E6893 Big Data Analytics Lecture 8:

Data Visualization and Graph Database

Ching-Yung Lin, Ph.D.
Adjunct Professor, Dept. of Electrical Engineering and Computer Science

October 29th, 2021
Presentation Format:

- Each group shall have 3 mins to present -> Do not overtime.
- CVN and Remote Student groups —> Please submit your presentation video
- Please sign up, including team members, project title, and links to slides (open to let people in Columbia can view) at https://docs.google.com/spreadsheets/d/1YwQ_ZBrpRDCOThySZ_uLwd3mlvxlPjUc356d_rf2WU/edit#gid=0

Proposal Scoring — preparing no more than 5 page slides (each item 1/5 of the proposal score):

- Goal — novel? challenging?
- Data — Big Data 3Vs? New or Existing dataset (from where)?
- Methods — potential methodologies and algorithms? Feasible?
- System — an overview of system. Multiple steps? Scalable?
- Schedule — what to achieve by what time, and by whom?
Data Visualization — SVG short Introduction

SVG Examples

Basic Shapes

- Lines
- Rectangles
- Circles
- Ellipses
- Polygons
- Polylines

Advanced Shapes (Paths)

- Arcs
- Quadratic Bezier Curves
- Cubic Bezier Curves
SVG Examples

Layering + Opacity

Text on Shape

Transformations

SVG Transformations
SVG Examples

Gradients

Linear Gradients

Radial Gradients

Opacity Gradients

Text with opacity gradient

Links

http://jenkov.com

Click Retangle:
SVG Examples

Animations

Use Cases

Graphs

Bar Charts

Pie Charts
A Simple SVG Example

```xml
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <rect x="10" y="10" height="100" width="100"
       style="stroke:#ff0000; fill: #0000ff"/>
</svg>
```
SVG as Background Images

Since the browsers treat SVG images just like bitmap images, you can use SVG images as background images via CSS. Here is an example:

```html
    div {
        background-image: url('my-svg-image.svg');
        background-size : 100px 100px;
    }
```

svg Element Inside HTML

Embedding an SVG image using the `<svg>` element can be done by embedding an `<svg>` element directly in an HTML page, like this:

```html
    <div>
        <svg>
            <circle cx="40" cy="40" r="24" style="stroke:#006600; fill:#00cc00"/>
        </svg>
    </div>
```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <g>
    <line x1="10" y1="10" x2="85" y2="10"
         style="stroke: #006600;"/>

    <rect x="10" y="20" height="50" width="75"
         style="stroke: #006600; fill: #006600"/>

    <text x="10" y="90" style="stroke: #660000; fill: #660000">
      Text grouped with shapes</text>
  </g>
</svg>
SVG strokes and fill

```xml
<circle cx="40" cy="40" r="24"
    style="stroke:#006600;
            stroke-width: 3;
            fill:#00cc00"/>

<circle cx="40" cy="40" r="24"
    style="stroke:#006600;
            stroke-width: 3;
            stroke-dasharray: 10 5;
            fill:#00cc00"/>

<circle cx="40" cy="40" r="24"
    style="stroke: none;
            fill:#00cc00"/>

<circle cx="40" cy="40" r="24"
    style="stroke: #006000;
            fill:none"/>
```
SVG path Element

```xml
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <path d="M50,50
         A30,30 0 0,1 35,20
         L100,100
         M110,110
         L100,0"
       style="stroke:#660000; fill:none;"/>
</svg>
```

A: Arc
M: Move
L: Line
The radius of the arc is set by the two first parameters set on the `A` command. The first parameter is $rx$ (radius in x-direction) and the second is $ry$ (radius in y-direction). Setting $rx$ and $ry$ to the same value will result in a circular arc. Setting $rx$ and $ry$ to different values will result in an elliptical arc. In the example above $rx$ was set to 30 and $ry$ to 50.

The third parameter set on the `A` command is the `x-axis-rotation`. This sets the rotation of the arc's x-axis compared to the normal x-axis. In the above example the `x-axis-rotation` is set to 0. Most of the time you will not need to change this parameter.

The fourth and fifth parameters are the `large-arc-flag` and `sweep-flag` parameters. The `large-arc-flag` determines whether to draw the smaller or bigger arc satisfying the start point, end point and $rx$ and $ry$.
SVG path Element

```html
<path d="M40,20 A30,30 0 0,0 70,70"
  style="stroke: #cccc00; stroke-width:2; fill:none;"/>

<path d="M40,20 A30,30 0 1,0 70,70"
  style="stroke: #ff0000; stroke-width:2; fill:none;"/>

<path d="M40,20 A30,30 0 1,1 70,70"
  style="stroke: #00ff00; stroke-width:2; fill:none;"/>

<path d="M40,20 A30,30 0 0,1 70,70"
  style="stroke: #0000ff; stroke-width:2; fill:none;"/>
```
SVG text Element

<text x="20" y="40"
    style="fill: #000000; stroke: none; font-size: 48px;">
    Fill only
</text>

<text x="20" y="100"
    style="fill: none; stroke: #000000; font-size: 48px;">
    Stroke only
</text>

<text x="20" y="150"
    style="fill: #999999; stroke: #000000; font-size: 48px;">
    Fill and stroke
</text>

