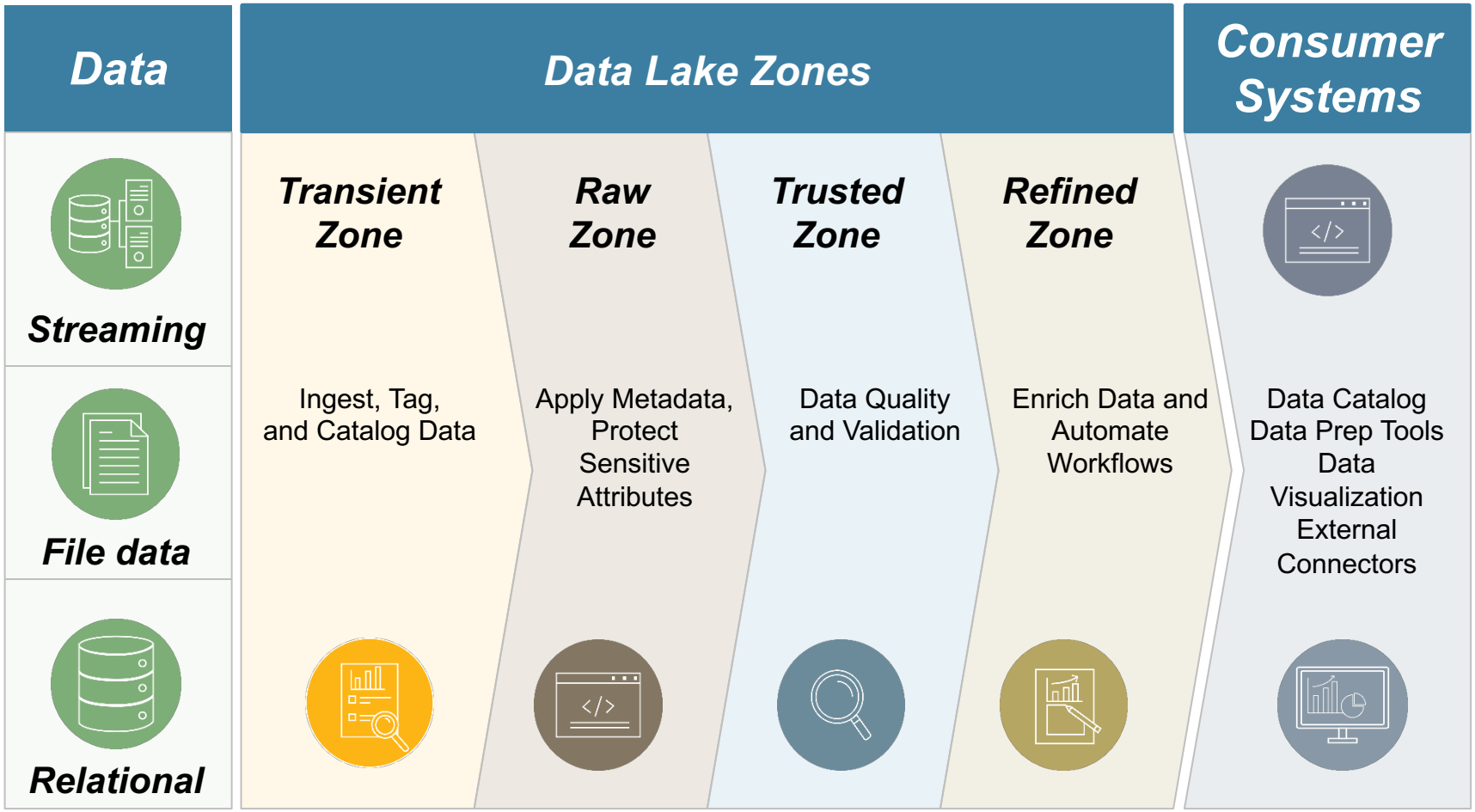
- Type 1: overwriting
  - Simple queries with no history
- Type 2: creating another dimension record
  - Complex queries and full history
- Type 3: creating a current value field
  - Traverse for history of changes
  - Simpler queries
- This can be extremely difficult to maintain



