Advanced Data Management (CSCI 490/680)

Scalable Database Systems

Dr. David Koop
Course Updates

• We want to help you continue learning… but we also realize that COVID-19 has changed a lot in our lives

• I understand that this is not class as normal

• If you have any circumstances that impact coursework, please contact me
Course Updates

- Office hours/Discussion/Questions: Blackboard Ultra Collaborate
  - 1:00-2:30pm on Tuesdays and Thursdays
- Lectures: Online, recorded videos on Blackboard
- Discussions/Questions: Either Blackboard Ultra Collaborate or Discussions
- Reading Responses: Same as before (turn in via Blackboard)
- Assignments: Same as before (turn in via Blackboard)
- Reading Quizzes: Online via Blackboard
- Test 2 & Final Exam: Online via Blackboard
Where have we been?

- Focused on how to deal with data
- Topics
  - Tools: Python & pandas
  - Understanding Data
  - Data wrangling, cleaning, and transformation
  - Data integration & fusion
  - Data exploration & visualization
Where are we going?

• Topics:
  - Scalable databases
  - Data curation
  - Different dataset types: graphs, time series, and spatial data
  - Provenance and reproducibility
  - Data Management & Machine Learning
Assignment 4

- COVID-19 data
- Data Integration
  - Population
  - Temperature
- Data Fusion:
  - Our World in Data
  - Johns Hopkins
  - Wikipedia
- Questions?
Reading Quiz

• Reading for Thursday:
  - Spanner: Google's Globally-Distributed Database

• Before watching Thursday's lecture, take Blackboard quiz on the reading

• Quiz will focus on key concepts not the details
Data Discovery and Visualization
Goal of Dataset Search: Accurate (A) vs. Timely (B)

[Chapman et al., 2020]
Goods: Organizing Google's Datasets

- Tool for Google to help its employees find internal data
- Keep data where it is, how it is, but extract metadata to aid search
- Challenges:
  - Dataset size and scale: >26 billion datasets
  - Variety: formats (text, csv, Bigtable), storage (GoogleFS, db server)
  - Churn: ~5% of datasets deleted each day
  - Metadata uncertainty: protocol buffers, primary key identification
  - Computing importance: need to understand users
  - Recovering semantics: understanding the data aids metadata extraction

[Halevy et al., 2016]
Goods: Organizing Google's Datasets

The Goods dataset catalog collects metadata about datasets from various storage systems as well as other sources. We infer metadata by processing additional sources such as logs and information about dataset owners and their projects, by analyzing content of the datasets, and by collecting input from the Goods users. We use the information in the catalog to build tools for search, monitoring, and visualizing flow of data.

Based on the information in its catalog, Goods provides a dashboard for the NLU team (in this case, dataset producers), which displays all their datasets and enables browsing them by facets (e.g., owner, data center, schema). Even if the team's datasets are in diverse storage systems, the engineers get a unified view of all their datasets and dependencies among them. Goods can monitor features of the dataset, such as its size, distribution of values in its contents, or its availability, and then alert the owners if the features change unexpectedly.

Another important piece of information that Goods provides is the dataset provenance: namely, the information about which datasets were used to create a given dataset (upstream datasets), and those that rely on it (downstream datasets). Note that both the upstream and downstream datasets may be created by other teams. When an engineer in the NLU team observes a problem with a dataset, she can examine the provenance visualization to determine whether a change in some upstream dataset had caused the problem. Similarly, if the team is about to make a significant change to its pipeline or has discovered a bug in an existing dataset that other teams have consumed already, they can quickly notify those affected by the problem.

From the perspective of dataset consumers, such as those not part of the NLU team in our example, Goods provides a search engine over all the datasets in the company, plus facets for narrowing search results, to find the most up-to-date or potentially important datasets. Goods presents a profile page for every dataset, which helps users unfamiliar with the data to understand its schema and to create boilerplate code to access and query the data. The profile page also contains the information on datasets with content similar to the content of the current dataset. The similarity information may enable novel combinations of datasets: for example, if two datasets share a primary key column, then they may provide complementary information and are therefore a good candidate for joining.

Goods allows users to expand the catalog with crowd-sourced metadata. For instance, dataset owners can annotate datasets with descriptions, in order to help users figure out which datasets are appropriate for their use (e.g., which analysis techniques are used in certain datasets and which pitfalls to watch out for). Dataset auditors can tag datasets that contain sensitive information and alert dataset owners or prompt a review to ensure that the data is handled appropriately. In this manner, Goods and its catalog become a hub through which users can share and exchange information about the generated datasets. Goods also exposes an API through which teams can contribute metadata to the catalog both for the teams own restricted use as well as to help other teams and users understand their datasets easily.

