Data systems rely on algorithms
Data structures define performance

As time goes by, data structures become ever more critical for data driven applications.

Jim Gray, Turing Award 1998

register = this room
caches = this city
memory = nearby city
disk = Pluto

[S. Idreos, 2019]
Tradeoffs in each structure

no perfect structure

Every data structure design is simply a point in the design space of possible solutions. There is no perfect design. Every design balances the fundamental tradeoffs of Read, Update, and Memory amplification. For example, Read amplification is defined as the excess data an algorithm needs to read on top of the data it wants to read. Typically a data structure would have some kind of metadata or navigation data that help locate the actual data, e.g., the internal nodes of a B-tree. Reading this navigation data is an excess cost, adding to read amplification. Creating a data structure without any navigation data would suffer update or even more read amplification. For example, we could choose to not have any structure in the data at all. Then every query would have to touch all the data. The other extreme would be to sort all data which effectively provides an implicit structure. But then updates get expensive. Overall, there is no perfect design.

[S. Idreos, 2019]
Many efforts in the field have been motivated by the vision of generating tailored systems for a specific scenario. In fact, even traditional databases are architected with this vision in mind. A generic database system can optimize a plan on the fly to match the query needs, it can choose from different storage and indexing options, etc. This is how generic database systems can be used in a wealth of applications! And then recent research has tried to push the boundaries of tailored designs by rethinking parts of the stack of a database system.

"Traditional" Database Research

[S. Idreos, 2019]
Learned Data Structures and Algorithms
B-Tree

Key
(e.g., spoon #1)

Model

[T. Kraska, 2019]
Model to Predict Data's Location on Disk

Frequency Distribution

Cumulative Distribution Function (CDF)

P(X<2017-11-27) * N

[Source: T. Kraska, 2019]
Challenges

Traditional model architectures do not work

Frameworks are not designed for nano-second execution

Overfitting can be good

ML+System Co-Design

Underfitting desired

T. Kraska, 2019
Recursive Model Index (RMI)

2-Stage RMI with Linear Model

\[
\begin{align*}
pos_0 &= a_0 + b_0 \times \text{key} \\
pos_1 &= m_1[pos_0].a + m_1[pos_0].b \times \text{key} \\
\text{record} &= \text{local-search}(\text{key}, \text{pos}_1)
\end{align*}
\]

[T. Kraska, 2019]
Sandwiched Bloom Filter

Is This **Key** In My Set?

- **No**
- **Maybe** Yes
- **Model**
- **Maybe** No
- **No**
- **Maybe** Yes
- **No**

[M. Mitzenmacher, 2018 via T. Kraska, 2019]
Sorting

(a) CDF Model Pre-Sorts

(b) Compact & local sort

[T. Kraska, 2019]
Sorting

(a) CDF Model Pre-Sorts

(b) Compact & local sort

32-bit ints; normal distribution ($\mu=0$, $\sigma=1e6$)

Running time (sec.)

- std:sort
- Radix sort
- Timsort
- Learned Sort (pre-trained)
- Learned Sort

[T. Kraska, 2019]
More...

Tree  Multi-Dim Index  Bloom-Filter  Sorting  Scheduling  Range-Filter  Hash-Map

Data Cubes  DNA-Search  SQL Query Optimizer  Cache Policy  Join  Nearest Neighbor

[T. Kraska, 2019]
Query Optimization

When a user submits a query, Bao parses the SQL statements and sends it to the query optimizer. The optimizer generates and evaluates multiple query plans, each using different hints from the available hint sets. These plans are then sent to the TCNN, which uses the Reward Predictions module to estimate the performance of each plan. The Execution Engine selects the best plan based on these predictions.

After each query execution, the selected plan is sent to the Experience Training module. This module stores the outcomes of queries and uses them to train the reward function. The trained function is then used to predict the performance of new query plans, informing the optimizer on which hints to apply for future queries.

Bao also uses Thompson sampling, a classical method in reinforcement learning, to balance the exploration of new query plans and the exploitation of known good plans. This method helps Bao to quickly converge to the best query plan combination.
Final Exam

- Wednesday, May 8, 8:00-9:50am, PM 252
- Similar format
- More comprehensive (questions from topics covered in Test 1 & 2)
- Will also have questions from graph/spatial/temporal data, provenance, reproducibility, machine learning
Questions?
Review
What did we do this semester?
What's involved in dealing with data?

**Data Acquisition**
- Structured data
- Unstructured data
- Event processing
- Sensor networks
- Protocols
- Real-time
- Data streams
- Multimodality

**Data Analysis**
- Stream mining
- Semantic analysis
- Machine learning
- Information extraction
- Linked Data
- Data discovery
- ‘Whole world’ semantics
- Ecosystems
- Community data analysis
- Cross-sectorial data analysis

**Data Curation**
- Data Quality
- Trust / Provenance
- Annotation
- Data validation
- Human-Data Interaction
- Top-down/Bottom-up
- Community / Crowd
- Human Computation
- Curation at scale
- Incentivisation
- Automation
- Interoperability

**Data Storage**
- In-Memory DBs
- NoSQL DBs
- NewSQL DBs
- Cloud storage
- Query Interfaces
- Scalability and Performance
- Data Models
- Consistency, Availability, Partition-tolerance
- Security and Privacy
- Standardization

**Data Usage**
- Decision support
- Prediction
- In-use analytics
- Simulation
- Exploration
- Visualisation
- Modeling
- Control
- Domain-specific usage

[Big Data Value Chain, Curry et al., 2014]
Python!

