Book Reviews

MARK LAMOURINE

Graph Databases

Ian Robinson, Jim Webber, and Emil Eifrem
Neo Technology Inc., O'Reilly Media, 2013, 210 pages
ISBN 978-1-449-35626-2

Graph Databases starts off really well. It opens with a definition and then a detailed description of what makes up a graph database and what makes graph databases special.

Some of it is pretty striking. I remember the "aha!" moment when I read that conventional relational databases don't contain relationships as first-class entities. It's true. Relational databases force all data into a table form, and then relationships are represented using second-class column types (FOREIGN KEY) and JOIN operations. In a graph database the relationships between data entities are first-class entities of their own.

The authors present ways of representing data relationships both graphically (naturally enough) and textually, as is required to communicate with a service over network (serialized) media. It's when they get to demonstration and implementation that I started to lose my enthusiasm.

While the title might lead a reader to believe that the book covers graph databases in general, it turns out that only one database system is presented. In the middle of Chapter 4, the authors begin talking about implementation options, but the only real option offered is Neo4j. I continued reading on, hoping to find some variation or alternatives, something to lead back to the general topic of graph databases, and alternate implementations, but I was disappointed. At some point I stopped and looked up two different things:

1. Graph database implementations—there are dozens
2. The authors—all employees of Neo4j

In fact, the copyright for the book is held by Neo4j Inc., not by the authors. To be fair, the forward and the bios on the back cover both make it clear that the three authors are the co-founders of Neo4j. If the title of the book had been "Graph Databases with Neo4j" I would not have felt so disappointed, but then I might not have picked it up in the first place. Once I realigned my expectations, I went back to reading.

In the remainder of the book, the authors talk about real-world instances of data well suited to graph modeling and representation in a graph database. Many applications today represent the links between people, objects, and concepts in a mesh or network (in the mathematical or social sense). Whether the application is an enterprise network or a social one, the links between nodes are more important than the individual attributes when trying to discover patterns of behavior or data flow.

Near the end of the book, the authors discuss database internals with an eye to performance, reliability, and scaling. They close with a chapter on how common problems can be represented more intuitively with a graph than with more traditional data structures.

Overall, the writing is clear, and the progression led me to a better understanding of how a graph database works. I will definitely be more likely to recognize an application that would suit modeling and implementation with a graph database, and I would certainly consider Neo4j for the implementation. I do wish that the focus on Neo4j had been more explicit in the title and on the front cover. If you want to learn about Neo4j, this is certainly a good place to begin.

Interactive Data Visualization for the Web

Scott Murray
O'Reilly Media Inc., 2013, 256 pages
ISBN 978-1-449-33973-9

I spend a fair amount of time drawing boxes and lines when I'm trying to explain things in documentation. Often I'm representing tabular data or collections with relationships. When it comes to representing data sets that change over time, I generally don't even try.

Recently, though, I had a problem I needed to understand myself: What are the relationships between RPMs with dependencies? And what effect does the addition of a single package with complex dependencies have on the total package set installed on a host? I didn't want to see just the list of new packages to add but where (and why) each new package was pulled in. I also wanted to be able to view what would change if I added just a subset or tried to remove a dependency. I remembered that a coworker had created some great dynamic visualizations using reveal.js, but that by itself didn't seem to have the data representation I was looking for. I asked him what he used. He pointed me at D3, and I went looking for books.

D3 is a JavaScript library (available at http://d3js.org). You embed it in your application in the same way you would JQuery or any other JavaScript library. D3 offers me a capability for browser-based graphics and data visualization that I just would have found impossible otherwise, and Scott Murray's book was a great way to get started.

Murray starts off with the traditional definition and features, but he's careful to outline what D3 is and is not good for. He devotes an entire section to other tools that might suit better than D3 depending on your needs and your application. He spends another large section bringing readers new to Web programming for browsers up to speed. There's a short section on HTML and

DOM programming as well as simple SVG and HTML canvas programming. This is sprinkled with outside resources so readers can get more detail and come back if needed.

This is when Murray really gets rolling. He shows how to use D3 to retrieve your data from a Web server and how to use it to populate your document with HTML and SVG elements, which the browser will draw for you. Simple graphing such as scatterplots and bar charts requires some labeling and scales for the axes, and each of these gets a section, as do dynamic updates and user interaction.

Things get really cool when Murray gets to D3 layouts. Layouts provide ways for D3 to automatically place the elements and draw them and even move them around in response to clicks and drags in the browser. When using layouts, you don't specify where each data element will be placed. Rather, D3 does it for you dynamically based on the data values themselves.