Fill only
Stroke only
Fill and stroke
SVG text Element

```xml
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <text x="20" y="40"
       transform="rotate(30 20,40)"
       style="stroke:none; fill:#000000;"
       >Rotated SVG text</text>
</svg>
```

Rotated SVG text

```xml
<text x="10" y="20" style="writing-mode: tb;">
  Vertical
</text>
```

Vertical
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <rect x="50" y="50" height="110" width="110"
        style="stroke:#ff0000; fill: #ccccff"
        transform="translate(30) rotate(45 50 50)"
    />
  <text x="70" y="100"
        transform="translate(30) rotate(45 50 50)"
        >Hello World</text>
</svg>
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <rect x="10" y="10" height="110" width="110"
        style="stroke:#ff0000; fill: #0000ff">
    <animateTransform
        attributeName="transform"
        begin="0s"
        dur="20s"
        type="rotate"
        from="0 60 60"
        to="360 60 60"
        repeatCount="infinite"
    />
  </rect>
</svg>
Overview of Animation Options

1. `<set>`
2. `<animate>`
3. `<animateColor>`
4. `<animateTransform>`
5. `<animateMotion>`

```xml
<circle cx="30" cy="30" r="25" style="stroke: none; fill: #0000ff;">
  <set attributeName="r" attributeType="XML"
       to="100"
       begin="0s" />
</circle>
```
<circle cx="30" cy="30" r="25" style="stroke: none; fill: #0000ff;">
    <animate attributeName="cx" attributeType="XML"
        from="30" to="470"
        begin="0s" dur="5s"
        fill="remove" repeatCount="indefinite"/>
</circle>
animateTransform

```xml
<rect x="20" y="20" width="100" height="40"
    style="stroke: #ff00ff; fill:none;" >
    <animateTransform attributeName="transform"
        type="rotate"
        from="0 100 100" to="360 100 100"
        begin="0s" dur="10s"
        repeatCount="indefinite"
    />
</rect>
```
animateMotion

<rect x="0" y="0" width="30" height="15"
     style="stroke: #ff0000; fill: none;">
    <animateMotion
        path="M10,50 q60,50 100,0 q60,-50 100,0"
        begin="0s" dur="10s" repeatCount="indefinite"
    />
</rect>
Linear Gradients

```xml
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
    <defs>
        <linearGradient id="myLinearGradient1"
            x1="0%" y1="0%"
            x2="0%" y2="100%"
            spreadMethod="pad">
            <stop offset="0%" stop-color="#00cc00" stop-opacity="1"/>
            <stop offset="100%" stop-color="#006600" stop-opacity="1"/>
        </linearGradient>
    </defs>

    <rect x="10" y="10" width="75" height="100" rx="10" ry="10"
        style="fill:url(#myLinearGradient1);
                stroke: #005000;
                stroke-width: 3;" />
</svg>
```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">

<defs>
    <linearGradient id="myLinearGradient1"
        x1="0%" y1="0%"
        x2="100%" y2="0%"
        spreadMethod="pad">
        <stop offset="10%" stop-color="#00cc00" stop-opacity="1"/>
        <stop offset="30%" stop-color="#006600" stop-opacity="1"/>
        <stop offset="70%" stop-color="#cc0000" stop-opacity="1"/>
        <stop offset="90%" stop-color="#000099" stop-opacity="1"/>
    </linearGradient>

</defs>

<rect x="10" y="10" width="500" height="50" rx="10" ry="10"
     style="fill:url(#myLinearGradient1); stroke: #005000;
            stroke-width: 3;" />

</svg>
SVG Gradients

Radial Gradients

```xml
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">

  <defs>
    <radialGradient id="myRadialGradient4"
                    fx="5%" fy="5%" r="65%"
                    spreadMethod="pad">
      <stop offset="0%" stop-color="#00ee00" stop-opacity="1"/>
      <stop offset="100%" stop-color="#006600" stop-opacity="1" />
    </radialGradient>
  </defs>

  <rect x="340" y="10" width="100" height="100" rx="10" ry="10"
        style="fill:url(#myRadialGradient4);
                 stroke: #005000; stroke-width: 3;" />

</svg>
```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <rect x="10" y="10" height="130" width="500" style="fill: #000000;"/>
  <image x="20" y="20" width="300" height="80"
  <line x1="25" y1="80" x2="350" y2="80"
       style="stroke: #ffffff; stroke-width: 3;"/>
</svg>
https://www.tutorialspoint.com/svg/maps.htm
Visualization — D3 Introduction and Examples
http://d3js.org
Installing

For NPM, `npm install d3`. For Yarn, `yarn add d3`. Otherwise, download the latest release. The released bundle supports AMD, CommonJS, and vanilla environments. Create a custom bundle using Rollup or your preferred bundler. You can also load directly from d3js.org:

```html
<script src="https://d3js.org/d3.v5.js"></script>
```

For the minified version:

```html
<script src="https://d3js.org/d3.v5.min.js"></script>
```

You can also use the standalone D3 microlibraries. For example, `d3-selection`:

```html
<script src="https://d3js.org/d3-selection.v1.min.js"></script>
```
D3 Selections

