

Database Systems

Department of Computer Engineering

Spring 2021

Design Report

Online Course Platform - Sapientia

Project Group 4:

* Ömer Ünlüsoy - Section 01 - 21702136
* Elif Gamze Güliter - Section 01 - 21802870
* İrem Tekin - Section 01 - 21803267
* Ece Ünal - Section 01 - 21703149

Supervisor:

* Uğur Güdükbay

Table of Contents

[Revised E/R Diagram 5](#_Toc68296405)

[Relation Schemas and SQL 6](#_Toc68296406)

[Notes 6](#_Toc68296407)

[1. Student 6](#_Toc68296408)

[Relational Model: 6](#_Toc68296409)

[SQL Definition: 6](#_Toc68296410)

[2. Instructor 6](#_Toc68296411)

[Relational Model: 6](#_Toc68296412)

[SQL Definition: 7](#_Toc68296413)

[3. Admin 7](#_Toc68296414)

[Relational Model: 7](#_Toc68296415)

[SQL Definition: 7](#_Toc68296416)

[4. Course 7](#_Toc68296417)

[Relational Model: 7](#_Toc68296418)

[SQL Definition: 7](#_Toc68296419)

[5. Section 8](#_Toc68296420)

[Relational Model: 8](#_Toc68296421)

[SQL Definition: 8](#_Toc68296422)

[6. Lecture 8](#_Toc68296423)

[Relational Model: 8](#_Toc68296424)

[SQL Definition: 8](#_Toc68296425)

[7. Quiz 9](#_Toc68296426)

[Relational Model: 9](#_Toc68296427)

[SQL Definition: 9](#_Toc68296428)

[8. Quiz\_Question 9](#_Toc68296429)

[Relational Model: 9](#_Toc68296430)

[SQL Definition: 10](#_Toc68296431)

[9. Note 10](#_Toc68296432)

[Relational Model: 10](#_Toc68296433)

[SQL Definition: 10](#_Toc68296434)

[10. Course\_Announcement 10](#_Toc68296435)

[Relational Model: 10](#_Toc68296436)

[SQL Definition: 11](#_Toc68296437)

[11. QnA\_Entry\_Student 11](#_Toc68296438)

[Relational Model: 11](#_Toc68296439)

[SQL Definition: 11](#_Toc68296440)

[12. QnA\_Entry\_Instructor 11](#_Toc68296441)

[Relational Model: 11](#_Toc68296442)

[SQL Definition: 11](#_Toc68296443)

[13. Complaint\_Entry\_Student 12](#_Toc68296444)

[Relational Model: 12](#_Toc68296445)

[SQL Definition: 12](#_Toc68296446)

[14. Complaint\_Entry\_Instructor 12](#_Toc68296447)

[Relational Model: 12](#_Toc68296448)

[SQL Definition: 12](#_Toc68296449)

[15. Wishlist 12](#_Toc68296450)

[Relational Model: 12](#_Toc68296451)

[SQL Definition: 13](#_Toc68296452)

[16. Enrolls 13](#_Toc68296453)

[Relational Model: 13](#_Toc68296454)

[SQL Definition: 13](#_Toc68296455)

[17. Certificate 13](#_Toc68296456)

[Relational Model: 13](#_Toc68296457)

[SQL Definition: 13](#_Toc68296458)

[18. Teaches 14](#_Toc68296459)

[Relational Model: 14](#_Toc68296460)

[SQL Definition: 14](#_Toc68296461)

[19. Evaluates\_Complaint\_Entry\_Student 14](#_Toc68296462)

[Relational Model: 14](#_Toc68296463)

[SQL Definition: 14](#_Toc68296464)

[20. Evaluates\_Complaint\_Entry\_Instructor 15](#_Toc68296465)

[Relational Model: 15](#_Toc68296466)

[SQL Definition: 15](#_Toc68296467)

[21. Discount 15](#_Toc68296468)

[Relational Model: 15](#_Toc68296469)

[SQL Definition: 15](#_Toc68296470)

[22. Zoom\_Session 16](#_Toc68296471)

[Relational Model: 16](#_Toc68296472)

[SQL Definition: 16](#_Toc68296473)

[23. Take\_Quiz 16](#_Toc68296474)

[Relational Model: 16](#_Toc68296475)

[SQL Definition: 16](#_Toc68296476)

[24. Take\_Quiz\_Question 17](#_Toc68296477)

[Relational Model: 17](#_Toc68296478)

[SQL Definition: 17](#_Toc68296479)

[25. Take\_Lecture 17](#_Toc68296480)

[Relational Model: 17](#_Toc68296481)

[SQL Definition: 17](#_Toc68296482)

[26. Evaluations 18](#_Toc68296483)

[Relational Model: 18](#_Toc68296484)

[SQL Definition: 18](#_Toc68296485)

[User Interface Design and Corresponding SQL Statements 19](#_Toc68296486)

[1. Login 19](#_Toc68296487)

# Revised E/R Diagram

# Relation Schemas and SQL

## Notes

General Assumptions:

ID -> INT AUTO\_INCREMENT

referencing ID -> INT NOT NULL

short string -> VARCHAR(30)

mid string -> VARCHAR(50)

long string -> VARCHAR(70)

description -> VARCHAR(300)

money -> NUMERIC(12, 2) OR NUMERIC(8, 2)

% -> NUMERIC(5, 2)

date -> DATE

URL -> VARCHAR(1024)

Special Cases:

membership\_type -> ENUM('BRZ', 'SLV', 'GLD')

refund\_request -> BOOLEAN

duration -> TIME

lecture\_content -> BLOB NOT NULL

membership\_type -> ENUM('BRZ', 'SLV', 'GLD') NOT NULL DEFAULT 'BRZ'

answer -> ENUM('choice1', 'choice2', 'choice3') NOT NULL DEFAULT 'choice1'

