Milestone: Final Report

Online Learning Platform Database

Group 3

Ziqi Li

Yiqian Ning

470-836-9925

857-250-6596

li.ziqi3@northeastern.edu

ning.yi@northeastern.edu

Percentage of Effort Contributed by Student1: 50%

Percentage of Effort Contributed by Student2: 50%

Signature of Student1: Ziqi Li

Signature of Student2: Yiqian Ning

Submission Date: Apr 16th 2024

Contents

Executive Summary:

In the modern educational environment, with the increase in the number of students and the diversification of educational needs, traditional methods of data management have gradually failed to meet the demands of high efficiency and flexibility. To this end, we have developed a database for a complex educational management system that utilizes advanced database management techniques and aims to provide a structured and efficient way of storing and processing large amounts of education-related data.

This system involves the management of information about students, teachers, courses, and employees within an educational organization. It includes not only basic personal information records, such as students' names, contact information, and dates of birth, but also more multidimensional data such as course schedules, management, teaching and learning activities, and students' VIP status and assignments. In this context, the system needs to deal with a variety of data relationships, such as the enrollment relationship between students and courses, between instructors and the course sections they are responsible for, and between administrators and the courses or departments under their jurisdiction.

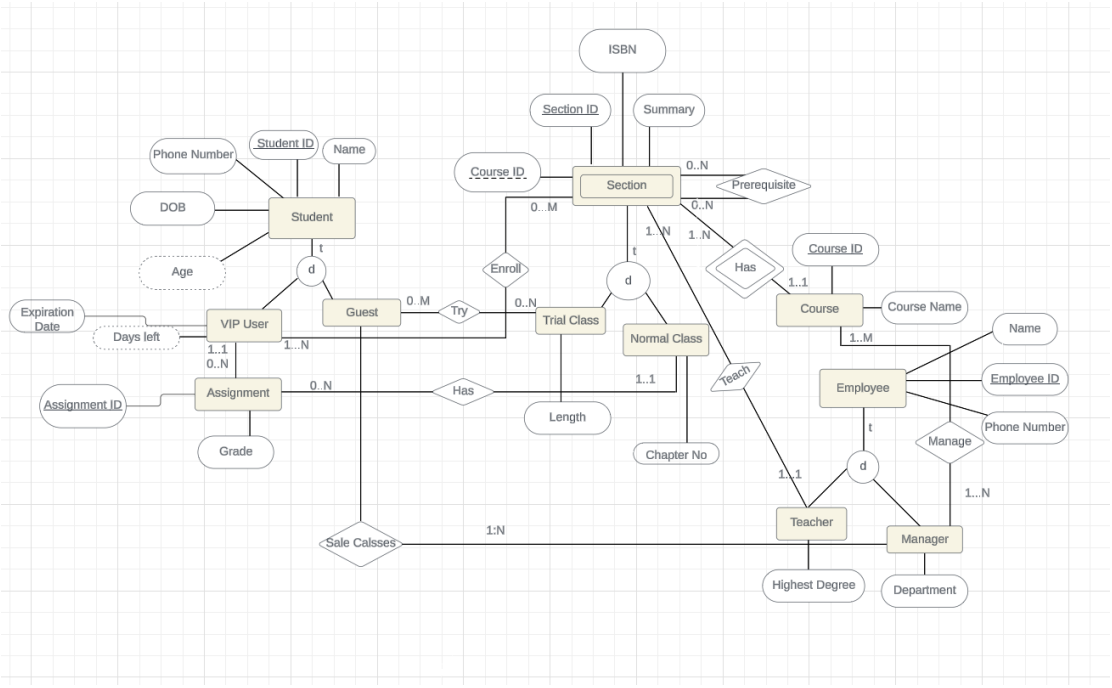
To accomplish this, we first designed Entity-Relationship (EER) diagrams that provide us with a visual representation of a conceptual model. We then mapped this conceptual model to a relational model, identifying the necessary primary and foreign keys to ensure data consistency and integrity. Subsequently, the database was developed using MySQL. We also use MongoDB to evaluate its suitability in a NoSQL environment. Additionally, to extract more insights from this data, we combined the database with Jupyter Notebook to leverage Python's powerful data analytics capabilities for in-depth research. This allowed us to run complex data queries, perform statistical analysis, and visualize data trends. By using data science libraries in Python, such as Pandas, NumPy, and Matplotlib, we can not only process and analyze data, but also build intuitive graphs and reports. This integrated approach enhances our database's role as more than just a store of information, but also facilitates the exploration of the deeper meaning behind our data.

