CPSC 5071 Project Specification (Group 5)

Overview

Mission Statement

The purpose of this project is to design, implement, and populate a database for storing information relevant to academic administration and performance at a university. The database will:

- support the day-to-day needs of students, professors, and administrators
- enable analysis of student academic outcomes by tracking relevant data such as section/room capacity, location, or number of TA's.

For us, a successful database is one that is both functional and easy to use and maintain.

Our ultimate goal is to develop a model to help us predict and understand student performance. By tracking detailed information about courses, enrollment, and outcomes, we aim to identify patterns and important predictors that will facilitate data-driven decisions to best improve student learning and achievement.

Description

Universities both *require* and *generate* a lot of data, from course offerings and student enrollments to room capacities and technological resources. As universities grow, managing this data with manual processes or ad-hoc systems becomes unsustainable; information inconsistencies, data redundancies, and coordination challenges emerge when different people maintain their own records without a unifying system. So, as universities grow, so too does the need for a systematic approach to data management. In this project, we develop a database for perhaps the most important part of the university data ecosystem: the academics.

One of the key challenges of developing a database of any kind is in clarifying who your users are and figuring out what they actually need. This involves iterative improvement via collaborative communication, modeling, and feedback. The users of *our* database are students, professors, and administrators, and each type of user has their own view, use case(s), and needs of the database. For example, administrators are probably a lot more concerned with room capacity and scheduling conflicts than professors.

The diversity of views and use cases naturally leads to increased database complexity–there are a lot of entities with a lot of relationships and a lot of constraints. The remedy to complexity is a systematic approach. For us, this means systematic designs and documentation (across varying levels of abstraction). For example, we construct diagrams for our users' view of the data (external models) and then merge these into a single, unified view (the conceptual model). But even organized database diagrams cannot save you from requirements that cannot (feasibly) be implemented in a DBMS, such as exceedingly complex constraints. Cases like this motivate the need for *project-wide* documentation so that such constraints aren't forgotten about and implemented at the application level.

One technical challenge in this project is in properly accounting for intermediate states/different modes of operation. For example, you'd usually think that each section of a course needs to have an assigned professor, but this isn't necessarily true *during registration* when things are still being decided. Handling these "different modes" requires (1) learning about them from the users and then (2) designing the database in a way that can accommodate all of them. In the previous example, we'd want to make the professor-section relationship *optional* so that sections can be created without an associated professor.

Universities both generate and store a lot of data, and they don't do it for fun but because it is useful and necessary. Whether it be in maintaining student private records or budget allocations, data is *important*. Earlier we talked about developing a database to support the use cases and needs of students, professors, and administrators, but *we* also have a use case for the database–data analysis–and it brings its own challenges, namely *performance*. We want to be able to analyze the database for patterns in and predictors of academic outcomes. Such analysis can be computationally expensive depending on the size and design of the database and queries. We address this in our choice of indexes and whether to denormalize certain tables to optimize the performance of common queries.

The Data

Data Modelling

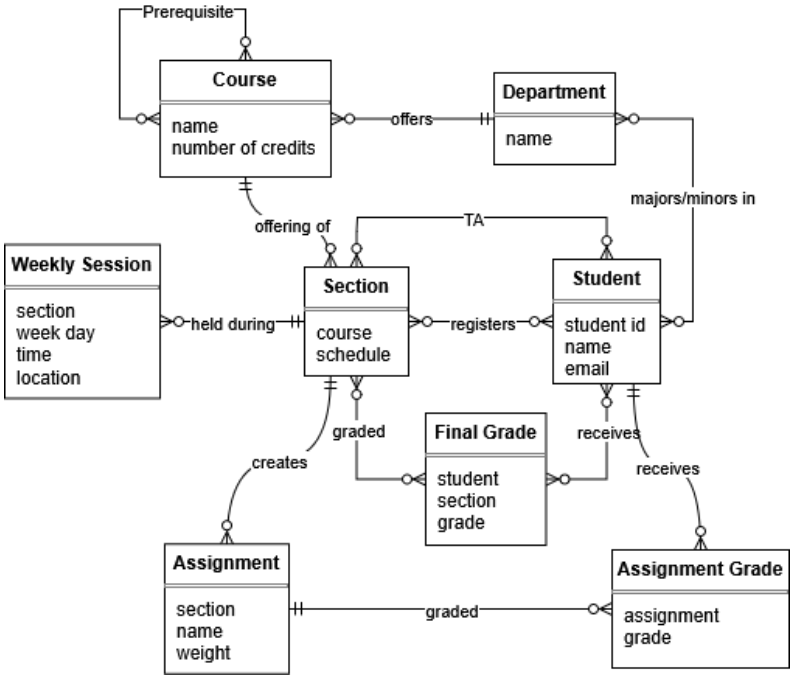
To model the data, we start by developing a model for each type of user. These models, called *external models*, represent each type of user's view of the data based on their use cases. We then unify the external models into a single model for the entire database, called the *logical model*.

External Models

The Student View

The student view includes:

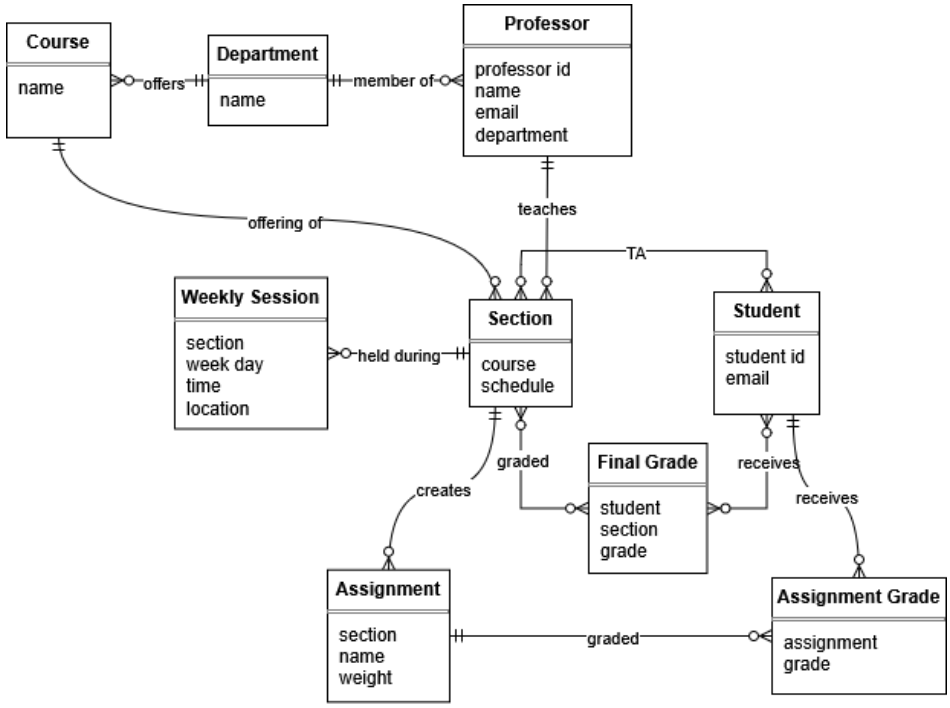
- Degree Progress: Students check their major/minor status as well as their transcript.
- Registration: Students search for course sections with weekly sessions that fit their schedules. They will need to have passed the prerequisite courses.
- Section Schedule: Students view their schedule for the quarter based on the sections they've enrolled in.
- Teaching Positions: Students with TA positions can view which sections they are assigned to teach.
- Assignments and Grades: Students access assignment details for enrolled sections, review their grades for completed work, and check their final grades at the end of each quarter.