In Javascript:

```javascript
var paragraphs = document.getElementsByTagName("p");
for (var i = 0; i < paragraphs.length; i++) {
    var paragraph = paragraphs.item(i);
    paragraph.style.setProperty("color", "white", null);
}
```

D3 employs a declarative approach, operating on arbitrary sets of nodes called selections. For example, you can rewrite the above loop as:

```javascript
d3.selectAll("p").style("color", "white");
```
Randomly color paragraphs:

```javascript
d3.selectAll("p").style("color", function() {
    return "hsl(" + Math.random() * 360 + ",100%,50%";
});
```

Alternate shades of gray for even and odd nodes:

```javascript
d3.selectAll("p").style("color", function(d, i) {
    return i % 2 ? "#fff" : "#eee";
});
```
D3 Dynamic Properties

Randomly color paragraphs:

d3.selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .style("font-size", function(d) { return d + "px"; });

Computed properties often refer to bound data. Data is specified as an array of values, and each value is passed as the first argument (d) to selection functions. With the default join-by-index, the first element in the data array is passed to the first node in the selection, the second element to the second node, and so on. For example, if you bind an array of numbers to paragraph elements, you can use these numbers to compute dynamic font sizes:
Using D3’s *enter* and *exit* selections, you can create new nodes for incoming data and remove outgoing nodes that are no longer needed.

When data is bound to a selection, each element in the data array is paired with the corresponding node in the selection. If there are fewer nodes than data, the extra data elements form the enter selection, which you can instantiate by appending to the enter selection. For example:

```javascript
// Using d3's enter and exit selections, you can create new nodes for incoming data.
// When data is bound to a selection, each element in the data array is paired with the corresponding node in the selection.
// If there are fewer nodes than data, the extra data elements form the enter selection.
// For example:

d3.select("body")
  .selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .enter().append("p")
  .text(function(d) { return "I’m number " + d + "!"; });
```
// Update...
var p = d3.select("body")
    .selectAll("p")
    .data([4, 8, 15, 16, 23, 42])
    .text(function(d) { return d; });

// Enter...
p.enter().append("p")
    .text(function(d) { return d; });

// Exit...
p.exit().remove();
D3 Transformation

For example, to fade the background of the page to black:

```javascript
D3 Transformation

For example, to fade the background of the page to black:

```javascript
d3.select("body").transition()
  .style("background-color", "black");
```

Or, to resize circles in a symbol map with a staggered delay:

```javascript
D3 Transformation

Or, to resize circles in a symbol map with a staggered delay:

```javascript
d3.selectAll("circle").transition()
  .duration(750)
  .delay(function(d, i) { return i * 10; })
  .attr("r", function(d) { return Math.sqrt(d * scale); });
```
D3 Bar Chart Tutorial

https://bost.ocks.org/mike/bar/

```
var data = [4, 8, 15, 16, 23, 42];
```

Selecting an Element

Javascript:
```
var div = document.createElement("div");
div.innerHTML = "Hello, world!";
document.body.appendChild(div);
```

D3:
```
var body = d3.select("body");
var div = body.append("div");
div.html("Hello, world!");
```
Chaining Methods

```javascript
var body = d3.select("body");
body.style("color", "black");
body.style("background-color", "white");

var section = d3.selectAll("section");

section.append("div")
  .html("First!");

section.append("div")
  .html("Second.");
```
Coding a Chart, Manually

```html
<!DOCTYPE html>
<style>
.chart div {
  font: 10px sans-serif;
  background-color: steelblue;
  text-align: right;
  padding: 3px;
  margin: 1px;
  color: white;
}
</style>
<div class="chart">
  <div style="width: 40px;"">4</div>
  <div style="width: 80px;"">8</div>
  <div style="width: 150px;"">15</div>
  <div style="width: 160px;"">16</div>
  <div style="width: 230px;"">23</div>
  <div style="width: 420px;"">42</div>
</div>
```
Coding a Chart, Automatically

d3.select(".chart")
  .selectAll("div")
    .data(data)
    .enter().append("div")
      .style("width", function(d) { return d * 10 + "px"; })
      .text(function(d) { return d; });

First, we select the chart container using a class selector.

var chart = d3.select(".chart");

Next we initiate the data join by defining the selection to which we will join data.

var bar = chart.selectAll("div");
Coding a Chart, Automatically

```javascript
var barUpdate = bar.data(data);

var barEnter = barUpdate.enter().append("div");

barEnter.style("width", function(d) { return d * 10 + "px"; });

barEnter.text(function(d) { return d; });
```
Scaling to Fit

```javascript
var x = d3.scale.linear()
  .domain([0, d3.max(data)])
  .range([0, 420]);
```
D3 Bar Chart Tutorial

```html
<!DOCTYPE html>
<style>
.chart rect {
  fill: steelblue;
}

.chart text {
  fill: white;
  font: 10px sans-serif;
  text-anchor: end;
}
</style>
<svg class="chart" width="420" height="120">
  <g transform="translate(0,0)">
    <rect width="40" height="19"></rect>
    <text x="37" y="9.5" dy=".35em">4</text>
  </g>
  <g transform="translate(0,20)">
    <rect width="80" height="19"></rect>
    <text x="77" y="9.5" dy=".35em">8</text>
  </g>
  <g transform="translate(0,40)">
    <rect width="150" height="19"></rect>
    <text x="147" y="9.5" dy=".35em">15</text>
  </g>
  <g transform="translate(0,60)">
    <rect width="160" height="19"></rect>
    <text x="157" y="9.5" dy=".35em">16</text>
  </g>
  <g transform="translate(0,80)">
    <rect width="230" height="19"></rect>
    <text x="227" y="9.5" dy=".35em">23</text>
  </g>
  <g transform="translate(0,100)">
    <rect width="420" height="19"></rect>
    <text x="417" y="9.5" dy=".35em">42</text>
  </g>
</svg>

Full code to do it manually
D3 Bar Chart Tutorial

Full code to do it automatically

