Advanced Data Management (CSCI 640/490)

Scalable Databases

Dr. David Koop
Data Integration: Combine Datasets with Different Data

[A. Doan et al., 2012]
Information Integration

Source A

Source B

Schema Mapping
Data Transformation
Duplicate Detection
Data Fusion

[<publication>
<title> Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases </title>
<author> Scheth & Larson </author>
<year> 1990 </year>
</publication>]

[L. Dong and F. Naumann, 2009]
Information Integration

[Source A]

<pub>
<Titel> Federated Database Systems </Titel>
<Autoren>
<Autor> Amit Sheth </Autor>
<Autor> James Larson </Autor>
</Autoren>
</pub>

Source B

<publication>
<title> Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases </title>
<author> Sheth & Larson </author>
<year> 1990 </year>
</publication>

Schema Mapping

Data Transformation

Duplicate Detection

Data Fusion

Schema Integration

[L. Dong and F. Naumann, 2009]
Information Integration

Source A

Source B

Transformation queries or views

Schema Mapping
Data Transformation
Duplicate Detection
Data Fusion

[L. Dong and F. Naumann, 2009]
Information Integration

Source A

Source B

<publication>
<title>Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases</title>
<author>Scheth & Larson</author>
<year>1990</year>
</publication>
"Duplicate Detection" has many Duplicates

[L. Dong and F. Naumann, 2009]
"Duplicate Detection" has many Duplicates

Household matching  Doubles  Duplicate detection
Mixed and split citation problem
Match
Deduplication
Entity resolution
Identity uncertainty
Hardening soft databases

Doubles
Fuzzy match
Approximate match
Entity clustering
Reference reconciliation
Reference matching
Merge/purge
Householding

Duplicate detection
Record linkage
Object identification
Object consolidation
Entity clustering
Reference reconciliation
Reference matching

[D. Koop, CSCI 640/490, Spring 2024]

[L. Dong and F. Naumann, 2009]
Record Linkage Process

Database A
- Data pre-processing
  - Indexing / Searching
    - Comparison
      - Classification
        - Matches
        - Non-matches
          - Evaluation
            - Potential Matches
              - Clerical Review
Record Linkage Techniques

- Deterministic matching
  - Rule-based matching (complex to build and maintain)

- Probabilistic record linkage [Fellegi and Sunter, 1969]
  - Use available attributes for linking (often personal information, like names, addresses, dates of birth, etc.)
  - Calculate match weights for attributes

- “Computer science” approaches
  - Based on machine learning, data mining, database, or information retrieval techniques
  - Supervised classification: Requires training data (true matches)
  - Unsupervised: Clustering, collective, and graph based
Information Integration

Source A

Source B

<pub>
<title>Federated Database Systems</title>
<Autoren>
<author>Amit Sheth</author>
<author>James Larson</author>
</Autoren>
</pub>

<pub>
<title>Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases</title>
<Autoren>
<author>Amit Sheth & Larson</author>
</Autoren>
<year>1990</year>
</pub>

Schema Mapping
Data Transformation
Duplicate Detection
Data Fusion

Preserve lineage

[L. Dong and F. Naumann, 2009]
Data Fusion

• Problem: Given a duplicate, create a single object representation while resolving conflicting data values.

• Difficulties:
  - Null values: Subsumption and complementation
  - Contradictions in data values
  - Uncertainty & truth: Discover the true value and model uncertainty in this process
  - Metadata: Preferences, recency, correctness
  - Lineage: Keep original values and their origin
  - Implementation in DBMS: SQL, extended SQL, UDFs, etc.
Conflict Resolution Strategies

- Conflict Ignorance: Pass it on
- Conflict Avoidance: Take the information no gossiping
- Conflict Resolution: Decide, mediate

Instance-based: Decide, meet in the middle
Metadata-based: Trust your friends, nothing is older than the news from yesterday

[L. Dong and F. Naumann, 2009]
Example Problem
## Example Problem

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stonebraker</td>
<td>MIT</td>
<td>Berkeley</td>
<td>MIT</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UWisc</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>AT&amp;T</td>
<td>BEA</td>
</tr>
<tr>
<td>Halevy</td>
<td>Google</td>
<td>Google</td>
<td>UW</td>
</tr>
</tbody>
</table>