1. Background

With the development of science and technology, the wide application of the Internet, and the occurrence of global events in recent years, the demand for online education is gradually increasing, and various countries have greatly promoted the popularization of online education. The platform provides students with more convenient and flexible learning opportunities, breaking through traditional education's time and space limitations. Students' online learning platforms can meet the needs of students to obtain learning resources anytime and anywhere. Improve the interaction between students and teachers. Students can also choose diversified and global teaching materials according to their level and interests and replay or focus on learning according to their own learning pace.

The realization of students' online learning platform database is of great practical significance, which can not only promote educational innovation, meet students' diversified learning needs, and improve teaching effects, but also improve students' ability to autonomous learning and self-discipline. By designing and implementing an efficient database of online learning platforms for students, our group will help to better manage information on students, courses, teachers, etc., provide a better learning experience, and promote the sustainable development of education. This is not only in line with the trend of contemporary education but also provides more opportunities for students and educational institutions to develop.

2. Conceptual Data Modeling (EER)



3. Mapping Conceptual Model to Relational Model

Primary Key- Underlined Foreign Key- *Italicized*

- Student (Student_ID, Name, Phone_Number, DOB DATE)
- VIP (*Student_ID*, vip_expiration_date)
  FOREIGN KEY (Student_ID) REFERENCES Student (Student_ID)
- Employee (Employee_ID, Name, Phone_Number)
- Manager (Employee_ID, Department)
  FOREIGN KEY (Employee_ID) REFERENCES Employee (Employee_ID))
- Guest (*Student_ID*, Advisor_ID,)
  FOREIGN KEY (Student_ID) REFERENCES Student (Student_ID)
  FOREIGN KEY (Advisor_ID) REFERENCES Manager (Employee_ID)
- Course (Course_ID, Course_Name)
- Teacher (*Employee_ID*, Highest_Degree);
  FOREIGN KEY (Employee_ID) REFERENCES Employee (Employee_ID)
- Sections (*Section_ID, Course_ID*, Textbook_ISBN, Summary, Prerequisite, Employee_ID)
  FOREIGN KEY (Course_ID) REFERENCES Course (Course_ID)
  FOREIGN KEY (Employee_ID) REFERENCES Teacher (Employee_ID)
- Trial_Class (*Section_ID, Course_ID*, Length)
  FOREIGN KEY (Section_ID, Course_ID) REFERENCES SECTIONS (Section_ID, Course_ID)
- Normal_Class (*Section_ID, Course_ID*, Chapter_No)
  FOREIGN KEY (Section_ID, Course_ID) REFERENCES SECTIONS (Section_ID, Course_ID)
- Manage (*Employee_ID, Course_ID*)
  FOREIGN KEY (Employee_ID) REFERENCES Manager (Employee_ID)
  FOREIGN KEY (Course_ID) REFERENCES Course (Course_ID))
- Prerequisite (*Required_Section_ID, Required_course_ID, Course_ID, Section_ID*)
  FOREIGN KEY (Required_Section_ID, Required_course_ID) REFERENCES Sections (Section_ID, Course_ID)
  FOREIGN KEY (Section_ID, Course_ID) REFERENCES Sections (Section_ID, Course_ID))
- Try (*Student_ID, Section_ID, Course_ID*)
  FOREIGN KEY (Student_ID) REFERENCES Guest (Student_ID)
  FOREIGN KEY (Section_ID, Course_ID) REFERENCES Trial_class (Section_ID, Course_ID))
- Enroll (*Section_ID, Student_ID, Course_ID*)
  FOREIGN KEY (Section_ID, Course_ID) REFERENCES Sections (Section_ID, Course_ID)
  FOREIGN KEY (Student_ID) REFERENCES VIP(Student_ID))
- Assignment (*Student_ID, Section_ID, Course_ID*, Grade)
  FOREIGN KEY (Student_ID) REFERENCES VIP(Student_ID)
  FOREIGN KEY (Section_ID, Course_ID) REFERENCES normal_class (Section_ID,

```
Course_ID))
    SET SQL_SAFE_UPDATES = 0;
    SELECT TIMESTAMPDIFF(YEAR, dob, NOW()) AS age1 FROM student;
    ALTER TABLE student ADD COLUMN age1 INT;
    UPDATE student SET age1 = TIMESTAMPDIFF(YEAR, dob, NOW()) WHERE student_id
IS NOT NULL;

    SELECT DATEDIFF(NOW(),vip_expiration_date) AS vip_remaining_date FROM vip;
    ALTER TABLE vip ADD COLUMN vip_remaining_date INT;
    UPDATE vip SET vip_remaining_date = TIMESTAMPDIFF(day,
NOW(),vip_expiration_date) WHERE student_id IS NOT NULL
```

