CS411 Final Project Report - Team DROP TABLE

In terms of changes in the direction of our project, not much changed regarding what we had planned for the application, and many of the things we were planning to implement we were able to get done. Our application achieved all that was intended regarding its usefulness save for some minor elements. We were able to compile a realistic dataset regarding the specs of different processors, especially with GPUs with which we had an extensive dataset to work with. Our dataset includes a comprehensive list of products, including ones that are rarely mentioned elsewhere. We also provide the user with quick and streamlined access to trending pages where he/she can see which products are in the most carts. Some elements we have yet to include that would help the application's usefulness are including datasets with actual dollar values of the products. This was difficult to do at the outset of our project since we were unable to find a reliable, accurate, and recent dataset online, and was only made more difficult by the ever-changing prices of CPUs and GPUs. Another feature that we were unable to include is displaying the dollar costs of the carts on the cart pages themselves. This was not officially stated to be a feature but was implied from the start. While users can still see the dollar costs on the cart list dashboard, we would still like them to be able to see the cost alongside their chosen CPU and GPU. This feature is a future consideration for this project. The majority of the rest of the work would likely involve polishing the application and cleaning up the codebase.

Minor changes to the schema were made. These include 1) adding a second primary key (Brand) to both the CPU and GPU tables and 2) implementing a Permission table to enforce the many-many relationship between the Cart and Person tables. As we continued work on the project, we also included various metadata attributes such as date a cart was created, the last time a cart was modified, and the date a user was registered. We also added persistent notes so that users can add details about what they like about products in their cart.

From the sourced database, those tables which did not have 1000 rows were filled with randomly generated entries. The data generation was largely restricted to the users relation and the permissions and carts. We used the pandas and Faker Python libraries to facilitate the generation of fictional users. For the tables with imported data, not much changed, although we did have to generate information regarding the price for each product.

Changes to the ER diagram directly reflect the changes in the schema which are already described above. Adding new parts to the diagram was very straightforward, and much of the planning and adaptations for the diagram were put together using Lucidapp. When we moved on to the database implementation, DBeaver facilitated much of the database actions such as data importing and schema editing. Implementing tables was also very straightforward and we were practically able to enter our database schema directly into Cloud Shell without any issues. Besides the aforementioned metadata and notes, we didn't stray too far from our original implementation. Overall, between our first iteration and our current design, our current design is much more suitable and allows for far easier record keeping and data collection.

We added minor functionalities which include 1) an "About" page, 2) the current date and time and 3) trending analytics. The "About" page adds a little more substance to our application and gives users an overview of the usefulness and functionality that the application will provide them. The timestamp functionality allows us to keep records pertaining to usage and helps us identify the most recently created users and carts. The trending analytics page helps the appeal

of our application since users would now be able to view the statistics on which items are most popular. We added these features to better reflect the bare-minimum expectation for modern website functionality, and more importantly uphold user convenience. One functionality we ended up scrapping is including images for each part. Doing so ended up not being feasible as there was no reliable way to collect all the sources for the images or ensure that the sources would stay functional. However, it would definitely be a consideration for future implementation as it would add a lot to the polish and feel of the application. We also planned on having an adaptive budget progress bar to indicate how much of a user's budget was allocated, but ended up scrapping it as well primarily due to time constraint and other features taking priority. However, we do feel that it would be relatively straightforward to integrate into our application.

The advanced queries include 1) AboveAverageCost, allowing the user to identify respected manufacturers which may overcharge, 2) MostExpensiveCart, displaying the highest-priced handful of carts as a community analytic which adds charm to the interface, 3) TopCostAfter2023, which provides a rundown of the higher-end and more modern products and 4) US_BasedCPU, which provides a rundown of the higher-end products manufactured by manufacturers based in the United States.

Advanced database programs definitely complemented our application and made management of the data much easier. We ultimately used MySQL as it was the same database used in our assignments, and doing so made the transition from coursework to project work very easy. Using DBeaver as a database tool was incredibly useful when it came to storing and importing information such as user data and cart specs. SQL made it straightforward to query the data and perform operations on the different relations. The stored procedures and triggers that we created helped streamline the functionality of the application and made operations such as adding permission to new carts far easier.

Following is a description of (a) technical challenge(s) faced by each team member.

Dennis Tomak - A technical challenge faced was familiarization with the frontend layout, and learning how to cross-integrate between Python and HTML files. An additional challenge was debugging and optimizing the transaction.

Hasan Colak - A challenge that I faced was being familiar with the frontend layout. We initially started with a different layout design, as we progressed in the project, that's when we started making changes to the interface. For instance, when it came to setting up the Navbar, implementing the Navbar was tough. When it came to connecting the backend to the frontend, I was confused and had a challenge with what to include in the interface that can complement the backend.

Jet Geronimo - Hosting the entire application and database with Google Cloud Platform was an admittedly tedious process. There was quite a bit of documentation that had to be read in order to facilitate local testing and building for Google Cloud Run. I would almost say that setting up Google Cloud Run was much more difficult than building the actual application itself. However, once it was done, progress on the project moved much quicker. GitHub integration with Google Cloud Build made iterating on the project incredibly easy, and since our application and database were

stored on the same platform, connecting the two together was also straightforward. There are some very important things to know if one is looking to host on Google Cloud Platform. Connecting your Google Cloud Run application to GitHub makes development much quicker, and using well-supported libraries such as cloud-sql-python-connector ensures that the database connection is properly configured. When working with GCP, it's important to understand service accounts and knowing how to set up networking — one important tip that saved us a lot of time is to make sure to whitelist the IP address when working on on the Google MySQL database, otherwise the database will reject the connection for no apparent reason. Finally, ignore everything else GCP tries to push, the only things needed to set up the project are **Cloud Run**, **SQL**, and **IAM**.

Brice Boutot - Learning all the various components was difficult at first because I've never used github for a project before or the frameworks that we used. Once I was familiar with flask, jinja, and GCP, it became easier to work on. I think the most difficult aspect was trying to get the front end to work with the backend and to display the data stored in the database. I was able to get data displayed very simply but after switching to a different form of connecting to the database, using GCP which was better, I wasn't able to figure out how to actually display the data we received from the DB. Also trying to fully understand and implement the transaction. I wasn't able to actually figure out where it could be implemented and almost opted to just keep it stored within a procedure. Ultimately I wasn't sure what was correct and left it In the documents but outside of the actual project.

Future considerations for this project are to calculate the total cart cost and reoptimize some of the queries, preferably the transaction which is currently nonfunctioning. We would also like to restructure much of the codebase, since currently many of our functions are stuffed into single files. It would be good to disperse them out into manageable chunks for better modularity and easier development. Another thing that did not make it to the final application that we would have liked to include is a dedicated page to each processor. However, adding one for each processor would be difficult and possibly not worth it since, as mentioned, we don't have images stored for each processor. We would also have to create descriptions for each processor, which would be incredibly tedious, although perhaps we could consider the use of an LLM for generating such content.
A summary of the division of labor is as follows:

Dennis Tomak: implemented backend database queries and complementary code as well as contributed to the frontend.

Brice Boutot: implemented queries into the backend and the frontend, developed the triggers, trending pages and procedure and implemented them into the back/frontend.

Jet Geronimo set up the application on Google Cloud Run, implemented the templates for the integrated interface, established connections between the Python, SQL and HTML files and the database, styled the web pages with Bootstrap, implemented queries and enforced an MD5 hash protocol for the login and registration.

Hasan Colak implemented backend code and fleshed out the interface, adding a navigation bar and numerous quality-of-life features.

Division of labor was managed efficiently and temporally, with frequent and persistent communication from all ends.