Murray demonstrates three common layouts. The Pie and Stack layouts yield fairly common-looking graphics. The Force layout, Murray admits, is overused because it is so cool. In a Force layout each data object is assigned a repulsive force. The elements are arranged randomly at first (constrained by links between elements that are related in some way), and the Force layout applies the forces, moving each element until they reach equilibrium. This is very slick to watch and it is seductive. I used the Force layout for my RPM dependency data, but a Tree layout might have been more appropriate, and I am going to try it to see.

The layout and writing of Interactive Data Visualization are themselves appealing. This is one of the first books of this type that I've read which uses full color for both the illustrations and for the example texts. The source code and HTML representations are taken from the Safari browser debugging tool set, and the colorization of the text is a welcome and familiar feature (although I use Emacs and Chrome).

As you may have noticed, I'm pretty enthusiastic about both D3 and Interactive Data Visualization. As a sysadmin with a strong coding background and some experience with JavaScript and browser development, I have with D3 a powerful new tool for understanding and explaining the behavior of the systems I'm working on using output from the CLI tools I already have, but it's unlikely that I'll use it often enough to stay fluent. Murray's book is one I'll return to for a refresher when I find a new question that cries out for a graphic representation.

Scratch Programming in Easy Steps

Sean McManus
In Easy Steps Ltd., 2014, 216 pages
ISBN 978-1-84078-1

I've looked at Scratch and reviewed books on it before, but something made me pull Scratch Programming off the shelf at my local bookstore and thumb through it. I was impressed instantly with its high-quality feel, the texture of the paper, and the weight of it in my hand. It's not a thick volume but its heft makes me think it's likely to be durable in the hands of the middle school students I think it is meant for.

Once I started working through the book, I noticed something else right away: Scratch has become a Web application. Scratch 2.0 is hosted by MIT at http://scratch.mit.edu. You can open the development screen in any modern browser. If you create an account and log in, you can save and publish your applications. Scratch 1.4 is still available as a standalone application. Scratch Programming uses Scratch 2.0 as its base, but it also includes information on running Scratch 1.4 on a Raspberry Pi. The text consistently and clearly includes graphics and instructions that show how 1.4 will differ from the default.

Getting and keeping the interest of middle schoolers can be a challenge. In books like this, I look for the hooks that will help hold the attention and enthusiasm of the students. McManus does a great job of mixing narrative with engaging graphics and layout to sustain interest.

Scratch Programming is color coded to make it easy to find one's place and return quickly to work when picking the book up, and each chapter presents a project, a mixture of simple graphical games, musical applications for sound, quiz games, and logic puzzles. Each section brings a new aspect of coding and makes good use of the Scratch graphical programming model to illustrate the points. McManus leaves enough room for the kids to experiment, make mistakes, and discover the solutions for themselves. He builds each concept or construct in a linear way, allowing for the reader to race ahead or off on a tangent and be guided gently back.

The last two sections bring in discussions of hardware sensors that can be used to provide additional inputs for game behavior and response. McManus covers using a computer's Webcam or a USB device called a "Picoboard," which can interface with Scratch to respond to sounds (such as a handclap) and changes in light or temperature.

In the final chapter, McManus provides seven short, complete programs and encourages the reader to experiment with them, changing parameters or logic and observing how the changes affect the behavior of the program.

I'm impressed with *Scratch Programming*, and I actually followed several of the projects to completion in Scratch because I was having fun in a way I hadn't since I'd done similar things on the Apple in my own high school. I have a couple of friends with girls the right age who've expressed interest in coding, and I mean to pass on the review copy and maybe even buy an additional one to give to them.

USENIX Board of Directors

Communicate directly with the USENIX Board of Directors by writing to board@usenix.org.

USENIX Member Benefits

Members of the USENIX Association receive the following benefits:

- **Free subscription** to *;login:*, the Association's magazine, published six times a year, featuring technical articles, system administration articles, tips and techniques, practical columns on such topics as security, Perl, networks, and operating systems, book reviews, and reports of sessions at USENIX conferences.
- **Access** to *;login:* online from December 1997 to the current month: www.usenix.org/publications/login/
- **Discounts** on registration fees for all USENIX conferences.
- **Special discounts** on a variety of products, books, software, and periodicals: www.usenix.org/member-services/discount-instructions
- **The right to vote** on matters affecting the Association, its bylaws, and election of its directors and officers.

For more information regarding membership or benefits:
please see www.usenix.org/membership/
or contact office@usenix.org. Phone: 510-528-8649