## Student

### Relational Model:

Student(SID, name, e\_mail, password, membership\_type)

### SQL Definition:

CREATE TABLE Student(

SID INT AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

e\_mail VARCHAR(50) NOT NULL UNIQUE,

password VARCHAR(30) NOT NULL,

membership\_type ENUM('BRZ', 'SLV', 'GLD') NOT NULL DEFAULT 'BRZ',

PRIMARY KEY (SID)

);

## Instructor

### Relational Model:

Instructor(IID, name, e\_mail, password)

### SQL Definition:

CREATE TABLE Instructor(

IID INT AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

e\_mail VARCHAR(50) NOT NULL UNIQUE,

password VARCHAR(30) NOT NULL,

PRIMARY KEY (IID)

);

## Admin

### Relational Model:

Admin(AID, name, e\_mail, password, salary)

### SQL Definition:

CREATE TABLE Admin(

AID INT AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

e\_mail VARCHAR(50) NOT NULL UNIQUE,

password VARCHAR(30) NOT NULL,

salary NUMERIC(12, 2) NOT NULL DEFAULT 0.00,

PRIMARY KEY (AID)

);

## Course

### Relational Model:

Course(CID, IID, course\_name, course\_photo, description, rating, category, level, cost, discount\_allowed, quiz\_threshold, lecture\_completed\_threshold)

### SQL Definition:

/\* Assumptions:

include functions [rating(), student\_count(), lecture\_count()]

constraints for category and level

0 <= cost <= 999,999.99

discount\_allowed = True OR False

lecture\_completed\_threshold = %70

0 <= certificate\_threshold <= 100.00

attention: FOREIGN KEY (creator\_IID) REFERENCES Instructor(IID) ON DELETE NO ACTION ON UPDATE CASCADE

UNIQUE (creator\_IID, course\_name)

\*/

CREATE TABLE Course(

CID INT AUTO\_INCREMENT,

IID INT NOT NULL,

course\_name VARCHAR(70) NOT NULL,

course\_photo BLOB,

description VARCHAR(300),

rating NUMERIC(5, 2) DEFAULT 0.00,

category VARCHAR(30),

level VARCHAR(30),

cost NUMERIC(8, 2) NOT NULL DEFAULT 0.00,

discount\_allowed BOOLEAN,

quiz\_threshold NUMERIC(5, 2),

lecture\_completed\_threshold NUMERIC(5, 2),

PRIMARY KEY (CID),

FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE NO ACTION ON UPDATE CASCADE,

UNIQUE (IID, course\_name)

);

### Section

### Relational Model:

Section(CID, section, title)

### SQL Definition:

CREATE TABLE Section(

CID INT NOT NULL,

section INT NOT NULL,

title VARCHAR(300),

PRIMARY KEY (CID, section),

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE,

INDEX (section)

);

## Lecture

### Relational Model:

Lecture(CID, content\_num, IID, section, title, lecture\_content, duration, date)

### SQL Definition:

/\* Assumptions:

lecture\_content is big data

lecture\_content is blob (not recommended, try to store videos in file directories and hold URLs)

Attention: UNIQUE (CID, title)

\*/

CREATE TABLE Lecture(

CID INT NOT NULL,

content\_num INT NOT NULL,

IID INT NOT NULL,

section INT NOT NULL,

title VARCHAR(300) NOT NULL,

lecture\_content BLOB NOT NULL,

duration TIME NOT NULL,

date DATE,

PRIMARY KEY (CID, content\_num),

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (section) REFERENCES Section(section) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE NO ACTION ON UPDATE CASCADE,

UNIQUE (CID, title),

INDEX (content\_num)

);

## Quiz

### Relational Model:

Quiz(CID, content\_num, IID, section, title, date)

### SQL Definition:

CREATE TABLE Quiz(

CID INT NOT NULL,

content\_num INT NOT NULL,

IID INT NOT NULL,

section INT NOT NULL,

title VARCHAR(300) NOT NULL,

date DATE,

PRIMARY KEY (CID, content\_num),

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (section) REFERENCES Section(section) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE NO ACTION ON UPDATE CASCADE,

UNIQUE (CID, title),

INDEX (content\_num)

);

## Quiz\_Question

### Relational Model:

Quiz\_Question(CID, content\_num, question\_num, question\_text, choice1, choice2, choice3, answer)

### SQL Definition:

/\*

Attention: answer ENUM('choice1', 'choice2', 'choice3') NOT NULL DEFAULT 'choice1'

answer = 1 => answer = choice1

\*/

CREATE TABLE Quiz\_Question(

CID INT NOT NULL,

content\_num INT NOT NULL,

question\_num INT NOT NULL,

question\_text VARCHAR(300) NOT NULL,

choice1 VARCHAR(50) NOT NULL DEFAULT '',

choice2 VARCHAR(50),

choice3 VARCHAR(50),

answer ENUM('choice1', 'choice2', 'choice3') NOT NULL DEFAULT 'choice1',

PRIMARY KEY (CID, content\_num, question\_num),

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (content\_num) REFERENCES Quiz(content\_num) ON DELETE CASCADE ON UPDATE CASCADE

);

## Note

### Relational Model:

Note(Note\_ID, SID, content\_num, time, text)

### SQL Definition:

CREATE TABLE Note(

Note\_ID INT AUTO\_INCREMENT,

SID INT NOT NULL,

content\_num INT NOT NULL,

time TIME,

text VARCHAR(300) NOT NULL,

PRIMARY KEY (Note\_ID),

FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (content\_num) REFERENCES Lecture(content\_num) ON DELETE CASCADE ON UPDATE CASCADE

);

## Course\_Announcement

### Relational Model:

Course\_Announcement(Ann\_ID, IID, CID, date, subject\_name, text)