The Professor View

The professor view includes:

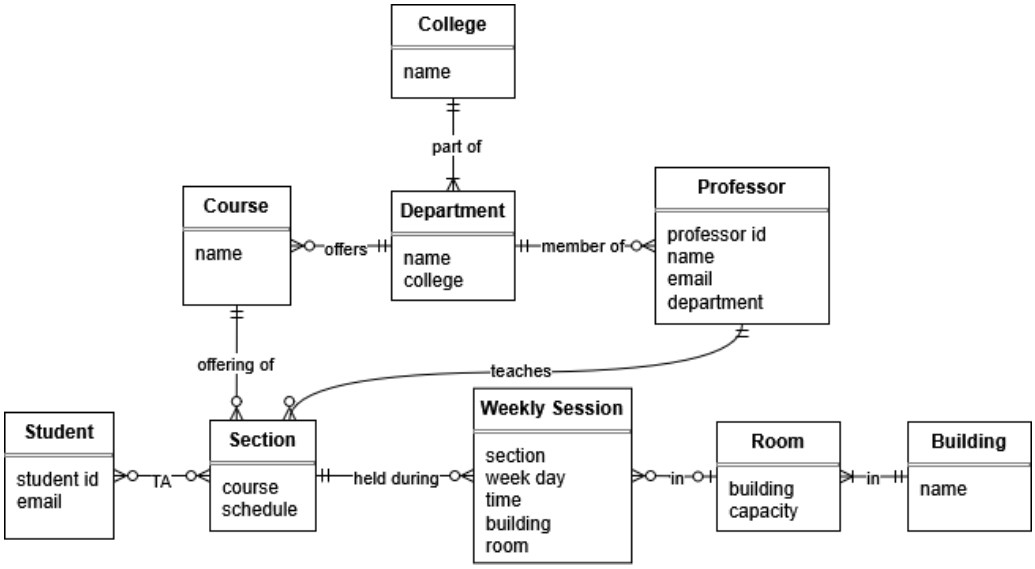
- Department Position: Professors can check their departmental position/membership.
- Teaching Schedule: Professors can see their teaching schedule, including which sections they're teaching and when and where those sections are held.
- Teaching Assistants: Professors can see who the teaching assistants of their sections are and contact them.
- Assignments and Grades: Professors create assignments and enter assignment/final grades.



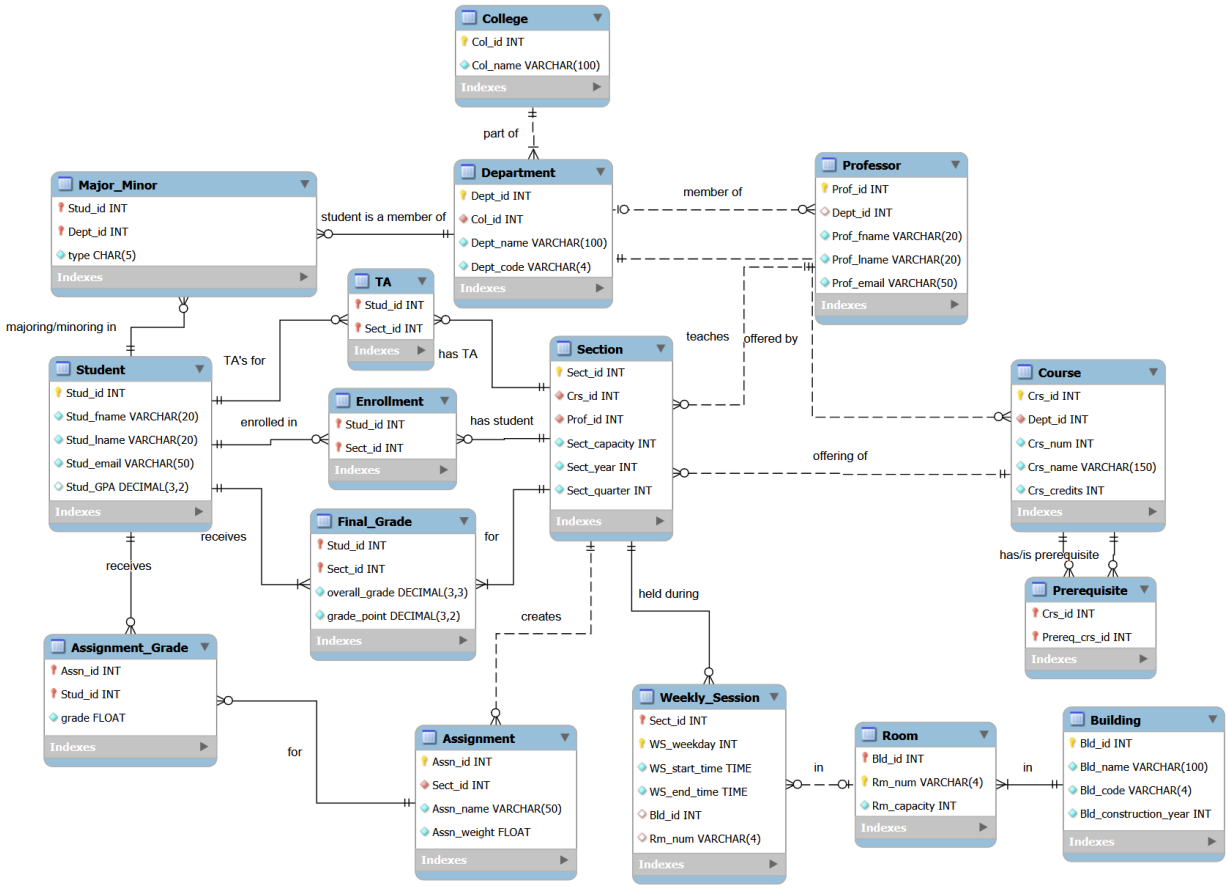
The Administrator View

Administrators are a catch-all term for the people tasked with managing the operations of a university. For the purposes of this database this is largely restricted to the duties that immediately affect the other types of users–students and professors. The administrator view includes:

- Section Logistics: Administrators schedule section sessions, including when and where the section will be held as well as which professor will teach the section. They need to make sure that resource usage does not conflict with other sections.
- Professor Positions: Administrators have the authority to decide professor positions.
- Teaching Assistant Positions: Administrators are responsible for the university budget, which requires that they be aware of things like payroll expenditures.
- College and Department Organization: Administrators are responsible for deciding the college-department hierarchy and degree options.

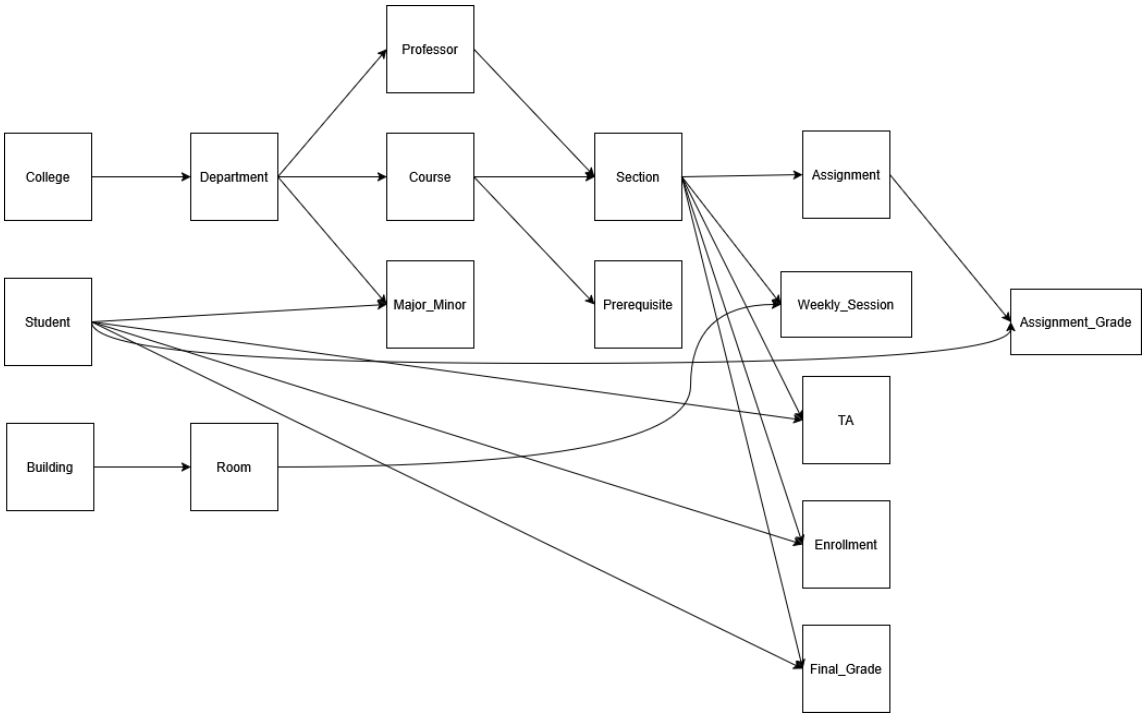


Logical Model



Data Generation

The first thing we did when generating the data was to topologically sort the tables with respect to foreign key dependency. The resulting hierarchical structure tells that, for example, we must complete the Assignment table before we can complete the Assignment_Grade table. Organizing the tables into levels based on their depth in the ordering gives us a logical order to create the data and a necessary order to insert it into MySQL later.



Topological ordering of the tables with respect to foreign key dependency.

We implemented data generation in Python, creating a DataFrame for each table and then writing that DataFrame to a CSV file that could be directly loaded into the corresponding SQL table. Keeping such a close correspondence between the Python data structures and the SQL tables streamlined the eventual loading process, but also made the data generation process tedious at times. For data elements that were difficult to randomize effectively, such as names, courses, and prerequisites, we created "seed" CSV files using external tools or data sources, with much of the data coming from the University of Washington. These seed files underwent further processing to create the final CSV files ready for MySQL import.

College

Several colleges were manually selected from the University of Washington based on target departments. An id column was added during processing.

Student

We generated 3,000 samples of student information using Mockaroo. Email addresses were standardized to use the "@uni.edu" domain and an id column was added during processing. We also created a temporary column "temp_aptitude" to be used in grade generation (and dropped afterwards).

Building

Manually selected some lecture buildings from the UW campus map. Added an id column during processing.

Room

For each building, we scraped the website https://www.washington.edu/classroom/[BUILDING_CODE] (example) to get all the rooms in the building, and traversed each room's individual page (example) to get the room capacity. Finally, we manually checked the resulting file to fix any irregularities (there was just one room that had an actual name instead of just a number).

Department

The department table was carefully chosen to include a diverse but manageable set of departments. The challenge lies in the fact that course prerequisites often span multiple departments, and so including a department often entails including other departments (ad infinitum). So, for example, if you wanted to include the chemical engineering department, you would also have to include the chemistry department. Department ids added and College ids inferred during processing.

Major_Minor

Students were randomly assigned either no major (25%), one major (50%), or two majors (25%), with students having at least one major also having a 25% chance of being assigned a minor. Departments were sampled uniformly.

Professor

We generated 200 samples of professor information using Mockaroo. Email addresses were standardized to use the "@uni.edu" domain, an id column was added, and professors were uniformly assigned to departments during processing. We also created a temporary column "temp_aptitude" to be used in grade generation (and subsequently dropped).

