Our cohort's approach to data generation for the database project was a multifaceted process that utilized both web-based data extraction and advanced scripting techniques to ensure a comprehensive and efficient dataset. Here's a more detailed breakdown of each step in our method::

1. Data Extraction from Web: We began by navigating to the University of Texas at Dallas club website(https://cometmail.sharepoint.com/sites/StudentOrganizationCenterSP/Lists/Student%20Organization%20Directory/All%20Items%20gallery.aspx?env=WebViewList), where we extracted initial data sets concerning various student clubs. This data was directly exported into an Excel spreadsheet, serving as our primary data source. This step was important as it provided a structured form of raw data that included essential details about each club. Based on this data, we based the DB on it since the data came with club name, president name, description, mission, and president's email. This CSV contained 416 clubs and 416 club leaders. Based on that, we populated the rest. Since there were so many entries for this, we decided to make less of the others, such as 50 of each user subclass, and so on.
2. Data Enrichment with ChatGPT: To augment the data with additional details on entities and relationships that were not covered by the initial extraction, we utilized ChatGPT. This AI tool helped us generate synthetic data that mimicked real-world complexities in relationships and entity attributes, thereby enhancing the depth and usability of our dataset. For instance, we could create realistic member profiles, event records, and club relationships without manual input. We asked ChatGPT to make us CSV files that would conform to the attributes we had per each table, and we made sure to make multiple data entries for each table. We also made sure to logically follow how the DB would be structured.
3. Scripting for Data Import: With the dataset prepared and enriched, the next challenge was to efficiently import it into our database. To do this, we employed SQL scripting, specifically starting with the command SET GLOBAL local_infile = 1; to enable the import of data from local files. Following this, we used the LOAD DATA INFILE command to transfer data from a text file—structured and formatted in CSV format—into our MySQL database. After that, we used the command CHARACTER SET utf8 to ensure that our text was readable for any machine. Lastly, we used the appropriate fields for FIELDS TERMINATED, OPTIONALLY ENCLOSED, and LINES TERMINATED to make sure that MariaDB accepted our input. This command allowed us to specify character encoding, field terminators, optional text qualifiers, and line terminators, ensuring that the data was parsed and stored correctly without errors.
4. SQL Command Execution: The specific SQL command we used not only facilitated the seamless integration of data into our database tables but also allowed us to handle large volumes of data efficiently. By managing the data import through scripted commands, we

could automate much of the process, reducing the potential for manual errors and significantly speeding up the time it took to populate the database.

This methodology leveraged the strengths of both automated data generation and precise script-driven database management, ensuring a thorough preparation and implementation phase for our database project. The combination of these techniques allowed for a scalable and flexible approach to database creation, capable of adapting to various data types and volumes.