Data Lakes: Consolidating the Data



Data Lakes: Governance Models



The End

Data Persistence and Storage

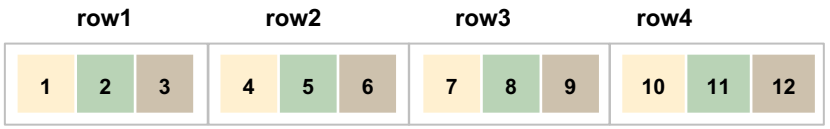
Row vs. Column-Oriented Storage

- In ACID systems, we want atomicity and act on individual rows in a transaction
- This is good for transactional workloads
- What if we want to do aggregates?
  - Column-oriented data allow us to aggregate on columns easily
- Choose based on your use case
- Orientation is based on serialization in HDFS/Spark, while orientation in databases is dependent on the architecture

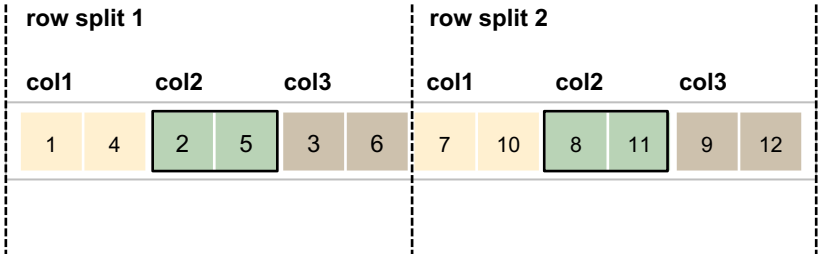
Logical Table

	col 1	col 2	col 3
row1	1	2	3
row2	4	5	6
row3	7	8	9
row4	10	11	12

Row-oriented layout (SequenceFile)



Column-oriented layout (RCFile)



