# CS 411

Summer 2024

Project Track 1 Stage 4

# Project Report (3%)

Ethan, Zachary, Ivan, Suyash

July 30, 2024

# Changes From Stage 1

Our changes from Stage 1 were mainly focused around reshaping the design of our database and website to simplify our design so that any further additions, like queries and stored procedures, would be slightly simpler in their implementation as well.

The initial plan for our project was to combine two different datasets, representing League of Legends and Rocket League, as a sort of example of a site that could be a central place to access game information and statistics for multiple games. Users would be able to pick a game and see general information and generic tables of information, such as top kills or most victories for a League player over a timeframe or top goals / assists for a Rocket League player.

However, we ran into issues with trying to implement this initial idea in actual design, so we decided to narrow the idea down to just working on collecting and displaying data for Rocket League. This shrunk the massive amount of data we were working with and also allowed us to focus on functionality while worrying less about differentiating game_ids and player_ids that appeared across both games, as an example. We also only had to worry about transferring the dataset to our own tables in one way, as the Rocket League and League of Legends databases both kept data relating to games and players in different complex ways. In essence, we narrowed our project down to a simpler project to envision and implement that could still hit every point needed for the project.

# Successes and Failures

One of the points we had on the original Stage 1 proposal was making a site that could hold tables and data for multiple games, since a player who was interested in looking at data across games would have to visit multiple sites focused on each game specifically, or look at a site that gives not too much more than surface level information like competitive results across all esports. Obviously, this point did not pan out because of the changes we made from Stage 1 and the changes and implementations that will be described later in this report, but it was an ambitious point to go after; it is not too worrying that the application did not end up becoming this in the end.

However, the application was successful in being able to display some basic information and load some tables that show top performers in basic stats like goals and appearances on the page. Search functions to search players by id or tag and display things like teams by region were also included. Being able to search by player or display leaders in a statistic were huge priorities in what we were trying to make, so we are satisfied with where the application went there.

An aspect of the application that hit kind of a middle ground was how the leaders in a statistic would be displayed. There were thoughts early on on trying to find a unique way to visualize this like on a chart or percentile visualization to try and go for the extra credit, but we moved off of that once the class actually got going and we put all focus into just getting the application working. Eventually we just moved to showing the top performers, like top scorers, into a simple table, which does not hit the extra credit points but does fulfill some of our necessary requirements for the project itself. We needed to do this and did, just not to the full extent of the extra credit heights we had in mind at the very beginning.

# Changed Schema/Source of Data

In our project, we initially utilized data from two different Kaggle datasets: one for Rocket League and one for League of Legends. The schema was designed to accommodate data from both games, with tables for players, teams, games, matches, and events.

We decided to drop the League of Legends dataset due to the complexity of integrating data from both games and the sufficiency of data from Rocket League to meet our project goals. This change allowed us to simplify our schema and focus solely on Rocket League data, which we had a significant amount of. Consequently, we removed tables specific to League of Legends and modified others to better handle Rocket League data. This includes removing/changing some foreign keys and relationships in the Rocket League tables that no longer needed to be connected to something like a game entity or anything else to distinguish that data as data for Rocket League and not from another game.

The primary motivation for these changes was to streamline the schema and improve data handling efficiency. By focusing exclusively on Rocket League, we removed unnecessary complexity and ensured that our database was better suited to our needs. This involved rewriting some of the table schema and relationships so that they would work better with the queries we were going to end up writing, which had to be done once during the entire project process.

The simplified schema has improved our ability to manage and query the database efficiently. With a focus on Rocket League, we have been able to optimize our data structure for better performance and more relevant insights.

# Changes to ER Diagram & Tables

Initially, our project included tables and entities that were designed to handle data from both Rocket League and League of Legends. The original UML diagram constrained many redundant entities and incorrect cardinality notations. The schema was overly complex and because of this it was difficult to manage and query.

Feedback from Stage 2:

- **Cardinality issues**: The UML diagram used incorrect cardinality notations (e.g., 10-10, 2-2) instead of standard expressions like 1-1, 1-many, or many-many.
- **Redundant Entities**: Entities such as game, team, and player were repeated for both datasets.

Updated ER Diagram and Tables:

- **Corrected Cardinality Notations**: We updated the cardinality expressions to standard forms like 1-1, 1-many, and many-many.
- Unified Entities: Removed redundant entities and combined related data into single instances for better query performance

Another thing we noticed from the original UML diagram was that some of the Entity Table names needed to be renamed. We changed ‘Match’ to ‘Matches’ because ‘Match’ is an actual function in SQL. The same issue appeared when we tried making an ‘Event’ table, so we renamed it to ‘Events’. We also realized that some name changes were necessary to better represent the relationships between tables.

One significant change we made was limiting the sizes of our tables. The data that we pulled from Kaggle was extremely large, with some tables having over 100,000 rows. This led to long periods of waiting whenever we tried using SELECT to view the entity tables that we would create from these large tables. For instance, even a simple query like "SELECT * FROM [table_name] LIMIT 15" would take over 10 minutes to execute. To solve this issue, we decided to create our new entity tables with a fixed size that we set to around 1000 rows. This helped us improve the responsiveness drastically so we were able to continue working on the project without having to wait extremely long periods to run simple queries.

The primary motivation for these changes was to address the feedback we received and improve the clarity, efficiency, and manageability of our database schema. We strongly believe our new design is more suitable because of these reasons.

