

**University Database Design and Implementation:
A Case Student In SQL and Database Management System**

Team Members: Jai Vang, Dasha Rizvanova, Daniel Seamon

PROJECT PROPOSAL

Content, Scope and Objectives

Briefly describe the overall context, scope and objectives of the software for which you propose to design and develop a database (*similar to the high level vision statement given to you for the case study we are using for classroom activities, but more concrete in scope*).

Overall context:

We will create a university enrollment database that will store information such as students' first names, last names, dates of birth, contact information, ID number, date of enrollment, expected graduation year, and intended majors. We will also include data such as instructors' and classes' information. Moreover, as we progress through this class, we will be more specific with the data we want to store. Our purpose is to create a database that will organize information, hence, help universities keep and manipulate valuable data safely.

Scope:

Project Description: Our project will be used to store, retrieve, and alter the information held by universities.

Project Resources: Our project resources are our team of highly experienced software programmers.

Project Goals and Deliverables: To store student information securely and enable authorized users to access data promptly.

Project Roadmap: project proposal by 10/07, sprint 1 by 11/04, sprint 2 by 11/18, sprint 3 by 12/02, and finally group video presentation by 12/07.

Project Exclusions: Our project will only include relevant data for a higher institution; no other data will be included.

Objectives:

Our end goal is to create a database all universities can use. One can have multiple purposes for this database- access student or faculty information, and find qualified students with high GPAs eligible for specific grants and scholarships. This database will enable users to find a student's advisor, background information, concentration, enrolled courses, completed courses, failed classes, grade point average, grade level, major, and professors.

Sprint 0

Part 1. PROJECT ENVIRONMENT

MySQL database management system will be our environment setup. The version number for our project is 1.0.

Part 2. HIGH LEVEL REQUIREMENTS

Initial user roles

User Role	Description
Role 1: Administrator	Role 1 description: The administrators are able to access information regarding the students.
Role 2: Advisors	Role 2 description: The advisors are able to access information regarding the students.
Role 3: Database Designer	Role 3 description: The database designer establishes the structure of the database and determines the data to be collected and stored.
Role 4: Database Administrator	Role 4 description: A database administrator should make sure that database is available and secure.

<p>Role 5:</p> <p>Database Programmer</p>	<p>Role 5 description:</p> <p>A database programmer uses query languages and programming languages to develop applications for database users.</p>
---	--

Initial user story descriptions

Story ID	Story description
100	As a Database Programmer, I want to be able to develop macros in the software so that I can efficiently manipulate the database.
200	As a Database Programmer, I want to service and update databases so they are up to date.
300	As a Database Programmer, I want to solve any database bugs, so users can have satisfactory experience using it.
400	As a Database designer, I want to design different database structures, including tables and fields, so the data is organized.
500	As a Database Designer, I want to navigate database software so that I may locate records effectively.

600	As a Database designer, I want to create data models, so I can understand how data is stored in a database and some ways it can be accessed.
700	As a Database designer, I want to be able to search the database so that I can easily find the information that I need.
800	As a Database Designer, I want to be able to review and enhance existing databases, so that I can improve performance.
900	As a Database Administrator, I want to design databases according to end users' goals so they can be satisfied with the services.
1000	As a Database Administrator, I want to secure the database system so that only authorized users can access it.
1100	As a Database Administrator, I want to be able to perform any tests regularly to ensure that the data is available and secure.
1200	As an Administrator, I want to manage data in the database to keep record reports up to date.
1300	As an Administrator, I want to maintain the budget plan so that each department does not spend over their yearly allowance.

1400	As an Advisor, I want to be able to have access to students' information to help evaluate and realize educational and career options.
1500	As an Advisor, I want to be able to schedule an appointment so that I can meet with students.