[X L Dong et al., 2009]
## Naive Voting Works

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stonebraker</td>
<td>MIT</td>
<td>Berkeley</td>
<td>MIT</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UWisc</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>AT&amp;T</td>
<td>BEA</td>
</tr>
<tr>
<td>Halevy</td>
<td>Google</td>
<td>Google</td>
<td>UW</td>
</tr>
</tbody>
</table>

[X L Dong et al., 2009]
Naive Voting Only Works if Data Sources are Independent

[X L Dong et al., 2009]
Naive Voting Only Works if Data Sources are Independent

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stonebraker</td>
<td>MIT</td>
<td>Berkeley</td>
<td>MIT</td>
<td>MIT</td>
<td>MS</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UWisc</td>
<td>UWisc</td>
<td>UWisc</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>AT&amp;T</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Halevy</td>
<td>Google</td>
<td>Google</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
</tbody>
</table>

[X L Dong et al., 2009]
S4 and S5 copy from S3

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stonebraker</td>
<td>MIT</td>
<td>Berkeley</td>
<td>MIT</td>
<td>MIT</td>
<td>MS</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UWisc</td>
<td>UWisc</td>
<td>UWisc</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>AT&amp;T</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Halevy</td>
<td>Google</td>
<td>Google</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
</tbody>
</table>

[X L Dong et al., 2009]
S4 and S5 copy from S3

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stonebraker</td>
<td>MIT</td>
<td>Berkeley</td>
<td>MIT</td>
<td>MIT</td>
<td>NS</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UWisc</td>
<td>UWisc</td>
<td>UWisc</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>AT&amp;T</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Halevy</td>
<td>Google</td>
<td>Google</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
</tbody>
</table>

[X L Dong et al., 2009]
# Challenges in Dependence Discovery

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stonebraker</td>
<td>MIT</td>
<td>Berkeley</td>
<td>MIT</td>
<td>MIT</td>
<td>MS</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UWisc</td>
<td>UWisc</td>
<td>UWisc</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>AT&amp;T</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Halevy</td>
<td>Google</td>
<td>Google</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
</tbody>
</table>

[X L Dong et al., 2009]
Challenges in Dependence Discovery

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stonebraker</td>
<td>MIT</td>
<td>Berkeley</td>
<td>MIT</td>
<td>MIT</td>
<td>MS</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UWisc</td>
<td>UWisc</td>
<td>UWisc</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>AT&amp;T</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Halevy</td>
<td>Google</td>
<td>Google</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
</tbody>
</table>

[X L Dong et al., 2009]
Challenges in Dependence Discovery

2. With only a snapshot it is hard to decide which source is a copier.

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stonebraker</td>
<td>MIT</td>
<td>Berkeley</td>
<td>MIT</td>
<td>MIT</td>
<td>MS</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UWisc</td>
<td>UWisc</td>
<td>UWisc</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>AT&amp;T</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Halevy</td>
<td>Google</td>
<td>Google</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
</tbody>
</table>

[X L Dong et al., 2009]
Challenges in Dependence Discovery

1. Sharing common data does not in itself imply copying.

2. With only a snapshot it is hard to decide which source is a copier.

3. A copier can also provide or verify some data by itself, so it is inappropriate to ignore all of its data.

[X L Dong et al., 2009]
Ideas

• If two sources share a lot of false values, they are more likely to be dependent.

• S1 is more likely to copy from S2, if the accuracy of the common data is highly different from the accuracy of S1.
Combining Accuracy and Dependence

Truth Discovery

Source-accuracy Computation

Dependence Detection

[X L Dong et al., 2009]
Combining Accuracy and Dependence

Step 2

Truth Discovery

Source-accuracy Computation  Dependence Detection

Step 1

Step 3

[XL Dong et al., 2009]
## The Motivating Example

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stonebraker</td>
<td>MIT</td>
<td>Berkeley</td>
<td>MIT</td>
<td>MIT</td>
<td>MS</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UWisc</td>
<td>UWisc</td>
<td>UWisc</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>AT&amp;T</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Halevy</td>
<td>Google</td>
<td>Google</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
</tbody>
</table>