Course and Prerequisite

We scraped the course catalog of each department (example) to get an initial set of courses and their prerequisite information, dropping graduate level courses and courses whose credit count couldn't be resolved to an integer (e.g., some courses gave a *range* of possible credits). We quickly realized that our model for prerequisites wasn't sophisticated enough to capture real prerequisite relationships. For example, there were courses that required you to take one prerequisite *or* another prerequisite. There were even cases with course not requiring *specific* courses (e.g., you must take one philosophy course, any philosophy course). We had to manually prune courses and/or prerequisites according to our department and changing course data and prerequisite model (e.g., if a course had a prerequisite offered by a department we didn't include, we either had to drop that course or drop the prerequisite; dropping a course may force you to drop other courses that rely on it).

Section and Weekly_Session

We created two sections per course per quarter, with section capacities determined by course level (larger for 100-level courses, smaller for 400-level courses). Each section was scheduled for three weekly sessions at consistent times and locations. Session scheduling was done so as to avoid conflicts in professor and room schedules.

Assignment

Each course was randomly chosen to have either 4 or 8 assignments, each titled "Homework i" and uniformly weighted. This assignment information was propagated to each course's sections.

Enrollment, TA, Assignment_Grade, and Final_Grade

To generate the quarterly enrollment, TA, Assignment_Grade, and Final_Grade data, we proceeded one quarter at a time in *chronological* order. For each quarter, we first assigned TA positions and section enrollments for each student. Students could receive at most one TA position per quarter, limited to courses they had previously passed, with higher-performing students given priority for these positions. Each student was then enrolled in four course sections for which they passed all the prequsuites, biased towards courses in their major(s) and minor.

After generating TA assignments and enrollments for the quarter, we generated assignment grades using a normal distribution model. The mean of this distribution was adjusted based on three

factors: student aptitude, professor aptitude, and the number of TAs available in each section. Final grades for each student were then calculated based on their assignment grades.

Working with the Data

Stored Procedures

GetTranscript

Description: GetTranscript takes an unsigned integer id as input and returns the transcript of the student with the provided id, sorted in the order the student took the courses. If the provided id is invalid, returns an error message indicating that no such student exists.

Example Usage and Output:

```
CALL GetTranscript(3);
```

first_name	last_name	year	quarter	department	course_number	course_name	credits	overall_grade	grade_point
Rudiger	Butler-Bowdon	2023	1	ECON	200	Introduction to Microeconomics	5	0.828	3.02
Rudiger	Butler-Bowdon	2023	1	ECON	216	Economics of Sports	5	0.877	3.42
Rudiger	Butler-Bowdon	2023	1	ECON	235	Introduction to Environmental Economics	5	0.995	4.00
Rudiger	Butler-Bowdon	2023	1	ECON	496	Honors Seminar	5	0.949	3.99
Rudiger	Butler-Bowdon	2024	2	PHYS	106	Facilitated Group Inquiry III	2	1.000	4.00
Rudiger	Butler-Bowdon	2024	2	ECON	282	Using Econometrics: A Practical Approach	5	1.000	4.00
Rudiger	Butler-Bowdon	2024	2	ECON	345	Global Health Economics	5	1.000	4.00
Rudiger	Butler-Bowdon	2024	2	LING	359	Language and Ethnicity	5	0.911	3.69
Rudiger	Butler-Bowdon	2024	3	ECON	201	Introduction to Macroeconomics	5	0.987	4.00
Rudiger	Butler-Bowdon	2024	3	ECON	230	Economics of Fisheries and Oceans	5	1.000	4.00
Rudiger	Butler-Bowdon	2024	3	RE	250	Introduction to Real Estate	3	1.000	4.00
Rudiger	Butler-Bowdon	2024	3	ECON	346	Using Economics to Solve Today's Health…	4	0.992	4.00
Rudiger	Butler-Bowdon	2024	4	PHIL	291	Ethics in Science	5	1.000	4.00
Rudiger	Butler-Bowdon	2024	4	RE	415	Real Estate Law	3	0.892	3.54
Rudiger	Butler-Bowdon	2024	4	ECON	494	Economy of Japan	5	1.000	4.00
Rudiger	Butler-Bowdon	2024	4	ECON	497	Honors Directed Study	5	0.953	4.00
Rudiger	Butler-Bowdon	2024	1	RE	413	Real Estate Finance and Investment	4	0.903	3.62
Rudiger	Butler-Bowdon	2024	1	RE	418	Best Practices in Sustainable Real Estate	3	1.000	4.00
Rudiger	Butler-Bowdon	2024	1	RE	462	Residential Real Estate Data Analytics	5	1.000	4.00
Rudiger	Butler-Bowdon	2024	1	RE	464	Affordable Housing	4	0.989	4.00
Rudiger	Butler-Bowdon	2025	2	RE	397	Introduction to Real Estate Data Modeling	5	1.000	4.00
Rudiger	Butler-Bowdon	2025	2	RE	416	Real Estate Economics and Market Analysis	4	0.949	3.99
Rudiger	Butler-Bowdon	2025	2	PHIL	466	Philosophy of the Social Sciences	5	1.000	4.00
Rudiger	Butler-Bowdon	2025	2	MATH	499	Undergraduate Research	8	0.996	4.00
Rudiger	Butler-Bowdon	2025	3	MATH	100	Algebra	5	1.000	4.00
Rudiger	Butler-Bowdon	2025	3	RE	361	Property Transactions	3	1.000	4.00
Rudiger	Butler-Bowdon	2025	3	RE	411	Real Estate Valuation and Appraisal	3	1.000	4.00
Rudiger	Butler-Bowdon	2025	3	RE	497	Real Estate Data Modeling	4	0.985	4.00
Rudiger	Butler-Bowdon	2025	4	RE	363	Real Estate Development Process	3	0.941	3.93
Rudiger	Butler-Bowdon	2025	4	RE	408	Financial Modeling for Real Estate I	3	0.989	4.00
Rudiger	Butler-Bowdon	2025	4	RE	431	Real Estate Portfolio Management	4	0.978	4.00
Rudiger	Butler-Bowdon	2025	4	RE	459	Risk and Reward in Sustainable Develop…	3	0.997	4.00

GetProfessorSchedule

Description: GetProfessorSchedule takes an id, year, and quarter as input (all unsigned integers) and returns the schedule of the professor with the provided id during the specified year and quarter, sorted by weekday and then start time. Returns an error message indicating that no such professor exists if the provided id is invalid. (Note: Returns an empty result table if the provided year and/or quarter are invalid.)

