University Database

By: Dasha Rizvanova, Daniel Seamon, Jai Vang

Outline of Presentation

1. Why we choose this topic?
2. Main Use of Cases, Roles, Business Requirements, and Priorities
3. Conceptual Design (E-R model)
4. Logical Designs: View Tables, Stored Procedures, Stored Functions, SQL Queries, and their Outputs
5. Future features

1. Why We Choose This Topic

1. Relevant to all of us since we all are university students
2. We consider this database to be of a high importance and usefulness
3. University database is a database that is easy to update and easy to search

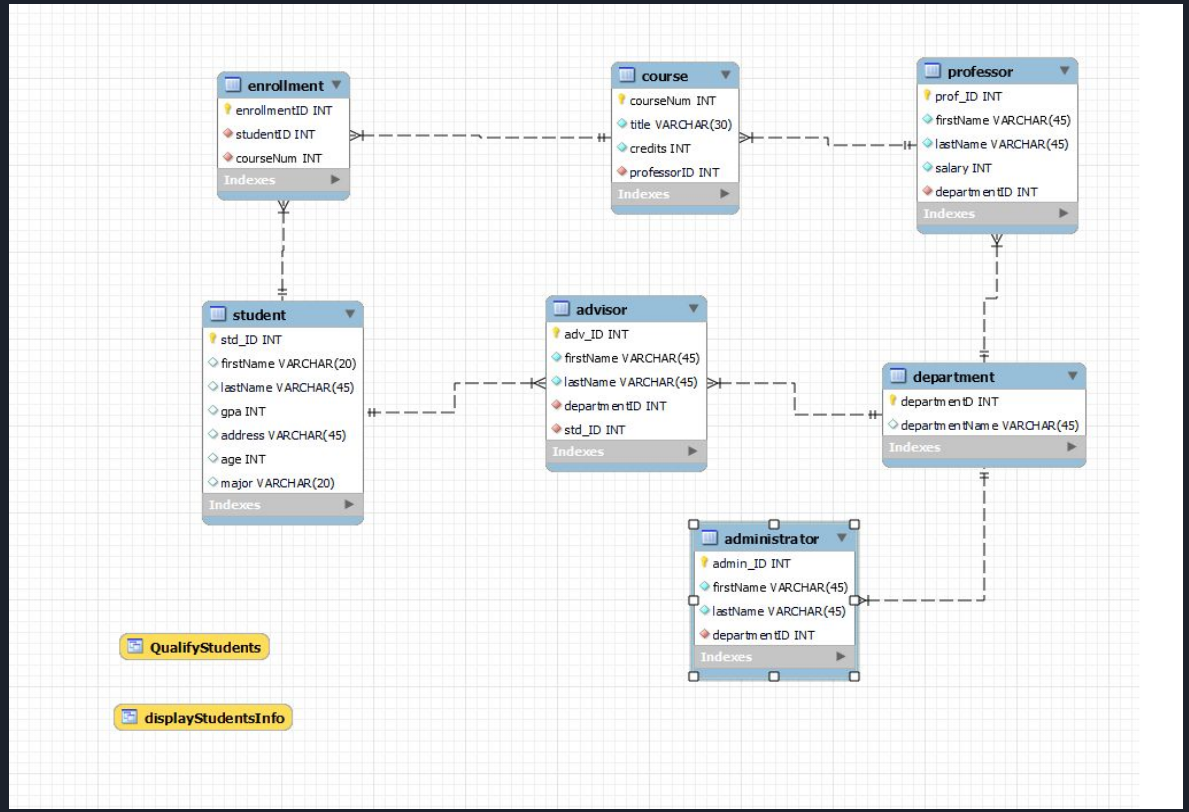
2. Main Use Cases, Roles, Business Requirements, and Priorities

- Our goal was to create a database that organizes information, hence, helps universities keep and manipulate valuable data safely.
- Our project only includes relevant data for a higher institution; no other data is be included.
- Design a schema that will be useful and accessible for a database designer, programmer, administrator and university advisors and and administrators
- Our database is used to store, retrieve, and alter the information held by universities.
- To be able to select students based on their gpa and major
- To be able to update any tables, for example, change the major of a student

Some of our user stories:

- As a Database Designer, I want to navigate to a specific table using a command line, so that I may locate records effectively
- As an Administrator, I want to manage data in the database to keep record reports up to date.
- As an Advisor, I want to be able to have access to a university database, so that I can advise students in a timely manner.

3. Conceptual Design (E-R Model)



4. Logical Designs

7 Tables: Administrator, Advisor, Professor, Student, Department, Course, Enrollment

2 View Tables: QualifyStudent & displayStudentInfo

2 Stored Procedures: getGpa() & getStudentsInfo()

2 Stored Functions: countStudent() & honorRoll()

Administrator Table

Table: Administrator

Columns:

Pk_1: admin_ID

Column_1a admin_ID, Column_1b firstName

Column_1c lastName, Column_1d departmentID

FK_1 Column_1d **REFERENCES** Department Table (DepartmentID)

Normalization Level:

The Administrator Table is in **3NF** because it's already in 2NF and there are no transitive functional dependencies.