```html
<!DOCTYPE html>
<meta charset="utf-8">
<style>
.chart rect {
  fill: steelblue;
}
.chart text {
  fill: white;
  font: 10px sans-serif;
  text-anchor: end;
}
</style>
<svg class="chart"></svg>
<script src="/d3js.org/d3.v3.min.js" charset="utf-8"></script>
```
D3 Bar Chart Tutorial

Full code to do it automatically

```javascript
<script>
var data = [4, 8, 15, 16, 23, 42];

var width = 420,
    barHeight = 20;

var x = d3.scale.linear()
    .domain([0, d3.max(data)])
    .range([0, width]);

var chart = d3.select(".chart")
    .attr("width", width)
    .attr("height", barHeight * data.length);

var bar = chart.selectAll("g")
    .data(data)
    .enter().append("g")
    .attr("transform", function(d, i) { return "translate(0," + i * barHeight + ")"; });

bar.append("rect")
    .attr("width", x)
    .attr("height", barHeight - 1);

bar.append("text")
    .attr("x", function(d) { return x(d) - 3; })
    .attr("y", barHeight / 2)
    .attr("dy", ".35em")
    .text(function(d) { return d; });
</script>
```
D3 Bar Chart Tutorial

Load data

```javascript
// 1. Code here runs first, before the download starts.

d3.tsv("data.tsv", function(error, data) {
    // 3. Code here runs last, after the download finishes.
});

// 2. Code here runs second, while the file is downloading.

name		value
Locke	4
Reyes	8
Ford	15
Jarrah	16
Shephard	23
Kwon	42

The equivalent of Javascript code:
```
<!DOCTYPE html>
<meta charset="utf-8">
<style>
.chart rect {
  fill: steelblue;
}

.chart text {
  fill: white;
  font: 10px sans-serif;
  text-anchor: end;
}
</style>
<svg class="chart"></svg>
<script src="//d3js.org/d3.v3.min.js" charset="utf-8"></script>
D3 Bar Chart Tutorial

```javascript
<script>
var width = 420,
    barHeight = 20;

var x = d3.scale.linear()
    .range([0, width]);

var chart = d3.select(".chart")
    .attr("width", width);

d3.tsv("data.tsv", type,
    function(error, data) {
        x.domain([0, d3.max(data, function(d) { return d.value; })]);
        chart.attr("height", barHeight * data.length);
    });

var bar = chart.selectAll("g")
    .data(data)
    .enter().append("g")
    .attr("transform", function(d, i) { return "translate(0," + i * barHeight + ")"; });

bar.append("rect")
    .attr("width", function(d) { return x(d.value); })
    .attr("height", barHeight - 1);

bar.append("text")
    .attr("x", function(d) { return x(d.value) - 3; })
    .attr("y", barHeight / 2)
    .attr("dy", ".35em")
    .text(function(d) { return d.value; });

function type(d) {
    d.value = +d.value; // coerce to number
    return d;
}
</script>
D3 Example Circles

https://bost.ocks.org/mike/circles/

```html
<svg width="720" height="120">
  <circle cx="40" cy="60" r="10"></circle>
  <circle cx="80" cy="60" r="10"></circle>
  <circle cx="120" cy="60" r="10"></circle>
</svg>
```

```javascript
var circle = d3.selectAll("circle");

circle.style("fill", "steelblue");
circle.attr("r", 30);
```
D3 Example Circles