• Just assign expressions to variables, no typing
  
  a = 12
  a = "abc"
  b = a + "de"

• Functions defined using def, called using parenthesis:
  
  def hello(name1="Joe", name2="Jane"):
      print(f"Hello {name1} and {name2}"")
  hello(name2="Mary")

• Always indent blocks (if-else-elif, while, for, etc.):
  
  z = 20
  if x > 0:
      if y > 0:
          z = 100
  else:
      z = 10
Python Containers

• List: [1, "abc", 12.34]
• Tuple: (1, "abc", 12.34)
• Indexing/Slicing:
  - x[0], x[:], x[1:2], x[::2]
• Set: {1, "abc", 12.34}
• Dictionary: {'x': 1, 'y': "abc", 'z': 12.34}
• Mutable vs. Immutable
• Stored by reference
• Iterators: objects that traverse containers, just know how to get next element
• You cannot index/slice an iterator (d.values()[-1] doesn't work)
Comprehensions

- **List Comprehensions:**
  - `squares = [i**2 for i in range(10)]`

- **Dictionary Comprehensions:**
  - `squares = {i: i**2 for i in range(10)}`

- **Set Comprehensions:**
  - `squares = {i**2 for i in range(10)}`

- **Comprehensions allow filters:**
  - `squares = [i**2 for i in range(10) if i % 2 == 0]`
JupyterLab

• An interactive, configurable programming environment
• Supports many activities including notebooks
• Runs in your web browser
• Notebooks:
  - Originally designed for Python
  - Supports other languages, too
  - Displays results (even interactive maps) inline
  - You decide how to divide code into executable cells
  - Shift+Enter to execute a cell
Relational Algebra

• Definition: A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.

• Six basic operators
  - select: σ
  - project: \( \Pi \)
  - union: \( \cup \)
  - set difference: –
  - Cartesian product: \( \times \)
  - rename: \( \rho \)
Components of SQL

• Data Definition Language (DDL): the specification of information about relations, including schema, types, integrity constraints, indices, storage

• Data Manipulation Language (DML): provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.

• An SQL relation is defined using the create table command:

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n, (C_1), ..., (C_k))

• A typical SQL query has the form:

  select A_1, A_2, ..., A_n
  from r_1, r_2, ..., r_m
  where P

  - A_i is an attribute
  - D_i is the data type
  - r_i represents a relation
  - P is a predicate

[A. Silberschatz et al.]
NumPy arrays and slicing

Figure 4-1. Indexing elements in a NumPy array

In multidimensional arrays, if you omit later indices, the returned object will be a lower dimensional ndarray consisting of all the data along the higher dimensions. So in the $2 \times 2 \times 3$ array $\text{arr3d}$:

$$\text{arr3d} = \text{np.array}([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])$$

In $\text{arr3d}[0]$ is a $2 \times 3$ array:

$$\text{arr3d}[0]$$

Out $\text{arr3d}[0]$: $\text{array}([[1, 2, 3], [4, 5, 6]])$

Both scalar values and arrays can be assigned to $\text{arr3d}[0]$:

```python
old_values = arr3d[0].copy()
arr3d[0] = 42
```

In $\text{arr3d}[0]$:

Out $\text{arr3d}[0]$: $\text{array}([[42, 42, 42], [42, 42, 42]])$

Figure 4-2. Two-dimensional array slicing

Suppose each name corresponds to a row in the data array and we wanted to select all the rows with corresponding name 'Bob'. Like arithmetic operations, comparisons (such as ==) with arrays are also vectorized. Thus, comparing names with the string 'Bob' yields a boolean array:

$$\text{names} == \text{'Bob'}$$

Out $\text{names} == \text{'Bob'}$: $\text{array([ True, False, False, True, False, False, False], dtype=bool)}$

This boolean array can be passed when indexing the array:

$$\text{data}[\text{names} == \text{'Bob'}]$$

Out $\text{data}[\text{names} == \text{'Bob'}]$: $\text{array}([[-0.048, 0.5433, -0.2349, 1.2792], [2.1452, 0.8799, -0.0523, 0.0672]])$

The boolean array must be of the same length as the axis it's indexing. You can even mix and match boolean arrays with slices or integers (or sequences of integers, more on this later):

$$\text{data}[\text{names} == \text{'Bob', 2:}]$$

Out $\text{data}[\text{names} == \text{'Bob', 2:}]$: $\text{array}([[ -0.2349, 1.2792]])$

[W. McKinney, Python for Data Analysis]
Boolean Indexing

• names == 'Bob' gives back booleans that represent the element-wise comparison with the array names

• Boolean arrays can be used to index into another array:
  - data[names == 'Bob']

• Can even mix and match with integer slicing

• Can do boolean operations (&, |) between arrays (just like addition, subtraction)
  - data[(names == 'Bob') | (names == 'Will')]

• Note: or and and do not work with arrays

• We can set values too! data[data < 0] = 0
What is Data?

- **Tables**
  - Attributes (columns)
  - Items (rows)
  - Cell containing value

- **Networks**
  - Link
  - Node (item)

- **Fields (Continuous)**
  - Grid of positions
  - Cell
  - Attributes (columns)
  - Value in cell

- **Geometry (Spatial)**
  - Position

- **Multidimensional Table**
  - Key 1
  - Key 2
  - Attributes
  - Value in cell