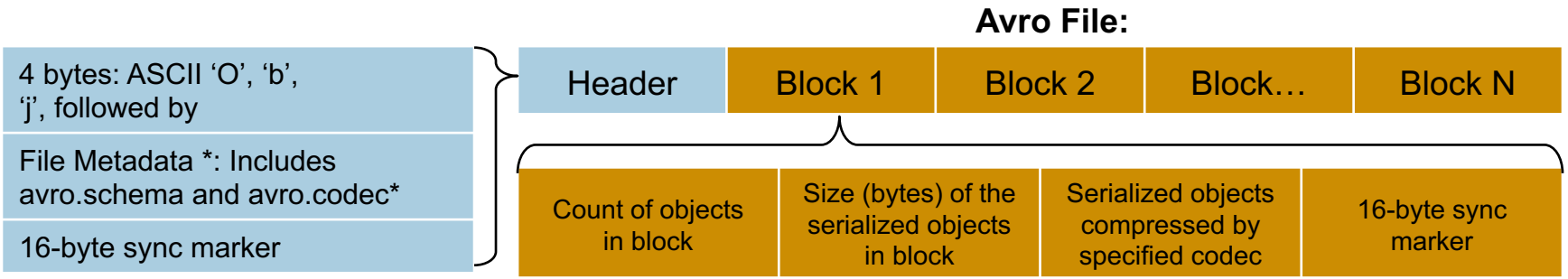
CSV

- Text files are human readable and easily parsable.
- Text files are slow to read and write.
- Data size is relatively bulky and not as efficient to query.
- No metadata are stored in the text files, so we need to know the structure of the fields.
- Text files are not splittable after compression.
- Limited support for schema evolution; new fields can only be appended at the end of the records, and existing fields can never be removed.

```
Generic,Storage Type,Hierarchy Type,Attributes Header,comment,bso data storage,aso data storage,
    two pass calculation,aso dimension formula,consolidation type,uda,parent,alias:Default,alias:English
Customers,SPARSE,STORED,,,LABELONLY,STOREDATA,N,,,UDA,,alias:Default,alias:English
NoCustomer,SPARSE,Disabled,,,StoreData,StoreData,N,,+,,,No Customer,No Customer
AllCustomers,SPARSE,Disabled,,,StoreData,StoreData,N,,+,,,All Customers,All Customers
Big Box,SPARSE,,,,StoreData,StoreData,N,,+,,AllCustomers,,
BB100,SPARSE,,,,StoreData,StoreData,N,,+,,Big Box,Q Mart,Q Mart
BB200,SPARSE,,,,StoreData,StoreData,N,,+,,Big Box,Bike Depot,Bike Depot
BB300,SPARSE,,,,StoreData,StoreData,N,,+,,Big Box,Mountain Adventures,Mountain Adventures
Specialty Retailers,SPARSE,,,,StoreData,StoreData,N,,+,,AllCustomers,,
SR100,SPARSE,,,,StoreData,StoreData,N,,+,,Specialty Retailers,Bobs Bikes,Bobs Bikes
SR200,SPARSE,,,,StoreData,StoreData,N,,+,,Specialty Retailers,Rose Town Bikes,Rose Town Bikes
SR300,SPARSE,,,,StoreData,StoreData,N,,+,,Specialty Retailers,The Cyclery,The Cyclery
Webstore,SPARSE,,,,StoreData,StoreData,N,,+,,AllCustomers,,
```

Avro

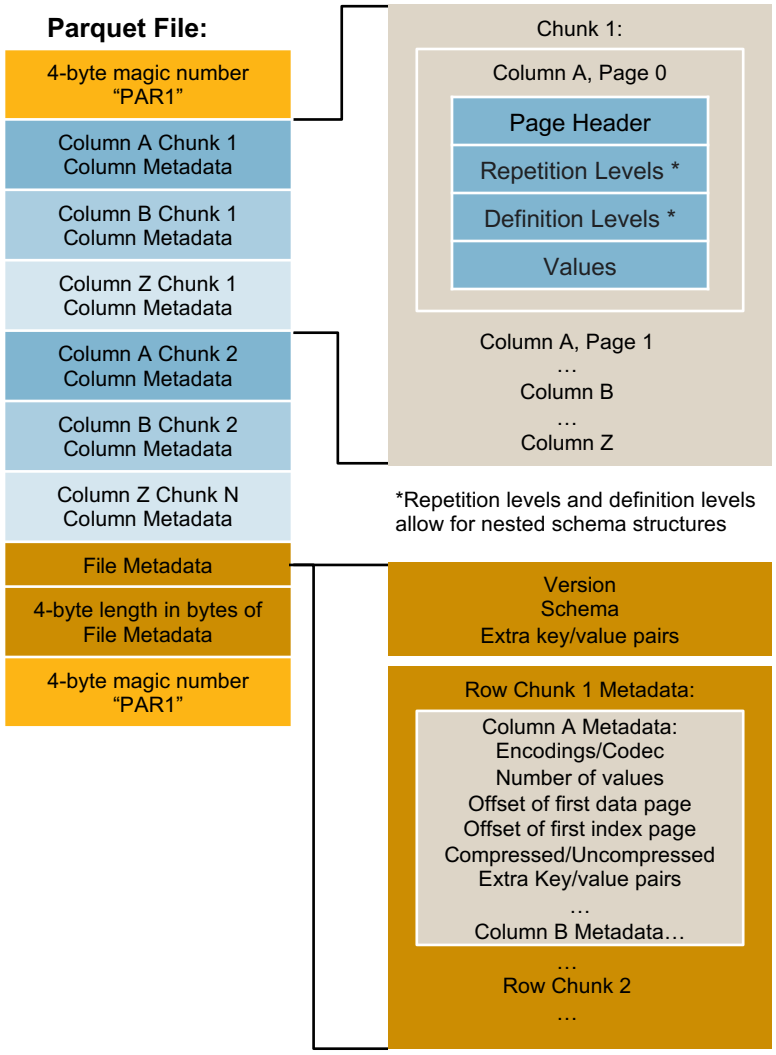
- Row-based
- Interoperability: can serialize into Avro/Binary or Avro/JSON
- Compact binary form
- Extensible schema language
- Untagged data
- Dynamic typing
- Supports block compression
- Splittable
- Best compatibility for evolving data schemas