https://bost.ocks.org/mike/circles/

```html
<svg width="720" height="120">
  <circle cx="40" cy="60" r="10"></circle>
  <circle cx="80" cy="60" r="10"></circle>
  <circle cx="120" cy="60" r="10"></circle>
</svg>
```

```javascript
var circle = d3.selectAll("circle");
circle.style("fill", "steelblue");
circle.attr("r", 30);

circle.attr("cx", function() { return Math.random() * 720; });
```
<table>
<thead>
<tr>
<th>Sunburst</th>
<th>Node-Link Tree</th>
<th>Treemap</th>
<th>Voronoi Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Sunburst" /></td>
<td><img src="image" alt="Node-Link Tree" /></td>
<td><img src="image" alt="Treemap" /></td>
<td><img src="image" alt="Voronoi Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hierarchical Edge Bundling</th>
<th>Voronoi Diagram</th>
<th>Symbol Map</th>
<th>Parallel Coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Hierarchical Edge Bundling" /></td>
<td><img src="image" alt="Voronoi Diagram" /></td>
<td><img src="image" alt="Symbol Map" /></td>
<td><img src="image" alt="Parallel Coordinates" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scatterplot Matrix</th>
<th>Zoomable Pack Layout</th>
<th>Hierarchical Bars</th>
<th>Epicyclical Gears</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Scatterplot Matrix" /></td>
<td><img src="image" alt="Zoomable Pack Layout" /></td>
<td><img src="image" alt="Hierarchical Bars" /></td>
<td><img src="image" alt="Epicyclical Gears" /></td>
</tr>
<tr>
<td>D3 Gallery</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fisheye Distortion</td>
<td>Hive Plot</td>
<td>Co-occurrence Matrix</td>
<td>Motion Chart</td>
</tr>
<tr>
<td><img src="image1" alt="Fisheye Distortion" /></td>
<td><img src="image2" alt="Hive Plot" /></td>
<td><img src="image3" alt="Co-occurrence Matrix" /></td>
<td><img src="image4" alt="Motion Chart" /></td>
</tr>
<tr>
<td>Chord Diagram</td>
<td>Animated Béziers</td>
<td>Zoomable Sunburst</td>
<td>Collatz Graph</td>
</tr>
<tr>
<td><img src="image5" alt="Chord Diagram" /></td>
<td><img src="image6" alt="Animated Béziers" /></td>
<td><img src="image7" alt="Zoomable Sunburst" /></td>
<td><img src="image8" alt="Collatz Graph" /></td>
</tr>
<tr>
<td>Parallel Sets</td>
<td>Word Cloud</td>
<td>Obama's Budget Proposal</td>
<td>Facebook IPO</td>
</tr>
<tr>
<td><img src="image9" alt="Parallel Sets" /></td>
<td><img src="image10" alt="Word Cloud" /></td>
<td><img src="image11" alt="Obama's Budget Proposal" /></td>
<td><img src="image12" alt="Facebook IPO" /></td>
</tr>
</tbody>
</table>
D3 Gallery

US Trade Deficit

Sequences sunburst

NFL Predictions

Koalas to the Max

Sankey Creator

Convert any page into bubbles

D3 Builder
Graphe Database — RDF and SPARQL
Network / Graph is the way we remember, we associate, and we understand.
Example: Graph Technology for Financial Service Sectors

- Graph Database is much more efficient than traditional relational database

- How does FINRA analyze ~50B events per day TODAY? - Build a graph of market order events from multiple sources [ref]

- How did journalists uncover the Swiss Leak scandal in 2014 and also Panama Papers in 2016? -- Using graph database to uncover information thousands of accounts in more than 20 countries with links through millions of files [ref]
WHAT DO RDF AND SPARQL BRING TO BIG DATA PROJECTS?

Bob DuCharme
TopQuadrant, Charlottesville, Virginia

Resource Description Format (RDF)

- A W3C standard since 1999
- Triples
- Example: A company has nine of part p1234 in stock, then a simplified triple representing this might be {p1234 inStock 9}.
- Instance Identifier, Property Name, Property Value.
- In a proper RDF version of this triple, the representation will be more formal. They require uniform resource identifiers (URIs).

```r
@prefix fbd:<http://foobarco.net/data/>.
@prefix fbv:<http://foobarco.net/vocab/>.

fbd:p1234 fbv:inStock "9".
fbd:p1234 fbv:supplier "Joe’s Part Company".
```
An example complete description

```xml
@prefix fbd:<http://foobarco.net/data/>.
@prefix fbv:<http://foobarco.net/vocab/>.
fbd:p1234 fbv:inStock "9".
fbd:p1234 fbv:name "Blue reverse flange".
fbd:p1234 fbv:supplier fbd:s9483.
fbd:s9483 fbv:name "Joe's Part Company".
fbd:s9483 fbv:homePage "http://www.joespartco.com".
fbd:s9483 fbv:contactName "Gina Smith".
fbd:s9483 fbv:contactEmail "gina.smith@joespartco.com".
```
Advantages of RDF

- Virtually any RDF software can parse the lines shown above as self-contained, working data file.
  - You can declare properties if you want.
  - The RDF Schema standard lets you declare classes and relationships between properties and classes.
  - The flexibility that the lack of dependence on schemas is the first key to RDF's value.

- Split trips into several lines that won't affect their collective meaning, which makes sharding of data collections easy.
  - Multiple datasets can be combined into a usable whole with simple concatenation.