### SQL Definition:

CREATE TABLE Course\_Announcement(

Ann\_ID INT AUTO\_INCREMENT,

IID INT NOT NULL,

CID INT NOT NULL,

date DATE,

subject\_name VARCHAR(30) NOT NULL,

text VARCHAR(300) NOT NULL,

PRIMARY KEY (Ann\_ID),

FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

## QnA\_Entry\_Student

### Relational Model:

QnA\_Entry\_Student(QandA\_ID, SID, CID, subject\_name, text, date)

### SQL Definition:

CREATE TABLE QnA\_Entry\_Student(

QnA\_ID INT AUTO\_INCREMENT,

SID INT NOT NULL,

CID INT NOT NULL,

subject\_name VARCHAR(30) NOT NULL,

text VARCHAR(300) NOT NULL,

date DATE,

PRIMARY KEY (QnA\_ID),

FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE NO ACTION ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

## QnA\_Entry\_Instructor

### Relational Model:

QnA\_Entry\_Instructor(QandA\_ID, IID, CID, subject\_name, text, date)

### SQL Definition:

CREATE TABLE QnA\_Entry\_Instructor(

QnA\_ID INT AUTO\_INCREMENT,

IID INT NOT NULL,

CID INT NOT NULL,

subject\_name VARCHAR(30) NOT NULL,

text VARCHAR(300) NOT NULL,

date DATE,

PRIMARY KEY(QnA\_ID),

FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE NO ACTION ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

## Complaint\_Entry\_Student

### Relational Model:

Complaint\_Entry\_Student(Complaint\_ID, SID, CID, subject\_name, text, date, refund\_request)

### SQL Definition:

CREATE TABLE Complaint\_Entry\_Student(

Complaint\_ID INT AUTO\_INCREMENT,

SID INT NOT NULL,

CID INT NOT NULL,

subject\_title VARCHAR(30) NOT NULL,

text VARCHAR(300) NOT NULL,

date DATE,

refund\_request BOOLEAN,

PRIMARY KEY (Complaint\_ID),

FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

## Complaint\_Entry\_Instructor

### Relational Model:

Complaint\_Entry\_Instructor(Complaint\_ID, SID, CID, subject\_name, text, date)

### SQL Definition:

CREATE TABLE Complaint\_Entry\_Instructor(

Complaint\_ID INT AUTO\_INCREMENT,

IID INT NOT NULL,

CID INT NOT NULL,

subject\_name VARCHAR(30) NOT NULL,

text VARCHAR(300) NOT NULL,

date DATE,

PRIMARY KEY (Complaint\_ID),

FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

## Wishlist

### Relational Model:

Wishlist(SID, CID, date)

### SQL Definition:

/\*

Binary relationship between Student and Course

Attributes: date

\*/

CREATE TABLE Wishlist(

SID INT NOT NULL,

CID INT NOT NULL,

date DATE,

PRIMARY KEY (SID, CID),

FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

## Enrolls

### Relational Model:

Enrolls(SID, CID, progress, rating, date)

### SQL Definition:

/\*

Binary relationship between Student and Course

Attributes: progress, rating

\*/

CREATE TABLE Enrolls(

SID INT NOT NULL,

CID INT NOT NULL,

progress NUMERIC(5, 2) DEFAULT 0.00,

rating NUMERIC(5, 2),

date DATE,

PRIMARY KEY (SID, CID),

FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

## Certificate

### Relational Model:

Certificate(SID, CID, certificate\_text, date)

### SQL Definition:

/\*

Binary relationship between Student and Course

Attributes: date

\*/

CREATE TABLE Certificate(

SID INT NOT NULL,

CID INT NOT NULL,

certificate\_text VARCHAR(300),

date DATE,

PRIMARY KEY (SID, CID),

FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

## Teaches

### Relational Model:

Teaches(IID, CID)

### SQL Definition:

/\*

Binary relationship between Instructor and Course

\*/

CREATE TABLE Teaches(

IID INT NOT NULL,

CID INT NOT NULL,

PRIMARY KEY (IID, CID),

FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

## Evaluates\_Complaint\_Entry\_Student

### Relational Model:

Evaluates\_Complaint\_Entry\_Student(AID, Complaint\_ID, response, isRefund)

### SQL Definition:

/\*

Binary relationship between Admin and Complaint\_Entry

Attention: FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE CASCADE

\*/

CREATE TABLE Evaluates\_Complaint\_Entry\_Student(

AID INT NOT NULL,

Complaint\_ID INT NOT NULL,

response VARCHAR(300),

isRefund BOOLEAN,

PRIMARY KEY (AID, Complaint\_ID),

FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE CASCADE,

FOREIGN KEY (Complaint\_ID) REFERENCES Complaint\_Entry\_Student(Complaint\_ID) ON DELETE CASCADE ON UPDATE CASCADE

);

## Evaluates\_Complaint\_Entry\_Instructor

### Relational Model:

Evaluates\_Complaint\_Entry\_Instructor(AID, Complaint\_ID, response)

### SQL Definition:

/\*

Binary relationship between Admin and Complaint\_Entry

Attention: FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE CASCADE

\*/

CREATE TABLE Evaluates\_Complaint\_Entry\_Instructor(

AID INT NOT NULL,

Complaint\_ID INT NOT NULL,

response VARCHAR(300),

PRIMARY KEY (AID, Complaint\_ID),

FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE CASCADE,

FOREIGN KEY (Complaint\_ID) REFERENCES Complaint\_Entry\_Instructor(Complaint\_ID) ON DELETE CASCADE ON UPDATE CASCADE

);

## Discount

### Relational Model:

Discount(AID, CID, rate)

### SQL Definition:

/\*

Binary relationship between Admin and Course

Attributes: rate (e.g. %30)