- **Trees**

[Munzner (ill. Maguire), 2014]
## Categorial, Ordinal, and Quantitative

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
<th>T</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10/14/06</td>
<td>5-Low</td>
<td>Large Box</td>
<td>0.8</td>
<td>10/21/06</td>
</tr>
<tr>
<td>6</td>
<td>2/21/08</td>
<td>4-Not Spec</td>
<td>Small Pack</td>
<td>0.55</td>
<td>2/22/08</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Small Pack</td>
<td>0.79</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Jumbo Box</td>
<td>0.72</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.6</td>
<td>7/18/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.65</td>
<td>7/18/07</td>
</tr>
<tr>
<td>35</td>
<td>10/23/07</td>
<td>4-Not Spec</td>
<td>Wrap Bag</td>
<td>0.52</td>
<td>10/24/07</td>
</tr>
<tr>
<td>35</td>
<td>10/23/07</td>
<td>4-Not Spec</td>
<td>Small Box</td>
<td>0.58</td>
<td>10/25/07</td>
</tr>
<tr>
<td>36</td>
<td>11/3/07</td>
<td>1-Urgent</td>
<td>Small Box</td>
<td>0.55</td>
<td>11/3/07</td>
</tr>
<tr>
<td>65</td>
<td>3/18/07</td>
<td>1-Urgent</td>
<td>Small Pack</td>
<td>0.49</td>
<td>3/19/07</td>
</tr>
<tr>
<td>66</td>
<td>1/20/05</td>
<td>5-Low</td>
<td>Wrap Bag</td>
<td>0.56</td>
<td>1/20/05</td>
</tr>
<tr>
<td>69</td>
<td>6/4/05</td>
<td>4-Not Spec</td>
<td>Small Pack</td>
<td>0.44</td>
<td>6/6/05</td>
</tr>
<tr>
<td>69</td>
<td>6/4/05</td>
<td>4-Not Spec</td>
<td>Small Pack</td>
<td>0.6</td>
<td>6/6/05</td>
</tr>
<tr>
<td>70</td>
<td>12/18/06</td>
<td>5-Low</td>
<td>Small Box</td>
<td>0.59</td>
<td>12/23/06</td>
</tr>
<tr>
<td>70</td>
<td>12/18/06</td>
<td>5-Low</td>
<td>Small Box</td>
<td>0.82</td>
<td>12/23/06</td>
</tr>
<tr>
<td>96</td>
<td>4/17/05</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.55</td>
<td>4/19/05</td>
</tr>
<tr>
<td>97</td>
<td>1/29/06</td>
<td>3-Medium</td>
<td>Medium Box</td>
<td>0.38</td>
<td>1/30/06</td>
</tr>
<tr>
<td>129</td>
<td>11/19/08</td>
<td>5-Low</td>
<td>Medium Box</td>
<td>0.37</td>
<td>11/28/08</td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.37</td>
<td>5/9/08</td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.38</td>
<td>5/10/08</td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.6</td>
<td>5/11/08</td>
</tr>
<tr>
<td>132</td>
<td>6/11/06</td>
<td>3-Medium</td>
<td>Medium Box</td>
<td>0.6</td>
<td>6/12/06</td>
</tr>
<tr>
<td>132</td>
<td>6/11/06</td>
<td>3-Medium</td>
<td>Jumbo Box</td>
<td>0.69</td>
<td>6/14/06</td>
</tr>
<tr>
<td>134</td>
<td>5/1/08</td>
<td>4-Not Spec</td>
<td>Large Box</td>
<td>0.82</td>
<td>5/3/08</td>
</tr>
<tr>
<td>135</td>
<td>10/21/07</td>
<td>4-Not Spec</td>
<td>Small Pack</td>
<td>0.64</td>
<td>10/23/07</td>
</tr>
<tr>
<td>166</td>
<td>9/12/07</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.55</td>
<td>9/14/07</td>
</tr>
<tr>
<td>193</td>
<td>8/8/06</td>
<td>1-Urgent</td>
<td>Medium Box</td>
<td>0.57</td>
<td>8/10/06</td>
</tr>
<tr>
<td>194</td>
<td>4/5/08</td>
<td>3-Medium</td>
<td>Wrap Bag</td>
<td>0.42</td>
<td>4/7/08</td>
</tr>
</tbody>
</table>

*quantitative*, *ordinal*, *categorical*
Pandas and Data Frames

Data Frames are tables with many database-like operations

Index shared across all columns

Can select, project, merge (join), and more

Read and write many file formats

In [1]:
import pandas as pd

In [2]:
# read the dataset using pandas
df = pd.read_csv("Food_Inspections.csv")

In [3]:
# look at the dataset, nice table formatting
df

In [4]:
# just the beginning of the dataset
df.head()

In [5]:
# number of records
len(df)

Out[2]:
<table>
<thead>
<tr>
<th>Inspection ID</th>
<th>DBA Name</th>
<th>AKA Name</th>
<th>License #</th>
<th>Facility Type</th>
<th>Risk</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Inspection Date</th>
<th>Inspection Type</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UNCOOKED LLC</td>
<td>UNCOOKED LLC</td>
<td>2709319.0</td>
<td>NaN</td>
<td>All</td>
<td>210 N CARPENTER ST</td>
<td>CHICAGO</td>
<td>IL</td>
<td>60607</td>
<td>01/13/2020</td>
<td>License</td>
<td>Not Ready</td>
</tr>
<tr>
<td>1</td>
<td>MOJO 33 NORTH LASALLE LLC</td>
<td>MOJO 33 NORTH LASALLE LLC</td>
<td>2689550.0</td>
<td>Restaurant</td>
<td>Risk 1 (High)</td>
<td>33 N LA SALLE ST</td>
<td>CHICAGO</td>
<td>IL</td>
<td>60602</td>
<td>01/13/2020</td>
<td>License Re-Inspection</td>
<td>Pass</td>
</tr>
<tr>
<td>2</td>
<td>LA BIZNAGA #2</td>
<td>LA BIZNAGA #2</td>
<td>2708992.0</td>
<td>NaN</td>
<td>Risk 1 (High)</td>
<td>2949 W BELMONT AVE</td>
<td>CHICAGO</td>
<td>IL</td>
<td>60618</td>
<td>01/10/2020</td>
<td>License</td>
<td>Not Ready</td>
</tr>
<tr>
<td>3</td>
<td>LAS TABLAS</td>
<td>LAS TABLAS</td>
<td>1617900.0</td>
<td>Restaurant</td>
<td>Risk 1 (High)</td>
<td>4920 W IRVING PARK RD</td>
<td>CHICAGO</td>
<td>IL</td>
<td>60641</td>
<td>01/09/2020</td>
<td>Canvas</td>
<td>Pass</td>
</tr>
<tr>
<td>4</td>
<td>GIORDANO'S OF BEVERLY</td>
<td>GIORDANO'S OF BEVERLY</td>
<td>2074456.0</td>
<td>Restaurant</td>
<td>Risk 1 (High)</td>
<td>9613 S WESTERN AVE</td>
<td>CHICAGO</td>
<td>IL</td>
<td>60643</td>
<td>01/09/2020</td>
<td>Canvas</td>
<td>Pass</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>199687</td>
<td>PANDA EXPRESS #236</td>
<td>PANDA EXPRESS #236</td>
<td>1801495.0</td>
<td>Restaurant</td>
<td>Risk 1 (High)</td>
<td>77 W JACKSON BLVD</td>
<td>CHICAGO</td>
<td>IL</td>
<td>60604</td>
<td>02/18/2010</td>
<td>Suspected Food Poisoning</td>
<td>Pass</td>
</tr>
<tr>
<td>199688</td>
<td>KENNY'S RIBS &amp; CHICKEN</td>
<td>UNCLE JOE’S</td>
<td>81000.0</td>
<td>Restaurant</td>
<td>Risk 1 (High)</td>
<td>1453 E HYDE PARK BLVD</td>
<td>CHICAGO</td>
<td>IL</td>
<td>60615</td>
<td>02/08/2010</td>
<td>Complaint</td>
<td>Pass</td>
</tr>
<tr>
<td>199689</td>
<td>Cafe Marbella</td>
<td>Cafe Marbella</td>
<td>2016764.0</td>
<td>Restaurant</td>
<td>Risk 1 (High)</td>
<td>5527-5531 N Milwaukee AVE</td>
<td>CHICAGO</td>
<td>IL</td>
<td>60630</td>
<td>01/29/2010</td>
<td>License Re-Inspection</td>
<td>Pass</td>
</tr>
<tr>
<td>199690</td>
<td>WALGREENS # 07876</td>
<td>WALGREENS # 07876</td>
<td>2004292.0</td>
<td>Grocery Store</td>
<td>Risk 3 (Low)</td>
<td>7544 S STONY ISLAND AVE</td>
<td>CHICAGO</td>
<td>IL</td>
<td>60649</td>
<td>02/18/2010</td>
<td>TASK FORCE LIQUOR 1474</td>
<td>Pass</td>
</tr>
<tr>
<td>199691</td>
<td>YSABEL'S FILIPINO CUISINE</td>
<td>YSABEL'S GRILL ASIAN CUISINE</td>
<td>2013419.0</td>
<td>Restaurant</td>
<td>Risk 1 (High)</td>
<td>4908 W Irving Park RD</td>
<td>CHICAGO</td>
<td>IL</td>
<td>60641</td>
<td>01/12/2010</td>
<td>License Re-Inspection</td>
<td>Pass</td>
</tr>
</tbody>
</table>