# Added/Removed Functionalities

Some of the removed functionalities that were originally proposed were referenced in earlier sections. There is, of course, the ability to search data by what video game it is from that was removed for reasons stated previously. Searching along any statistical attribute in game or player tables was also not fully implemented in exchange for certain key stats, just because there would have needed to be a lot more procedures written for each individual statistic, of which there are dozens. Comparing data between players was also something we couldn't exactly figure out how to effectively display in our UI, so that was also something we did not get around to.

Some functionalities were added that were not described in the original proposal because of requirements in stage 4 that needed functionality to be implemented for. This includes a trigger that prevents duplicate players from being added to the database and a stored procedure that should be able to filter down players into separate categories for viewing. Constraints on primary keys and foreign keys weren't explicitly defined in some of the early proposals, but they were added in while creating the schema for the database we used. (That one was kind of a given with how tables are made but technically it is new from the proposal). Any of the other functionalities were already described in the original proposal or any of the checkpoints.

# Complements to Application

The advanced database programs add an essential part of how the application works. The tables to see top scorer and defender and sections like that are all generated and managed by advanced queries and procedures. Without the advanced database programs, players would not be able to consider their statistics with those in the professional scene. We wanted to give players the ability to compare themselves to a more professional standard because it is important for those who may consider joining the professional scene to see just how they measure up against the professionals. In regards to this, we consider the advanced database programs to be essential to the application and how we want users to use it.

# Technical Challenges

## Suyash:

A significant challenge was connecting the database to the front end. This required a lot of middle steps in order to get the UI to display the tables correctly. Since the database is hosted on GCP rather than being hosted locally, it is a little difficult to directly access the database and use it in the front end part of the application. The solution we decided to use was express with cors and body-parser modules and host the retrieved data from the GCP SQL server locally and update the front end using that local data. Sending too many active requests to GCP servers could be resource heavy and is not time efficient.

## Zach:

One significant challenge I encountered during the project was handling and processing a large dataset from Kaggle. Initially, we imported 6 tables of data from Kaggle into the project. These tables were extremely large, with some having over 100,000 rows. We then used those tables to make our new entity tables that contained values that we intended to use for our website.

Given the substantial size of the original tables from the dataset, any query operation, particularly the creation of these new entity tables and using SELECT to view them, would take an excessively long time. For instance, even a simple query like "SELECT * FROM [table_name] LIMIT 15" would take over 10 minutes to execute. This long execution time is due to the fact that we still need to scan through every row in the table to apply any conditions or limits specified in the query. Therefore, limiting the result set to 15 rows does not prevent the database from processing the entire dataset, leading to significant delays. This was impractical and negatively impacted our ability to quickly iterate and test our database design.

To overcome this, we used a strategy that we felt was simple and practical. Instead of attempting to process the entire dataset, we decided to create our new entity tables with a fixed size that was manageable yet fit the requirements for table sizes. We set the size to 1000 rows, and this helped us improve the responsiveness of our database. We were finally able to proceed with the project without the excessive wait times we previously encountered.

## Ethan:

The team encountered technical challenges when it came to getting SQL lines to run in the database server. Unexpected was the necessity of including DELIMITER statements when creating stored procedures. In addition, there were never before seen errors discovered when trying to run programs that were modeled after examples from coursework. Additionally, we

discovered some discrepancies regarding data types with tables that we created for our data compared to auto-generated tables when importing the data through GCP.

## Ivan:

One area of challenges was working and reworking the conceptual database design, which is then translated into the tables used for working the actual application. Thinking about how primary and foreign keys are set up is one of the most important parts of the initial design, and any setbacks in that area hindered the advancement of the project entirely. Taking care of how table relationships are set up and how foreign keys are connected between all the tables sets up all work that happens on these tables after, so this was an important point in the process that definitely took a lot of thought.

# Other Changes:

Almost all changes have been covered in other sections in this report. The only thing to put here is that the UI ended up different than what was originally envisioned.

# Future Improvements

Beyond what was first mentioned in the “Removed Functionalities” section with the reasonable enhancements to the application, an impressive comparison feature with multiple statistics would be a very compelling feature for the application to have. However, this feature would be theoretically complex in how it might be composed and displayed. So, adding more statistics and perhaps optimizing table combinations to allow for more searches could be an interesting improvement route. Again, back to the original proposal, being able to see the leaders in advanced statistics in multiple games and making comparisons between them is a super cool idea in theory but one that wasn’t realized in this attempt at creating this application.

# Division of Labor

Throughout the project, our team maintained regular communication to make sure that all parts of the project were completed and integrated smoothly. We divided tasks based on our individual skills and interests with the work, and because of this we were able to work very efficiently to complete all aspects to the best of our ability. The distribution was as follows:

## Suyash:

Handled the organization of the UI/UX design and the programming for the front end of the application. As well as setting up the connection in the backend from the GCP SQL server database to the application.

## Zach:

Mainly handled tasks such as database management, data storage, and any operations involving SQL queries. Was responsible for setting up all of the GCP-related tasks. Set up all of the documents submitted, including extras to help keep track of info such as tables and indexing. Helped with creating the UML diagram.

## Ethan:

Assisted other team members in completing their tasks throughout the entirety of the project. I chipped in with the schema generation, the UML diagramming, writing the SQL code for the database tables in Stage 3, writing half of the advanced queries, and later writing more SQL code for the frontend-backend of Stage 4.

## Ivan:

Created the UI mockup for Stage 1 and added parts to the proposal for the project. Took the lead position in diagram generation for Stage 2. Later, chipped in to fill in missing parts for Stages 3 and 4.