COMP.3100 Database II Spring 2022

Term Paper

Author: _Joseph Scannell_____

(Do not change the format of the term paper.)

I. Approach

So for our database, we had to first develop the schema of the database. In it we had to create accounts, create view and modify comments, add or remove items, add or remove publishers from items, create carts, order history's, and add a system for creating orders, and a subscription model. On top of that we needed to add shipping methods, and costs for it. So for it we had multiple tables, one table named Items, which were items, the price and the name, and then books, so that if we had wanted to expand items to more like amazon we could. The books table contained the item id, the book type (hardcover, softcover, digital, audio), the ISBN if it contains one, the publisher ID. Then we had the Users table which had usernames and hashed passwords. Then on top of that we had a toggle for root users on it. Then we had a authors table, and since authors can sometimes be users, it contained the authorid, and a userid which could be null which linked to a table called BookAuthors, which related AuthorIDs to BookIDs. Then we related the reviews to the books via a Ratings table, which contained the UserID, ItemID, the comment and the rating. Then for Shipping we created a Shipping table containing shipping name, price, and speed, then related it to the Items table using ShippedVia, relating ItemID to ShippingID, which then we can relate in the Orders table, which relates a OrderID to ShippingPairID. We can relate to a Ordered table, which we related to the UserID. We can then calculate the total by doing the sum of the items in that.

For this part of the project, we equally did the work, and it was pretty obvious. We brainstormed ideas for the schema as we worked on it. One of us worked on the visual of the schema, which was because working on it together is a pain. Overall, 33%, 33%, 33% each.

For the next segment, it was to develop a webportal for the database. In it we had to write PHP code for the assignment. Now I have my gripes with how the assignment was handled, but that's down to using older strategies for making projects like this. Modern development doesn't rely on the server for rendering pages. This is bad, as its

easy to get a issue with webpage rendering on it. Its also browser dependent. Ignoring that for now, we can go over how we implemented it. Each page does a single query except for one, which is the sixth and eighth queries are grouped together. The reason was the fact that the queries were virtually similar as each page renders books we ordered, just I needed to add quick orders and add a page for ratings. So I made it so that you can submit ratings under each book and press a reorder button. I also made one more query. Overall though we had divied the tasks roughly evenly.

For the final part of the project, it was different. Rather than developing the app using Java natively, we chose to make the project just a web-browser instance, and used the WebView engine to render the page. This allowed for fast development of an app. Most apps on the app store work on this way, even desktop apps now use similar methods, like Discord, Slack, and Visual Studio Code. For this we chose to make the project use a different method than expected. The PHP code would dump the JSON out, which would use the builtin JSON Parser, to create a an object tree, in which we could render client side using scripts and style it. This allowed us to create dynamic user rendered apps in a more modular approach. It also made developing this way faster, as it was not required to do things in a complicated manner. Also the pages would use a standard JSON form for it, so that the names of each field is the same. The reason is for quick adaption of PHP code for the queries. Then for the page mapping, rather than a page for every query, it uses a user derived format for handling the queries. It uses the normal use of a webpage rather than a page for every query, allowing it to easily handle the interaction between the user and the page, seperating the pages into simple pages.

Overall, for this we contributed roughly equal. Most of the PHP is a variation of what was used before, but with changes to add JSON support, and the pages. For the Java app, I made it in Kotlin, then Keith moved it over. I worked on the updated search function, and the user interaction components. James and Keith worked on some of the PHP code. Overall, we worked on it roughly equally. 33%, 33%, and 33% each.

II. Amazon’s Dynamo

Amazon’s Dynamo is a different form for handling data compared to the SQL implementation. Rather than the normal relational database system, where relations define the way data is connected. With data handling, there are many forms of database handling systems, than a simple relational database system. For example, there are distributed databases, and page based database systems. Dynamo has to account for the nature of the scalability they need, as it was intended to be implemented on Amazon Web Services.

In the paper they go over related works, such as Peer To Peer database systems. Ways of handling data where each member of the node provides for the storage and

availability of data so it can scale. Think file sharing systems. They also refer to Distributed database systems. These are systems are similar to the Peer To Peer systems, but support hierarchical datasystems.

Dynamo provides two operations, get() and put(), where put puts a key, value pair into the context, while get just grabs a key from a context. Dynamo considers the key and value pair to be just a set of bytes. The context is a scope where all the data is around, similar to a table. It uses MD-5 to hash the key. (This may cause issues, as MD-5 is known to have the ability to have collisions on different data, but same results.) The MD-5 key then indicates where the data is stored, similar to a pointer. Dynamo then uses a ring to scale it across multiple nodes. The ring, has the values assigned to multiple nodes, with each node holding the items key and its ring location. It rotates this ring to find the position of the value, allowing for it to figure out which machine holds which keys. This means if the keys are between X and Y, then its on B machine, but if its above Y, then its on C, but if its below X, then its on A machine. This means if they want to scale the amount of machines to increase the amount of capacity, its just means they have to add a node, and then it can reconstruct to add devices. This is similar to how RAID works on hardware. However this can mean that sometimes each device is on average hit more often then others, so instead, the devices are assigned via many nodes along the ring, meaning each device has more than 1 region, so its storage region is non contiguous. To achieve data redundancy, it creates clones on multiple hosts. However it does this through N devices. This means that the N devices before it on the ring care cloned on the other devices, such that if a failure occurs, it can reconstruct lost data, and it can serve during the other machines downtime. This is done through the preference list. Basically the main device is highest on the preference list, but devices closer to it are more likely to be on it.

So data redundancy and scalability are upsides. However what are some downsides. One of the downsides is the fact that all this is occurring asynchronously, so it means that sometimes calls right after a call on the same data will not show up, or fail. This is the concept of synchronization. Data synchronization is important in many areas of computing, such as multiprocessing and multithreading. The solution is not quite perfect for dynamo. It handles this by giving you a copy of the deltas of the data. This means that if the data is not the same as what was handed you, it shows the deltas on top, computing it such that when its complete it can add the write job to the queue. This means the database your working on may not have that key value pair in it at that moment. This can be a real issue, as if you have multiple places all requesting highly in time updated data, this means that the database may not provide fully up to date data at that moment, as it takes time for it to synchronize.