As we discuss in the rest of the paper, we addressed many challenges in designing and building Goods, arising from the sheer number of datasets (tens of billions in our case), the high churn in terms of updates, the sizes of individual datasets (gigabytes or terabytes in many cases), the many different data formats and stores they reside in, and the varying quality and importance of information collected about each dataset. Many of the challenges that we addressed in Goods were precipitated by the scale and characteristics of the data lake at Google. However, we believe that our experience and the lessons that we learned will apply to similar systems in other enterprises.

[Halevy et al., 2016]
Figure 1: An overview of the Dataset Search components. Google crawler collects the metadata from the Web; Dataset Search backend normalizes and reconciles the metadata; we then index the reconciled metadata and rank results for user queries.

For a set of triples from each page, we traverse the graph to collect all the properties and related objects for each dataset in a protocol buffer. A dataset record can point to other records such as organizations that provided a dataset or a record describing the distribution of a dataset. A single Web page can have multiple dataset records on it.

The specification of the graph traversal captures the mapping from Schema.org and DCAT vocabularies to the corresponding elements in the protocol buffer definition (e.g., example fields in Figure 2). The schema of the protocol buffer for the metadata largely corresponds to http://schema.org/Dataset and therefore the transformation of metadata at this stage is rather small.

To improve scalability, we use the graph query independently on the triples from each individual page rather than try to extract information from a graph that includes all metadata triples on the Web. Because the links across different pages must specify objects on another page directly through a URL (e.g., a provider of this dataset on page A is described on page B), we can do this reconciliation post-hoc. So, essentially, each page corresponds to its own, possibly disconnected graph. At the same time, doing graph traversal only for a single page is dramatically more scalable.

The information that we extract through graph traversal constitutes the raw metadata, metadata that closely mimics the structure of Schema.org properties on the original page.

In the next few steps, we describe how we create reconciled metadata for each dataset, accounting for the different levels of quality and variety of the modeling patterns used.

5.2 Normalizing and cleaning the metadata

As we mentioned in Section 4.1, we must assume that we will encounter every possible misuse and mis-interpretation of Schema.org properties when we operate at the scale of the whole Web. Thus, we perform a number of operations to normalize and clean up the metadata.

First, for the properties where we observe different patterns on the Web, we analyze the common patterns used and try to account for all of them. For instance Figure 2 shows the different patterns that we observed for defining downloads and distribution. In the figure, the first example of raw metadata defines the format of the dataset (CSV) at the level of the dataset itself and stores the download URL as the value of the http://schema.org/distribution property. Other examples in the figure deal with these two pieces of information differently. All these patterns are commonly used in our corpus. We mine these patterns by traversing either the initial graph or the resulting protocol buffer. Once we identify the patterns, we write adapters to convert all of them into the same modeling pattern in the reconciled metadata record. The right-hand side of Figure 2 shows this reconciled result.

Similarly, we have developed adapters for other metadata fields: We understand a lot more representations of dates than the ISO standard required by the Schema.org specification (Section 4.1. We will pick up digital object identifiers (DOIs) for a dataset from a variety of fields, and not just http://schema.org/identifier. We will use a uniform field, provider, for the many different fields that dataset providers used to identify this property. As we collect more metadata, our set of such adapters grows. Our decisions in these steps are guided by two factors: (1) the frequent usage patterns that we observed in the data; and (2) our understanding of what we expect the users to see in Dataset Search results.

[N. Noy et al., 2019]
Requirements

• System must be **open** so new providers can add their own datasets
• Search is over **metadata** (a provider may require users to pay/create account)
• Metadata must be published by the data publishers themselves, adhering to a **standard**

[N. Noy et al., 2019]
Challenges

- Metadata Quality: providers don't adhere to the specs
- Metadata Duplication in Search Results: search results vs. profile pages
- Dataset Replication and Provenance: identify replicas across providers
- Churn and Stale Sites:
  - 3% deleted, 7-10% added per day
  - standard web crawlers check high-traffic sites more often
- Ranking/Relevance: data citation might help
- Multiple Dataset-Metadata Standards: schema.org vs DCAT

[N. Noy et al., 2019]
Data Visualization
What does it mean to summarize data?