Attention: FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE CASCADE

\*/

CREATE TABLE Discount(

AID INT NOT NULL,

CID INT NOT NULL,

rate NUMERIC(5, 2),

PRIMARY KEY (AID, CID),

FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

## Zoom\_Session

### Relational Model:

Zoom\_Session(Session\_ID, IID, CID, invite\_link, start\_date);

/\*

Binary relationship between Instructor and Course

Attributes: invite\_link, start\_date

Attention: it uses its own key to allow instructors to create several links

\*/

CREATE TABLE Zoom\_Session(

Session\_ID INT AUTO\_INCREMENT,

IID INT NOT NULL,

CID INT NOT NULL,

invite\_link VARCHAR(1024),

start\_date TIMESTAMP,

PRIMARY KEY (Session\_ID),

FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

### SQL Definition:

## Take\_Quiz

### Relational Model:

Take\_Quiz(SID, CID, content\_num, grade)

### SQL Definition:

/\*

Binary relationship between Student and Quiz

Attributes: grade (updated according to Student\_Take\_Quiz\_Question)

Attention: its key is Student(SID) U Quiz(CID, QID)

\*/

CREATE TABLE Take\_Quiz(

SID INT NOT NULL,

CID INT NOT NULL,

content\_num INT NOT NULL,

grade NUMERIC(5, 2),

PRIMARY KEY (SID, CID, content\_num),

FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (content\_num) REFERENCES Quiz(content\_num) ON DELETE CASCADE ON UPDATE CASCADE

);

## Take\_Quiz\_Question

### Relational Model:

Take\_Quiz\_Question(SID, CID, content\_num, question\_num, answer, isTrue)

### SQL Definition:

/\*

Binary relationship between Student and Quiz\_Question

keeps students' answers to each quiz question

Attributes: isTrue

\*/

CREATE TABLE Take\_Quiz\_Question(

SID INT NOT NULL,

CID INT NOT NULL,

content\_num INT NOT NULL,

question\_num INT NOT NULL,

answer ENUM('choice1', 'choice2', 'choice3') NOT NULL,

isTrue BOOLEAN,

PRIMARY KEY (SID, CID, content\_num, question\_num),

FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID, content\_num, question\_num) REFERENCES Quiz\_Question(CID, content\_num, question\_num) ON DELETE CASCADE ON UPDATE CASCADE

);

## Take\_Lecture

### Relational Model:

Take\_Quiz\_Lecture(SID, CID, content\_num, isCompleted)

### SQL Definition:

/\*

Binary relationship between Student and Lecture

keeps students' completion to each lecture (lecture\_completed\_threshold in lecture (e.g. %70 for each lecture))

Attributes: isCompleted

\*/

CREATE TABLE Take\_Lecture(

SID INT NOT NULL,

CID INT NOT NULL,

content\_num INT NOT NULL,

isCompleted BOOLEAN,

PRIMARY KEY (SID, CID, content\_num),

FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (CID, content\_num) REFERENCES Lecture(CID, content\_num) ON DELETE CASCADE ON UPDATE CASCADE

);

## Evaluations

### Relational Model:

Evaluations(SID, CID, evaluation)

### SQL Definition:

CREATE TABLE Evaluations(

SID INT NOT NULL,

CID INT NOT NULL,

evaluation VARCHAR(300),

PRIMARY KEY (SID, CID),

FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,

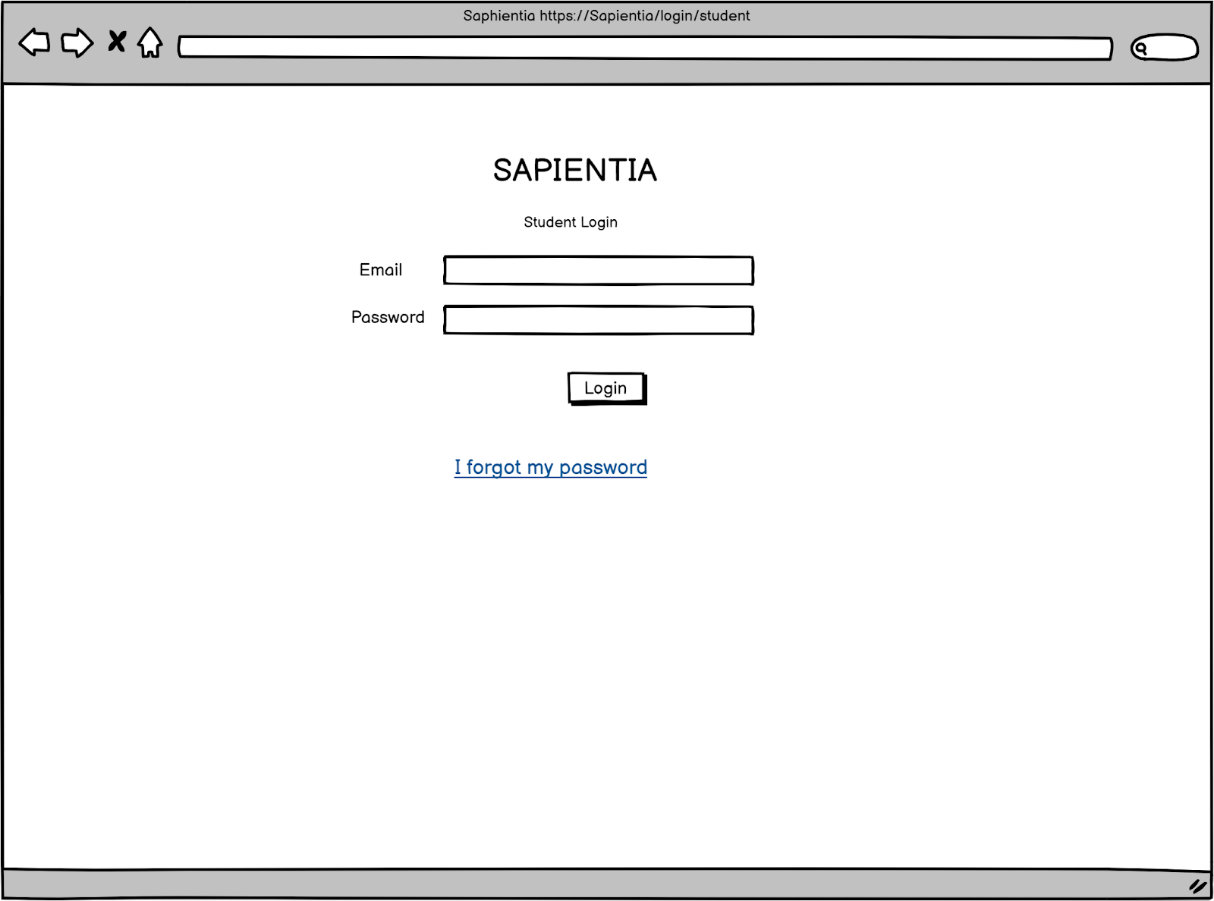
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE

);

# User Interface Design and Corresponding SQL Statements

## Login

**Student Login**



Students will log-in to Sapientia using their email and password.

Inputs: @e\_mail , @password

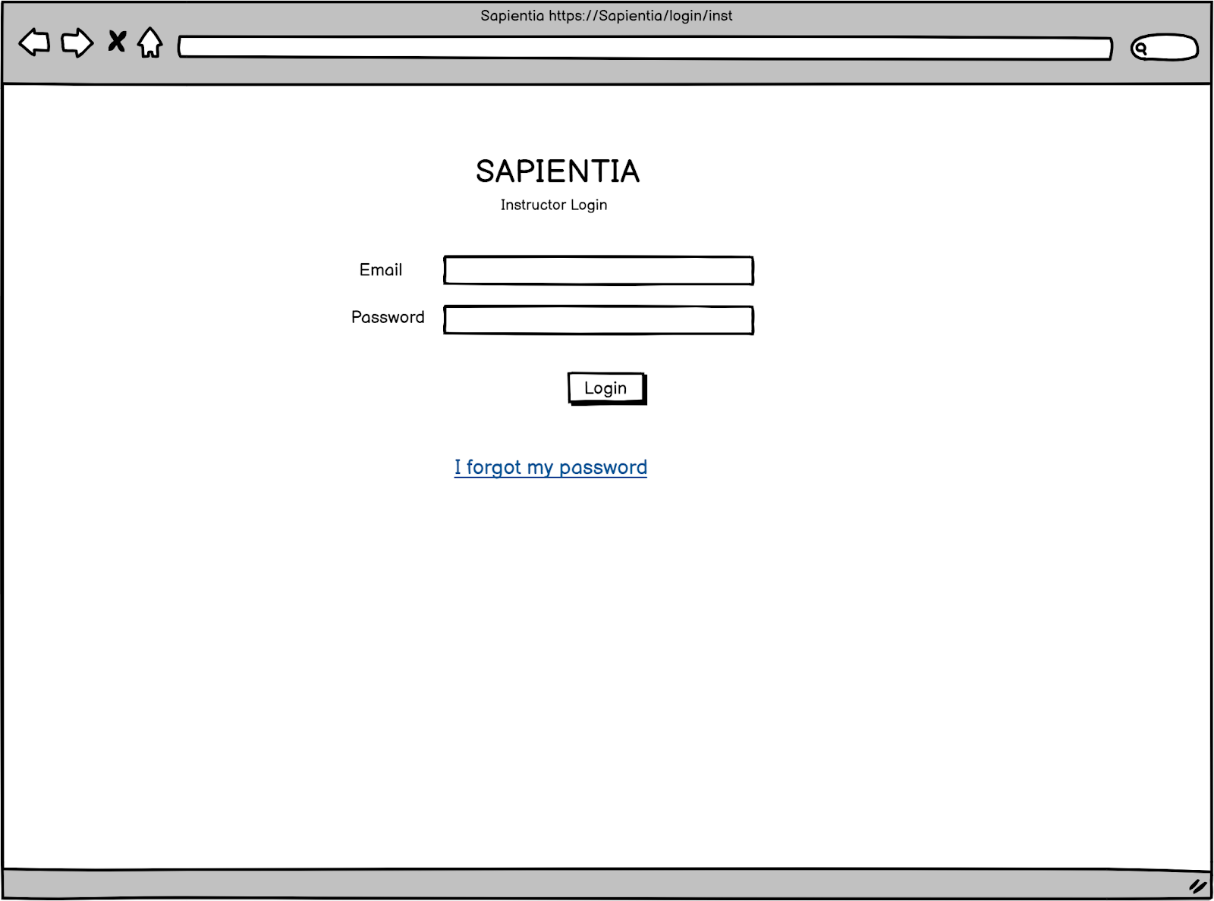
On Login button pressed:

SELECT \*

FROM Student

WHERE e\_mail = @e\_mail AND password = @password

**Instructor Login**



Instructors will log-in to Sapientia using their email and password.

Inputs: @e\_mail , @password

On Login button pressed:

SELECT \*

FROM Instructor

WHERE e\_mail = @e\_mail AND password = @password

**Admin login**



Admin will log-in to Sapientia using the email and password associated with the admin role.