Part 3. HIGH LEVEL CONCEPTUAL DESIGN

Entities:

Entity 1: Administrators

Entity 2: Advisors

Entity 3: Professors

Entity 4: Students

Entity 5: Departments

Entity 6: Course

Entity 7: Enrollment

Relationships:

Professors works for a Department

Administrators Govern Advisors

Administrators Govern Students

Advisors Advise Students

Professors Teach Students

Students enrolls in a Courses

Professors teach Courses

Students Consult Database

Sprint 1

REQUIREMENTS

List your updated user stories and any notes you wish to include in decreasing order of priority and **highlight the stories chosen for Sprint 1**. *There is no need to show your story refinement process - just the list of updated stories suffices*. Use the format shown below.

Initial user roles

User Role	Description
Role 1: Administrator	Role 1 description: The administrators are able to access information regarding the students.
Role 2: Advisors	Role 2 description: The advisors are able to access information regarding the students.
Role 3: Database Designer	Role 3 description: The database designer establishes the structure of the database and determines the data to be collected and stored.

<p>Role 4:</p> <p>Database Administrator</p>	<p>Role 4 description:</p> <p>A database administrator should make sure that database is available and secure.</p>
<p>Role 5:</p> <p>Database Programmer</p>	<p>Role 5 description:</p> <p>A database programmer uses query languages and programming languages to develop applications for database users.</p>

Initial user story descriptions

Story ID	Story description
100	As a Database Programmer, I want to be able to develop macros in the software so that I can efficiently manipulate the database.
200	As a Database Programmer, I want to service and update databases so they are up to date.
300	As a Database Programmer, I want to solve any database bugs, so users can have satisfactory experience using it.
400	As a Database designer, I want to design different database structures, including tables and fields, so the data is organized.

500	As a Database Designer, I want to navigate database software so that I may locate records effectively.
600	As a Database designer, I want to create data models, so I can understand how data is stored in a database and some ways it can be accessed.
700	As a Database designer, I want to be able to search the database so that I can easily find the information that I need.
800	As a Database Designer, I want to be able to review and enhance existing databases, so that I can improve performance.
900	As a Database Administrator, I want to design databases according to end users' goals so they can be satisfied with the services.
1000	As a Database Administrator, I want to secure the database system so that only authorized users can access it.
1100	As a Database Administrator, I want to be able to perform any tests regularly to ensure that the data is available and secure.
1200	As an Administrator, I want to manage data in the database to keep record reports up to date.
1300	As an Administrator, I want to maintain the budget plan so that each department does not spend over their yearly allowance.

1400	As an Advisor, I want to be able to have access to students' information, such as GPA and studentID to help evaluate and realize educational and career options.
1500	As an Advisor, I want to be able to schedule an appointment so that I can meet with students.

CONCEPTUAL DESIGN

Include your detailed conceptual design here. Use the format shown below.

Entity: Administrator

Attributes: admin_ID, firstName, lastName, departmentID

PK admin_ID[simple]

firstName[simple]

lastName[simple]

departmentID[simple]

Entity: Advisor

Attributes: adv_D, firstName, lastName, departmentID, studentID

PK adv_ID[simple]

firstName[simple]

lastName[simple]

departmentID[simple]

studentID[simple]

Entity: Professor

Attributes: prof_ID, firstName, lastName, salary, departmentID

PK prof_ID[simple],

firstName[simple]

lastName[simple]

salary[simple]

departmentID[simple]

Entity: Student

Attributes: studentID, firstName, lastName, gpa, address, age, major,

PK studentID[simple]

firstName[simple]

lastName[simple]

gpa[simple]

address[simple]

age[simple]

major[simple]

Entity: Department

Attributes: departmentID, departmentName

PK departmentID[simple]

departmentName[simple]

Entity: Course

Attributes: courseNum, title, credits, professorID

PK courseNum[simple],

title[simple],

credits[simple],

professorID[simple]

Entity: Enrollment

Attributes: enrollmentID, studentID, courseNum

PK ID [simple],

studentID[simple],

courseID[simple]

Relationship: **Professor** works for a **Department**

Cardinality: <Many> to <One>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Administrator** governs **Advisor**

Cardinality: <Many> to <Many>

Participation:

Entity1 has <Partial> participation

Entity2 has <Partial> participation

Relationship: **Administrator** governs **Student**

Cardinality: <Many> to <Many>

Participation:

Entity1 has <Partial> participation

Entity2 has <Partial> participation

Relationship: **Advisor** advises **Student**

Cardinality: <One> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Professor** teaches **Student**

Cardinality: <One> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Total> participation

Relationship: **Student** consults **The Database**

Cardinality: <One> to <One>

Participation:

Entity1 has <Partial> participation

Entity2 has <Partial> participation

Relationship: **Student** enrolls in a **Course**

Cardinality: <Many> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Professor** teaches **Course**

Cardinality: <One> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

LOGICAL DESIGN

Include your logical design here. Use the format shown below

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)

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)

Table: **Professor**

Columns:

Pk_3 prof_ID

Column_3a professorID

Column_3b firstName

Column_3c lastName

Column_3d departmentID

Column_3e salary

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

Table: **Student**

Columns:

Pk_4: studentID

Column_4a std_ID

Column_4b firstName

Column_4c lastName

Column_4d gpa

Column_4e address

Column_4f age

Column_4g major

Table: **Department**

Columns:

Pk_5 departmentID

Column_5a departmentID

Column_5b departmentName

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 (professorID)

Table: **Enrollment**

Columns:

Pk_7 enrollmentID

Column_7a enrollmentID

Column_7b studentID

Column_7c courseNum

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

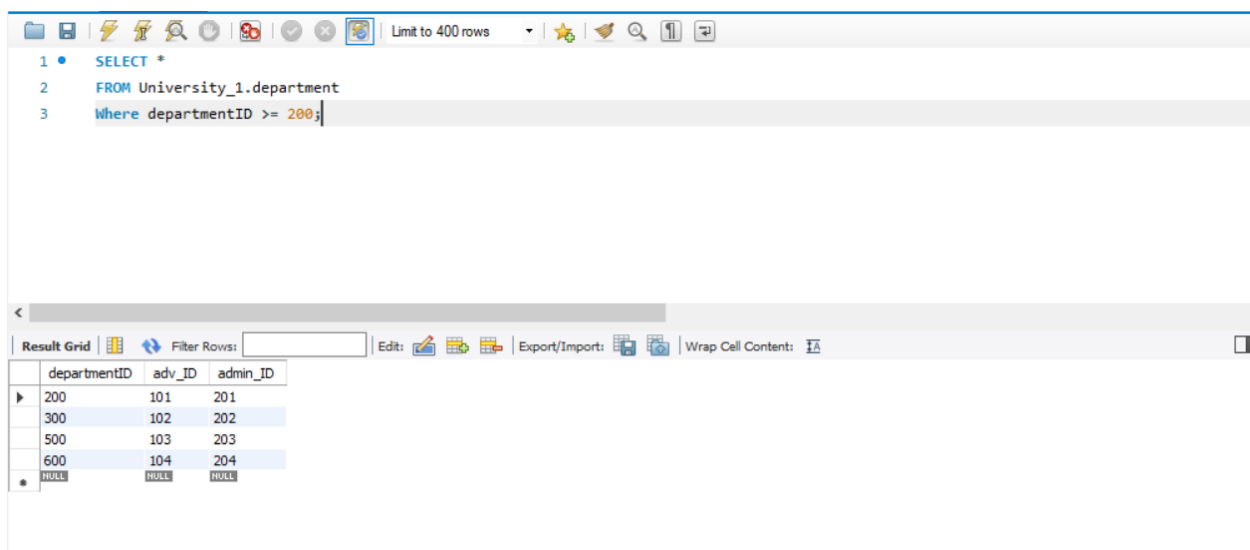
FK_7 Column_7b studentID **REFERENCES** Student Table (std_ID)

Query #2:

Story ID: 1200

Story Description: As an Administrator, I want to manage data in the database to keep record reports up to date.