* File metadata follows:
{"type": "map", "values": "bytes"}

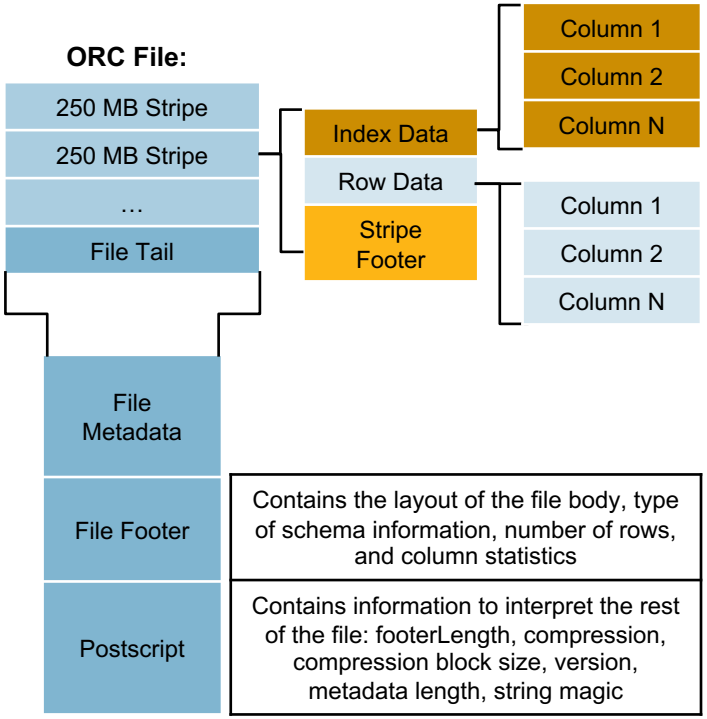
Parquet

- Column-oriented
- Efficient in terms of disk I/O and memory
- Efficiently encoding of nested structures and sparsely populated data
- Provides extensible support for per-column encodings
- Provides extensibility of storing multiple types of data in column data
- Offers better write performance by storing metadata at the end of the file
- Records in columns are homogeneous, so it's easier to apply encoding schemes
- Nonappendable due to metadata



ORC

- Column-oriented
- Lightweight indexes stored within the file
- Ability to skip row groups that don't pass predicate filtering
- Block-mode compression based on data type
- Includes basic statistics (min, max, sum, count) on columns
- Concurrent reads of the same file using separate RecordReaders
- Ability to split files without scanning for markers
- Metadata are stored using protobuf, which allows schema evolution by supporting addition and removal of fields



The End

Working With Data

SQL and MapReduce

SQL

```
SELECT
        customer_id,
        review_date,
        review_rating,
        review_votes
FROM
        customer_reviews_rowstore
LIMIT 10
```

```
"customer_id","review_date","review_rating","review_votes"
AE22YDHSBFYIP,1970-12-30,5,10
AE22YDHSBFYIP,1970-12-30,5,9
ATVPDKIKX0DER,1995-06-19,4,19
AH7OKBE1Z35YA,1995-06-23,5,4
ATVPDKIKX0DER,1995-07-14,5,0
A102UKC71I5DU8,1995-07-18,4,2
A1HPIDTM9SRBLP,1995-07-18,5,0
A1HPIDTM9SRBLP,1995-07-18,5,0
ATVPDKIKX0DER,1995-07-18,5,1
ATVPDKIKX0DER,1995-07-18,5,1
```