Introduction to the relational model above:

1. Student table contains either VIP table or guest student table. VIP students can enroll in both normal class and trial class. But only normal classes have assignments and will give a grade. Guest students can only try trial class.
2. Employee table contains either teacher table or manager table. There are two types of managers in manger table: manager and sales. Managers in computer science department course manage courses named computer science and each of the corresponding sections and arrange teacher for each section. For sales, at current stage, we don't have too many guest students. So, there is only one sale in sale department. In the future, if our online learning platform becomes large, we will hire more salespeople.
3. Course table has two attribute types of course name and course id.
4. Section table includes each level of a specific course. For example: course 'computer science' has 5 sections. The difficulty of section1 to section 4 becomes harder in turn. Section5 is a trial class.
5. Prerequisite table is about the enroll requirement. If a VIP student wants enroll section 4 in any course, he/she must enroll section 3. There is no requirement for section 1, section 2, and section 3.

4. Implementation of Relation Model via MySQL and NoSQL

4.1 MySQL Implementation：

4.1.1 Simple Query

To inquire the name and age of the student:

SELECT Name, age1 FROM Student;

Name	age1
Patricia Barnes	20
Ryan Smith	18
Jessica Kelley	20
Michael Vargas	21
William Petty	18
Gary Adams	20
Mr. Michael Townsend	18

4.1.2 Aggregate Query

To know the average age of students in the platform's audience:

SELECT AVG(age1) AS average_age FROM Student;

average_age
19.6900

4.1.3 Inner Join Query

Query each student's grade, as well as the corresponding course name, section id:

```
SELECT
    s.Name AS Student_Name,
    a.Grade AS Score,
    c.Course_Name,
    c.Course_ID,
    a.Section_ID
FROM
    Student s
INNER JOIN
    Assignment a ON s.Student_ID = a.Student_ID
INNER JOIN
    Course c ON a.Course_ID = c.Course_ID
INNER JOIN
    Sections sec ON a.Course_ID = sec.Course_ID AND a.Section_ID = sec.Section_ID
order by s.name;
```

Student_Name	Score	Course_Name	Course_ID	Section_ID
Adam Casey	B	Economics	1212	1
Adam Casey	B	Mathematics	101	3
Adam Casey	C+	Political Science	1313	1
Adam Casey	C+	Biology	505	2
Adam Smith	C	Medicine	1717	4
Adam Smith	A	Medicine	1717	3
Adam Smith	B	Sociology	1515	2
Allison Wiggins	A	Mathematics	101	1
Allison Wiggins	B-	Biology	505	1
Allison Wiggins	B+	Music	1010	3
Amanda Vela…	A	History	707	3
Amanda Vela…	A	History	707	2
Amanda Vela…	B	Engineering	1818	1
Amanda Vela…	B	History	707	4
Amanda Vela…	A+	Law	1616	3
Amanda Vela…	B	Physical Educ…	1111	4
Amanda Vela…	C+	Physical Educ…	1111	3
Bradley Smith	C	Engineering	1818	1
Bradley Smith	A	Chemistry	404	3
Bradley Smith	A	Chemistry	404	4
Bradley Smith	B+	Physics	303	1
Bradley Smith	B+	Art	909	3