- For the inventory dataset's property name URIs, sharing of vocabulary makes easy to aggregate.
The following SPQRL query asks for all property names and values associated with the fbd:s9483 resource:

```sparql
PREFIX fbd:<http://foobarco.net/data/>
SELECT ?property ?value
WHERE {fbd:s9483 ?property ?value.}
```

The heart of any SPARQL query is the WHERE clause, which specifies the triples to pull out of the dataset. Various options for the rest of the query tell the SPARQL processor what to do with those triples, such as sorting, creating, or deleting triples. The above query’s WHERE clause has a single triple pattern, which resembles a triple but may have variables substituted for any or all of the triple’s three parts. The triple pattern above says that we’re interested in triples that have fbd:s9483 as the subject and—because variables function as wildcards—anything at all in the triple’s second and third parts.
The SPAQRL Query Result from the previous example

<table>
<thead>
<tr>
<th>property</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;<a href="http://foobarto.net/vocab/contactEmail">http://foobarto.net/vocab/contactEmail</a>&gt;</td>
<td>&quot;<a href="mailto:gina.smith@joespartco.com">gina.smith@joespartco.com</a>&quot;</td>
</tr>
<tr>
<td>&lt;<a href="http://foobarto.net/vocab/contactName">http://foobarto.net/vocab/contactName</a>&gt;</td>
<td>&quot;Gina Smith&quot;</td>
</tr>
<tr>
<td>&lt;<a href="http://foobarto.net/vocab/homePage">http://foobarto.net/vocab/homePage</a>&gt;</td>
<td>&quot;<a href="http://www.joespartco.com">http://www.joespartco.com</a>&quot;</td>
</tr>
<tr>
<td>&lt;<a href="http://foobarto.net/vocab/name">http://foobarto.net/vocab/name</a>&gt;</td>
<td>&quot;Joe's Part Company&quot;</td>
</tr>
</tbody>
</table>
Another SPARQL Example

What is this query for?

```
PREFIX fbd:<http://foobarco.net/data/>
PREFIX fbv:<http://foobarco.net/vocab/>

SELECT ?flangeContactEmail
WHERE
{
  ?part fbv:name "Blue reverse flange".
  ?supplier fbv:contactEmail ?flangeContactEmail.
}
```

Data

```
@prefix fbd:<http://foobarco.net/data/>.
@prefix fbv:<http://foobarco.net/vocab/>.

fbd:p1234 fbv:inStock "9".
fbd:p1234 fbv:name "Blue reverse flange".
fbd:p1234 fbv:supplier fbd:s9483.
fbd:s9483 fbv:name "Joe’s Part Company".
fbd:s9483 fbv:homePage "http://www.joespartco.com".
fbd:s9483 fbv:contactName "Gina Smith".
fbd:s9483 fbv:contactEmail "gina.smith@joespartco.com".
```
Open Source Software – Apache Jena

A free and open source Java framework for building Semantic Web and Linked Data applications.

➤ Get started now!  Download

RDF

RDF API

Interact with the core API to create and read Resource Description Framework (RDF) graphs. Sanitise your triples using popular formats such as RDF/XML or Turtle.

ARQ (SPARQL)

Query your RDF data using ARQ, a SPARQL 1.1 compliant engine. ARQ supports remote federated

Triple store

TDB

Persist your data using TDB, a native high performance triple store. TDB supports the full range of Jena APIs.

Fuseki

Expose your triples as a SPARQL end-point accessible over HTTP. Fuseki provides REST-style interaction with your RDF data.

OWL

Ontology API

Work with models, RDFS and the Web Ontology Language (OWL) to add extra semantics to your RDF data.

Inference API

Reason over your data to expand and check the content of your triple store. Configure your own inference rules or
Graph Database — Property Graphs
A usual example

![Graph Database Diagram]

**Figure 2-1.** Semantic relationships are hidden in a relational database.
Query Example – I

![Diagram of Person and PersonFriend tables]

Figure 2-2. Modeling friends and friends-of-friends in a relational database

Asking “who are Bob’s friends?” is easy, as shown in Example 2-1.

Example 2-1. Bob’s friends

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
    ON PersonFriend.FriendID = p1.ID
JOIN Person p2
    ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```
Example 2-2. Who is friends with Bob?

```sql
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
    ON PersonFriend.PersonID = p1.ID
JOIN Person p2
    ON PersonFriend.FriendID = p2.ID
WHERE p2.Person = 'Bob'
```