SQL: Toward Analytics

```
SELECT
        AVG(review_rating)
FROM
        customer_reviews_rowstore
```

```
"avg"
4.3045247643869191
```

SQL: Toward Analytics

```
SELECT
        product_id,
        ROUND(AVG(review_rating),2),
        COUNT(review_rating)
FROM
        customer_reviews_rowstore
GROUP BY
        product_id
LIMIT 10
```

```
"product_id","round","count"
"0471412767",5.00,11
"0802773613",5.00,1
B000001OU6,4.88,8
"0882681206",5.00,3
"0312971842",4.29,34
"0895264544",4.08,88
"6303444016",2.50,6
"0671003526",4.67,3
"0198227868",5.00,3
"0879759178",5.00,6
```

SQL: Toward Analytics

```
SELECT
product_id,
review_date,
review_rating,
ROUND(AVG(review_rating) OVER (
    ORDER BY product_id,review_date
    ROWS BETWEEN 3 PRECEDING
     AND CURRENT ROW
),2) AS average,
COUNT(review_rating) OVER (
    ORDER BY product_id,review_date
) AS cnt
FROM customer_reviews_rowstore
LIMIT 10
```

```
"product_id","review_date","review_rating","average","cnt"
"0001047655",1998-04-09,5,5.00,1
"0001047655",1998-05-11,4,4.50,2
"0001047655",1998-07-07,4,4.33,3
"0001047655",1998-07-27,2,3.75,4
"0001047655",1998-08-23,5,3.75,5
"0001047655",1998-12-30,5,4.00,6
"0001053736",1997-04-08,4,4.00,7
"0001053736",1999-05-12,3,4.25,10
"0001053736",1999-05-12,3,3.75,10
"0001053736",1999-05-12,3,3.25,10
```

SQL: Comparing Row vs. Column, Transactional

- Let's update a review in our customer reviews

```
UPDATE customer_reviews_rowstore
SET review_rating = 4
WHERE customer_id='AE22YDHSBFYIP'
AND product_id='1551803542';
```

- Row store
  - 0.1 second update
- Column store
  - Operation not supported
    - Would require traversing the entire column