4.1.4 Left Outer Join Query

List each section of all courses:

```
SELECT c.Course_ID, c.Course_Name, sec.Section_ID, sec.Summary
FROM Course c
LEFT OUTER JOIN Sections sec ON c.Course_ID = sec.Course_ID;
```

Course_ID	Course_Name	Section_ID	Summary
101	Mathematics	1	This is course 0101 level 1
101	Mathematics	2	This is course 0101 level 2
101	Mathematics	3	This is course 0101 level 3
101	Mathematics	4	This is course 0101 level 4
101	Mathematics	5	This is course 0101 trial class
202	Computer Science	1	This is course 0202 level 1
202	Computer Science	2	This is course 0202 level 2
202	Computer Science	3	This is course 0202 level 3
202	Computer Science	4	This is course 0202 level 4
202	Computer Science	5	This is course 0202 trial class

4.1.5 Nested Query

List the names and ages of all students above the average age:

```
SELECT Name, age1
FROM Student
WHERE age1 > (SELECT AVG(age1) FROM Student);
```

Name	age1
Patricia Barnes	20
Jessica Kelley	20
Michael Vargas	21
Gary Adams	20
Veronica Nichols	20
Shelby Mcdowell	22
Eduardo Gibson	21
Holly Campos	21
Jordan Garcia	22

4.1.6 Correlated Subquery

List each student who took section 3 of the same course and got an A, but did not take Section 4:
In the same way, we can also identify students who did not take section1 and section 2 at a certain course but can get an A in section 4 and who enroll a section without taking the trial class.

```
SELECT DISTINCT s.Student_ID, s.Name, a.Course_ID, c.Course_Name
FROM Student s
JOIN Assignment a ON s.Student_ID = a.Student_ID
JOIN Course c ON a.Course_ID = c.Course_ID
WHERE a.Grade = 'A'
AND a.Section_ID = 3
AND NOT EXISTS (
    SELECT 1
    FROM Enroll e
    WHERE e.Section_ID = 4
    AND e.Course_ID = a.Course_ID
    AND e.Student_ID = s.Student_ID)
AND EXISTS (
    SELECT 1
    FROM Sections sec
    WHERE sec.Course_ID = a.Course_ID
    AND sec.Section_ID = 3);
```

Student_ID	Name	Course_ID	Course_Name
1001	Patricia Barnes	505	Biology
1005	William Petty	101	Mathematics
1017	Holly Campos	606	Geography
1029	Emily Henderson	303	Physics
1033	Stephanie Tucker	606	Geography
1034	Mike Mercer	505	Biology
1036	James Escobar	2020	Agriculture
1037	Regina Haynes	1616	Law
1062	Nicholas Chambers	1616	Law
1081	Tiffany Hill	909	Art
1085	Dylan Ramos	303	Physics
1095	Tanya Bell	707	History
1097	Tami Sanchez	101	Mathematics

4.1.7 >ALL Query

List the students who have taken the most courses:

```
SELECT s.Student_ID, s.Name, COUNT(*) as Sections_Count
FROM Student s
JOIN Enroll e ON s.Student_ID = e.Student_ID
GROUP BY s.Student_ID, s.Name
HAVING COUNT(*) > ALL (
    SELECT COUNT(*)
    FROM Enroll
    GROUP BY Student_ID
    HAVING Student_ID != s.Student_ID);
```

Student_ID	Name	Sections_Count
1088	Joseph Lynch	11

4.1.8 Union Set Operation

List the names and numbers of all staff and students together:

```
SELECT student_id as ID, Name FROM Student
UNION
SELECT employee_id, name FROM Employee
```