199692 rows x 17 columns
FINDINGS

we got about the future of the data science, the most salient takeaway was how excited our respondents were about the evolution of the field. They cited things in their own practice, how they saw their jobs getting more interesting and less repetitive, all while expressing a real and broad enthusiasm about the value of the work in their organization.

As data science becomes more commonplace and simultaneously a bit demystified, we expect this trend to continue as well. After all, last year's respondents were just as excited about their work (about 79% were "satisfied" or better).

How a Data Scientist Spends Their Day

Here's where the popular view of data scientists diverges pretty significantly from reality. Generally, we think of data scientists building algorithms, exploring data, and doing predictive analysis. That's actually not what they spend most of their time doing, however.

As you can see from the chart above, 3 out of every 5 data scientists we surveyed actually spend the most time cleaning and organizing data. You may have heard this referred to as "data wrangling" or compared to digital janitor work. Everything from list verification to removing commas to debugging databases—-that time adds up and it adds up immensely. Messy data is by far the more time-consuming aspect of the typical data scientist's work flow. And nearly 60% said they simply spent too much time doing it.

Data scientist job satisfaction

<table>
<thead>
<tr>
<th>Activity</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building training sets</td>
<td>3%</td>
</tr>
<tr>
<td>Cleaning and organizing data</td>
<td>60%</td>
</tr>
<tr>
<td>Collecting data sets</td>
<td>19%</td>
</tr>
<tr>
<td>Mining data for patterns</td>
<td>9%</td>
</tr>
<tr>
<td>Refining algorithms</td>
<td>4%</td>
</tr>
<tr>
<td>Other</td>
<td>5%</td>
</tr>
</tbody>
</table>

What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets: 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%
Data Wrangling

- Automated Transformation Suggestions
- Editable Natural Language Explanations
- Visual Transformation Previews
- Transformation History

Fill **Bangladesh** by copying values from above

Fill **Bangladesh** by interpolating

Fill **Bangladesh** by averaging the 5 values from above

---

[S. Kandel et al., 2011]
TDE: Transform Data by Example

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Date</td>
<td>output</td>
</tr>
<tr>
<td>Wed, 12 Jan 2011</td>
<td>2011-01-12-Wednesday</td>
</tr>
<tr>
<td>Thu, 15 Sep 2011</td>
<td>2011-09-15-Thursday</td>
</tr>
<tr>
<td>Mon, 17 Sep 2012</td>
<td>2012-09-17-Monday</td>
</tr>
<tr>
<td>2010-Nov-30 11:10:41</td>
<td>2010-11-30-Tuesday</td>
</tr>
<tr>
<td>2011-Jan-11 02:27:21</td>
<td>2011-02-17-Tuesday</td>
</tr>
<tr>
<td>2011-Jan-12</td>
<td>2011-01-12-Wednesday</td>
</tr>
<tr>
<td>2010-Dec-24</td>
<td>2010-12-24-Friday</td>
</tr>
<tr>
<td>9/22/2011</td>
<td>2011-09-22-Thursday</td>
</tr>
<tr>
<td>7/11/2012</td>
<td>2012-07-11-Wednesday</td>
</tr>
<tr>
<td>2/12/2012</td>
<td>2012-02-12-Sunday</td>
</tr>
</tbody>
</table>