[XL Dong et al., 2009]
The Motivating Example

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round 1</td>
<td>.52</td>
<td>.42</td>
<td>.53</td>
<td>.53</td>
<td>.53</td>
</tr>
<tr>
<td>Round 2</td>
<td>.63</td>
<td>.46</td>
<td>.55</td>
<td>.55</td>
<td>.55</td>
</tr>
<tr>
<td>Round 3</td>
<td>.71</td>
<td>.52</td>
<td>.53</td>
<td>.53</td>
<td>.37</td>
</tr>
<tr>
<td>Round 4</td>
<td>.79</td>
<td>.57</td>
<td>.48</td>
<td>.48</td>
<td>.31</td>
</tr>
<tr>
<td>Round 11</td>
<td>.97</td>
<td>.61</td>
<td>.40</td>
<td>.40</td>
<td>.21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Confidence</th>
<th>Carey</th>
<th>Halevy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UCI</td>
<td>AT&amp;T</td>
</tr>
<tr>
<td>Round 1</td>
<td>1.61</td>
<td>1.61</td>
</tr>
<tr>
<td>Round 2</td>
<td>1.68</td>
<td>1.3</td>
</tr>
<tr>
<td>Round 3</td>
<td>2.12</td>
<td>1.47</td>
</tr>
<tr>
<td>Round 4</td>
<td>2.51</td>
<td>1.68</td>
</tr>
<tr>
<td>Round 11</td>
<td>4.73</td>
<td>2.08</td>
</tr>
</tbody>
</table>

[X L Dong et al., 2009]
Assignment 4

- Data Integration & Data Fusion
- Out soon
Paper Critique

• Read What’s Really New with NewSQL?
• Submit critique **before class** on Wednesday, March 20
• Discussion ideas:
  - What are the advantages or disadvantages of NewSQL vs NoSQL?
  - Are they really different from standard RDBMS?
  - Which category of NewSQL databases is most exciting?
Scalable Database Systems
Database Architecture

Data Analytics

- Analytics
- Data Warehouse

Operative Database

Applications

Reporting

Data Mining

[F. Gessert et al., 2017]
Database Architecture

Data Analytics
- Analytics
- Data Warehouse

Data Management
- Operative Database
- Applications

NoSQL

Reporting
Data Mining

[F. Gessert et al., 2017]
The era of one-size-fits-all database systems is over

→ Specialized data systems

[References: F. Gessert et al., 2017]
NoSQL Motivation

Scalability

User-generated data, Request load

Impedance Mismatch

ID
Customer
Line Item 1: ...
Line Item2: ...
Payment: Credit Card, ...
Orders
Payment
Customers

Two main motivations:
User-generated data, Request load
Payment: Credit Card, ...

[F. Gessert et al., 2017]
Introduction

Fig. 1.1 Main components of a DBMS.

At heart, a typical RDBMS has five main components, as illustrated in Figure 1.1. As an introduction to each of these components and the way they fit together, we step through the life of a query in a database system. This also serves as an overview of the remaining sections of the paper.

Consider a simple but typical database interaction at an airport, in which a gate agent clicks on a form to request the passenger list for a flight. This button click results in a single-query transaction that works roughly as follows:

1. The personal computer at the airport gate (the "client") calls an API that in turn communicates over a network to establish a connection with the Client Communications Manager of a DBMS (top of Figure 1.1). In some cases, this connection...
Relational Databases: One size fits all?

- Lots of work goes into relational database development:
  - B-trees
  - Cost-based query optimizers
  - ACID (Atomicity, Consistency, Isolation, Durability)
- Vendors largely stuck with this model from the 1980s through 2000s
- Having different systems leads to business problems:
  - cost problem
  - compatibility problem
  - sales problem
  - marketing problem

[Stonebraker and Çetinetmel, 2005]
ACID Transactions

- Make sure that transactions are processed reliably
- **Atomicity**: leave the database as is if some part of the transaction fails (e.g., don't add/remove only part of the data) using rollbacks
- **Consistency**: database moves from one valid state to another
- **Isolation**: concurrent execution matches serial execution
- **Durability**: endure hardware failures, make sure changes hit disk
How to Scale Relational Databases?
Shared Nothing Architecture

Shift towards higher distribution & less coordination:

- **Shared Memory** e.g. "Oracle 11g"
- **Shared Disk** e.g. "Oracle RAC"
- **Shared Nothing** e.g. "NoSQL"

[Source: F. Gessert et al., 2017]
TrafficDB: Shared-Memory Data Store

- Traffic-aware route planning
- Want up-to-date data for all
- Thousands of requests per second
  - High-Frequency Reads
  - Low-Frequency Writes
- "Data must be stored in a region of RAM that can be shared and efficiently accessed by several different application processes"

[R. Fernandes et al., 2016]
3.2 Shared-Nothing

A shared-nothing parallel system (Figure 3.2) is made up of a cluster of independent machines that communicate over a high-speed network interconnect or, increasingly frequently, over commodity networking components. There is no way for a given system to directly access the memory or disk of another system.