ID	Name
1094	Brian Martin
1095	Tanya Bell
1096	Leah Wood
1097	Tami Sanchez
1098	Denise Pham
1099	Brooke Anderson
1100	Angela Davis
AA0101_01	Liam
AA0101_02	Olivia
AA0101_03	Noah
AA0202_01	Emma
AA0202_02	Oliver
AA0202_03	Ava
AA0303_01	Elijah
AA0303_02	Charlotte

4.1.9 Subqueries in SELECT

List the total number of sections taken by each vip student:

```
SELECT
    v.Student_ID,
    s.Name,
    (SELECT COUNT(*)
     FROM Enroll e
     WHERE e.Student_ID = v.Student_ID) AS Sections_Count
FROM vip v
INNER JOIN student s ON v.Student_ID = s.Student_ID;
```

Student_ID	Name	Sections_Count
1001	Patricia Barnes	6
1002	Ryan Smith	0
1004	Michael Vargas	9
1005	William Petty	7
1006	Gary Adams	5
1008	Nicole Butler MD	0
1010	Amanda Velazquez	7
1011	William Wood	5
1013	Lucas Hoffman	0
1015	Eduardo Gibson	7
1016	Dr. Alicia Hess DDS	5
1017	Holly Campos	6
1019	Jordan Garcia	6

4.1.10 Additional Query: CTE

Look for the lowest level section taught by the teacher with the highest degree in each course. Then we can rearrange the teacher.

```
WITH TeacherDegreeRank AS (
      SELECT se.Course_ID,se.Section_ID,t.Employee_ID,t.Highest_Degree,
ROW_NUMBER() OVER (
       PARTITION BY se.Course_ID
       ORDER BY
          CASE t.Highest_Degree
             WHEN 'PhD' THEN 1
            WHEN 'Master' THEN 2
             WHEN 'Bachelor' THEN 3
             ELSE 4
          END) AS degree_rank
      FROM Sections se
 JOIN Teacher t ON se.Employee_ID = t.Employee_ID)
 SELECT c.Course_Name, tdr.Highest_Degree,tdr.Section_ID
 FROM Course c
 JOIN TeacherDegreeRank tdr ON c.Course_ID = tdr.Course_ID
 WHERE tdr.degree_rank = 1;
```

Course_Name	Highest_Degree	Section_ID
Mathematics	Bachelor	1
Computer Science	Master	1
Physics	PhD	1
Chemistry	PhD	3
Biology	Master	1
Geography	Master	1
History	PhD	1
Literature	PhD	1
Art	Master	3
Music	Master	1
Physical Education	PhD	2
Economics	PhD	1
Political Science	PhD	2
Psychology	Master	1
Sociology	PhD	3
Law	PhD	1
Medicine	PhD	3
Engineering	PhD	3
Environmental Sc...	PhD	1
Agriculture	Master	1

4.1.11 Additional Query:

Check the distribution of teachers. Then we can arrange courses according to the highest degree of teachers effectively and hire more suitable teacher.

```
Select s.course_id, e.name, tea.highest_degree, s.section_id
From sections s
Join teacher tea on tea.employee_id=s.employee_id
Join employee e on e.employee_id=tea.employee_id
Order by course_id,section_id,name,highest_degree
```

course_id	name	highest_degr...	section_id
101	Liam	Bachelor	1
101	Olivia	Bachelor	2
101	Noah	Bachelor	3
101	Noah	Bachelor	4
101	Liam	Bachelor	5
202	Emma	Master	1
202	Oliver	Bachelor	2
202	Ava	Master	3
202	Ava	Master	4
202	Emma	Master	5
303	Elijah	PhD	1
303	Char...	PhD	2
303	William	PhD	3
303	William	PhD	4
303	Elijah	PhD	5