Administrator Table

Indexes:

Index #: <type (clustered/non-clustered)>

Clustered and non-clustered

Columns: <ordered list of columns forming the index>

Clustered - admin_ID

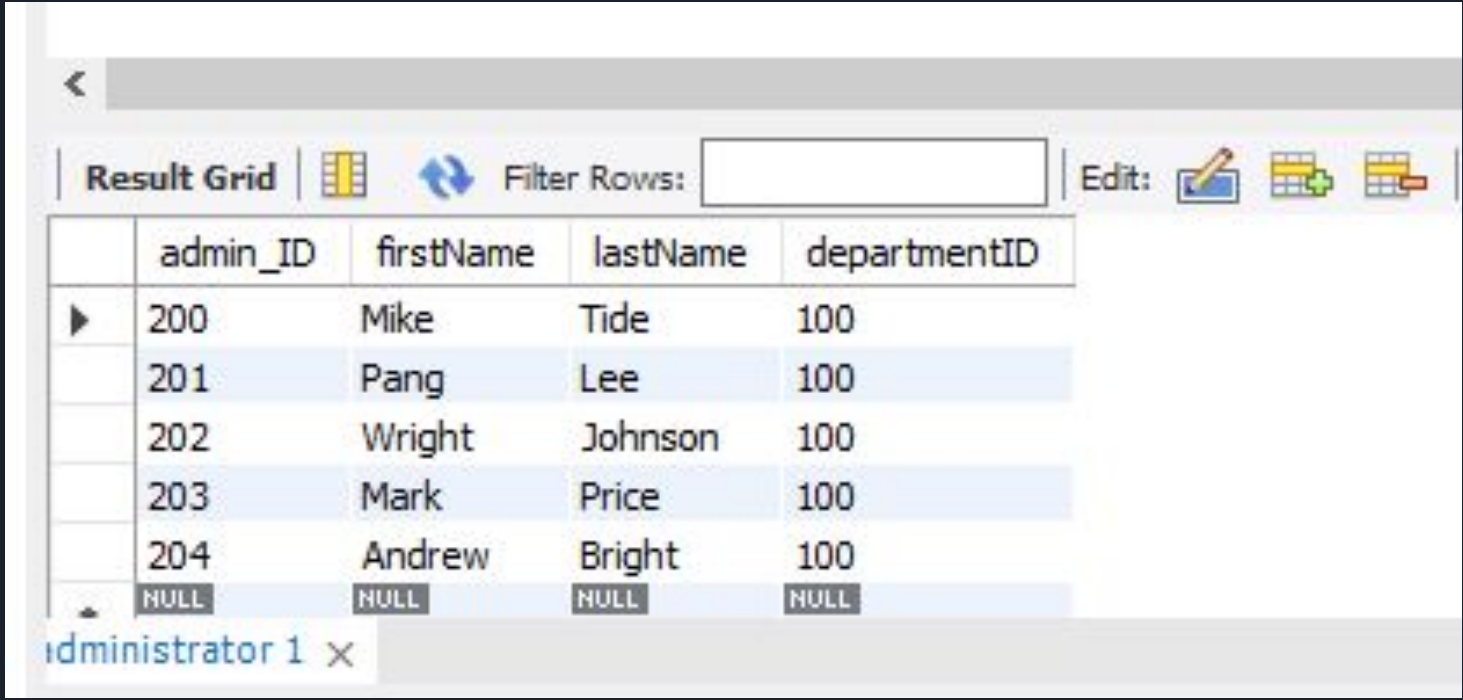
Non-clustered - departmentID

Justification:

By default a clustered index is created on a PK.

Since departmentID is FK; hence, the column is used frequently for queries, we've decided to use a non-clustered index on departmentID.

Administrator Table



Advisor Table

Table: Advisor

Columns:

Pk_2: advis_ID

Column_2a advis_ID, Column_2b firstName, Column_2c lastName

Column_2d departmentID, Column_2e std_ID

FK_2 Column_2d **REFERENCES** Department Table (departmentID)

FK_2 Column_2e **REFERENCES** Student Table (std_ID)

Normalization Level:

The Advisor Table is in 3NF because it's already in 2NF and there are no transitive functional dependencies.

Advisor Table

Indexes:

Index #: <type (clustered/non-clustered)>

Clustered and non-clustered

Columns: <ordered list of columns forming the index>

Clustered - admin_ID

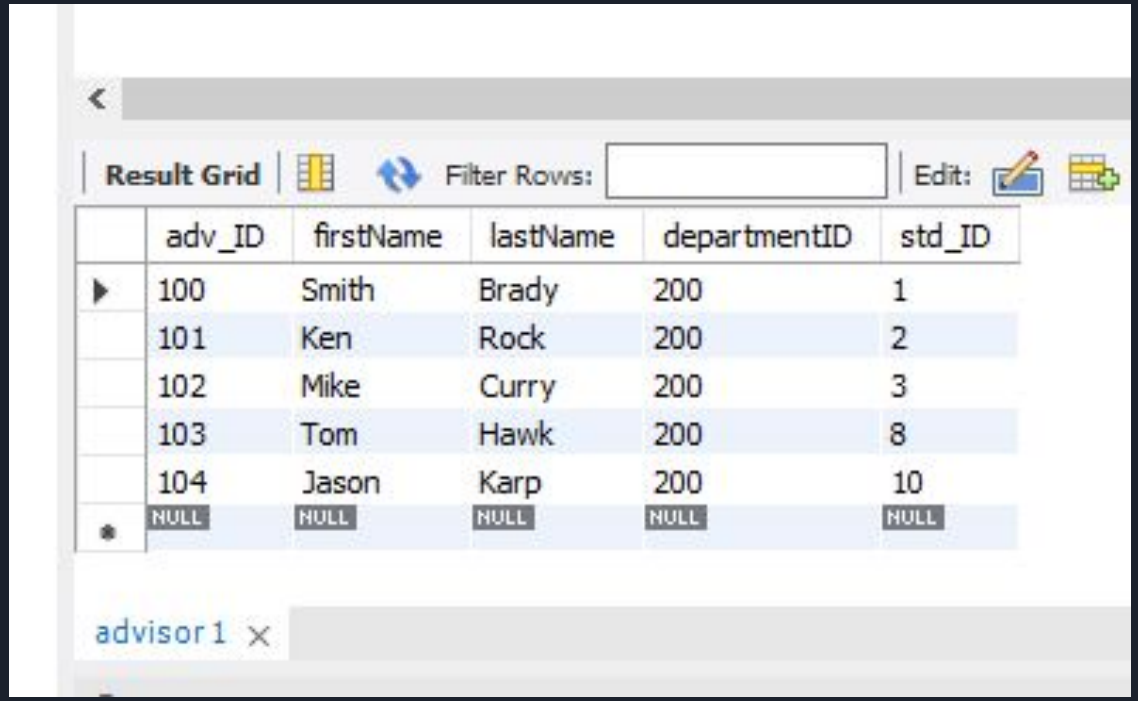
Non-clustered - departmentID

Justification:

By default a clustered index is created on a PK.

Since departmentID is FK; hence, the column is used frequently for queries, we've decided to use a non-clustered index on departmentID.

Advisor Table



adv_ID	firstName	lastName	departmentID	std_ID
100	Smith	Brady	200	1
101	Ken	Rock	200	2
102	Mike	Curry	200	3
103	Tom	Hawk	200	8
104	Jason	Karp	200	10
NULL	NULL	NULL	NULL	NULL

Professor Table

Table: Professor

Columns:

Pk_3 prof_ID

Column_3a prof_ID, Column_3b firstName, Column_3c lastName

Column_3d departmentID , Column_3e salary

FK_3 Column_3d departmentID **REFERENCES** Department Table (departmentID)

Normalization level:

The Professor Table is in 3NF because it’s already in 2NF and there are no transitive functional dependencies.

Professor Table

Indexes:

Index #: <type (clustered/non-clustered)>

Clustered and non-clustered.

Columns: <ordered list of columns forming the index>

Clustered - prof_ID

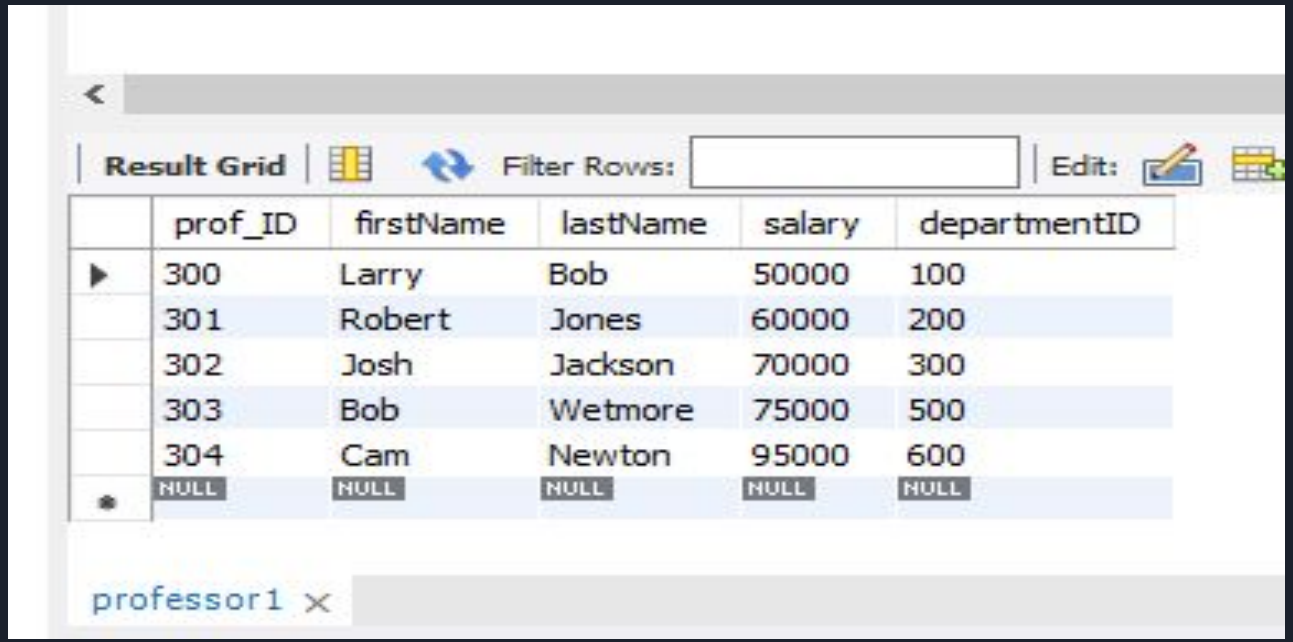
Non-clustered - departmentID

Justification:

PK is indexed by default.

departmentID is a FK; hence, will be used frequently in our queries.

Professor Table



Result Grid | Filter Rows: | Edit:

prof_ID	firstName	lastName	salary	departmentID
300	Larry	Bob	50000	100
301	Robert	Jones	60000	200
302	Josh	Jackson	70000	300
303	Bob	Wetmore	75000	500
304	Cam	Newton	95000	600
NULL	NULL	NULL	NULL	NULL

professor1

Student Table

Table: student

Columns:

Pk_4: std_ID

Column_4a std_ID, Column_4b firstName, Column_4c lastName

Column_4d gpa, Column_4e address, Column_4f age ,Column_4g major

Normalization Level:

The Student Table is in 3NF because it’s already in 2NF and there are no transitive functional dependencies

Student Table

Indexes:

Index #: <type (clustered/non-clustered)>

Clustered and non-clustered

Columns: <ordered list of columns forming the index>

Clustered - std_ID

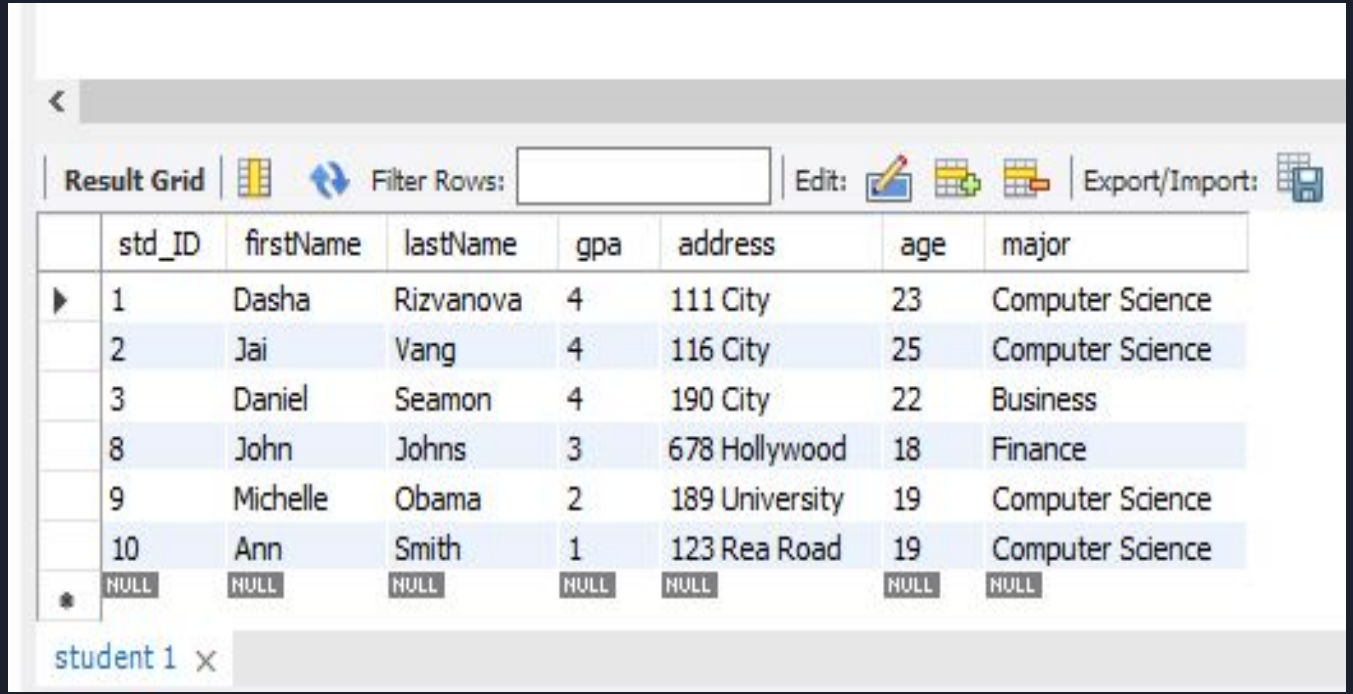
Non-clustered - gpa, major

Justification:

PK is indexed by default.

Gpa and major are one of the most important columns in the Student table, especially for advisors; hence, they will be used often. Therefore,we've decided that gpa and major columns should form a non-clustered index.

Student Table



Result Grid | Filter Rows: | Edit: | Export/Import:

std_ID	firstName	lastName	gpa	address	age	major
1	Dasha	Rizvanova	4	111 City	23	Computer Science
2	Jai	Vang	4	116 City	25	Computer Science
3	Daniel	Seamon	4	190 City	22	Business
8	John	Johns	3	678 Hollywood	18	Finance
9	Michelle	Obama	2	189 University	19	Computer Science
10	Ann	Smith	1	123 Rea Road	19	Computer Science
NULL	NULL	NULL	NULL	NULL	NULL	NULL

student 1

Department Table

Table: Department

Columns:

Pk_5: departmentID

Column_5a departmentID

Column_5b departmentName

Normalization Level:

The Department Table is in 3NF because it’s already in 2NF and there are no transitive functional dependencies.

Department Table

Indexes:

Index #: <type (clustered/non-clustered)>

Clustered

Columns: <ordered list of columns forming the index>

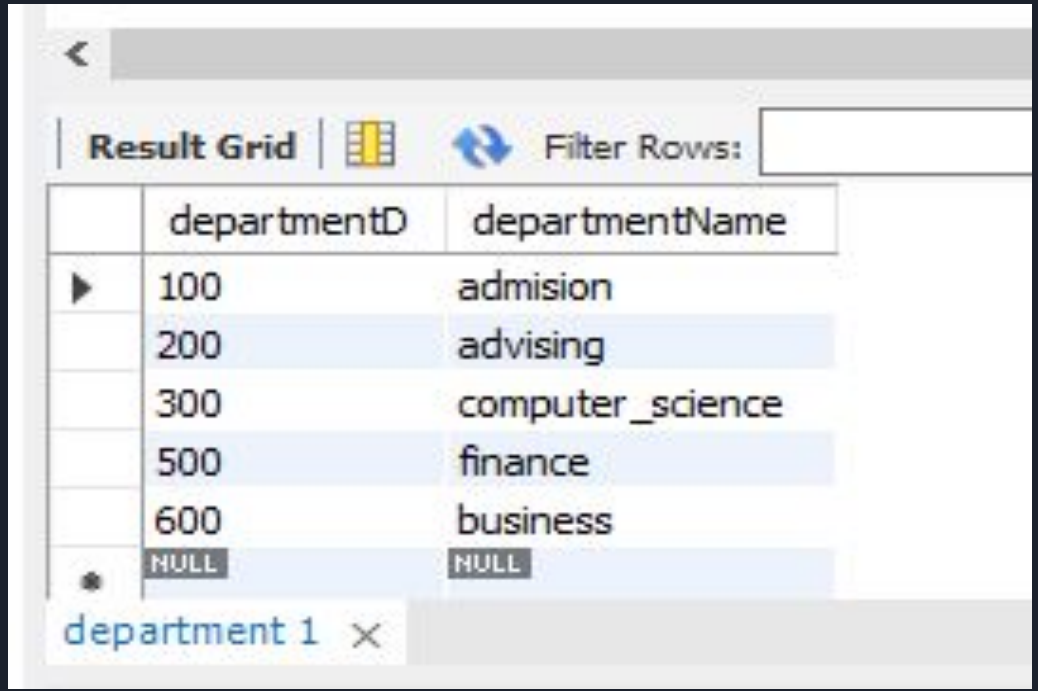
Clustered - departmentID

Justification:

By default, PK forms a clustered index.

Non-clustered index is not needed for this table because we do not have FK, and we only use PK to manipulate the queries.

Department Table



Course Table

Table: Course

Columns:

Pk_6: courseNum

Column_6a courseNum, Column_6b title

Column_6c credits, Column_6d professorID

FK_6 Column_6d professorID **REFERENCES** Professor Table (professortID)

Normalization Level:

Course Table is in 3NF because it’s already in 2NF and there are no transitive functional dependencies.

Course Table

Indexes:

Index #: <type (clustered/non-clustered)>

Clustered and non-clustered

Columns: <order>

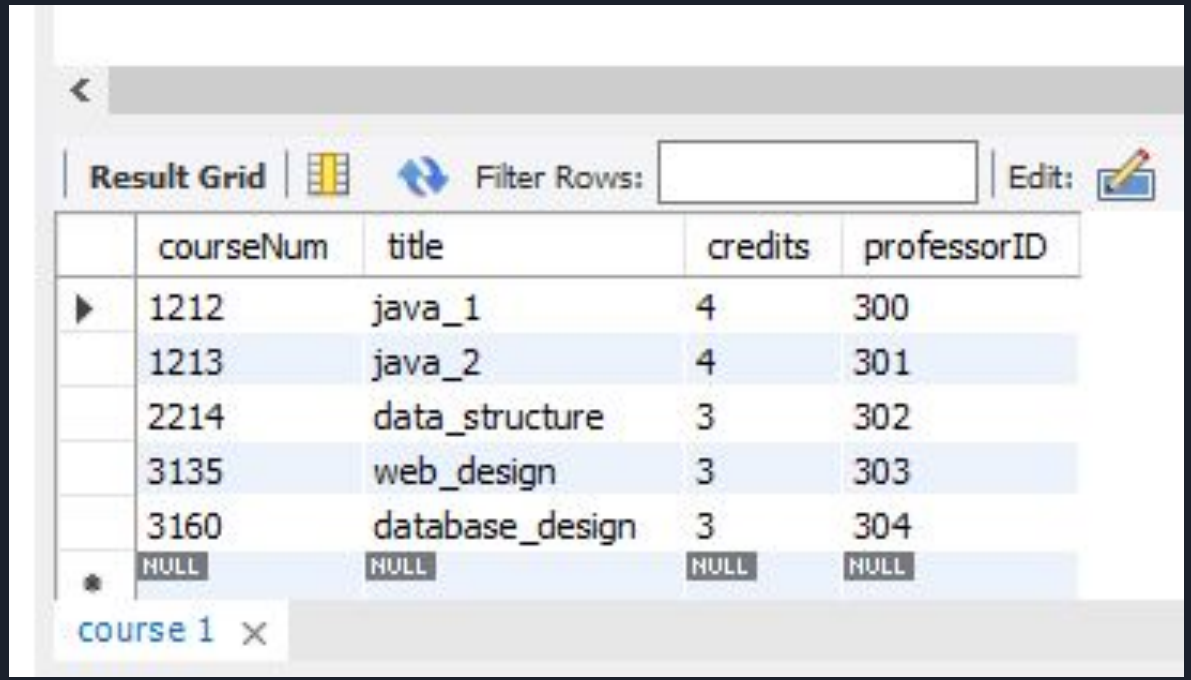
Clustered - courseNum Non-clustered - professorID

Justification:

By default, PK forms a clustered index.

The professorID column will be frequently used in our queries, so we decided to use a non-clustered index to speed up the query performance.

Course Table



Result Grid | Filter Rows: | Edit:

courseNum	title	credits	professorID
1212	java_1	4	300
1213	java_2	4	301
2214	data_structure	3	302
3135	web_design	3	303
3160	database_design	3	304
NULL	NULL	NULL	NULL

course 1

Enrollment Table

Table: Enrollment

Columns:

Pk_7: enrollmentID

Column_7a enrollmentID, Column_7b studentID, Column_7c courseNum

FK_7 Column_7c courseNum **REFERENCES** Course Table (courseNum)

FK_7 Column_7b studentID **REFERENCES** Student Table (std_ID)

Normalization Level:

Enrollment Table is in 3NF because it’s already in 2NF and there are no transitive functional dependencies.