Shared-nothing systems provide no hardware sharing abstractions, leaving coordination of the various machines entirely in the hands of the DBMS. The most common technique employed by DBMSs to support these clusters is to run their standard process model on each machine, or node, in the cluster. Each node is capable of accepting client SQL requests in parallel. All three models run well on these systems and support the execution of multiple, independent SQL requests in parallel. The main challenge is to modify the query execution layers to take advantage of the ability to parallelize a single query across multiple CPUs; we defer this to Section 5.
Sharding

Collection I

1 TB

Shard A

Collection I

256 GB

Shard B

256 GB

Shard C

256 GB

Shard D

256 GB

[MongoDB]
Stonebraker: The End of an Architectural Era

- "RDBMSs were designed for the business data processing market, which is their sweet spot"
- "They can be beaten handily in most any other market of significant enough size to warrant the investment in a specialized engine"
- Changes in markets (science), necessary features (scalability), and technology (amount of memory)
- RDBMS Overhead: Logging, Latching, and Locking
- Relational model is not necessarily the answer
- SQL is not necessarily the answer
OLTP vs. OLAP

- Online Transactional Processing (OLTP) often used in business applications, data entry and retrieval transactions

- OLTP Examples:
  - Add customer's shopping cart to the database of orders
  - Find me all information about John Hammond's death

- OLTP is focused on the day-to-day operations while Online Analytical Processing (OLAP) is focused on analyzing that data for trends, etc.

- OLAP Examples:
  - Find the average amount spent by each customer
  - Find which year had the most movies with scientists dying
## Row Stores

A typical table structure for row stores includes id, scientist, death_by, and movie_name columns.

<table>
<thead>
<tr>
<th>id</th>
<th>scientist</th>
<th>death_by</th>
<th>movie_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reinhardt</td>
<td>Crew</td>
<td>The Black Hole</td>
</tr>
<tr>
<td>2</td>
<td>Tyrell</td>
<td>Roy Batty</td>
<td>Blade Runner</td>
</tr>
<tr>
<td>3</td>
<td>Hammond</td>
<td>Dinosaur</td>
<td>Jurassic Park</td>
</tr>
<tr>
<td>4</td>
<td>Soong</td>
<td>Lore</td>
<td>Star Trek: TNG</td>
</tr>
<tr>
<td>5</td>
<td>Morbius</td>
<td>The machine</td>
<td>Forbidden Planet</td>
</tr>
<tr>
<td>6</td>
<td>Dyson</td>
<td>SWAT</td>
<td>Terminator 2: Judgment Day</td>
</tr>
</tbody>
</table>

[Primary Key]

Row Stores

Primary Key

Row

Row Stores

D. Koop, CSCI 640/490, Spring 2024

J. Swanhart, Introduction to Column Stores
Inefficiency in Row Stores for OLAP

\[
\text{select sum(metric) as the\_sum from fact}
\]

1. Storage engine gets \textit{a whole row} from the table

2. SQL interface extracts only requested portion, adds it to \textit{“the\_sum”}

3. IF all rows scanned, send results to client, else GOTO 1

[J. Swanhart, Introduction to Column Stores]
### Column Stores

Each column has a file or segment on disk

<table>
<thead>
<tr>
<th>id</th>
<th>Title</th>
<th>Person</th>
<th>Genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mrs. Doubtfire</td>
<td>Robin Williams</td>
<td>Comedy</td>
</tr>
<tr>
<td>2</td>
<td>Jaws</td>
<td>Roy Scheider</td>
<td>Horror</td>
</tr>
<tr>
<td>3</td>
<td>The Fly</td>
<td>Jeff Goldblum</td>
<td>Horror</td>
</tr>
<tr>
<td>4</td>
<td>Steel Magnolias</td>
<td>Dolly Parton</td>
<td>Drama</td>
</tr>
<tr>
<td>5</td>
<td>The Birdcage</td>
<td>Nathan Lane</td>
<td>Comedy</td>
</tr>
<tr>
<td>6</td>
<td>Erin Brokovich</td>
<td>Julia Roberts</td>
<td>Drama</td>
</tr>
</tbody>
</table>