4.1.12 Additional Query:

Find students who took the same course and the same section, but the younger ones scored higher than the older ones.

```
SELECT
    young.Name AS Younger_Student_Name,
    old.Name AS Older_Student_Name,
    young.Grade AS Younger_Student_Grade,
    old.Grade AS Older_Student_Grade,
    sec.Course_ID,
    sec.Section_ID
FROM
    (SELECT s.Student_ID, s.Name, s.DOB, a.Grade, a.Course_ID, a.Section_ID, CASE
a.Grade
                WHEN 'A+' THEN 12
                WHEN 'A' THEN 11
                WHEN 'A-' THEN 10
                WHEN 'B+' THEN 9
                WHEN 'B' THEN 8
                WHEN 'B-' THEN 7
                WHEN 'C+' THEN 6
                WHEN 'C' THEN 5
                WHEN 'C-' THEN 4
                WHEN 'F' THEN 3
                ELSE 0
            END AS Numeric_Grade
    FROM Student s
    INNER JOIN Assignment a ON s.Student_ID = a.Student_ID) AS young
INNER JOIN
    (SELECT s.Student_ID, s.Name, s.DOB, a.Grade, a.Course_ID, a.Section_ID, CASE
a.Grade
                WHEN 'A+' THEN 12
                WHEN 'A' THEN 11
                WHEN 'A-' THEN 10
                WHEN 'B+' THEN 9
                WHEN 'B' THEN 8
                WHEN 'B-' THEN 7
                WHEN 'C+' THEN 6
                WHEN 'C' THEN 5
                WHEN 'C-' THEN 4
                WHEN 'F' THEN 3
                ELSE 0
            END AS Numeric_Grade
```

```
    FROM Student s
    INNER JOIN Assignment a ON s.Student_ID = a.Student_ID) AS old
ON
    young.Course_ID = old.Course_ID
    AND young.Section_ID = old.Section_ID
    AND young.Student_ID != old.Student_ID
    AND young.DOB > old.DOB
    AND young.Numeric_Grade > old.Numeric_Grade
INNER JOIN
    Sections sec ON young.Section_ID = sec.Section_ID
    AND young.Course_ID = sec.Course_ID
ORDER BY young.Name, old.Name;
```

Younger_Student_Na...	Older_Student_Na...	Younger_Student_Gra...	Older_Student_Gra...	Course_ID	Section_ID
Adam Casey	Holly Campos	B	F	1212	1
Adam Casey	Monica Holland	C+	C-	1313	1
Adam Casey	Nancy Banks	B	B-	1212	1
Adam Smith	Pamela Rodriguez	A	B	1717	3
Allison Wiggins	Desiree Yates	B+	F	1010	3
Allison Wiggins	Holly Brewer	B+	C	1010	3
Allison Wiggins	Joe Bates	A	C-	101	1
Allison Wiggins	Karen Joseph	A	B	101	1
Allison Wiggins	Michael Perry	B+	C-	1010	3
Allison Wiggins	Michael Vargas	B+	C	1010	3
Allison Wiggins	Tami Sanchez	A	B+	101	1
Amanda Velazquez	Bradley Smith	B	C	1818	1
Amanda Velazquez	Colin Freeman	A	C-	707	2
Amanda Velazquez	Denise Pham	A	B	707	2
Amanda Velazquez	Denise Pham	B	B-	707	4
Amanda Velazquez	Desiree Yates	A+	C	1616	3
Amanda Velazquez	Desiree Yates	A	C	707	2
Amanda Velazquez	Hannah Potter	B	C+	1818	1
Amanda Velazquez	Jennifer Chan	B	F	707	4

4.1.13 Additional Query:

Find student got more than 1 F grade. And send a reminder.

```
WITH FailingGrades AS (
    SELECT st.Student_ID, st.Name, COUNT(*) AS F_Count
    FROM Student st
    INNER JOIN Enroll en ON st.Student_ID = en.Student_ID
    INNER JOIN
        Assignment asgn ON en.Student_ID = asgn.Student_ID
        AND en.Section_ID = asgn.Section_ID
        AND en.Course_ID = asgn.Course_ID
    INNER JOIN VIP v ON st.Student_ID = v.Student_ID
    WHERE asgn.Grade = 'F'
    GROUP BY st.Student_ID, st.Name)
SELECT Student_ID, Name,    F_Count
FROM FailingGrades
WHERE F_Count > 1;
```