Inputs: @e\_mail , @password

On Login button pressed:

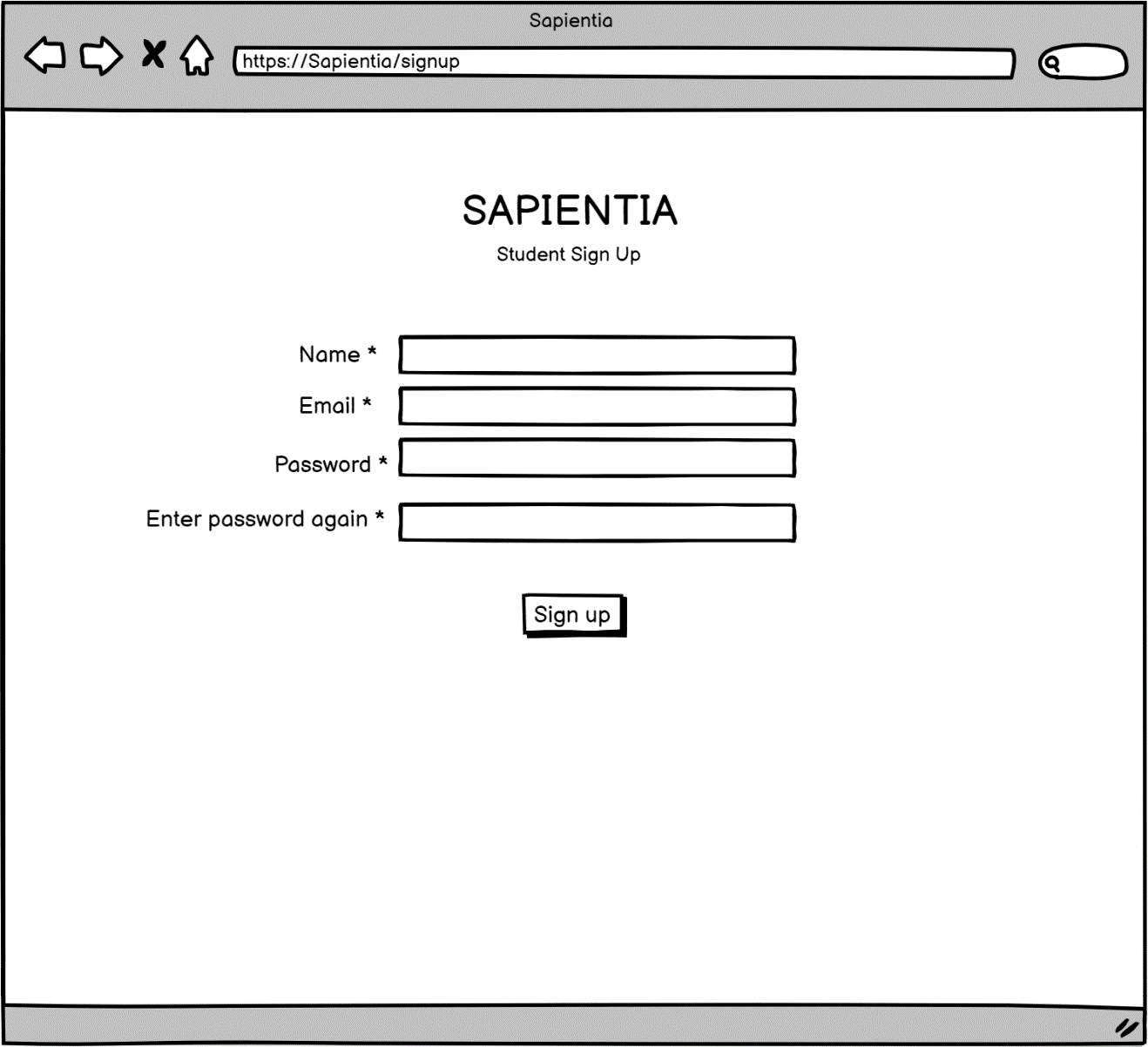
SELECT \*

FROM Admin

WHERE e\_mail = @e\_mail AND password = @password

## Sign Up

**Student Sign Up**



Students will sign up by their name, email and password. The password will be checked twice.

Inputs: @name, @e\_mail, @password

On Sign up button pressed:

BEGIN

IF NOT EXISTS ( SELECT \* FROM Student

WHERE e\_mail = @e\_mail)