---

*J. Swanhart, Introduction to Column Stores*
## Horizontal Partitioning vs. Vertical Partitioning

### Original Table

<table>
<thead>
<tr>
<th>CUSTOMER ID</th>
<th>FIRST NAME</th>
<th>LAST NAME</th>
<th>FAVORITE COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAEKO</td>
<td>OHNUKI</td>
<td>BLUE</td>
</tr>
<tr>
<td>2</td>
<td>O.V.</td>
<td>WRIGHT</td>
<td>GREEN</td>
</tr>
<tr>
<td>3</td>
<td>SELDA</td>
<td>BAĞCAN</td>
<td>PURPLE</td>
</tr>
<tr>
<td>4</td>
<td>JIM</td>
<td>PEPPER</td>
<td>AUBERGINE</td>
</tr>
</tbody>
</table>
## Horizontal Partitioning vs. Vertical Partitioning

### Vertical Partitions

<table>
<thead>
<tr>
<th>VP1</th>
<th>VP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMER ID</td>
<td>FIRST NAME</td>
</tr>
<tr>
<td>1</td>
<td>TAEKO</td>
</tr>
<tr>
<td>2</td>
<td>O.V.</td>
</tr>
<tr>
<td>3</td>
<td>SELDA</td>
</tr>
<tr>
<td>4</td>
<td>JIM</td>
</tr>
</tbody>
</table>

### Original Table

<table>
<thead>
<tr>
<th>CUSTOMER ID</th>
<th>FIRST NAME</th>
<th>LAST NAME</th>
<th>FAVORITE COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAEKO</td>
<td>OHNUKI</td>
<td>BLUE</td>
</tr>
<tr>
<td>2</td>
<td>O.V.</td>
<td>WRIGHT</td>
<td>GREEN</td>
</tr>
<tr>
<td>3</td>
<td>SELDA</td>
<td>BAĞCAN</td>
<td>PURPLE</td>
</tr>
<tr>
<td>4</td>
<td>JIM</td>
<td>PEPPER</td>
<td>AUBERGINE</td>
</tr>
</tbody>
</table>

### Horizontal Partitions

<table>
<thead>
<tr>
<th>HP1</th>
<th>HP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMER ID</td>
<td>FIRST NAME</td>
</tr>
<tr>
<td>1</td>
<td>TAEKO</td>
</tr>
<tr>
<td>2</td>
<td>O.V.</td>
</tr>
<tr>
<td>3</td>
<td>SELDA</td>
</tr>
<tr>
<td>4</td>
<td>JIM</td>
</tr>
</tbody>
</table>

[M. Drake]
NoSQL Paradigm Shift

- Commercial DBMS
- Specialized DB hardware (Oracle Exadata, etc.)
- Highly available network (Infiniband, Fabric Path, etc.)
- Highly Available Storage (SAN, RAID, etc.)

- Open-Source DBMS
- Commodity hardware
- Commodity network (Ethernet, etc.)
- Commodity drives (standard HDDs, JBOD)

[F. Gessert et al., 2017]
Problems with Relational Databases

ID: 1001
Customer: Ann

Line Items:
- 0321293533 2 $48 $96
- 0321601912 1 $39 $39
- 0131495054 1 $51 $51

Payment Details:
Card: Amex  
CC Number: 12345  
Expiry: 04/2001

Orders
Customers
Order Lines
Credit Cards

[P. Sadalage]
NoSQL Classification Criteria

**Data Model**
- Key-Value
- Wide-Column
- Document
- Graph

**Consistency/Availability Trade-Off**
- **AP**: Available & Partition Tolerant
- **CP**: Consistent & Partition Tolerant
- **CA**: Not Partition Tolerant

---

[F. Gessert et al., 2017]
Key-Value Stores

- **Data model:** (key) -> value
- **Interface:** CRUD (Create, Read, Update, Delete)