[Y. He et al., 2018]
Transform by Pattern: Automating Unify/Repair

• Auto-Unify

<table>
<thead>
<tr>
<th>S-timestamp</th>
<th>S-phone</th>
<th>S-coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-12-23</td>
<td>(425) 882-8080</td>
<td>(38°57'N, 95°15'W)</td>
</tr>
<tr>
<td>2019-12-24</td>
<td>(425) 882-8080</td>
<td>(38°61'N, 95°21'W)</td>
</tr>
<tr>
<td>2019-12-23</td>
<td>(206) 876-1800</td>
<td>(39°19'N, 95°18'W)</td>
</tr>
<tr>
<td>2019-12-24</td>
<td>(206) 876-1800</td>
<td>(39°26'N, 95°23'W)</td>
</tr>
<tr>
<td>2019-12-23</td>
<td>(206) 903-8010</td>
<td>(39°42'N, 96°38'W)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R-timestamp</th>
<th>R-phone</th>
<th>R-coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov. 16 2019</td>
<td>650-853-1300</td>
<td>N37°31′ W122°14′</td>
</tr>
<tr>
<td>Nov. 17 2019</td>
<td>650-853-1300</td>
<td>N37°18′ W122°19′</td>
</tr>
<tr>
<td>Nov. 16 2019</td>
<td>425-421-1225</td>
<td>N37°48′ W122°17′</td>
</tr>
<tr>
<td>Nov. 17 2019</td>
<td>425-421-1225</td>
<td>N37°60′ W123°08′</td>
</tr>
<tr>
<td>Nov. 16 2019</td>
<td>650-253-0827</td>
<td>N37°01′ W123°72′</td>
</tr>
</tbody>
</table>

• Auto-Repair

(a) EN-Wiki: Dates

(b) EN-Wiki: Currency values

(c) EN-Wiki: time

(d) EN-Wiki: Date
TBP: Learning from Tables

Table Corpus

Pair & Link Related Table-Cols

Invoke TBE to learn programs

Global optimization of TBP Graph

---

<table>
<thead>
<tr>
<th>T_1</th>
<th>Name</th>
<th>Born</th>
<th>Died</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_2</td>
<td>Date of birth</td>
<td>President</td>
<td>Birthplace</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>State of birth</td>
</tr>
<tr>
<td>T_3</td>
<td>Name</td>
<td>State of birth</td>
<td>Born</td>
</tr>
<tr>
<td>T_4</td>
<td>Name</td>
<td>State of birth</td>
<td>Birth Date</td>
</tr>
<tr>
<td>T_5</td>
<td>Name and (party)</td>
<td>Term</td>
<td>State of birth</td>
</tr>
<tr>
<td>T_6</td>
<td>President</td>
<td>Birth Date</td>
<td>Birth Place</td>
</tr>
</tbody>
</table>
## Tidy Data

### Initial Data

<table>
<thead>
<tr>
<th></th>
<th>treatmenta</th>
<th>treatmentb</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>—</td>
<td>2</td>
</tr>
<tr>
<td>Jane Doe</td>
<td>16</td>
<td>11</td>
</tr>
<tr>
<td>Mary Johnson</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

### Transpose

<table>
<thead>
<tr>
<th></th>
<th>John Smith</th>
<th>Jane Doe</th>
<th>Mary Johnson</th>
</tr>
</thead>
<tbody>
<tr>
<td>treatmenta</td>
<td>—</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>treatmentb</td>
<td>2</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

### Tidy Data

<table>
<thead>
<tr>
<th>name</th>
<th>trt</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>a</td>
<td>—</td>
</tr>
<tr>
<td>Jane Doe</td>
<td>a</td>
<td>16</td>
</tr>
<tr>
<td>Mary Johnson</td>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>John Smith</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>Jane Doe</td>
<td>b</td>
<td>11</td>
</tr>
<tr>
<td>Mary Johnson</td>
<td>b</td>
<td>1</td>
</tr>
</tbody>
</table>

[H. Wickham, 2014]
AutoSuggest

- Automate "Complex" Data Preparation steps
- Focus on frame transformations (not per-cell transformations)
- Learn from Jupyter Notebooks
- Two Types of Predictions:
  - Single-Operator Prediction
  - Next-Operator Prediction
Data Cleaning: SampleClean

Dirty Data

Sample Creation

Aggregate Queries

Data Cleaning

Cleaned Sample

Result Estimation (RawSC)

Result Estimation (NormalizedSC)

Results with Confidence Intervals

Results with Confidence Intervals

Dirty Sample

[J. Wang et al., 2014]
## Data Cleaning: HoloClean

### Input
- **Dataset to be cleaned**
  - **DBAName**: t1, t2, t3, t4
  - **Address**: John Veliotis Sr., John Veliotis Sr., John Veliotis Sr., John Veliotis Sr.
  - **City**: Chicago, Chicago, Chicago, Chicago
  - **State**: IL, IL, IL, IL
  - **Zip**: 60608, 60609, 60609, 60608

### The HoloClean Framework

#### 1. Error detection module
- Denial Constraints:
  - t1: DBAName -> Zip
  - t2: Zip -> City, State
  - t3: City, State, Address -> Zip

#### 2. Automatic compilation to a probabilistic graphical model
- Matching Dependencies:
  - t1: Zip = Ext.Zip -> City = Ext.City
  - t2: Zip = Ext.Zip -> State = Ext.State
  - t3: City = Ext.City & State = Ext.State
  - Address = Ext.Address -> Zip = Ext.Zip

#### 3. Repair via statistical learning and inference

### Output
- **Proposed Cleaned Dataset**

<table>
<thead>
<tr>
<th>DBAName</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>t2 John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t3 John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t4 John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
</tbody>
</table>

- Marginal Distribution of Cell Assignments

<table>
<thead>
<tr>
<th>Cell</th>
<th>Possible Values</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2.Zip</td>
<td>60608, 60609</td>
<td>0.84, 0.16</td>
</tr>
<tr>
<td>t4.City</td>
<td>Chicago</td>
<td>0.95</td>
</tr>
<tr>
<td>t4.DBAName</td>
<td>John Veliotis Sr.</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>Johnny's</td>
<td>0.01</td>
</tr>
</tbody>
</table>
Merges (aka Joins)

• Need to merge data from one DataFrame with data from another DataFrame
• Example: Football game data merged with temperature data