Example Usage and Output:

```
CALL GetProfessorSchedule(5,2023,1);
```

first_name	last_name	department	course_num	course_name	building	room_num	weekday	start_time	end_time
Idaline	Matushevitz	CSE	160	Data Programming	Mary Gates Hall	389	1	10:00:00	10:50:00
Idaline	Matushevitz	CSE	122	Introduction to Computer Programming II	Mary Gates Hall	389	2	09:00:00	09:50:00
Idaline	Matushevitz	CSE	122	Introduction to Computer Programming II	Mary Gates Hall	389	3	09:00:00	09:50:00
Idaline	Matushevitz	CSE	160	Data Programming	Mary Gates Hall	389	3	10:00:00	10:50:00
Idaline	Matushevitz	CSE	122	Introduction to Computer Programming II	Mary Gates Hall	389	5	09:00:00	09:50:00
Idaline	Matushevitz	CSE	160	Data Programming	Mary Gates Hall	389	5	10:00:00	10:50:00

GetCoursePrerequisites

Description: GetCoursePrerequisites takes a department code (VARCHAR(4)) and course number (unsigned integer) as input and returns the prerequisites required to take the specified course. Prerequisites are sorted by their *depth*, where the depth of a course is the depth of its prerequisite tree, which is the number of quarters it would take you to be able to finish its prerequisites if you could take an infinite number of courses per quarter. Returns an error message indicating that no such course exists if the specified course cannot be found.

Example Usage and Output: The prerequisites for CSE 446 (Machine Learning) are

```
CALL GetCoursePrerequisites('CSE',446);
```

department_code	course_number	course_name	course_depth	prerequisites
CSE	123	Introduction to Computer Programming III	0	0
MATH	124	Calculus with Analytic Geometry I	0	0
MATH	125	Calculus with Analytic Geometry II	1	1
MATH	126	Calculus with Analytic Geometry III	2	2
MATH	208	Matrix Algebra with Applications	3	3
CSE	311	Foundations of Computing I	3	4
CSE	312	Foundations of Computing II	4	5
CSE	332	Data Structures and Parallelism	4	5

Example Queries

What is the average course depth and number of prerequisites for each department?

```
WITH course_depth_and_prereqs AS (
WITH RECURSIVE prereq_path_length AS (
    -- Immediate prerequisites have a path of length 1
    SELECT
        P.Crs_id AS start_crs_id,
        P.Prereq_crs_id AS end_crs_id,
        1 AS path_length
    FROM Prerequisite AS P

    UNION ALL

    -- Extend current paths
    SELECT
        PPL.start_crs_id AS start_crs_id,
        P.Prereq_crs_id AS end_crs_id,
        (PPL.path_length + 1) AS path_length
    FROM prereq_path_length AS PPL, Prerequisite AS P
    WHERE PPL.end_crs_id = P.Crs_id
)
SELECT
    D.Dept_code,
    C.Crs_num,
    C.Crs_name,
    (CASE
        WHEN MAX(PPL.path_length) IS NULL THEN 0
        ELSE MAX(PPL.path_length)
    END) AS depth,
    COUNT(DISTINCT PPL.end_crs_id) AS prerequisites
FROM Course AS C
JOIN Department D ON C.Dept_id = D.Dept_id
LEFT JOIN prereq_path_length AS PPL ON C.Crs_id = PPL.start_crs_id
GROUP BY C.Crs_id
)
SELECT Dept_code, AVG(depth), AVG(prerequisites) FROM course_depth_and_prereqs GROUP BY Dept_code;
```

Dept_code	AVG(depth)	AVG(prerequisites)
ARCH	0.7347	0.7347
CSE	3.2464	4.4493
ECON	1.9828	3.1552
LING	0.8605	0.8837
MATH	3.3333	3.6667

What are the five courses with the lowest passing rate?

```
SELECT D.Dept_code department, C.Crs_num course_num, C.Crs_name course_name,
       COUNT(DISTINCT FG.Sect_id) num_sections,
       COUNT(FG.Stud_id) num_students,
       -- use 1.0 as passing cut off
       ROUND(SUM(CASE WHEN FG.grade_point >= 1.0 THEN 1 ELSE 0 END)/COUNT(*),2) passing_rate
FROM COURSE C, Section S, Department D, Final_Grade FG
WHERE C.Crs_id = S.Crs_id AND C.Dept_id = D.Dept_id AND S.Sect_id = FG.Sect_id
GROUP BY C.Crs_id
HAVING COUNT(FG.Stud_id) >= 20  -- only courses with more than 20 sample size
ORDER BY passing_rate ASC
LIMIT 5;
```

department	course_num	course_name	num_sections	num_students	passing_rate
ARCH	301	Introduction to Architectural Design II	5	48	0.83
ARCH	481	3D Modeling and Rendering	12	194	0.92
ECON	497	Honors Directed Study	10	135	0.92
MATH	207	Introduction to Differential Equations	6	186	0.92
ARCH	201	Architectural Design and Representation II	7	207	0.93

Who are the five hardest professors?

```
SELECT P.Prof_id id, P.Prof_fname first_name, P.Prof_lname last_name,
       COUNT(DISTINCT FG.Sect_id) num_sections,
       COUNT(FG.Stud_id) num_students,
       ROUND(AVG(FG.overall_grade),3) average_grade,
       ROUND(AVG(FG.grade_point),2) average_grade_point
FROM Professor P, Section S, Final_Grade FG
WHERE P.Prof_id = S.Prof_id AND S.Sect_id = FG.Sect_id
GROUP BY P.Prof_id
HAVING COUNT(FG.Stud_id) >= 20  -- only courses with at least 20 sample size
ORDER BY average_grade ASC
LIMIT 5;
```

id	first_name	last_name	num_sections	num_students	average_grade	average_grade_point
97	Lishe	Christian	10	151	0.723	2.19
116	Lambert	Geffen	5	123	0.743	2.33
39	Lanie	Johnsey	8	180	0.772	2.55
154	Willis	Marriner	4	190	0.773	2.55
96	Cleavland	Millam	8	150	0.793	2.71