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>country</th>
<th>primary_fuel</th>
<th>capacity_mw</th>
<th>generation_gwh</th>
<th>latitude</th>
<th>longitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PHILIPPSBURG-2</td>
<td>DEU</td>
<td>Nuclear</td>
<td>1468.000</td>
<td>12763.886134</td>
<td>49.2529</td>
<td>8.4364</td>
</tr>
<tr>
<td>1</td>
<td>Kvinen</td>
<td>NOR</td>
<td>Hydro</td>
<td>90.000</td>
<td>403.532191</td>
<td>58.9281</td>
<td>7.0838</td>
</tr>
<tr>
<td>2</td>
<td>Torrance Farm Wind Park Extension</td>
<td>GBR</td>
<td>Wind</td>
<td>5.700</td>
<td>10.342942</td>
<td>55.8766</td>
<td>-3.7452</td>
</tr>
<tr>
<td>3</td>
<td>Nam Phom</td>
<td>THA</td>
<td>Gas</td>
<td>710.000</td>
<td>2906.785455</td>
<td>16.6820</td>
<td>102.7422</td>
</tr>
<tr>
<td>4</td>
<td>Winterset</td>
<td>USA</td>
<td>Oil</td>
<td>13.600</td>
<td>0.103000</td>
<td>41.3372</td>
<td>-94.0126</td>
</tr>
<tr>
<td>5</td>
<td>Twin Elm Solar Farm</td>
<td>GBR</td>
<td>Solar</td>
<td>1.000</td>
<td>0.499932</td>
<td>51.3580</td>
<td>-2.7959</td>
</tr>
<tr>
<td>6</td>
<td>Tesoro Hawaii</td>
<td>USA</td>
<td>Oil</td>
<td>20.000</td>
<td>156.520696</td>
<td>21.3032</td>
<td>-158.0914</td>
</tr>
<tr>
<td>7</td>
<td>Prospector Windfarm (Burnt River)</td>
<td>USA</td>
<td>Wind</td>
<td>10.000</td>
<td>1.774076</td>
<td>44.4181</td>
<td>-117.2550</td>
</tr>
<tr>
<td>8</td>
<td>Caterpillar</td>
<td>USA</td>
<td>Oil</td>
<td>13.300</td>
<td>33.826000</td>
<td>40.4175</td>
<td>-86.8447</td>
</tr>
<tr>
<td>9</td>
<td>Bourouge</td>
<td>FRA</td>
<td>Biomass</td>
<td>8.862</td>
<td>34.817396</td>
<td>47.5647</td>
<td>6.9062</td>
</tr>
<tr>
<td>10</td>
<td>CED Ducor 2</td>
<td>USA</td>
<td>Solar</td>
<td>20.000</td>
<td>8.918553</td>
<td>35.8442</td>
<td>-119.0681</td>
</tr>
<tr>
<td>11</td>
<td>Edenderry</td>
<td>IRL</td>
<td>Oil</td>
<td>117.600</td>
<td>19.196229</td>
<td>53.2902</td>
<td>-7.0845</td>
</tr>
<tr>
<td>12</td>
<td>Kangaroo Valley (Shoalhaven Scheme)</td>
<td>AUS</td>
<td>Hydro</td>
<td>160.000</td>
<td>140.743389</td>
<td>-34.7230</td>
<td>150.4795</td>
</tr>
<tr>
<td>13</td>
<td>Wheelabrator Baltimore Refuse</td>
<td>USA</td>
<td>Waste</td>
<td>64.500</td>
<td>336.882000</td>
<td>39.2660</td>
<td>-76.6297</td>
</tr>
<tr>
<td>14</td>
<td>Guangxi Yulin Darongshan Wind</td>
<td>CHN</td>
<td>Wind</td>
<td>25.000</td>
<td>76.523288</td>
<td>22.8722</td>
<td>110.2332</td>
</tr>
<tr>
<td>15</td>
<td>Bernards Solar</td>
<td>USA</td>
<td>Solar</td>
<td>3.700</td>
<td>1.649932</td>
<td>40.7027</td>
<td>-74.5816</td>
</tr>
<tr>
<td>16</td>
<td>Manor Farm - Shripney</td>
<td>GBR</td>
<td>Solar</td>
<td>5.000</td>
<td>2.499661</td>
<td>50.8148</td>
<td>-0.6825</td>
</tr>
<tr>
<td>17</td>
<td>Saint-Cirgues-en-Montagne</td>
<td>FRA</td>
<td>Wind</td>
<td>18.450</td>
<td>35.116073</td>
<td>44.7679</td>
<td>4.0883</td>
</tr>
<tr>
<td>18</td>
<td>Turweston Solar Farm</td>
<td>GBR</td>
<td>Solar</td>
<td>16.700</td>
<td>8.348868</td>
<td>52.0328</td>
<td>-1.1053</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Counts & Stats?