<table>
<thead>
<tr>
<th>Game</th>
<th>Weather</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Id</strong></td>
<td><strong>wId</strong></td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td><strong>9/2</strong></td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td><strong>9/9</strong></td>
</tr>
<tr>
<td><strong>Home</strong></td>
<td><strong>12</strong></td>
</tr>
<tr>
<td><strong>Away</strong></td>
<td><strong>1</strong></td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td><strong>9/23</strong></td>
</tr>
<tr>
<td><strong>Temp</strong></td>
<td><strong>15</strong></td>
</tr>
</tbody>
</table>

| No data for San Diego |
## Inner Strategy

<table>
<thead>
<tr>
<th>Id</th>
<th>Location</th>
<th>Date</th>
<th>Home</th>
<th>Away</th>
<th>Temp</th>
<th>wld</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Boston</td>
<td>9/2</td>
<td>1</td>
<td>15</td>
<td>72</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Boston</td>
<td>9/9</td>
<td>1</td>
<td>7</td>
<td>75</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Cleveland</td>
<td>9/16</td>
<td>12</td>
<td>1</td>
<td>81</td>
<td>36</td>
</tr>
</tbody>
</table>

**No San Diego entry**
## Outer Strategy

### Merged

<table>
<thead>
<tr>
<th>Id</th>
<th>Location</th>
<th>Date</th>
<th>Home</th>
<th>Away</th>
<th>Temp</th>
<th>wld</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Boston</td>
<td>9/2</td>
<td>1</td>
<td>15</td>
<td>72</td>
<td>0</td>
</tr>
<tr>
<td>NaN</td>
<td>Boston</td>
<td>9/3</td>
<td>NaN</td>
<td>NaN</td>
<td>68</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Boston</td>
<td>9/9</td>
<td>1</td>
<td>7</td>
<td>75</td>
<td>7</td>
</tr>
<tr>
<td>NaN</td>
<td>Boston</td>
<td>9/10</td>
<td>NaN</td>
<td>NaN</td>
<td>76</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td>Cleveland</td>
<td>9/2</td>
<td>NaN</td>
<td>NaN</td>
<td>61</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Cleveland</td>
<td>9/16</td>
<td>12</td>
<td>1</td>
<td>81</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>San Diego</td>
<td>9/23</td>
<td>21</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>
Data Integration

**Movie**: Title, director, year, genre  
**Actors**: title, actor  
**Plays**: movie, location, startTime  
**Reviews**: title, rating, description

```sql
select title, startTime
from Movie, Plays
where Movie.title=Plays.movie AND
    location="New York" AND
    director="Woody Allen"
```

Sources S1 and S3 are relevant, sources S4 and S5 are irrelevant, and source S2 is relevant but possibly redundant.

---

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

Source A

Source B

Schema Mapping

Data Transformation

Duplicate Detection

Data Fusion

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

Source A

Source B

Schema Mapping

Duplication Detection

Data Transformation

Data Fusion

Source A

Source B

[Scheth & Larson, 1990]

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

Transformation queries or views

Source A
<publication>
<title>Federated Database Systems</title>
<author>Amit Sheth</author>
<author>James Larson</author>
<year>1990</year>
</publication>

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

Schema Mapping
Data Transformation
Duplicate Detection
Data Fusion

XQuery

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

Source A

Source B

Schema Mapping

Data Transformation

Duplicate Detection

Data Fusion

[L. Dong and F. Naumann, 2009]
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</author>
<author>James Larson</author>
</Autoren>
<year>1990</year>
</pub>

Schema Mapping
Data Transformation
Duplicate Detection
Data Fusion

Preserve lineage

[Source A

Source B]

[L. Dong and F. Naumann, 2009]
Challenges in Data Fusion when Sources Copy

<table>
<thead>
<tr>
<th>Source</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>
# Challenges in Data Fusion when Sources Copy

[XL Dong et al., 2009]

<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>
Challenges in Data Fusion when Sources Copy

<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>

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

[X L Dong et al., 2009]
### Challenges in Data Fusion when Sources Copy

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.

<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>

Source: X L Dong et al., 2009
Source Dependence: Iteration on Truth and Sources

Truth Discovery

Source-accuracy Computation

Dependence Detection

[X L Dong et al., 2009]
Source Dependence: Iteration on Truth and Sources

Truth Discovery

Source-accuracy Computation

Dependence Detection

Step 1

Step 2

Step 3

[X L Dong et al., 2009]
NoSQL Motivation

**Scalability**

User-generated data, Request load

**Impedance Mismatch**

ID
Customer
Line Item 1: ...
Line Item 2: ...
Payment: Credit Card, ...

Orders
Payment
Customers

Line Items

---

Two main motivations:
User-generated data, Request load

[NoSQL Motivation][1856x25]

F. Gessert et al., 2017

D. Koop, CSCI 640/490, Spring 2024

[1856x25]: [F. Gessert et al., 2017]
## Column Stores

<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>

Each column has a file or segment on disk

[J. Swanhart, *Introduction to Column Stores*]
CAP Theorem

Scalability: CAP Theorem

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

Pick Two!

CA: Traditional relational databases: PostgreSQL, MySQL, etc.
AP: Voldemort, Riak, Cassandra, CouchDB, Dynamo-like systems
CP: HBase, MongoDB, Redis, MemcacheDB, BigTable-like systems

[E. Brewer]
Cassandra: Replication and Consistency
Three Types of NewSQL Systems

• New Architectures
  - New codebase without architectural baggage of legacy systems
  - Examples: VoltDB, Spanner, Clustrix

• Transparent Sharding Middleware:
  - Transparent data sharding & query redirecting over cluster of single-node DBMSs
  - Examples: citusdata, ScaleArc (usually support MySQL/postgres wire)

• Database-as-a-Service:
  - Distributed architecture designed specifically for cloud-native deployment
  - Examples: xeround, GenieDB, FathomDB (usually based on MySQL)
Spanner: Google's NewSQL Cloud Database

- Which type of system is Spanner?
  - C: consistency, which implies a single value for shared data
  - A: 100% availability, for both reads and updates
  - P: tolerance to network partitions

- Which two?
  - CA: close, but not totally available
  - So actually CP
Dataframe Data Model