Student_ID	Name	F_Count
1048	Jeremy Rivers	2
1069	Jennifer Chan	2

4.1.14 Additional Query:

Find students whose memberships expire within 50 days and send reminders.

```
SELECT s.Name, s.Phone_Number,v.vip_remaining_date
FROM VIP v
INNER JOIN Student s ON v.Student_ID = s.Student_ID
WHERE    v.vip_remaining_date < 50;
```

Name	Phone_Number	vip_remaining_date
Richard Morgan	91743495823	48
Joseph Jones	66286165225	21

4.2 NoSQL

4.2.1 A Simple Query

Find all informations with a section_id of 5:

db.section.find({ "Section_id": 5 });

```
{
  _id: ObjectId('661c8086dedec9372b1ca7e8'),
  Section_id: 5,
  Textbook_isbn: 'ISBN_5342',
  Course_id: 101,
  Summary: 'This is course 0101 trial class',
  Prerequisite: 'NA',
  Employee_id: 'AA0101_01'
}
{
  _id: ObjectId('661c8086dedec9372b1ca7ed'),
  Section_id: 5,
  Textbook_isbn: 'ISBN_1234',
  Course_id: 202,
  Summary: 'This is course 0202 trial class',
  Prerequisite: 'NA',
  Employee_id: 'AA0202_01'
}
```

4.2.2 A More Complex Query

Calculate the average age in the student collection:

db.student.aggregate([{$group: {_id: null, averageAge: { $avg: "$age1" } }}]);

```
> db.student.aggregate([
    {
      $group: {
        _id: null,
        averageAge: { $avg: "$age1" }
      }
    }
  ]);
< {
    _id: null,
    averageAge: 19.7
```

4.2.3 An Aggregare(or MapReduce)

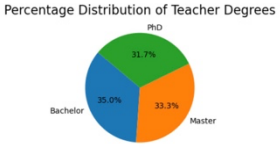
Link the student and enroll tables.

```
db.student.aggregate([
  {
    $lookup: {
      from: "enroll",
      localField: "student_id",
      foreignField: "student_id",
      as: "enrollments"
    }
  }
```

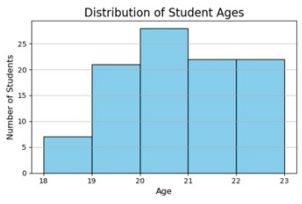
```
{
  _id: ObjectId('661c805ddedec9372b1ca77f'),
  Student_ID: 1001,
  Name: 'Patricia Barnes',
  Phone_Number: 74914347209,
  DOB: 2003-10-01T00:00:00.000Z,
  age1: 20,
  enrollments: []
}
{
  _id: ObjectId('661c805ddedec9372b1ca780'),
  Student_ID: 1002,
  Name: 'Ryan Smith',
  Phone_Number: 51145019215,
  DOB: 2005-12-14T00:00:00.000Z,
  age1: 18,
  enrollments: []
}
```

```
]);
```

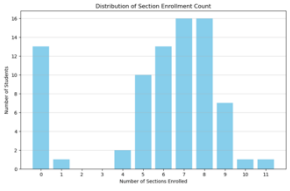
5. Database Access Via Python



Graph1



Graph2



Graph3

Connect Jupyter notebook to MySQL to further explore the data.

6. Improvement in the Future

1. In the current database, I set the number of chapters in the normal class to an attribute type, that is, the total number of chapters in this section. The entire content of each section is contained together. But in practice, this should be a primary key combined with section id and course id, and each chapter is independent.
2. In the current database, there are only 20 guest students, so only 1 salesperson is assigned. So, the cardinality is 1: N. In the future, when the online learning platform gets bigger, we will hire more salespeople. In this case, the cardinality will become N: N.
3. In the current database, there is only one assignment per normal class and thus only one score. But in practice, each course may have many assignments, and finally need to calculate a total score. So, this is also an area that will be improved in the future.