Portfolio 3

Wikipedia Pathfinding Web App Implemented using Neo4j

Ben Meeder and Noah Eigenfeld

Table of Contents

Overview

The reddit community r/DegreesToHitler is a popular internet community that attempts to connect the Wikipedia article on “Adolf Hitler” with any other article on Wikipedia. They often find funny and creative paths such as `Comcast -> Literal -> Hitler`. It was this internet community that first proposed the “Six degrees to Hitler” theory. This theory states that any article on Wikipedia can be linked to Hitler in six clicks or less. This theory is what inspired this portfolio project. The idea behind this project was to find the shortest series of clicks that it takes from any article on Wikipedia to get to Adolf Hitler. After completion it has become a challenge amongst people that have used to website to find an article that does not link back to Hitler and as of writing this none have been discovered. This Once again proves that everything in this world is closely related to Adolf Hitler.

The simplest way to represent any website is a series of pages with links to other pages or as nodes with relationships to other nodes. So step one is to convert a website into a graphical database. To do this I found a java library that parses the Wikipedia XML file which has a approximate size of 15Gb and converts each file so that only the title and links remain. This operation reduces the file size to around 5Gb, still a rather large data dump. Finally it goes back and constructs a Neo4j data store using those articles and links between them. The final database has 14 million nodes and over 100 million relationships between nodes.

The next step was to provide a search box to query all of Wikipedia's articles. The easiest way to do this is to make calls to Wikipedia’s REST API using an HTTP request that returns JSON. The hardest part implementing this was the cross domain policy which blocks http requests across domains. To get around this I set the data type of the request to be JSONP which opens up the website to javascript attacks if not sending the request to a trusted source. However since the Wikimedia Foundation is rather trustworthy it was decided this method could be used.

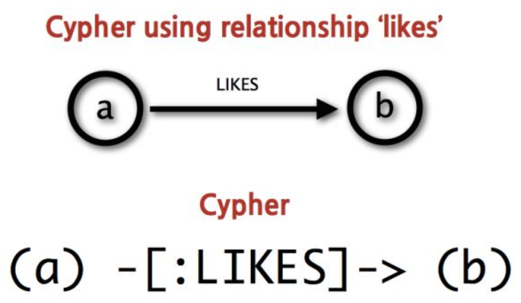
Next was finding a library that connects to a Neo4j database to allow the running of queries against it. After looking into several options it was determined that the best way to do this was using a Composer package called Graphaware. Composer is a PHP dependency manager and it allowed the package and all of its dependencies to be downloaded to the host with one simple command of `composer require Graphaware`. Once a connection is made to the database it can run commands in a way very similar to the PHP built in mysqli commands.

The UI was constructed using basic javascript and jquery calls to manipulate the DOM. The idea was to construct a UI that was easy for non-computer science people to understand. In future versions it could be helpful for the search bar to also be incorporated on this page to make for easier use.

New and Complex

Neo4j
Neo4j is a nosql graphical database implemented using the java programming language. It is currently the most popular graphical database in the world and is used by thousands of organizations such as Walmart and ebay in their daily operations. The idea behind graphical databases is to focus on relationships as opposed to data itself. A mysql or document database might store large amounts of user or document data but they struggle to describe the relationships between those users and documents efficiently. This is where a graphical database comes in. The best use case for a graphical database is in a social network. Each user has a node that links to a post node. Other users can like that post which is another link from that other user to the post node. This schema makes for calls such as “How many users like this post” and “How many users who like this post also like this other post” very easy and quick to compute when compared to other forms of databases.

Neo4j also developed its own query language in order to add, manipulate, and extract data from Neo4j databases. This query language is called Cypher. A simple Cypher query has a MATCH value and a return value. To find a node with a certain property let’s say “name” all the programmer has to do is run `MATCH n WHERE n.name=”Adam” RETURN n`. Relationships between nodes can be described by using the -[]-> operator where the arrow specifies the direction and the area between the square brackets can be used to specify the type of relationship required. This syntax allows for very complex queries to be written in a short amount of text and forqueries to be easily human readable. By implementing a custom query language like Cypher, Neo4j continues to be on the leading edge of nosql graphical database design.



REST API
Restful API’s are a way for websites to allow for simple external API’s to services. RESTful API’s are often HTTP based and return JSON. Wikipedia did just that by implementing many API’s by sending a GET request to a URL with parameters. The

main URL that Wikipedia hosts their RESTful API's is at: https://en.Wikipedia.org/w/API.php. To call a function of this API the only thing that needs to be done is the change the URL at which the GET request is being called. For our project we wanted to be able to run a text search against the API and get article titles matching that search value. To do this one simply files a get request to: http://en.Wikipedia.org/w/API.php?action=opensearch&search=SEARCH_VALUE&format=json&callback=spellcheck. The parameters specified to the URL are what make the API calls non-static.