- Combines parts of matrices, databases, and spreadsheets
- Ordered, but not necessarily sorted
  - Rows and columns
- No predefined schema necessary
  - Types can be induced at runtime
- Typed Row/column labels
  - Labels can become data
- Indexing by label or row/column number
  - “Named notation” or “Positional notation”
Differences between Databases & Dataframes

<table>
<thead>
<tr>
<th>Convenience</th>
<th>Entire query at once</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexible</td>
<td>Strict schema</td>
</tr>
<tr>
<td>Versatility</td>
<td>SFW or bust</td>
</tr>
</tbody>
</table>

Incremental + inspection
Mixed types, R/C and data/metadata equiv.
600+ functions

[D. Petersohn, 2022]
Modin as a Way to Scale Dataframes

New Data Source → Prototyping
New spec → Exploring
New requirements

Prototyping → Testing

Testing → Production

Laptop/Workstation → Small Cluster → Large Cluster

Feedback

[D. Petersohn]
Data Science Jungle

Extending Python ecosystem

Extending SQL databases

[A. Jindal et al., 2021]
Magpie Goals

Pythonic Environment
Unified Dataframe API
Magpie Middleware
Polyengines & Mappers
Database Backends

Familiar Python surface
Ongoing standardization
Batching Pandas into large query expressions
Backend selection using past workloads
Cache commonly seen dataframes
Multi-backend environments and libraries
Cloud backends

D. Koop, CSCI 640/490, Spring 2024
Time Series Data

US Treasury bill contracts

Australian electricity production

Sales of new one-family houses, USA

Annual Canadian Lynx trappings

[R. J. Hyndman]
Time Series Data

Trend

US Treasury bill contracts

Australian electricity production

Sales of new one-family houses, USA

Annual Canadian Lynx trappings

[R. J. Hyndman]
Time Series Data

**US Treasury bill contracts**

**Australian electricity production**

**Sales of new one–family houses, USA**

**Annual Canadian Lynx trappings**

**Trend**

<table>
<thead>
<tr>
<th>Day</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>91</td>
</tr>
<tr>
<td>10</td>
<td>89</td>
</tr>
<tr>
<td>20</td>
<td>87</td>
</tr>
<tr>
<td>30</td>
<td>85</td>
</tr>
<tr>
<td>40</td>
<td>83</td>
</tr>
</tbody>
</table>

**Trend + Seasonality**

**Year | GWh**
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>8000</td>
</tr>
<tr>
<td>1985</td>
<td>10000</td>
</tr>
<tr>
<td>1990</td>
<td>12000</td>
</tr>
<tr>
<td>1995</td>
<td>14000</td>
</tr>
</tbody>
</table>

**Total sales**

<table>
<thead>
<tr>
<th>Year</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1975</td>
<td>30</td>
</tr>
<tr>
<td>1980</td>
<td>40</td>
</tr>
<tr>
<td>1985</td>
<td>50</td>
</tr>
<tr>
<td>1990</td>
<td>60</td>
</tr>
<tr>
<td>1995</td>
<td>70</td>
</tr>
</tbody>
</table>

**Number trapped**

<table>
<thead>
<tr>
<th>Time</th>
<th>Lynx trapped</th>
</tr>
</thead>
<tbody>
<tr>
<td>1820</td>
<td>0</td>
</tr>
<tr>
<td>1840</td>
<td>0</td>
</tr>
<tr>
<td>1860</td>
<td>0</td>
</tr>
<tr>
<td>1880</td>
<td>0</td>
</tr>
<tr>
<td>1900</td>
<td>0</td>
</tr>
</tbody>
</table>

[R. J. Hyndman]
Time Series Data

Trend

US Treasury bill contracts

Australian electricity production

Seasonality + Cyclic

Sales of new one-family houses, USA

Annual Canadian Lynx trappings

Trend + Seasonality

[R. J. Hyndman]
Time Series Data

US Treasury bill contracts

Australian electricity production

Sales of new one-family houses, USA

Annual Canadian Lynx trappings

Trend

Seasonality + Cyclic

Trend + Seasonality

Stationary

[R. J. Hyndman]
Gorilla Time Series Data Compression

Figure 2: Visualizing the entire compression algorithm. For this example, 48 bytes of values and time stamps are compressed to just under 21 bytes/167 bits.

4.1 Time series compression

In evaluating the feasibility of building an in-memory time series database, we considered several existing compression schemes to reduce the storage overhead. We identified techniques that applied solely to integer data which didn’t meet our requirement of storing double precision floating point values. Other techniques operated on a complete dataset but did not support compression over a stream of data as was stored in Gorilla [7, 13]. We also identified lossy time series approximation techniques used in data mining to make the problem set more easily fit within memory [15, 11], but Gorilla is focused on keeping the full resolution representation of data.

Our work was inspired by a compression scheme for floating point data derived in scientific computation. This scheme leveraged XOR comparison with previous values to generate a delta encoding [25, 17].

Gorilla compresses data points within a time series with no additional compression used across time series. Each data point is a pair of 64 bit values representing the time stamp and value at that time. Timestamps and values are compressed separately using information about previous values. The overall compression scheme is visualized in Figure 2, showing how time stamps and values are interleaved in the compressed block.

Verifying Correctness

The correctness of Gorilla’s data compression scheme was evaluated on our production dataset and its performance measured using SMAC [16]. Gorilla achieved a compression rate of 97.9%, with a range of 97.3% to 99.8% and a mean of 98.9% across 100 different time series.

Gorilla tolerates single node failures, network cuts, and entire datacenter failures by writing each time series value to two hosts in separate geographic regions. On detecting a failure, all read queries are failed over to the alternate region ensuring that users do not experience any disruption.