Examples:
- `users:2:friends` -> `{23, 76, 233, 11}`
- `users:2:inbox` -> `[234, 3466, 86, 55]`
- `users:2:settings` -> Theme → "dark", cookies → "false"

Value: An opaque blob

[F. Gessert et al., 2017]
Key-Value Stores

- Always use primary-key access
- Operations:
  - Get/put value for key
  - Delete key
Wide-Column Stores

- **Data model:** (rowkey, column, timestamp) -> value
- **Interface:** CRUD, Scan

Examples: Cassandra (AP), Google BigTable (CP), HBase (CP)

(F. Gessert et al., 2017)
Column Stores

- Instead of having rows grouped/sharded, we group columns
- …or families of columns
- Put similar columns together
**Document Stores**

- **Data model:** (collection, key) -> document
- **Interface:** CRUD, Querys, Map-Reduce

**ID/Key**

```
order-12338
```

**JSON Document**

```
{
  order-id: 23,
  customer: { name: "Felix Gessert", age: 25 },
  line-items: [ {product-name: "x"}, ... ]
}
```

- **Examples:** CouchDB (AP), RethinkDB (CP), MongoDB (CP)

[F. Gessert et al., 2017]
Document Stores

- Documents are the main entity
  - Self-describing
  - Hierarchical
  - Do not have to be the same
- Could be XML, JSON, etc.
- Key-value stores where values are "examinable"
- Can have query language and indices overlaid

```json
<Key=CustomerId>
{
  "customerid": "fc986e48ca6",
  "customer":
  {
    "firstname": "Pramod",
    "lastname": "Sadalage",
    "company": "ThoughtWorks",
    "likes": [ "Biking", "Photography"
  ]
  
  "billingaddress":
  {
    "state": "AK",
    "city": "DILLINGHAM",
    "type": "R"
  }
}
```
Graph Databases

- **Data model**: $G = (V, E)$: Graph-Property Modell
- **Interface**: Traversal algorithms, queries, transactions

- **Examples**: Neo4j (CA), InfiniteGraph (CA), OrientDB (CA)

[D. Koop, CSCI 640/490, Spring 2024]
Graph Databases

- Focus on entities and relationships
- Edges may have properties
- Relational databases required a set traversal
- Traversals in Graph DBs are faster
NoSQL Classification Criteria

Data Model
- Key-Value
- Wide-Column
- Document
- Graph

Consistency/Availability Trade-Off
- **AP**: Available & Partition Tolerant
- **CP**: Consistent & Partition Tolerant
- **CA**: Not Partition Tolerant

[F. Gessert et al., 2017]
CAP Theorem

[Diagram showing the CAP Theorem]

- **A** (Availability): Remains accessible and operational at all times.
- **C** (Consistency): Commits are atomic across the entire distributed system.
- **P** (Partition Tolerance): Only a total network failure can cause the system to respond incorrectly.

The diagram illustrates that you cannot have all three properties (A, C, P) simultaneously. You must pick exactly two of these.

Traditional relational databases: PostgreSQL, MySQL, etc.

- **CA**: Voldemort, Riak, Cassandra, CouchDB, Dynamo-like systems

- **CP**: HBase, MongoDB, Redis, MemcacheDB, BigTable-like systems

[Image credits: E. Brewer]
CAP Theorem

- Consistency: every read would get you the most recent write
- Availability: every node (if not failed) always executes queries
- Partition tolerance: system continues to work even if nodes are down
- Theorem (Brewer): It is impossible for a distributed data store to simultaneously provide more than two of Consistency, Availability, and Partition Tolerance
CAP Theorem "Proof"

- If there is a network partition, one of consistency or availability will not be possible.