<table>
<thead>
<tr>
<th>primary_fuel</th>
<th>num_plants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hydro</td>
<td>7155</td>
</tr>
<tr>
<td>Solar</td>
<td>5929</td>
</tr>
<tr>
<td>Wind</td>
<td>5188</td>
</tr>
<tr>
<td>Gas</td>
<td>3922</td>
</tr>
<tr>
<td>Coal</td>
<td>2390</td>
</tr>
<tr>
<td>Oil</td>
<td>2290</td>
</tr>
<tr>
<td>Biomass</td>
<td>1396</td>
</tr>
<tr>
<td>Waste</td>
<td>1087</td>
</tr>
<tr>
<td>Nuclear</td>
<td>198</td>
</tr>
<tr>
<td>Geothermal</td>
<td>189</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>country</th>
<th>num_plants</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>8686</td>
</tr>
<tr>
<td>CHN</td>
<td>3041</td>
</tr>
<tr>
<td>GBR</td>
<td>2536</td>
</tr>
<tr>
<td>BRA</td>
<td>2340</td>
</tr>
<tr>
<td>FRA</td>
<td>2017</td>
</tr>
<tr>
<td>CAN</td>
<td>1154</td>
</tr>
<tr>
<td>DEU</td>
<td>982</td>
</tr>
<tr>
<td>IND</td>
<td>861</td>
</tr>
<tr>
<td>ESP</td>
<td>614</td>
</tr>
<tr>
<td>RUS</td>
<td>505</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>generation_gwh</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>806.709784</td>
</tr>
<tr>
<td>median</td>
<td>47.308833</td>
</tr>
<tr>
<td>std</td>
<td>3696.765694</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>capacity_mw</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>186.294810</td>
</tr>
<tr>
<td>median</td>
<td>18.900000</td>
</tr>
<tr>
<td>std</td>
<td>525.703572</td>
</tr>
</tbody>
</table>
Visual Summary
Visual Summary
Visual Summary Problems

• Too much data
  - Cannot display all of it without overlap (occlusion)
  - A limited number of pixels
  - A limited amount of human-resolvable resolution

• Prioritizing the display data is non-trivial
  - Show a lot of tiny power plants and occlude
  - Show only the big power plants
Visual Information-Seeking Mantra

• Overview First
• Zoom & Filter
• Details on Demand

—Schneiderman, 1996
Visual Summarization Projects

Graph Collection Summaries

Graph Summarization

Map Summarization

Trajectory Summarization
Scalable Database Systems
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 involves the Catalog Manager, Memory Manager, Administration, Monitoring & Utilities, Replication and Loading Services, Batch Utilities, Shared Components and Utilities (Section 7).

Relational Database Architecture

[Hellerstein et al., Architecture of a Database System]
How to Scale Relational Databases?
3.3 Shared-Disk

A shared-disk parallel system (Figure 3.3) is one in which all processors can access the disks with about the same performance, but are unable to access each other’s RAM. This architecture is quite common with two prominent examples being Oracle RAC and DB2 for zSeries SYMPLEX. Shared-disk has become more common in recent years with the increasing popularity of Storage Area Networks (SAN). A SAN allows one or more logical disks to be mounted by one or more host systems making it easy to create shared disk configurations.

One potential advantage of shared-disk over shared-nothing systems is their lower cost of administration. DBAs of shared-disk systems do not have to consider partitioning tables across machines in order to achieve parallelism. But very large databases still typically do require partitioning so, at this scale, the difference becomes less pronounced.

Another compelling feature of the shared-disk architecture is that the failure of a single DBMS processing node does not affect the other nodes’ ability to access the entire database. This is in contrast to both shared-memory systems that fail as a unit, and shared-nothing systems that lose access to at least some data upon a node failure (unless some alternative data redundancy scheme is used). However, even with these advantages, shared-disk systems are still vulnerable to some single node failure.

[Figure 3.3 Shared-disk architecture.]