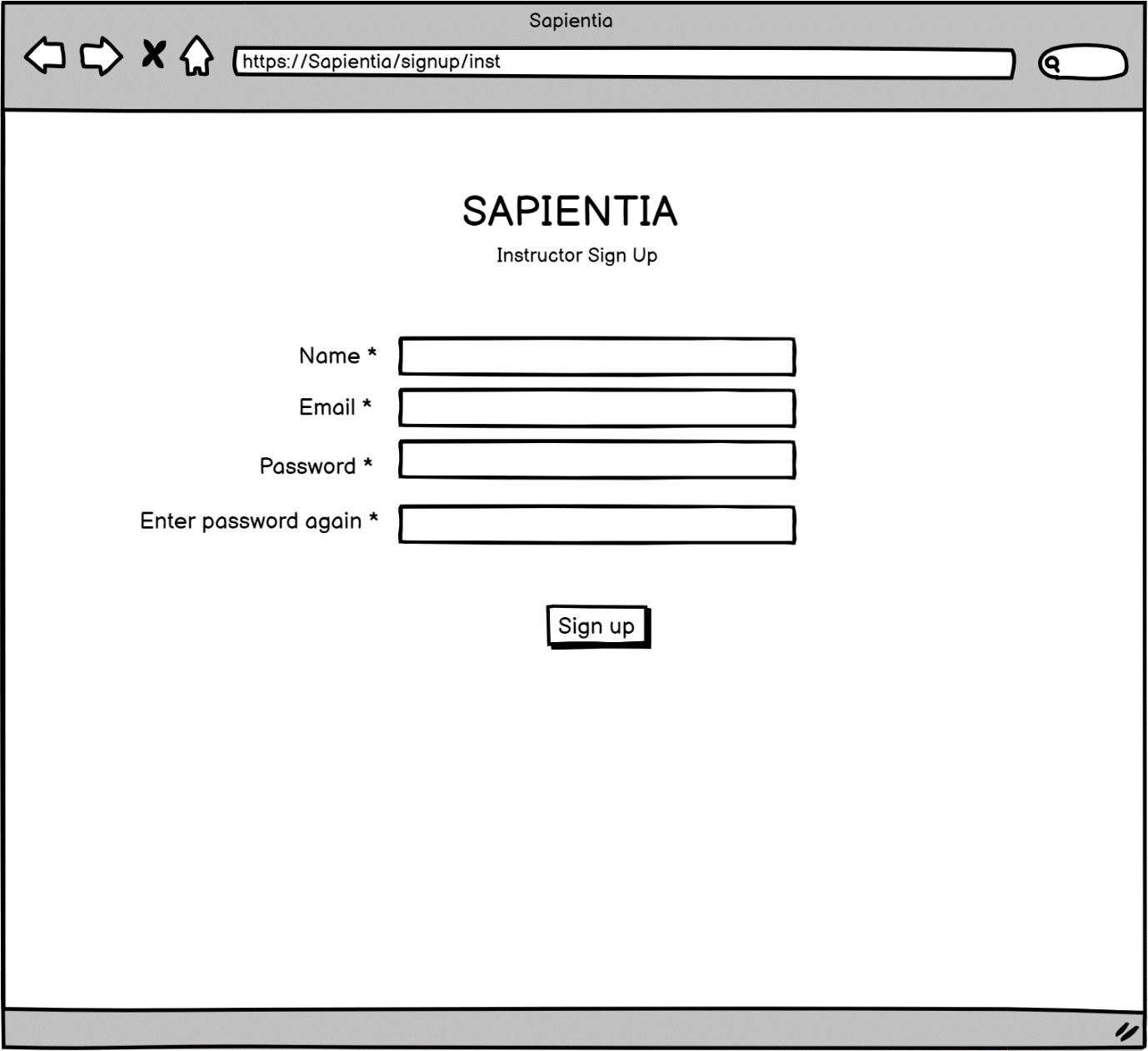
BEGIN

INSERT INTO Student (name, e\_mail, password)

VALUES (@name, @e\_mail, @password)

END

END

**Instructor Sign Up**

Instructors will sign up by their name, email and password. The password will be checked twice.

Inputs: @name, @e\_mail, @password

On Sign up button pressed:

BEGIN

IF NOT EXISTS ( SELECT \* FROM Student

WHERE e\_mail = @e\_mail)

BEGIN

INSERT INTO Instructor (name, e\_mail, password)

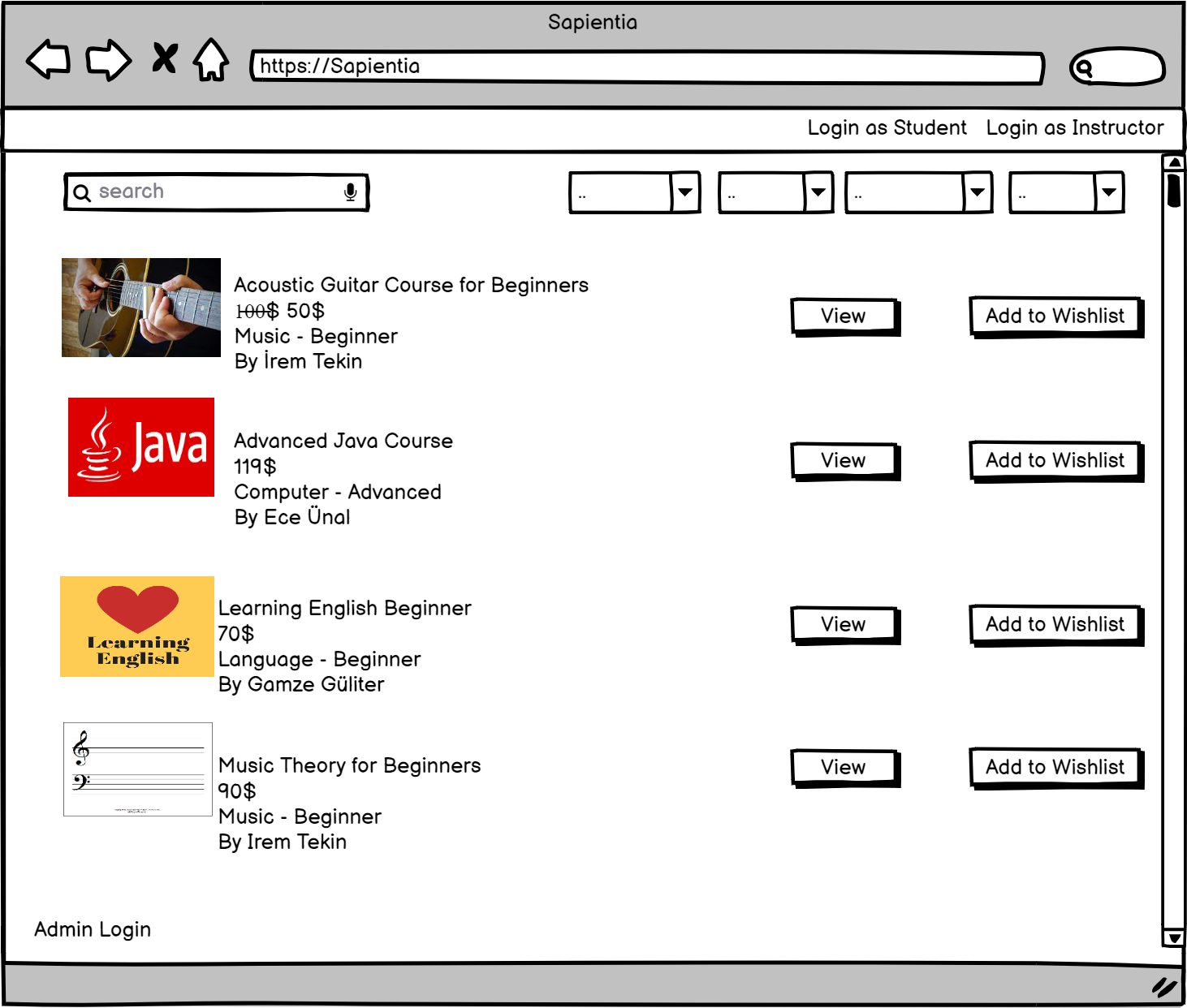
VALUES (@name, @e\_mail, @password)

END

END

## 

## Main Page Without Logging in



Students will be able to see courses in Sapientia without logging in. However, for a student to add a course to the wishlist or purchase it , logging in is needed. 