SQL: Comparing Row vs. Column, Analytical

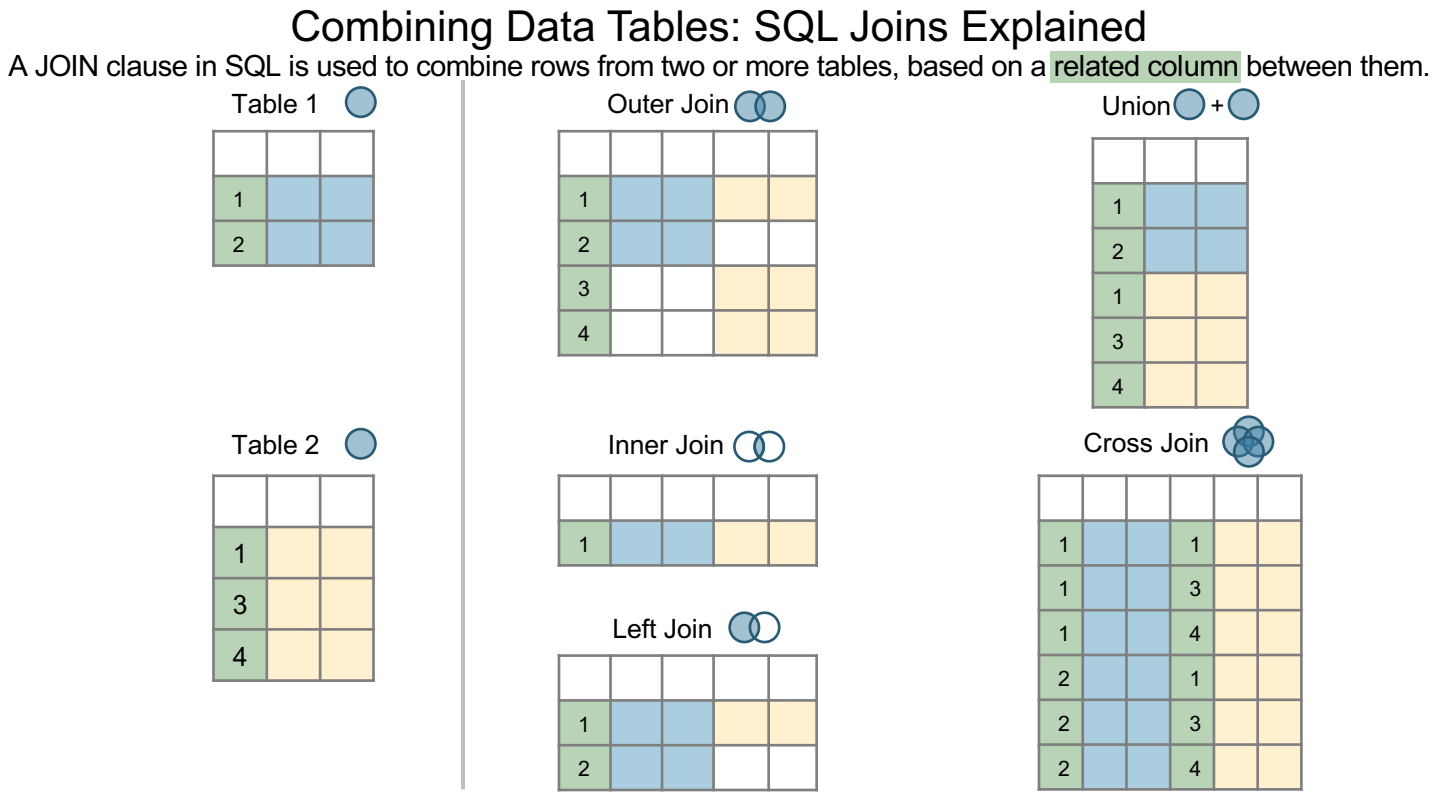
- Let's test the aggregation we built up to in a row and column-oriented table
- 100 million records

```
SELECT
product_id,
review_date,
review_rating,
ROUND(AVG(review_rating) OVER (
    ORDER BY product_id,review_date
    ROWS BETWEEN 3 PRECEDING
     AND CURRENT ROW
),2) AS average,
COUNT(review_rating) OVER (
    ORDER BY product_id,review_date
) AS cnt
FROM customer_reviews_{row,c}store
LIMIT 10
```

- Row-oriented
  - 300 seconds
- Column-oriented
  - 250 seconds
- 20% faster operation with the column-oriented table
- Performance will change drastically based on indexes defined on the tables and query types