Administrator tries to access information for department that have ID # 200 and higher



The screenshot shows a database query editor interface. The top toolbar includes icons for file operations, query execution, and a 'Limit to 400 rows' dropdown. The SQL editor contains the following query:

```

1 • SELECT *
2 FROM University_1.department
3 Where departmentID >= 200;

```

Below the query editor, the 'Result Grid' is displayed with a table of results. The table has three columns: departmentID, adv_ID, and admin_ID. The results show five rows of data, with the first four rows having values and the fifth row being NULL. A filter bar above the table shows 'Filter Rows:' with an empty input field. The table interface includes buttons for 'Edit', 'Export/Import', and 'Wrap Cell Content'.

departmentID	adv_ID	admin_ID
200	101	201
300	102	202
500	103	203
600	104	204
NULL	NULL	NULL

Query #3

Story ID:1400

Story Description: As an Advisor, I want to be able to have access to students' information, such as GPA and studentID to help evaluate and realize educational and career options.

Advisor tries to access information of students with a high GPA

Sprint 2

REQUIREMENTS

List your updated user stories in decreasing order of priority. Highlight the stories for which database design was completed in Sprint 1 in one color. Highlight the updated/new stories chosen for Sprint 2 in a different color. *There is no need to explicitly show your story refinement process.* Use the format shown below.

Initial user story descriptions

Story ID	Story description
100	As a Database Programmer, I want to be able to develop macros in the software so that I can efficiently manipulate the database.
200	As a Database Programmer, I want to service and update databases so they are up to date.
300	As a Database Programmer, I want to be able to use a debugger, so that I can solve any errors in my database quickly.
400	As a Database Designer, I want to design different database structures, including tables and fields, so the data is organized.
500	As a Database Designer, I want to navigate to a specific table using a command line, so that I may locate records effectively.

600	As a Database Designer, I want to use a reverse engineer feature, so I can understand how tables are connected in my database
700	As a Database Designer, I want to be able to archive, enhance, and review existing databases, so that I can improve performance.
800	As a Database Administrator, I want to design, modify, and test databases according to users' end goals, so they can be satisfied with the services.
900	As a Database Administrator, I want to use strong authentication and implement security measures, so that only authorized users can access it.
1000	As a Database Administrator, I want to perform white box testing, so that I can ensure that the data is available.
1100	As an Administrator, I want to manage data in the database to keep record reports up to date.
1200	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.

CONCEPTUAL DESIGN

Include your **complete updated conceptual design** here. Use the format shown below.

Entity: Administrator

Attributes: admin_ID, firstName, lastName, departmentID

PK admin_ID[simple]

firstName[simple]

lastName[simple]

departmentID[simple]

Entity: Advisor

Attributes: adv_ID, firstName, lastName, departmentID, studentID

PK adv_ID[simple]

firstName[simple]

lastName[simple]

departmentID[simple]

studentID[simple]

Entity: Professor

Attributes: prof_ID, firstName, lastName, salary, departmentID

PK prof_ID[simple],
firstName[simple]
lastName[simple]
salary[simple]
departmentID[simple]

Entity: Student

Attributes: studentID, firstName, lastName, gpa, address, age, major,

PK studentID[simple]
firstName[simple]
lastName[simple]
gpa[simple]
address[simple]
age[simple]
major[simple]

Entity: Department

Attributes: departmentID, departmentName

PK departmentID[simple]
departmentName[simple]

Entity: Course

Attributes: courseNum, title, credits, professorID

PK courseNum[simple],

title[simple],

credits[simple],

professorID[simple]

Entity: Enrollment

Attributes: enrollmentID, studentID, courseNum

PK ID [simple],

studentID[simple],

courseID[simple]

Relationship:

Relationship: **Professor** works for a **Department**

Cardinality: <Many> to <One>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Administrator** works for a **Department**

Cardinality: <Many> to <One>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Advisor** works for a **Department**

Cardinality: <Many> to <One>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Advisor** advises **Student**

Cardinality: <One> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Professor** teaches **Student**

Cardinality: <One> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Total> participation

Relationship: **Student** enrolls in a **Course**

Cardinality: <Many> to<Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Professor** teaches **Course**

Cardinality: <One> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

LOGICAL DESIGN WITH NORMAL FORM IDENTIFICATION

Include your **complete updated logical design** here. Use the format shown below.