What is the relationship between student grades and TA frequency?

```
WITH TA_counts AS (
    SELECT
        Stud_id,
        COUNT(DISTINCT Sect_id) as times_TAd
    FROM TA
    GROUP BY Stud_id
)
SELECT
    CASE
        WHEN TC.times_TAd > 5 THEN 'Frequent TA (>5 sections)'
        WHEN tc.times_TAd IS NOT NULL THEN 'Occasional TA (1-5 sections)'
        ELSE 'Never TA'
    END TA_frequency,
    COUNT(DISTINCT s.Stud_id) student_count,
    ROUND(AVG(S.Stud_GPA),2) average_GPA,
    ROUND(MIN(s.Stud_GPA),2) minimum_GPA,
    ROUND(MAX(s.Stud_GPA),2) maximum_GPA
FROM Student S
LEFT JOIN TA_counts TC ON S.Stud_id = TC.Stud_id
WHERE S.Stud_GPA IS NOT NULL
GROUP BY TA_frequency
ORDER BY average_GPA DESC;
```

TA_frequency	student_count	average_GPA	minimum_GPA	maximum_GPA
Frequent TA (>5 sections)	521	3.734	2.24	4.00
Occasional TA (1-5 sections)	2402	3.263	1.08	4.00
Never TA	77	1.801	0.90	3.01

What is the average student GPA for each major?

```
SELECT D.Dept_name AS department, AVG(S.Stud_GPA) AS average_gpa
FROM Major_Minor AS MM, Department AS D, Student AS S
WHERE MM.Stud_id = S.Stud_id AND MM.Dept_id = D.Dept_id AND MM.type = 'Major'
GROUP BY D.Dept_name
ORDER BY average_gpa DESC;
```

department	average_gpa
Department of Physics	3.386580
Mechanical Engineering	3.326646
Department of Philosophy	3.309277
Department of Economics	3.304169
Department of Linguistics	3.296036
Department of Mathematics	3.295096
Paul G. Allen School of Computer Science & Engi...	3.200813
Architecture	3.197543
Real Estate	3.196867

How does student performance change across quarters?

```
SELECT Sect_quarter quarter,
       ROUND(AVG(fg.grade_point),3) as avg_grade_point,
       ROUND(AVG(fg.overall_grade*100), 2) as avg_percentage
FROM Section S, Final_Grade FG
WHERE S.Sect_id = FG.Sect_id
GROUP BY Sect_quarter
ORDER BY Sect_quarter;
```

quarter	avg_grade_point	avg_percentage
1	3.056	84.35
2	3.385	89.09
3	3.401	89.40
4	3.376	88.94

Connecting with Python

To connect to MySQL within a Python program, simply import mysql.connector and call the function mysql.connector.connect() with the appropriate host, username, password, and database. For example, we can connect to our database with the following code (you would need to change at least the password):

```
import mysql.connector

db = mysql.connector.connect(
    host = "localhost",
    user = "root",
    password = "redacted",
    database = "University"
)
```

Once a connection to the database has been established, you can execute any queries that the provided user has permission to execute.

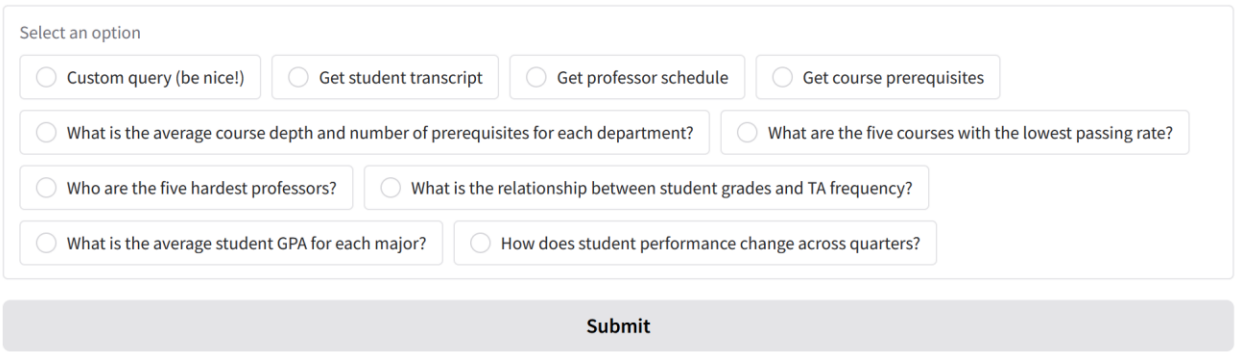
Python GUI Demo

An example of how to work with MySQL in Python can be seen in milestone3code.ipynb, where we wrote a simple GUI that can execute the example queries above as well as custom queries provided by the user. In addition to mysql.connector, you will also need to have need to have pandas and gradio installed. **Warning:** Use pip to install gradio, not anaconda!

Once you have all the dependencies, you just need to (1) make sure the MySQL server is running and (2) change the database credentials according to your setup. Then you can press “Run All” and scroll to the bottom to see the GUI. There is also a link to open it in the browser if you prefer.

```
# TODO: Change these according to your setup
host = "localhost"
user = "root"
password = "YOUR PASSWORD"
database = "University"
```

Make sure to change this code block.