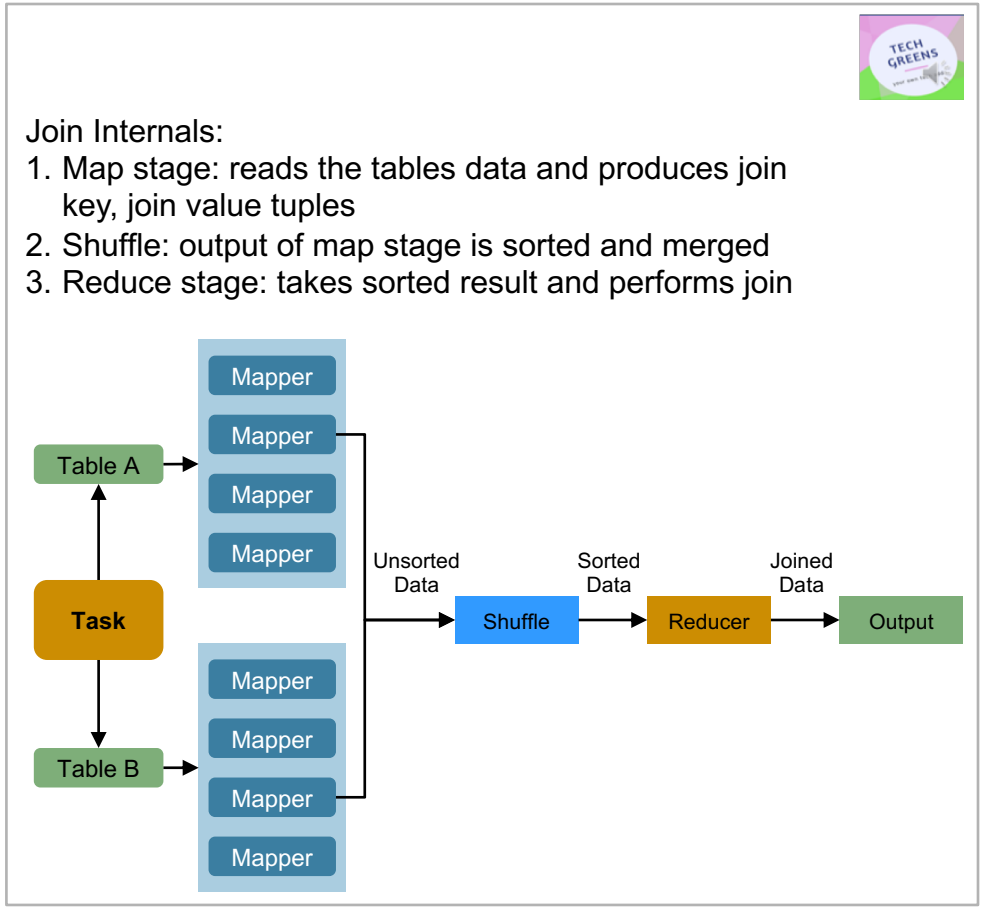
SQL: Joins

- Sometimes we need to combine data in multiple tables
- We do this with a join or union
- Let's talk through the diagram