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)

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)

Table: **Professor**

Columns:

Pk_3 prof_ID

Column_3a professorID

Column_3b firstName

Column_3c lastName

Column_3d departmentID

Column_3e salary

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

Table: **Student**

Columns:

Pk_4: studentID

Column_4a std_ID

Column_4b firstName

Column_4c lastName

Column_4d gpa

Column_4e address

Column_4f age

Column_4g major

Table: **Department**

Columns:

Pk_5 departmentID

Column_5a departmentID

Column_5b departmentName

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 (professorID)

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)

Highest normalization level: <1NF/2NF/3NF/4NF>

Justification (if below 4NF):

Our database is in 1NF because all attributes in tables are atomic.

Our database is in 2NF because the database is in 1NF and there are no partial key dependencies.

Our database is in 3NF because it's already in 2NF and there are no transitive functional dependencies.

SQL QUERIES

List at least **three** SQL queries that perform data retrievals relevant to the features chosen in the current sprint. For each query, paste a **screenshot** of the output, as shown through your user interface.

Query #1:

Story ID #1200

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

SQL File 9* x enrollment student course professor department administrator advisor

Limit to 400 rows

```

1 • Use university_1;
2
3 /*
4   Story ID #1200 (Highlighted orange)
5   Task: Advisor needs to access database to advise students with low GPA
6 */
7
8 • Select std_ID as Student_ID, firstName as First_Name, lastName as Last_Name, GPA as Grade_Piont_Average
9   FROM University_1.student
10  Where GPA <= 2;
11
12

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

Student_ID	First_Name	Last_Name	Grade_Piont_Average
9	Michelle	Obama	2
10	Ann	Smith	1

Result Grid
Form Editor

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.

SQL File 9* x enrollment student course professor department administrator advisor

Limit to 400 rows

