Zaki Refai – Answers to Questions

1. At the start, I examined the RealGM International Transactions website. Upon realizing that the data is not available for download, I knew that I needed to scrape the data off of the website. Before I started writing any code to scrape the data, I needed to understand its structure on the website. For every season, there's a league, teams associated with a league (or multiple leagues), and transactions that occurred with players between different teams. After determining this structure, I created an ER diagram and relationship schema (you can find this ERD/schema in /Supporting_Documents/ERD_Relationship.pdf). An ERD diagram, along with the relationship schema, helps with visualizing the database, and it is something I create before I start any project of this nature. The relationship schema helps when I have to create the tables in SQL. Overall, the ERD and relation schema help me understand how the data should be formatted when the functions scrape data off of RealGM. I also decided to code the entire project in Python since it's my favorite language.

   The next task was to create the functions to scrape the data off of the pages of RealGM. To do this, I used a library called BeatifulSoup. It's a library that allows one to search specific classes of tags in an HTML page and generate a list of all tags found along with the contents they contain. RealGM's database follows a specific structure where leagues within a season are in a class called "page-nav-option clearfix." Transactions are in another class called "portal widget fullpage." I created a function that loops through all seasons, gathering all league names and ids. This function also builds a link off of this information and stores everything in a CSV file (this function can is in the main project folder titled League_Scraper.py). I do the same thing when I go over each transaction, except I can gather data on the players, teams, transactions, and the information to link all of these things together (the functions that do this are in a file called Major_Scaper.py). While League_Scaper.py outputs files for each league during each season, Major_Scraper.py outputs five other file data types for each season during each pass. For comparison, League_Scraper.py outputs a total of eight data files, while Major_Scraper.py outputs forty data files. These files are used when creating the database. The TransactionsDB.py file creates tables based on the ERD. It also fills them with the corresponding data generated from the files mentioned above. I automated this through Main_Program.py, a file that runs each step by gathering the data, creating the database, and finally adding the data to the database.

2. As stated above, the ERD shows the structure of the database. I found that leagues, teams, transactions, and players needed their own entity sets; these are the significant kinds of data sets. Entities like league, team, and players have attributes like ID, name, and link (a link to the RealGM website for more information about a specific team). Leagues and teams needed a relationship between them because a league and team share data (for an overall description of how I describe relationships between entities check out Supporting_Documents/Database_Description.docx). I tied them together with the season relationship, because then I could track which teams were a part of

what leagues during different seasons. I found that I needed to do it like this because new teams were added to the RealGM database under different seasons, and some were deleted over other seasons. I then created a Transaction entity. By looking at descriptors like “has signed with,” “previously with,” or “has left” within each transaction, I realized different types of transactions exist. That’s why the Transaction entity is the parent of Signing, Transfer, and Exit. These children entities contain attributes corresponding to teams a player has signed with, transferred to, or left. The Transaction entity is connected to Teams through the relationship Record because we need to be able to search for transactions within a specific team. Finally, the Player entity set connects to the Transaction entity through a Has relationship. Since Player has a one-to-many relationship with Transaction, I added a foreign-key reference to Player’s primary key in Transactions. Doing this simplifies the database, as we do not need to create a relationship table between Transaction and Player.

This is the overall structure of the database. I created everything with simplicity and efficiency in mind. If I could select the storage file (i.e., heap, hash, B+, or sequential) for each table created, I could also make the database more efficient in searching.

3. The kinds of questions that could be answered pertain to the transactions of each team. We could query for the total number of transactions a team has made over a season. We can do the same thing for a league but instead count the number of transactions in different seasons. If we wanted to be more specific, we could even count the number of transactions a player has gone through across seasons. We can also query each transaction a player has gone through and count the teams a player has joined.

   Here is an example of a question that is more about a team and players. I was able to leverage the database to generate an HTML document to see which players, on what dates, were involved in a transaction with the team “Real Madrid” across all seasons (you can view this file under HTML_File_Outputs/All_Real_Madrid_Transactions.html). In short, these are all the players that have gone through the team “Real Madrid” across all seasons.