MY THOUGHTS ON POSTGRES vs. MONGODB

I experienced a handful of pros and cons in creating a database for Bob’s Pizzeria in Postgres and MongoDB. These pros and cons are discussed below.

The syntax required to create tables in Postgres and insert collections (and accompanying data) into MongoDB is not terribly complicated – I would consider that a pro for both databases. I knew that each table in Postgres and collection in MongoDB would need to have a unique identifier. MongoDB is capable of autogenerating unique identifiers, but they are long, jumbled mix of alphanumeric characters. This would’ve made linking the collections later on in the build process quite complicated, so I took the extra time to generate these identifiers myself. In Postgres, I utilized the ‘serial PRIMARY KEY’ to create my unique identifiers; this simple numerical incrementing process saved time and is easy to comprehend; this is a definite pro for Postgres.

When it came to actually inserting data into the collections and tables that I had created in both databases, the process was more streamlined in MongoDB. The collections in which that data needed to be stored were created and populated simultaneously. This process in Postgres takes two steps, as tables must be created before any data can be inserted into them. This is a pro for MongoDB.

It became apparent early on in the process of building a database for Bob’s Pizzeria in both MongoDB and Postgres that mapping the ‘many-to-many’ relationship between recipes and ingredients would be vital to success. Grasping this concept and creating a table to manage the relationship was easier in Postgres, as like many of the other tables, it needed a few references and standard data types. Having a standalone table whose sole purpose was to track what ingredients were required for each recipe along with their quantity provided transparency into how the other tables were to be brought together and rounded out my understanding of how the schema was supposed to work. In Mongo, this many to many relationship between ingredients and recipes was more difficult to create/envision, given its schema-less nature. The ingredients required in each recipe were instead nested in the recipe collection, which was quite difficult to grasp at first. That said, once this concept of nesting became more clear, I favored it to that of the intermediate join table in Postgres; nesting allowed for one less collection/table to be created and managed. Additionally, when it came to creating the take\_order function and the Orders collection in MongoDB, I knew that the Inventory collection and the Recipes collection had all the data that I needed to track orders and ensure inventory was present. There was no consulting an extra collection, and writing the function was more streamlined, as it was just a matter of iterating over the nested portion of the recipes table to extract the information about a particular recipe and then performing computations against the Inventory collection. On the flipside, performing the same tasks in PL/PGSQL was not as straightforward; it instead involved aliasing and joining several tables, which took lots of trial and error and did not ‘flow’ as easily as JavaScript did for me (likely because I only have experience using Python, and no real experience with SQL syntax). Overall, once I was able to wrap my head around nested fields in MongoDB documents, I would give the edge to MongoDB over Postgres when it comes to the all-important mapping of the relationship between recipes and inventory.

Obviously, the process of creating both databases did not come without some major speedbumps. It was in the process of overcoming these speedbumps that I realized that MongoDB is much easier to update/change on the fly. I deleted and reloaded my collections several times, which while not ideal in general, was easily accomplished in Mongo. Deleting tables in Postgres once they’d been created was a pain, and typically left me quite nervous/unclear of what would happen to the reference tables of the table that I was trying to delete. Even changing a column name, so that it was more sensical when used in the take\_order Postgres function was a pain, and it also had to be edited in the corresponding CREATE TABLE instructions. This ability to edit/delete/update in MongoDB was a pro, and its difficulty in Postgres is a definite con. One area where I gained efficiency in Postgres was in the generation of the orders data. Rather than writing the code in Python, I was, with minimal Googling, able to create data for the Orders table by using PL/PGSQL syntax, which I thought was cool. Additionally, when it came to testing the performance of queries once indexes had been added to Orders in both databases, I greatly preferred the results output of Postgres to that of MongoDB. MongoDB returned a large JSON object with a significant amount of data that was difficult to decipher/find what I needed. Postgres returned five rows of information, all of which were easier to read and interpretable.

Overall, I can see the benefits of using both databases. Personally, the only ‘database’ experience that I have is in the form of tables found in Excel, and the occasional Access database, both of which are relational. Because of this, I was able to initially understand/envision what the process would be like to bring Bob’s Pizzeria’s database to life. When it came to implementing the vital mapping between Inventory and Recipe, the join table was easy enough to create, but utilizing/traversing it in the take\_order function was where my vague familiarity with and fondness of relational databases stopped. As I gained experience with MongoDB throughout this project, I became more comfortable with its schema-less nature, and can see why, if you don’t know what your data is going to look like and need to maintain flexibility, it is a great choice. Nesting in collections, combined with MongoDB’s use of JavaScript to create server-side functions made it seem like a more dynamic, modern product. Once one is comfortable with it, I think MongoDB’s schema-less nature and features gives it broader applicability than Postgres – I look forward to testing that theory in future classes and professional environments.