```

1 • Use university_1;
2
3 /*
4   Story ID #1100 (Highlighted orange)
5   Task: Administrator manages data in the database to keep record reports up to date.
6   The student, Jai Vang, as changed his major from Finance to Computer Science.
7 */
8
9 • UPDATE `University_1`.`student` SET `major` = 'Computer Science' WHERE (`std_ID` = '2');
10
11
12

```

Jai's previous major:

Result Grid						
Filter Rows:						
	std_ID	firstName	lastName	gpa	address	major
1	Dasha	Rizvanova	4	111 City	23	Computer Science
2	Jai	Vang	4	116 City	25	Finance
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

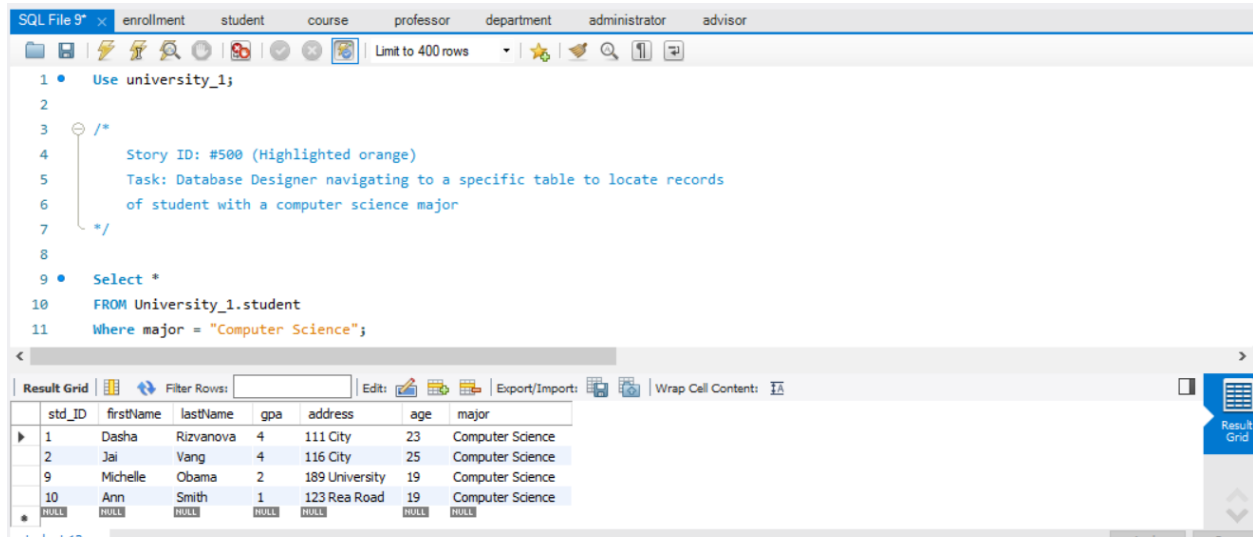
Jai's new major:

Result Grid						
Filter Rows:						
	std_ID	firstName	lastName	gpa	address	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

Query #3:

Story ID: #500 (Highlighted orange)

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



Sprint 3

REQUIREMENTS

List your updated user stories in decreasing order of priority. Highlight the stories that were completed in Sprint 1 in one color. Highlight the stories that were completed in Sprint 2 in a different color. Highlight the updated/new stories chosen for Sprint 3, if any, in a third color. *There is no need to explicitly show your story refinement process.* Use the format shown below.

Story ID	Story description
----------	-------------------

100	As a Database Programmer, I want to be able to develop macros in the software so that I can efficiently manipulate the database.
200	As a Database Programmer, I want to service and update databases so they are up to date.
300	As a Database Programmer, I want to be able to use a debugger, so that I can solve any errors in my database quickly.
400	As a Database Designer, I want to design different database structures, including tables and fields, so the data is organized.
500	As a Database Designer, I want to navigate to a specific table using a command line, so that I may locate records effectively.
600	As a Database Designer, I want to use a reverse engineer feature, so I can understand how tables are connected in my database
700	As a Database Designer, I want to be able to archive, enhance, and review existing databases, so that I can improve performance.
800	As a Database Administrator, I want to design, modify, and test databases according to users' end goals, so they can be satisfied with the services.

900	As a Database Administrator, I want to use strong authentication and implement security measures, so that only authorized users can access it.
1000	As a Database Administrator, I want to perform white box testing, so that I can ensure that the data is available.
1100	As an Administrator, I want to manage data in the database to keep record reports up to date.
1200	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.

CONCEPTUAL DESIGN

Include your **complete updated conceptual design** here. Use the format shown below.

Entity: Administrator

Attributes: admin_ID, firstName, lastName, departmentID

PK admin_ID[simple]

firstName[simple]

lastName[simple]

departmentID[simple]

Entity: Advisor

Attributes: adv_D, firstName, lastName, departmentID, studentID

PK adv_ID[simple]

firstName[simple]

lastName[simple]

departmentID[simple]

studentID[simple]

Entity: Professor

Attributes: prof_ID, firstName, lastName, salary, departmentID

PK prof_ID[simple],

firstName[simple]

lastName[simple]

salary[simple]

departmentID[simple]

Entity: Student

Attributes: studentID, firstName, lastName, gpa, address, age, major,

PK studentID[simple]

firstName[simple]

lastName[simple]

gpa[simple]

address[simple]

age[simple]

major[simple]

Entity: Department

Attributes: departmentID, departmentName

PK departmentID[simple]

departmentName[simple]

Entity: Course

Attributes: courseNum, title, credits, professorID

PK courseNum[simple],

title[simple],

credits[simple],

professorID[simple]

Entity: Enrollment

Attributes: enrollmentID, studentID, courseNum

PK ID [simple],

studentID[simple],

courseID[simple]

Relationship:

Relationship: **Professor** works for a **Department**

Cardinality: <Many> to <One>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Administrator** works for a **Department**

Cardinality: <Many> to <One>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Advisor** works for a **Department**

Cardinality: <Many> to <One>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Advisor** advises **Student**

Cardinality: <One> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Professor** teaches **Student**

Cardinality: <One> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Total> participation

Relationship: **Student** enrolls in a **Course**

Cardinality: <Many> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

Relationship: **Professor** teaches **Course**

Cardinality: <One> to <Many>

Participation:

Entity1 has <Total> participation

Entity2 has <Partial> participation

LOGICAL DESIGN WITH HIGHEST NORMAL FORMS AND INDEXES

Include your **complete updated logical design** here. Use the format shown below.

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)

Highest normalization level: <1NF/2NF/3NF/4NF>

Justification (if below 4NF):

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

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.

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)

Highest normalization level: <1NF/2NF/3NF/4NF>

Justification (if below 4NF):

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

Indexes:

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

Clustered and non-clustered

Columns: <ordered list of columns forming the index>

Clustered - advis_ID

Non-clustered - std_ID, departmentID

Justification:

PK is indexed by default.

Std_ID and departmentID are both FK; hence, will be used frequently in our queries.

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)

Highest normalization level: <1NF/2NF/3NF/4NF>

Justification (if below 4NF):

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

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.

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

Highest normalization level: <1NF/2NF/3NF/4NF>

Justification (if below 4NF):

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

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.

Table: **Department**

Columns:

Pk 5 departmentID

Column_5a departmentID

Column_5b departmentName

Highest normalization level: <1NF/2NF/3NF/4NF>

Justification (if below 4NF):

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

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.

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 (professorID)

Highest normalization level: <1NF/2NF/3NF/4NF>

Justification (if below 4NF):

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

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.

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)

Highest normalization level: <1NF/2NF/3NF/4NF>

Justification (if below 4NF):

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

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.

VIEWS AND STORED PROGRAMS

List the views relevant to your application here. Use the format specified below.

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.

The screenshot shows the SQL Developer interface with a SQL script editor and a results grid.

SQL Script:

```

1 • Use university_1;
2
3 /*
4  Create a view table consisting of students
5  who qualify for scholarships based on their GPA.
6
7  Output: Dasha Rizvanova, Daniel Seamon, Jai Vang
8 */
9 • CREATE VIEW QualifyStudents (ID, FirstName, GPA)
10 AS SELECT std_ID, firstName, gpa
11 FROM University_1.student
12 WHERE gpa = 4
13 WITH CHECK OPTION;
14
15
16 • select *
17 from QualifyStudents;
18
19
20

```

Results Grid:

ID	FirstName	GPA
1	Dasha	4
2	Jai	4
3	Daniel	4

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.

[illegible]

List the stored programs relevant to your application thus far here. Use the format specified below for the different kinds of stored programs. **Note: if you do not have a particular type of stored program in your application, just leave that part out.**

Stored procedure: getGpa()

Parameters: IN studGPA

Goal: This procedure displays students' information from the student table based on their GPA.

```
DELIMITER //

CREATE PROCEDURE getGPA(
    IN studGPA TINYINT(4) )

BEGIN

    SELECT *

    FROM student

    WHERE gpa = studGPA;

END//

DELIMITER ;
```

```
1 • ⊖ CREATE DEFINER='admin'@'%' PROCEDURE `getGPA` (
2   ⊖ IN studGPA TINYINT(4) )
3   ⊖ BEGIN
4     SELECT *
5     FROM student
6     WHERE gpa = studGPA;
7   END
```

//To find students with gpa 3, pass 3 as a parameter to the stored procedure

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

1 • CALL getGpa(3);

100% 17:1

Result Grid Filter Rows: Search Export:

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

Stored procedure: getStudentsInfo()

Parameters: IN studentID INT

Goal: This stored procedure displays students' information based on their student ID.

DELIMITER //

CREATE PROCEDURE getStudentInformation(IN studentID INT)

BEGIN

SELECT *

FROM University_1.student

WHERE studentID = std_ID;

END //

DELIMITER ;

```

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

```

// To find student information by passing 10 as a parameter to the stored procedure

Call getStudentInformation(10);

39

Result Grid

Filter Rows: Export: Wrap Cell Content:

	std_ID	firstName	lastName	gpa	address	age	major
▶	10	Ann	Smith	1	123 Rea Road	19	Computer Science

Result Grid

Form Editor

Stored function: countStudents()

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 function: honorRoll()

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;
```

Trigger: <type of trigger> on <table name> (NOT APPLICABLE)

Goal: <1-2 sentence description of what the trigger does>

Event: <type of event> (NOT APPLICABLE)

Goal: <1-2 sentence description of what the event does>