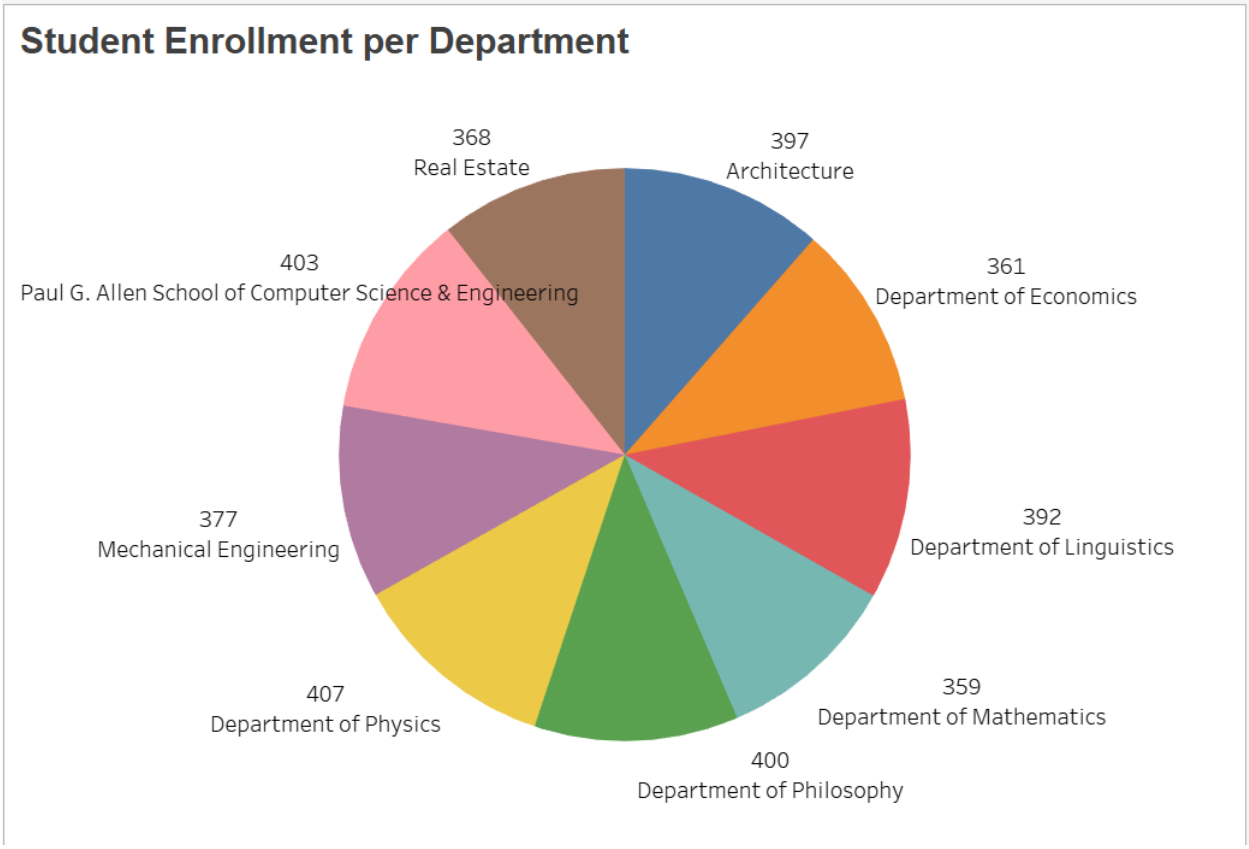
GUI created upon pressing "Run All." To use, just select one of the options and press "Submit." Some of the options take parameters, in which case textboxes will appear upon selecting the option and you will have to enter values before pressing “Submit.”

Connecting with Tableau

When you first open Tableau, there will be a blue banner on the left-hand side providing different options for connecting data. To connect Tableau to a MySQL server, click “MySQL” under “To a Server.” If this your first time doing so, then you will likely be prompted to install a driver. After you have successfully installed the driver, repeating the process above will instead prompt you to enter your database credentials, after which you Tableau will be able to access your data (more detailed information on connecting Tableau to a MySQL server can be found here).