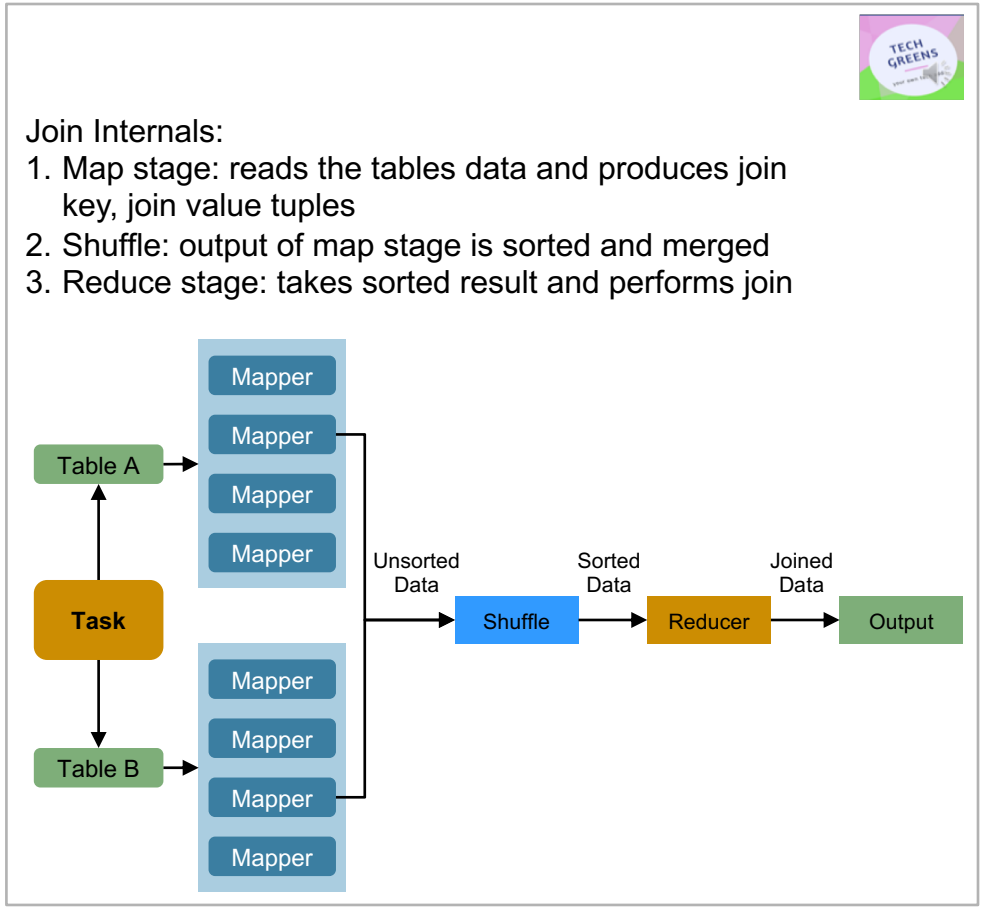
MapReduce: Reduce-Side Join

- Sometimes we encounter data that need to be joined together, such as:
  - Clickstream
  - User demographics
- Both of these might be rather large datasets for large companies
- Clickstream data are effectively unbounded and could be trillions of rows



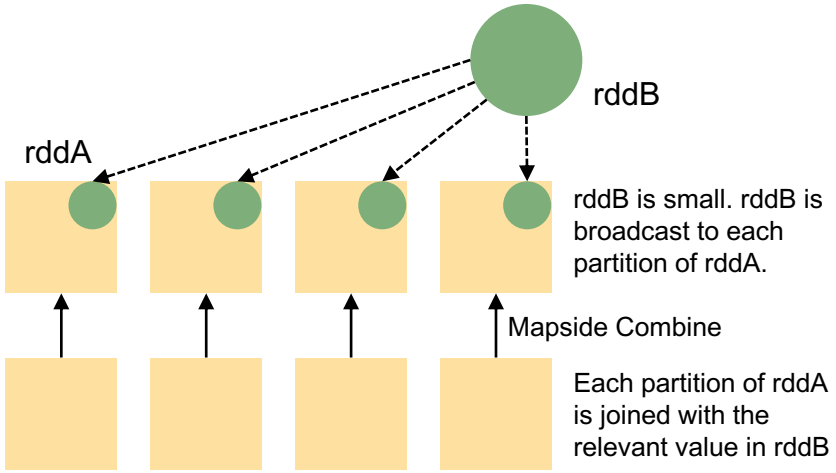
MapReduce: Reduce-Side Join

- User demographics for some companies might be in the billions (Netflix, Google, Facebook) but be smaller for other companies (startups).
- How do we join these datasets together efficiently?



MapReduce: Map-Side (Broadcast) Join

- The reduce-side join had a shuffle, and we know that's expensive now.
- What if our user data were small?
- We could share the user data with every executor.
  - In Spark, this is a broadcast.
- This removes the shuffle for the reduce phase and makes the joins significantly more efficient.
- We make join keys and do the lookup in our shared user data in one mapping phase.



The End

Summary

Summary

- We won’t be experts in a day.
- SQL has been around for more than 40 years and has a long history of use.
- When working with systems with extremely large amounts of data (1TB or more), sometimes you need to break away from the traditional database.
- Breaking away from traditional databases has some advantages and disadvantages.
- If you’re able to talk to the how/why of a use case, you can make a good design choice for technologies to use.

Data Systems and Pipelines: Outcomes

1. Understand the history of data systems and how they evolved into the current landscape.
2. Be able to articulate why a given class of databases should be used over another based on a use case.
3. Understand how traditional data warehouses were constructed and their weaknesses.
4. Understand how data serialization can affect performance.
5. Have a basic working knowledge of SQL and how joins work in the MapReduce framework.

The End