The biggest challenge with implementing this RESTful API was getting around the same-origin policy. It is a security feature of many modern day browsers to block HTTP requests from one website to another website on a different domain. To get around this problem we set to AJAX dataType to "JSONP". JSON with Padding or JSONP is a JSON extension developed with the intention of overcoming the same-origin policy. JSONP works by attaching a script tag to the DOM with the src value set to the specified REST URL. Then javascript functions are used to interpret the inner value of the script tag which is the current return value from the API call.

CSS Animations

CSS Animations allow the changing of DOM element styles in a cycle without the use of HTML or Javascript. This greatly improves the complexity and performance of any animations on a website. This project used CSS animations to render a loading animation to an asynchronous database request. Since the request is not instantaneous the animation allows the user to see some feedback that the page did not throw an error or time out. A very basic animation is simply changing the color of a div element.

```
/* The animation code */
@keyframes example {
        from {background-color: red;}
        to {background-color: yellow;}
}

/* The element to apply the animation to */
div {
        width: 100px;
        height: 100px;
        background-color: red;
        animation-name: example;
        animation-duration: 4s;
}
```

The @keyframes defines an artbritarty number of states and then the amination can be attached as a css-selector animation-name. The animation then changes state every amount of time as defined by animation-duration css option. Our spinner is slightly more complicated due to there being multiple different style changes but the basic idea is the same.

Bloom’s Taxonomy

Analysis

A graphical database is much more abstract than a nosql database. In a nosql database tuples are defined within a rigid schema and relations between those tuples are defined by unique foreign key values. In graphical databases, specifically neo4j, nodes have a much looser schema and can be defined on the fly. In Neo4j a node has a type and properties. A type is just a label that is used for search queries and the properties are the values that get attached to the node which are much like the values in a sql tuple.

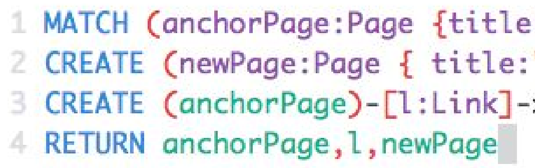
The creation of a node is one of the most basic operations that can be performed in a graphical database. The syntax for node creation as seen below is a basic example but all forms of node creation follow this same template. The command `CREATE` starts the command. The text following the colon is the node “type” which really just serves as a placeholder label. Finally the parameters inside the curly brackets define this node’s properties, in this example those are title and url. In summary, the command below creates a Page node with a title and a url. This statement is equivalent to the INSERT statement in SQL.

```
$ CREATE (:Page {title:"title", url:"url"})
```

To query a node for a certain value the MATCH keyword is used. As seen in the query below the node to be searched for is defined as n. Secondly the WHERE clause defines parameters for the selection of this node, in this example the parameter is all nodes with name “Adam”. Finally the node/nodes are returned. This statement is equivalent to the SELECT statement in SQL.

```
$ MATCH n WHERE n.name=”Adam” RETURN n
```

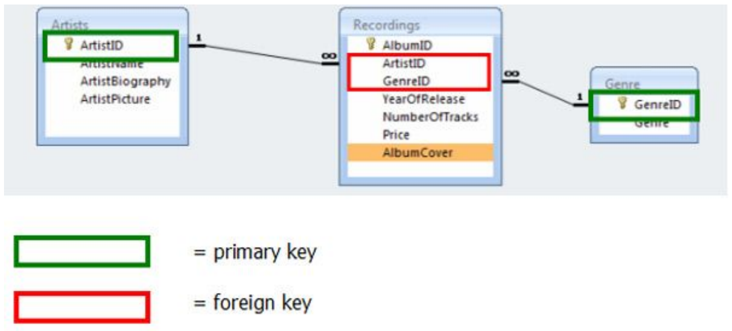
The creation of a relationship between two nodes is slightly more complex, but the syntax of the commands to make it happen are easily human readable. The command below finds an anchor node, creates a new node and then creates a link between those two nodes that link can have a type and properties of its own. The match command as seen in the previous example stores the page node with a title of “title” in a variable called anchorPage. Then creates a new Page node with a title and url and stores it in a variable called newPage. Finally the same CREATE keyword that can create nodes. This statement is equivalent to inserting a tuple with a foreign key in a relational database.