If there is a network partition, one of consistency or availability will not be possible. The CAP Theorem states that it is not possible to achieve all three of the following properties simultaneously:

- Consistency
- Availability
- Partition tolerance

In any distributed system, at least two of these three properties must be compromised. The theorem is illustrated with a network partition, showing how replication can block responses until an ACK arrives to maintain consistency, while responses before successful replication can lead to availability. The diagram shows two nodes, N1 and N2, with values V1 and V0, respectively, and a replication failure indicating that the network is partitioned.
NoSQL Techniques

Functional Techniques

- Scan Queries
- ACID Transactions
- Conditional or Atomic Writes
- Joins
- Sorting
- Filter Queries
- Full-text Search
- Aggregation and Analytics

Non-Functional Techniques

- Sharding: Range-Sharding, Hash-Sharding, Entity-Group Sharding, Consistent Hashing, Shared-Disk
- Replication: Commit/Consensus Protocol, Synchronous, Asynchronous, Primary Copy, Update Anywhere
- Storage Management: Logging, Update-in-Place, Caching, In-Memory Storage, Append-Only Storage
- Query Processing: Global Secondary Indexing, Local Secondary Indexing, Query Planning, Analytics Framework, Materialized Views

Non-Functional Techniques

- Data Scalability
- Write Scalability
- Read Scalability
- Elasticity
- Consistency
- Write Latency
- Read Latency
- Write Throughput
- Read Availability
- Write Availability
- Durability

[F. Gessert et al., 2017]
Distributing Data

- Aggregate-oriented databases
- Sharding (horizontal partitioning): Sharding distributes different data across multiple servers, so each server acts as the single source for a subset of data
- Replication: Replication copies data across multiple servers, so each bit of data can be found in multiple places. Replication comes in two forms,
  - Source-replica replication makes one node the authoritative copy that handles writes, replica synchronizes with the source and may handle reads.
  - Peer-to-peer replication allows writes to any node; the nodes coordinate to synchronize their copies of the data.
Sharding

Collection I

1 TB

Shard A

Collection I

256 GB

Shard B

256 GB

Shard C

256 GB

Shard D

256 GB

[MongoDB]
Sharding Approaches

- Hash-based Sharding
  - Hash of data values (e.g. key) determines partition (shard)
  - Pro: Even distribution, Con: No data locality

[D. DeWitt & J. Gray, 1992, via F. Gessert, Image: MongoDB]
Sharding Approaches

- Range-based Sharding
  - Assigns ranges defined over fields (shard keys) to partitions
  - Pro: Enables Range Scans & Sorting, Con: Repartitioning/balancing req'd

[D. DeWitt & J. Gray, 1992, via F. Gessert, Image: MongoDB]
Sharding Approaches

- **Entity-Group Sharding**
  - Explicit data co-location for single-node-transactions
  - Pro: Enables ACID Transactions, Con: Partitioning not easily changable

[D. DeWitt & J. Gray, 1992, via F. Gessert, Image: J. Kim]
Replication

- Store $N$ copies of each data item
- Consistency model: synchronous vs. asynchronous
- Coordination: Multiple Primary, Primary/Replica

\[ \text{DB Node} \xrightarrow{\text{synchronous/ asynchronous}} \text{DB Node} \xrightarrow{\text{synchronous/ asynchronous}} \text{DB Node} \]

[F. Gessert et al., 2017]
Replication: When

- Asynchronous (lazy)
  - Writes are acknowledged immediately
  - Performed through log shipping or update propagation
  - Pro: Fast writes, no coordination needed
  - Con: Replica data potentially stale (inconsistent)

- Synchronous (eager)
  - The node accepting writes synchronously propagates updates/transactions before acknowledging
  - Pro: Consistent
  - Con: needs a commit protocol (more roundtrips), unavailable under certain network partitions
Replication: Where

- **Primary-Replica (Primary Copy)**
  - Only a dedicated primary is allowed to accept writes, replicas are read-replicas
  - Pro: reads from the primary are consistent
  - Con: primary is a bottleneck and SPOF

- **Multi-Primary (Update anywhere)**
  - The server node accepting the writes synchronously propagates the update or transaction before acknowledging
  - Pro: fast and highly-available
  - Con: either needs coordination protocols (e.g. Paxos) or is inconsistent

[F. Gessert et al., 2017]
Consistency Levels

- **Linearity**
  - Causal Consistency
    - Writes Follow Reads
    - Read Your Writes
    - Monotonic Reads
    - Monotonic Writes
  - PRAM
    - Bounded Staleness

[via F. Gessert et al., 2017]
Next Class's Paper Critique

• Read What’s Really New with NewSQL?
• Submit critique before class on Wednesday, March 20
• Discussion ideas:
  - What are the advantages or disadvantages of NewSQL vs NoSQL?
  - Are they really different from standard RDBMS?
  - Which category of NewSQL databases is most exciting?