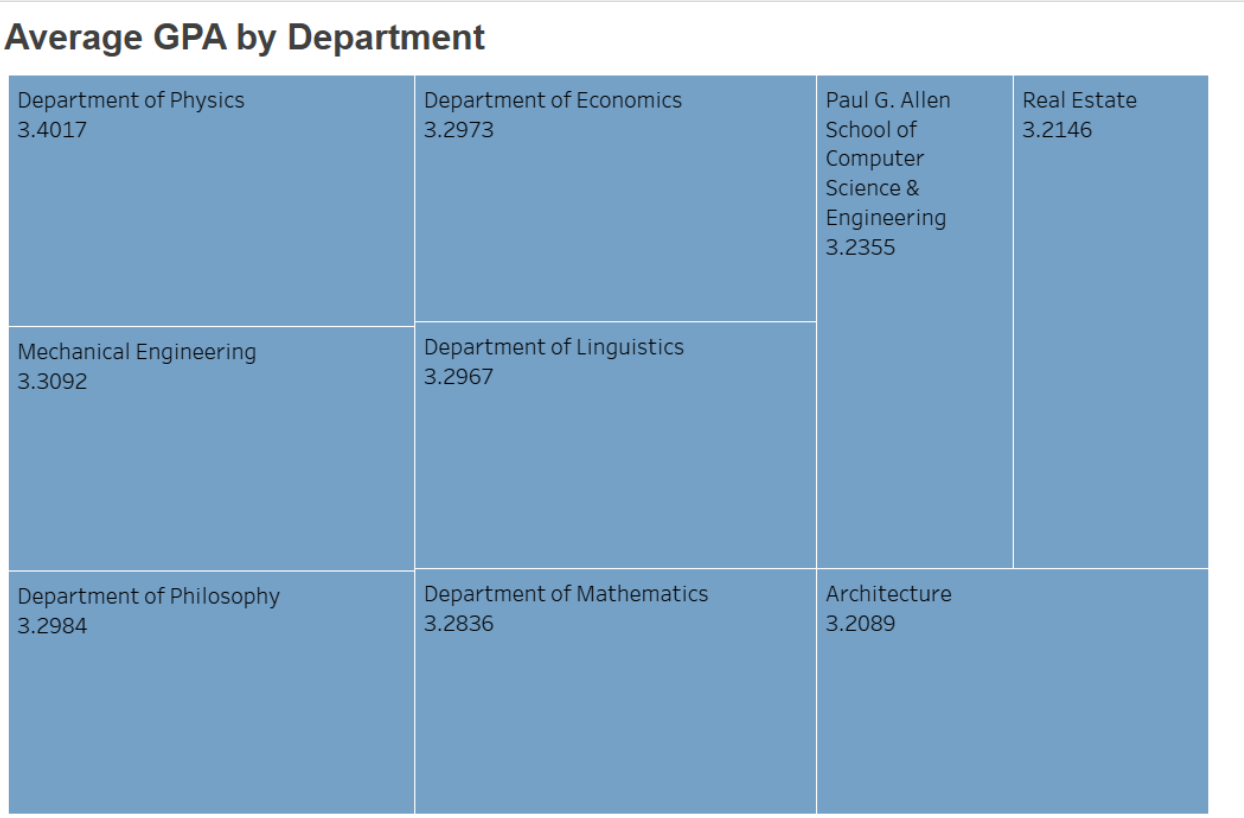
Tableau Visualization Demo

Pie Chart



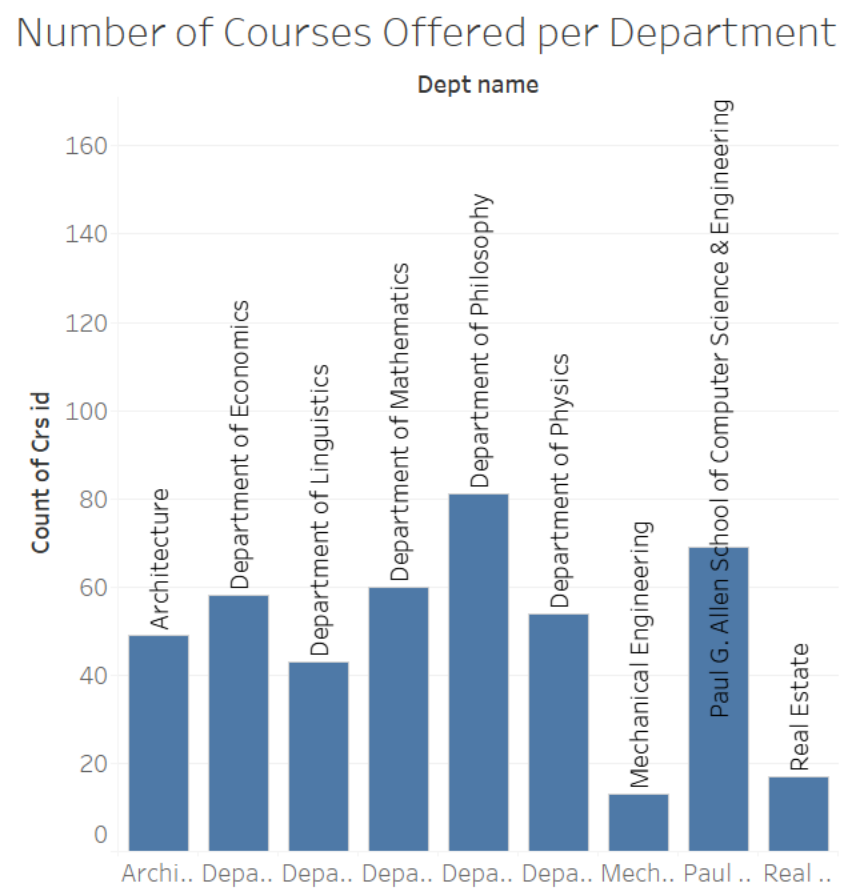
The pie chart helps in understanding which departments have higher or lower student enrollments relative to others. For instance, if a department like Linguistics has enrollments of 407 and 359, it suggests a significant number of students compared to a department like Economics with 361 enrollments. The chart provides a quick visual comparison, making it easier to identify trends and disparities in student interest or department capacity.

TreeMap



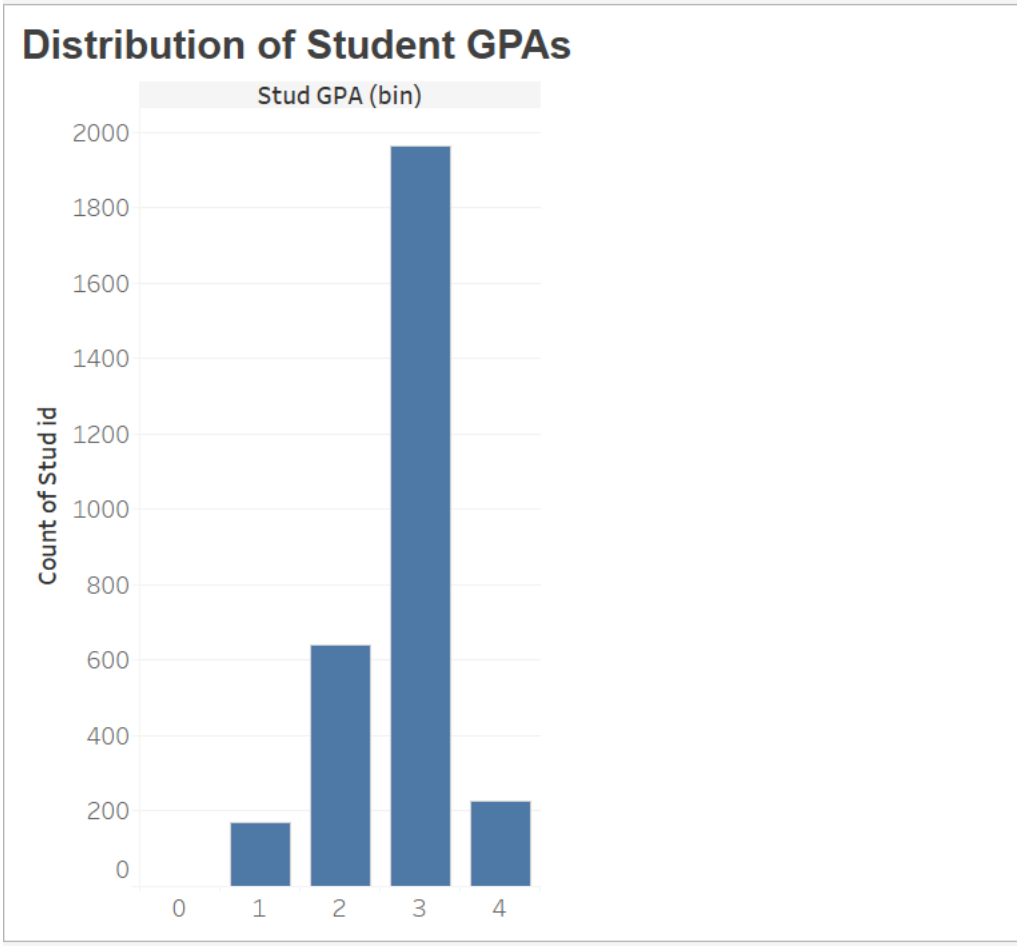
The treemap allows for a quick visual comparison of the average GPAs across departments. Larger or more prominently colored rectangles may indicate higher average GPAs, making it easy to identify which departments have higher or lower academic performance metrics. For example, the Department of Physics has the highest average GPA (3.4017), while Architecture has the lowest (3.2089). This visualization helps in understanding the academic performance distribution across different departments at a glance.

Bar Graph



The bar graph represents the frequency or count of departments within each academic field. The height of each bar corresponds to the number of departments in that category. For example, the bar for Architecture would be the tallest, reflecting the five departments listed under it, while Mechanical Engineering would have a shorter bar, representing its single department.

Bar Graph



- The distribution can help identify trends in academic performance. For instance, if the bar for the GPA bin "3" is the tallest, it indicates that the majority of students have GPAs in the 3.0 to 3.99 range.
- A shorter bar for the GPA bin "0" or "1" would suggest fewer students with very low GPAs.
- The bar for the GPA bin "4" shows the number of students achieving the highest possible GPA.

Here is the Link to Dashboard

https://public.tableau.com/app/profile/fariha.shah/viz/Student_Grading_System/DraftDashboard?publish=yes