Enrollment Table

Indexes:

Index #: <type (clustered/non-clustered)>

Clustered and non-clustered

Columns: <order>

Clustered- enrollmentID

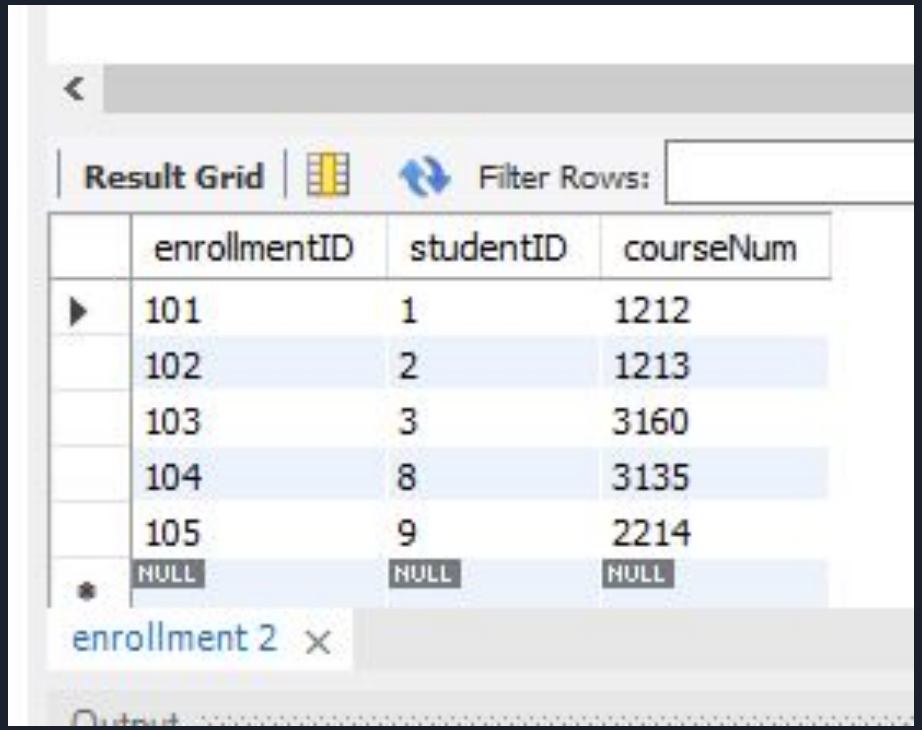
Non-clustered - studentID, courseNum

Justification:

By default, PK forms a clustered index.

studentID and courseNum are FK columns that reference 2 tables; they will be frequently used in our queries. Hence, that is why they both form non-clustered indexes.

Enrollment Table



View Tables

2 View Tables:

-QualifyStudents

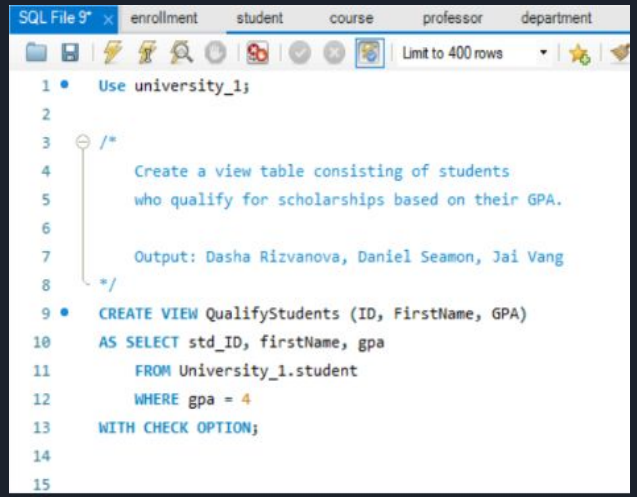
-displayStudentsInfo

View: QualifyStudents

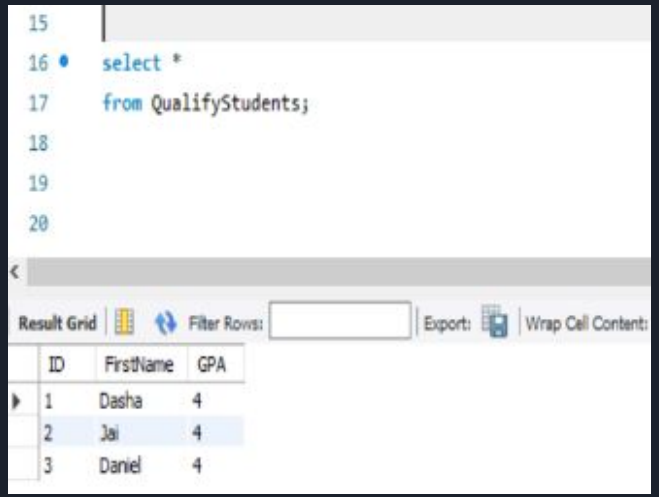
View: QualifyStudents

Goal: We created a view table consisting of students who qualify for scholarships based on their GPA. The administrator will use this view table to see which student qualifies. The table displays student's with an overall GPA of 4.

Create



Output



View: DisplayStudentsInfo

View: displayStudentsInfo

Goal: Create a view table that displays students' information based on their major. Database Designer navigating to a specific table to locate records of students with a computer science major.

Create

```
1   CREATE VIEW displayStudentsInfo(ID, first, last, Sgpa, Smajor)
2   AS SELECT std_ID, firstName, lastName, gpa, major
3       FROM University_1.student
4       WHERE major = "Computer Science";
5
```

Output

```
1   select *
2   from displayStudentsInfo;
```

100% 26:2

Result Grid | Filter Rows: Search | Export:

ID	first	last	Sgpa	Smajor
1	Dasha	Rizvanova	4	Computer Science
2	Jai	Vang	4	Computer Science
9	Michelle	Obama	2	Computer Science
10	Ann	Smith	1	Computer Science

Stored Procedures

2 Stored Procedures:

- getStudentInformation()
- getGpa()

Procedure: getStudentInformtaion()