Example 2-3. Alice’s friends-of-friends

```sql
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
    ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
    ON pf2.PersonID = pf1.FriendID
JOIN Person p2
    ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```
Graph Database Example

Figure 2-5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph.
Execution Time in the example of finding extended friends (by Neo4j)

Partner and Vukotic’s experiment seeks to find friends-of-friends in a social network, to a maximum depth of five. Given any two persons chosen at random, is there a path that connects them that is at most five relationships long? For a social network containing 1,000,000 people, each with approximately 50 friends, the results strongly suggest that graph databases are the best choice for connected data, as we see in Table 2-1.

Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

<table>
<thead>
<tr>
<th>Depth</th>
<th>RDBMS execution time (s)</th>
<th>Neo4j execution time (s)</th>
<th>Records returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.016</td>
<td>0.01</td>
<td>~2500</td>
</tr>
<tr>
<td>3</td>
<td>30.267</td>
<td>0.168</td>
<td>~110,000</td>
</tr>
<tr>
<td>4</td>
<td>1543.505</td>
<td>1.359</td>
<td>~600,000</td>
</tr>
<tr>
<td>5</td>
<td>Unfinished</td>
<td>2.132</td>
<td>~800,000</td>
</tr>
</tbody>
</table>
Modeling Order History as a Graph

Figure 2-6. Modeling a user’s order history in a graph
Figure 3-1. A simple graph pattern, expressed using a diagram

This pattern describes three mutual friends. Here’s the equivalent ASCII art representation in Cypher:

(a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
Cypher Example

Like most query languages, Cypher is composed of clauses. The simplest queries consist of a START clause followed by a MATCH and a RETURN clause (we’ll describe the other clauses you can use in a Cypher query later in this chapter). Here’s an example of a Cypher query that uses these three clauses to find the mutual friends of user named Michael:

START a=node:user(name='Michael')
MATCH (a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
RETURN b, c
Other Cypher Clauses

WHERE
   Provides criteria for filtering pattern matching results.

CREATE and CREATE UNIQUE
   Create nodes and relationships.

DELETE
   Removes nodes, relationships, and properties.

SET
   Sets property values.

FOREACH
   Performs an updating action for each element in a list.

UNION
   Merges results from two or more queries (introduced in Neo4j 2.0).

WITH
   Chains subsequent query parts and forward results from one to the next. Similar to piping commands in Unix.
Property Graph Example – Shakespeare

Figure 3-6. Three domains in one graph
Creating the Shakespeare Graph

CREATE (shakespeare { firstname: 'William', lastname: 'Shakespeare' }),(juliusCaesar { title: 'Julius Caesar' }),(shakespeare)-[:WROTE_PLAY { year: 1599 }]->(juliusCaesar),(theTempest { title: 'The Tempest' }),(shakespeare)-[:WROTE_PLAY { year: 1610 }]->(theTempest),(rsc { name: 'RSC' }),(production1 { name: 'Julius Caesar' }),(rsc)-[:PRODUCED]->(production1),(production1)-[:PRODUCTION_OF]->(juliusCaesar),(performance1 { date: 20120729 }),(performance1)-[:PERFORMANCE_OF]->(production1),(production2 { name: 'The Tempest' }),(rsc)-[:PRODUCED]->(production2),(production2)-[:PRODUCTION_OF]->(theTempest),(performance2 { date: 20061121 }),(performance2)-[:PERFORMANCE_OF]->(production2),(performance3 { date: 20120730 }),(performance3)-[:PERFORMANCE_OF]->(production1),(billy { name: 'Billy' }),(review { rating: 5, review: 'This was awesome!' }),(billy)-[:WROTE_REVIEW]->(review),(review)-[:RATED]->(performance1),(theatreRoyal { name: 'Theatre Royal' }),(performance1)-[:VENUE]->(theatreRoyal),(performance2)-[:VENUE]->(theatreRoyal),(performance3)-[:VENUE]->(theatreRoyal),(greyStreet { name: 'Grey Street' }),(theatreRoyal)-[:STREET]->(greyStreet),(newcastle { name: 'Newcastle' }),(greyStreet)-[:CITY]->(newcastle),(tyneAndWear { name: 'Tyne and Wear' }),(newcastle)-[:COUNTY]->(tyneAndWear), (england { name: 'England' }),(tyneAndWear)-[:COUNTRY]->(england),(stratford { name: 'Stratford upon Avon' }),(stratford)-[:COUNTRY]->(england),(rsc)-[:BASED_IN]->(stratford),(shakespeare)-[:BORN_IN]->(stratford)
Query on the Shakespeare Graph

```
START theater=node:venue(name='Theatre Royal'),
    newcastle=node:city(name='Newcastle'),
    bard=node:author(lastname='Shakespeare')
MATCH (newcastle)<-[[: STREET | CITY*1..2]]-(theater)
    <-[: VENUE ]()-[: PERFORMANCE_OF ]>()-[[: PRODUCTION_OF ]]-
    (play)<-[[: WROTE_PLAY ]]- (bard)
WHERE w.year > 1608
RETURN DISTINCT play.title AS play
```

Adding this WHERE clause means that for each successful match, the Cypher execution engine checks that the WROTE_PLAY relationship between the Shakespeare node and the matched play has a year property with a value greater than 1608. Matches with a WROTE_PLAY relationship whose year value is greater than 1608 will pass the test; these plays will then be included in the results. Matches that fail the test will not be included in the results. By adding this clause, we ensure that only plays from Shakespeare’s late period are returned:

```
+-------------+
| play        |
+-------------+
| "The Tempest" |
+-------------+
1 row
```
Another Query on the Shakespeare Graph

```
START theater=node:venue(name='Theatre Royal'),
    newcastle=node:city(name='Newcastle'),
    bard=node:author(lastname='Shakespeare')
MATCH (newcastle)<-[[:STREET|CITY*1..2]]-(theater)
    <[:VENUE]-(p:PERFORMANCE_OF)->()-[:PRODUCTION_OF]->(play)<-[:WROTE_PLAY]-(bard)
RETURN play.title AS play, count(p) AS performance_count
ORDER BY performance_count DESC
```

The RETURN clause here counts the number of PERFORMANCE_OF relationships using the identifier p (which is bound to the PERFORMANCE_OF relationships in the MATCH clause) and aliases the result as performance_count. It then orders the results based on performance_count, with the most frequently performed play listed first:

```
+--------------------------+------------------+
| play                     | performance_count |
+--------------------------+------------------+
| "Julius Caesar"          | 2                |
| "The Tempest"            | 1                |
+--------------------------+------------------+
2 rows
```
Building Application Example – Collaborative Filtering

Figure 4-1. Data model for the book reviews user story

Because this data model directly encodes the question presented by the user story, it lends itself to being queried in a way that similarly reflects the structure of the question we want to ask of the data:

```
START reader=node:users(name={readerName})
MATCH reader-[LIKE]->book<-[LIKE]-other_readers-[LIKE]->books
RETURN book.title
```
Chaining on the Query

```
START bard=node:author(lastname='Shakespeare')
MATCH (bard)-[w:WROTE_PLAY]->(play)
WITH play
ORDER BY w.year DESC
RETURN collect(play.title) AS plays
```

Executing this query against our sample graph produces the following result:

```
+-----------------+
| plays           |
+-----------------+
| ["The Tempest","Julius Caesar"] |
+-----------------+
1 row
```
Example – Email Interaction Graph

What's this query for?

START bob=node:user(username='Bob')
MATCH (bob)-[:SENT]->(email)-[:CC]->(alias),
  (alias)-[:ALIAS_OF]->(bob)
RETURN email

Figure 3-10. A graph of email interactions
How to make graph database fast?

Figure 6-1. Nonnative graph processing engines use indexing to traverse between nodes
Use Relationships, not indexes, for fast traversal
Storage Structure Example

Figure 6-4. Neo4j node and relationship store file record structure
An experiment

**Dataset:** 12.2 million edges, 2.2 million vertices

**Goal:** Find paths in a property graph. One of the vertex property is call TYPE. In this scenario, the user provides either a particular vertex, or a set of particular vertices of the same TYPE (say, "DRUG"). In addition, the user also provides another TYPE (say, "TARGET"). Then, we need find all the paths from the starting vertex to a vertex of TYPE “TARGET”. Therefore, we need to 1) find the paths using graph traversal; 2) keep trace of the paths, so that we can list them after the traversal. Even for the shortest paths, it can be multiple between two nodes, such as: drug->assay->target , drug->MOA->target

<table>
<thead>
<tr>
<th></th>
<th>Avg time (100 tests)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>First test (cold-start)</td>
</tr>
<tr>
<td>NativeStore C++</td>
<td>39 sec</td>
</tr>
<tr>
<td>NativeStore JNI</td>
<td>57 sec</td>
</tr>
<tr>
<td>Neo4j (Blueprints 2.4)</td>
<td>105 sec</td>
</tr>
<tr>
<td>Titan (Berkeley DB)</td>
<td>3861 sec</td>
</tr>
<tr>
<td>Titan (HBase)</td>
<td>3046 sec</td>
</tr>
</tbody>
</table>

**First full test** - full depth 23. All data pulled from disk. Nothing initially cached.

**Modes** - All tests in default modes of each graph implementation. Titan can only be run in transactional mode. Other implementations do not default to transactional mode.
Native Store Overview

- **Native store represents graphs in-memory and on-disk**
  - Organizing graph data for representing a graph that stores both graph structure and vertex properties and edge properties
  - Caching graph data in memory in either batch-mode or on-demand from the on-disk streaming graph data
  - Accepting graph updates and modifying graph structure and/or property data accordingly and incorporating time stamps
    - Add edge, remove vertex, update property, etc.
  - Persisting graph updates along with the time stamps from in-memory graph to on-disk graph
  - Performing graph queries by loading graph structure and/or property data
    - Find neighbors of a vertex, retrieve property of an edge, traverse a graph, etc.

Diagram:

- Graph data
- Graph queries
- In-memory cached graph
- On-disk persistent graph
- Graph engine
- Time stamp control
On-Disk Graph Organization

- Native store organizes graph data for representing a graph with both structure and the vertex properties and edge properties using multiple files in Linux file system
  - Creating a list called ID → Offset where each element translates a vertex (edge) ID into two offsets, pointing to the earliest and latest data of the vertex/edge, respectively
  - Creating a list called Time_stamp → Offset where each element has a time stamp, an offset to the previous time stamp of the vertex/edge, and a set of indices to the adjacent edge list and properties
  - Create a list of chained block list to store adjacent list and properties
Impact from Storage Hardware

- Convert csv file (adams.csv 20G) to datastore
  - Similar performance: 7432 sec versus 7806 sec
- CPU intensive
  - Average CPU util.: 97.4 versus 97.2
- I/O pattern
  - Maximum read rate: 5.0 vs. 5.3
  - Maximum write rate: 97.7 vs. 85.3

SSD offers consistently higher performance for both read and write

<table>
<thead>
<tr>
<th>Ratio HDD/SDD</th>
<th>TYPE1</th>
<th>TYPE2</th>
<th>TYPE3</th>
<th>TYPE4</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.79</td>
<td>6.36</td>
<td>19.93</td>
<td>2.44</td>
<td></td>
</tr>
</tbody>
</table>

Queries
- Type 1: find the most recent URL and PCID of a user
- Type 2: find all the URLs and PCIDs
- Type 3: find all the most recent properties
- Type 4: find all the historic properties
Impact from Storage Hardware — 2

- Dataset: Knowledge Repository
  - 138614 Nodes, 1617898 Edges
- OS buffer is flushed before test
- Processing 320 queries in parallel
- In memory graph cache size: 4GB (default value)