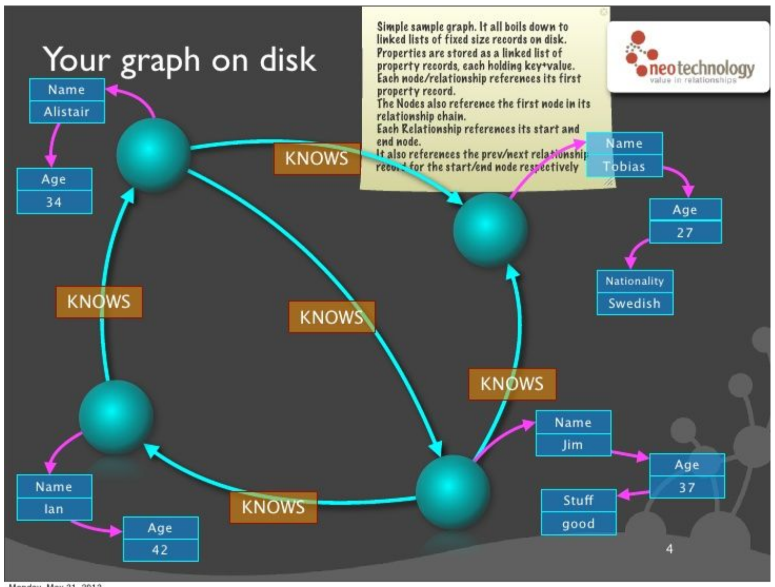


```
MATCH (anchorPage:Page {title:"title"})
CREATE (newPage:Page { title:"title",url:"url" })
CREATE (anchorPage)-[l:Link]->(newPage)
RETURN anchorPage,l,newPage
```

Evaluating

The same tasks that can be completed by a graphical database (GD) can also be accomplished with a relational database(RD). In a RD the data is stored as records in a table and is easily accessed from a point on the disk. The basis of RDs is the table. Records have several columns defined and then records are inserted and selected based on data. Each row has a unique identifier called a primary key. A row can relate back to another row by storing the other rows primary key as one of its column values, this is called a foreign key. As seen in the figure to the right a table can be set up that only holds rows with foreign keys to link 2 records from different tables together. These “relationships” are where a relational database gets its name. A graphical database could be implemented with a table of nodes and their corresponding information, and a table of relationships between them. What is ironic about relational databases is even though they are built to show relationships between objects, they are not performant when





Simple sample graph. It all boils down to linked lists of fixed size records on disk.
Properties are stored as a linked list of property records, each holding key+value.
Each node/relationship references its first property record.
The Nodes also reference the first node in its relationship chain.
Each Relationship references its start and end node.
It also references the prev/next relationship record for the start/end node respectively

Monday, May 21, 2012

the nodes and relationships between them get larger in number. For example to reference a foreign key, the computer must look at all the nodes to find the one it is looking for, then look for all of its foreign keys, then look for all nodes that have relationships with the original node. This would take a Big-O notation of O(n). For example the graphical database used in this project has 14 million nodes and +100 million relationships. So just to find a one degree path between two nodes by their properties the number of operations is: 10 million + 100 million + 10 million or 120 million operations. This number is infeasible when graphical algorithms that have higher polynomial Big-O notations start piling on producing Big-O notations of O(n^4) or more. Neo4j solves all of these problems by storing nodes as Java objects and relationships as memory mapped links to other node objects. These objects are stored in linked list data structures to achieve maximum efficiency. Neo4j can handle single object lookup in a constant time meaning basic queries are almost instantaneous. This vast improvement in algorithm efficiency makes graphical databases the only way to efficiently express and manipulate data that has a graph.

Creating

The internet is one of the greatest inventions of all time. This web stretching across the entire globe is responsible for much of our daily life, but that is just what it is, a web. Everything on the internet can be represented as nodes and vertices. Everything from the entire internet to the smallest website can be represented as a graph. This ability to represent the internet in this way allows the use of several very powerful analytical tools, one of which is a shortest path algorithm. In theory there is a shortest path between any two websites on the internet. That also applies to pages within the same website. This is the basic idea behind the Degrees To Hitler website. Once the path is found it is returned as a Graphaware path object which is parsed for the titles and those are returned in the HTTP request as an array. Javascript then dynamically renders that path in a card like fashion in order to make it easily digestible for users. This webapp can be used as an informational tool to see in what important ways things can link to Hitler, or as a game to see if users can find a web app that does not link to Hitler. The game has been the most popular use case amongst test users.

Works Cited

http://www.slideshare.net/thobe/an-overview-of-neo4j-internals
http://www.teach-ict.com/as_a2_ict_new/ocr/AS_G061/315_database_concepts/terminology/miniweb/images/foreignkey.jpg
https://neo4j.com/
https://www.reddit.com/r/degreestohitler/
http://www.w3schools.com/css/css3_animations.asp