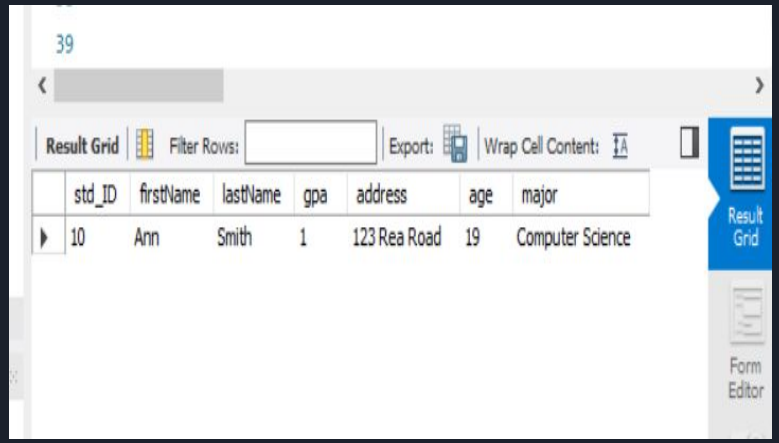
Parameters: IN studentID INT

Goal: This stored procedure displays students' information based on their student ID. It finds student information by passing 10 as a parameter to the stored procedure (Call getStudentsInformation(10);)

Create

```
2
3   /*
4   -- Procedure
5   */
6   DELIMITER //
7   CREATE PROCEDURE getStudentInformation(IN studentID INT)
8   BEGIN
9       SELECT *
10      FROM University_1.student
11      WHERE studentID = std_ID;
12  END //
13  DELIMITER ;
14
```

Output



Procedure: getGpa()

Parameters: IN studGPA

Goal: This procedure displays students' information from the student table based on their GPA. It finds students with gpa of 3 and pass 3 as a parameter to the stored procedure.

Create

```
CREATE DEFINER=`admin`@`%` PROCEDURE `getGPA`(
IN studGPA TINYINT(4) )
BEGIN
    SELECT *
    FROM student
    WHERE gpa = studGPA;
END
```

Output

```
CALL getGpa(3);
```

Result Grid | Filter Rows: | Search | Export:

std_ID	firstName	lastName	gpa	address	age	major
8	John	Johns	3	678 Hollywood	18	Finance

Stored Functions

2 Stored Functions:

- countStudents()
- honorRoll()

Function: countStudents()

Parameters: none

Goal: This stored function calculates the number of students in the Database.

```
/*
-- Stored Function
*/
DELIMITER $$
CREATE FUNCTION countStudents()
RETURNS INTEGER
DETERMINISTIC
BEGIN
	return (SELECT count(*) as Number_Of_Students From University_1.student);
END$$
DELIMITER ;
```

Stored Functions

Parameters: gpa(INT)

Goal: This stored function determines whether the student is eligible for an honor roll. They need to have a GPA 3 or above.

```
DELIMITER //
CREATE FUNCTION honorRoll(gpa INT) RETURNS CHAR(3)
DETERMINISTIC
BEGIN
        DECLARE honor_roll CHAR(3);

   IF gpa >= 3 THEN
                SET honor_roll = 'yes';
        ELSEIF gpa < 3 THEN
                SET honor_roll = 'no';
        END IF;
   RETURN (honor_roll);
END$$
DELIMITER;
```

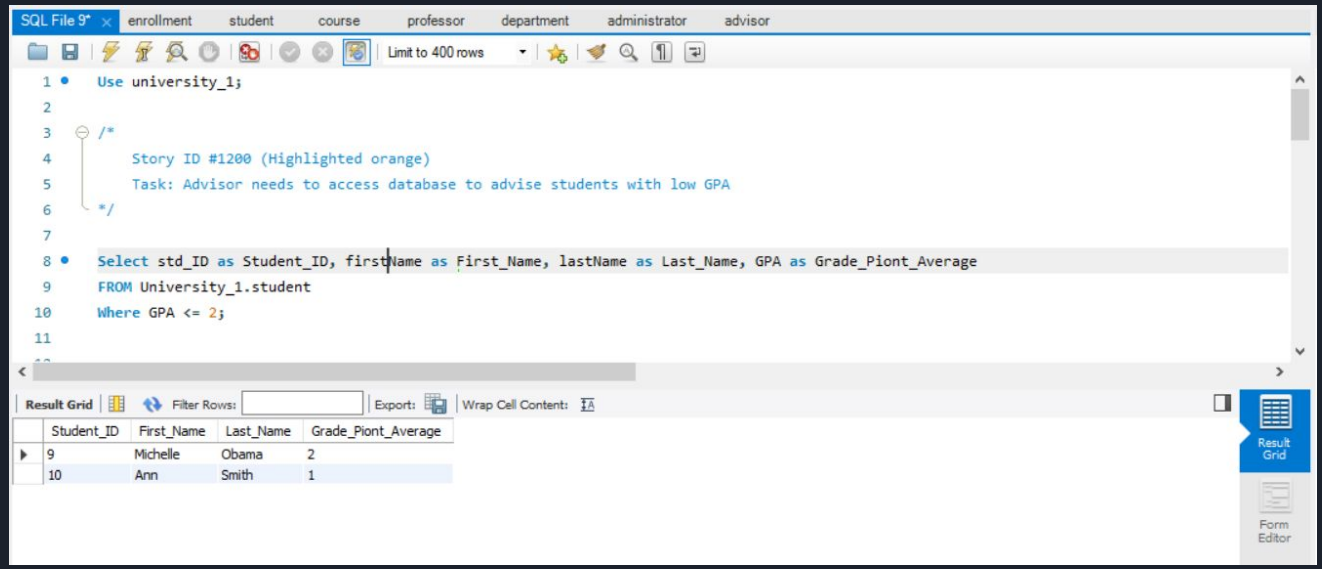
Other SQL Queries & Outputs

These SQL queries perform data retrievals relevant to the features chosen in the current sprint.