[Pelkonen et al., 2015]
Graph Databases focus on relationships

- Directed, labelled, attributed multigraph
- Properties are **key/value pairs** that represent metadata for nodes and edges
Graph DBMS Problems

- **performance**
  - Slow loading speeds
  - Query speeds over magnitude slower than RDBMS
- **scalability**
  - Low datasize limit, typically << RAM
  - Little benefit from parallelism
- **reliability**
  - Loads never terminate
  - Query run out of memory or crash
  - Bugs

[P. Boncz, 2022]
Supporting Scalable Visualization

• Two Problems:
  - **Lots of data**, how to display (encode) it
  - User *interaction* is key to gaining insight, requires *low latency*

• Addressing big data:
  - Encoding should focus on *available resolution*, not size of data
  - Approaches:
    • Sampling
    • Modeling
    • Binning
    - Bin $\rightarrow$ Aggregate ($\rightarrow$ Smooth) $\rightarrow$ Plot
Time Series Aggregation

1M samples -> 2,653 plotted points

↑ Value

Sat 03  Mon 05  Wed 07  Fri 09  Dec 11

Time →
Interactive Exploration of Spatial Data

SELECT lat, lng, (b4-b6)/(b4+b6) as ndsi
FROM modis_data
WHERE ndsi > 0.7
Interactive Exploration of Spatial Data

SELECT lat, lng, (b4-b6)/(b4+b6) as ndsi
FROM modis_data
WHERE ndsi > 0.7
Visualization: Minimize Latency

A full data cube is often too big to fit in memory and query in real-time. The size of a cube is in real-time. The size of a cube is $O(n^d)$, where $n$ is the number of data values and $d$ is the number of dimensions. As the number of dimensions or bins increases, the data cube size may become unwieldy. To address this issue, we decompose the full cube into sub-cubes. After decomposition, individual sub-cubes may still be prohibitively large if the bin count is high. In some plots, we can treat the bin count as a free parameter, and adjust accordingly. For others – particularly geographic heatmaps – we may wish to zoom in to see fine-grained details, requiring finer grained bins, as in Figure 5. For example, four 3-dimensional cubes can cover all combinations of Month, Hour, and Lon; when Month = 5, Hour = 0, and Lon = 0, the bin is selected in the corresponding data in other views, requiring computation of the filtered aggregation (or “roll-up”). Sending queries require partially de-aggregated data over which to compute the projected data. Figure 3: [Image -1356x489 to 74x884]

[Z. Liu et al., 2013]
Visualization: Task-Prioritized Prefetching

- `brushes in the precomputed view`
- `serves requests from a data cube`
- `interacts with a new view`
- `query for new data cubes`
Visualization: Prefetching

- Predict which tiles a user will need next and prefetch those
  - Use common patterns (zoom, pan)
  - Use regions of interest (ROIs)

[Image of visualization with ROI tiles highlighted]
Spatial Data: Beast Architecture

The On-top Approach

- Spatial Modules
- User Programs
- SQL Spark Java/Scala APIs
- Job Monitoring and Scheduling
- RDD Runtime
- Storage (HDFS)

From Scratch Approach

(Spatial) User Program + RDD APIs + Job Monitoring and Scheduling + RDD Runtime + Storage + ...

The Built-in Approach (Beast)

User Programs
- Spatial Language
- Spatial Operators
- Early Pruning
- Spatial Indexing
- SQL Spark Java/Scala APIs
- Job Monitoring and Scheduling
- RDD Runtime
- Storage (HDFS)

[A. Eldawy, 2021]
Spatial Data: Partitioning/Indexing & Range Query

- Read a sample
- Partition the sample using an R-tree index
- Use MBR of leaf nodes as partition boundaries for all the data

Use the partition information to prune disjoint partitions
Scan matching partitions in parallel to find matching records

[A. Eldawy, 2021]
The DCC Curation Lifecycle Model

Full Lifecycle Actions
- Conceptualise
- Create or Receive
- Appraise and Select
- Ingest
- Preservation Action
- Store
- Access, Use and Reuse
- Transform
- Preserve
- Community Watch & Participation
- Preservation Planning
- Description
- Representation Information

Occasional Actions
- Reappraise
- Migrate
- Dispose

Data (Digital Objects or Databases)

Data Curation

Data, any information in binary digital form, is at the centre of the Curation Lifecycle. This includes:
- Simple Digital Objects are discrete digital items; such as textual files, images or sound files, along with their related identifiers and metadata.
- Complex Digital Objects are discrete objects, made by combining a number of other digital objects, such as websites.
- Structured collections of records or data stored in a computer system.

The DCC Curation Lifecycle Model provides a graphical high level overview of the stages required for successful curation and preservation of data from initial conceptualisation or receipt. The model can be used to plan activities within an organisation or consortium to ensure that all necessary stages are undertaken, each in the correct sequence. The model enables granular functionality to be mapped against it; to define roles and responsibilities, and build a framework of standards and technologies to implement. It can help with the process of identifying additional steps which may be required, or actions which are not required by certain situations or disciplines, and ensuring that processes and policies are adequately documented.

Data, any information in binary digital form, is at the centre of the Curation Lifecycle. This includes:
- Simple Digital Objects are discrete digital items; such as textual files, images or sound files, along with their related identifiers and metadata.
- Complex Digital Objects are discrete objects, made by combining a number of other digital objects, such as websites, structured collections of records or data stored in a computer system.
Data Curation: FAIR Principles

• Findable: Metadata and data should be easy to find for both humans and computers

• Accessible: Users need to know how data can be accessed, possibly including authentication and authorization

• Interoperable: Can be integrated with other data, and can interoperate with applications or workflows for analysis, storage, and processing

• Reusable: Optimize the reuse of data. Metadata and data should be well-described so they can be replicated and/or combined in different settings
Provenance
Prospective and Retrospective Provenance

• Recipe for baking a cake versus the actual process & outcome
• Prospective provenance is what was specified/intended
  - a workflow, script, list of steps
• Retrospective provenance is what actually happened
  - actual data, actual parameters, errors that occurred, timestamps, machine information
• Do not need prospective provenance to have retrospective provenance!
Reproducibility

Reproducibility Spectrum

Publication only
Publication +
- Code
- Code and data
- Linked and executable code and data

Not reproducible
Gold standard
Full replication

[R. D. Peng]
Machine Learning and Databases
Questions?
Final Exam

• Wednesday, May 8, 8:00-9:50am, PM 252
• Similar format
• More comprehensive (questions from topics covered in Test 1 & 2)
• Will also have questions from graph/spatial/temporal data, provenance, reproducibility, machine learning