[Hellerstein et al., Architecture of a Database System]
A shared-disk parallel system (Figure 3.3) is one in which all processors can access the disks with about the same performance, but are unable to access each other's RAM. This architecture is quite common with two prominent examples being Oracle RAC and DB2 for zSeries Systmplex. Shared-disk has become more common in recent years with the increasing popularity of Storage Area Networks (SAN). A SAN allows one or more logical disks to be mounted by one or more host systems making it easy to create shared disk configurations.

One potential advantage of shared-disk over shared-nothing systems is their lower cost of administration. DBAs of shared-disk systems do not have to consider partitioning tables across machines in order to achieve parallelism. But very large databases still typically do require partitioning so, at this scale, the difference becomes less pronounced.

Another compelling feature of the shared-disk architecture is that the failure of a single DBMS processing node does not affect the other nodes' ability to access the entire database. This is in contrast to both shared-memory systems that fail as a unit, and shared-nothing systems that lose access to at least some data upon a node failure (unless some alternative data redundancy scheme is used). However, even with these advantages, shared-disk systems are still vulnerable to some single failure.

[Figure 3.3 Shared-disk architecture.

[Hellerstein et al., Architecture of a Database System]
Parallel DB Architecture: Shared Memory

The dominant cost for DBMS customers is typically paying qualified people to administer high-end systems. This includes Database Administrators (DBAs) who configure and maintain the DBMS, and System Administrators who configure and maintain the hardware and operating systems. Today, nearly all serious database deployments involve multiple processors, with each processor having more than one CPU. DBMS architectures need to be able to fully exploit this potential parallelism. Fortunately, all three of the DBMS architectures described in Section 2 run well on modern shared-memory hardware architectures.
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"

[Diagram showing shared memory and 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. (Hellerstein et al., Architecture of a Database System)
Sharding

![Diagram of sharding in MongoDB](MongoDB)
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 have stuck with this model since the 1980s
• 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
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
### Row Stores

<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

Row Stores

[If you mentioned any specific information, such as a website or publication, please include that here.]

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

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

1. Storage engine gets a whole row from the table

2. SQL interface extracts only requested portion, adds it to “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>

### 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>

*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>

[M. Drake]
Problems with Relational Databases
NoSQL: Key-Value Databases

• Always use primary-key access

• Operations:
  - Get/put value for key
  - Delete key

• Examples
  - Memcached
  - Amazon DynamoDB
  - Project Voldemort
  - Couchbase
NoSQL: Document Databases

- 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
- Examples: MongoDB, CouchDB, Terrastore

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

[P. Sadalage]
NoSQL: Column Stores

• Instead of having rows grouped/sharded, we group columns
• …or families of columns
• Put similar columns together
• Examples: Cassandra, HBase
NoSQL: Graph Databases

- Focus on entities and relationships
- Edges may have properties
- Relational databases required a set traversal
- Traversals in Graph DBs are faster
- Examples:
  - Neo4j
  - Pregel
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,
  - Master-slave replication makes one node the authoritative copy that handles writes while slaves synchronize with the master and may handle reads.
  - Peer-to-peer replication allows writes to any node; the nodes coordinate to synchronize their copies of the data.

[P. Sadalage]
CAP Theorem

Scalability: CAP Theorem

Remains accessible and operational at all times.

Pick Two!

Traditional relational databases: PostgreSQL, MySQL, etc.

Voidemort, Riak, Cassandra, CouchDB, Dynamo-like systems

Consistency
Commits are atomic across the entire distributed system.

CP
HBase
MongoDB
Redis
MemcacheDB
BigTable-like systems

Partition Tolerance
Only a total network failure can cause the system to respond incorrectly.

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
Think about RDBMS Transactions…
Cassandra:
A Decentralized Structured Storage System

A. Lakshman and P. Malik
What is Cassandra?

• Fast Distributed (Column Family NoSQL) Database
  - High availability
  - Linear Scalability
  - High Performance
• Fault tolerant on Commodity Hardware
• Multi-Data Center Support
• Easy to operate
• Proven: CERN, Netflix, eBay, GitHub, Instagram, Reddit
Cassandra and CAP

RDBMS

Availability

Partition Tolerance

Consistency (ACID)

Atomicity
Consistency
Isolation
Durability

G. Atil
Cassandra: Ring for High Availability

No Master

No Slave

Peer to Peer

gossip

I'm online!

gossip

No, no...
This isn't gossip.
It's the truth.
Slides: Introduction to Cassandra

Robert Stupp
Next Class's Reading & Quiz

- **Spanner: Google's Globally-Distributed Database**
- Quiz available on Thursday:
  - Will focus on main concepts in the paper, not details
  - On Blackboard