Query #1

Story ID: 1200

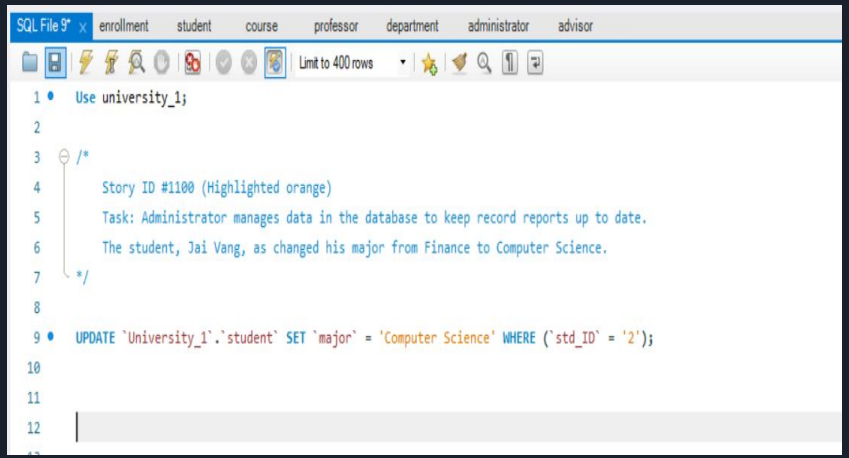
Story Description: Advisor access database to advise students with low GPA



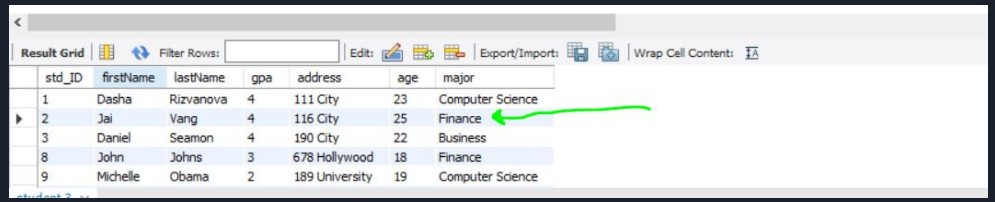
Query #2

Story ID: 1100

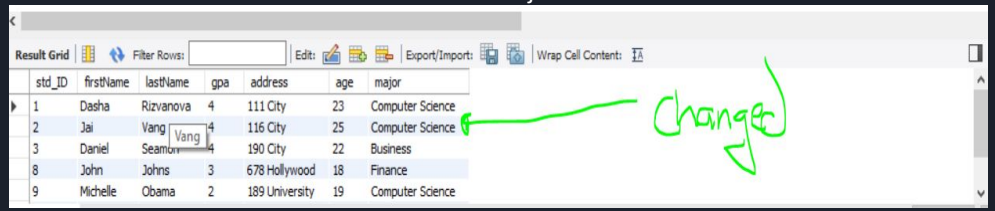
Story Description: Administrator manages data in the database to keep record reports up to date. The student, Jai Vang, has changed his major from Finance to Computer Science.



Previous Major



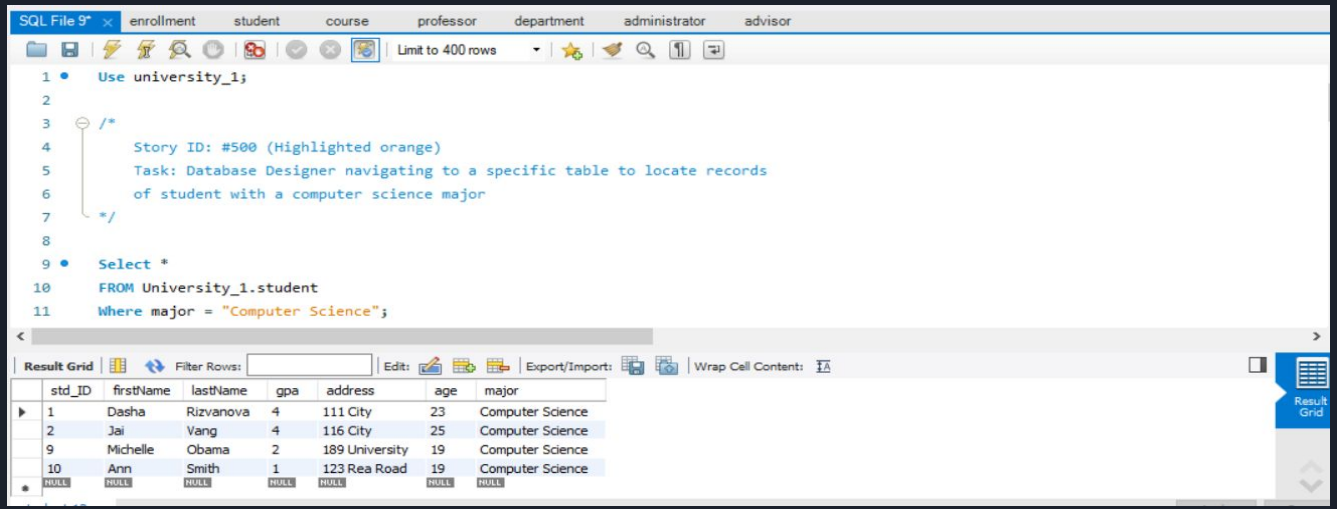
New Major



Query #3

Story ID: 500

Story Description: Database Designer navigating to a specific table to locate records of students with a computer science major.



Future Features

- End product will be a fully functional database that any university can use.
- It will accurately reflect the real world data.
- It will be a valuable asset to any higher education institution.
- Our product will be easy for all users in a university to use from students, advisors, and database administrators.