Inputs: @category, @price\_threshold, @discount, @keyword, @level

SELECT course\_name, course\_photo, cost, category, level, name

FROM Course FULL OUTER JOIN Discount NATURAL JOIN Instructor

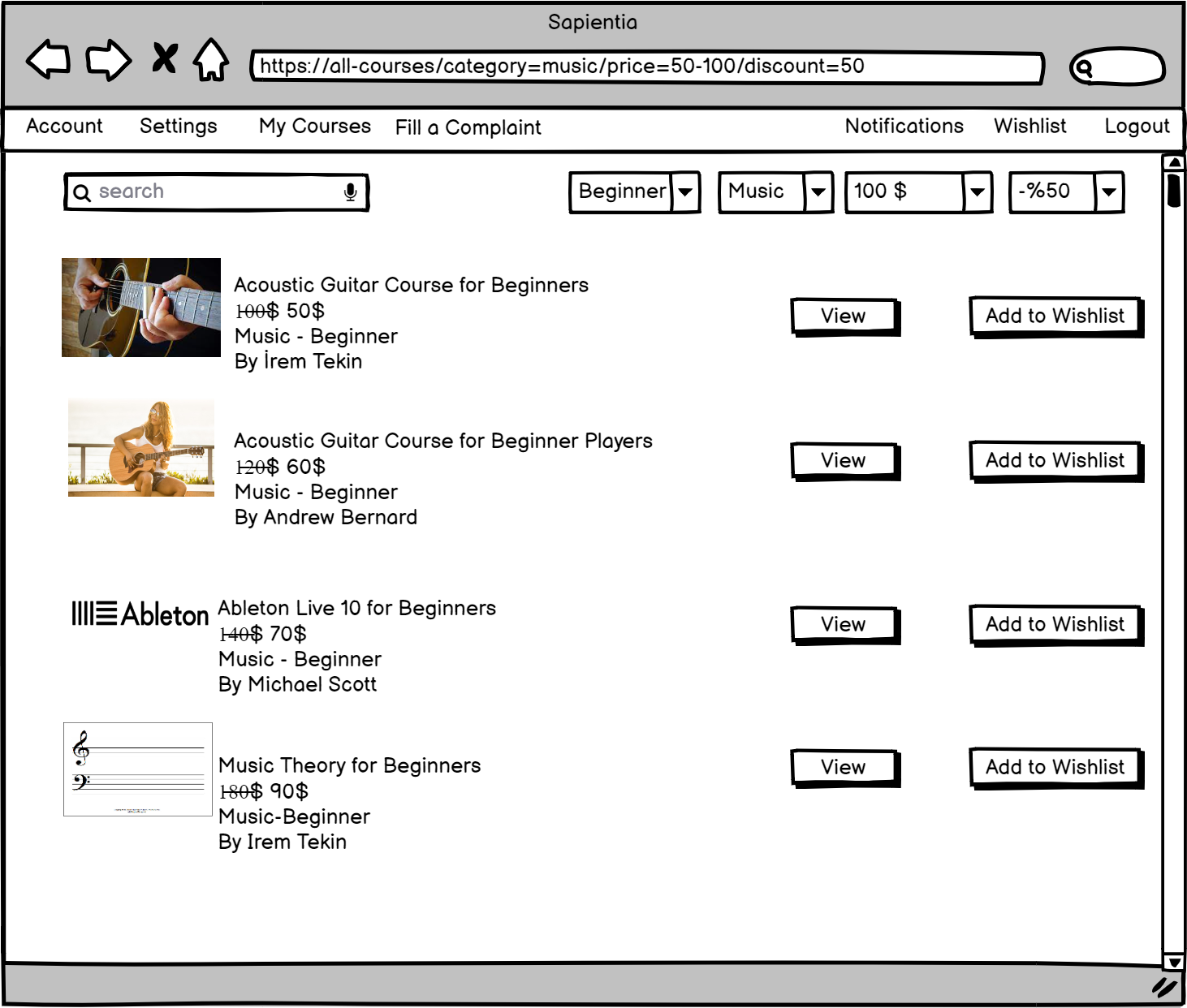
WHERE category = @category AND price < @price\_threshold

AND rate = @discount

AND ( course\_name like ‘%keyword%’

          OR instructor\_name like ‘%keyword%’ )

## Filtered Main Page After Student Login



When students log in to their account, they will be able to find courses to enroll. At first , the most popular courses based on the enrolled students will be shown in their main page. However, students will be able to filter courses in their main page by a keyword or level of course, the category , maximum price and the discount rate.

**Case:** In the above example, the user wanted to see the courses in the beginner level of music category. In addition the courses are under 100$ and have the discount rate of %50.

Inputs: @category, @price\_threshold, @discount, @keyword, @level

SELECT course\_name, course\_photo, cost, category, level, name

FROM Course FULL OUTER JOIN Discount NATURAL JOIN Instructor

WHERE category = @category AND price < @price\_threshold

AND rate = @discount

AND ( course\_name like ‘%keyword%’

          OR instructor\_name like ‘%keyword%’ )

           EXCEPT ( SELECT course\_name, course\_photo, cost, category, level

            FROM Enrolls NATURAL JOIN Course

            WHERE SID = @student\_id  )

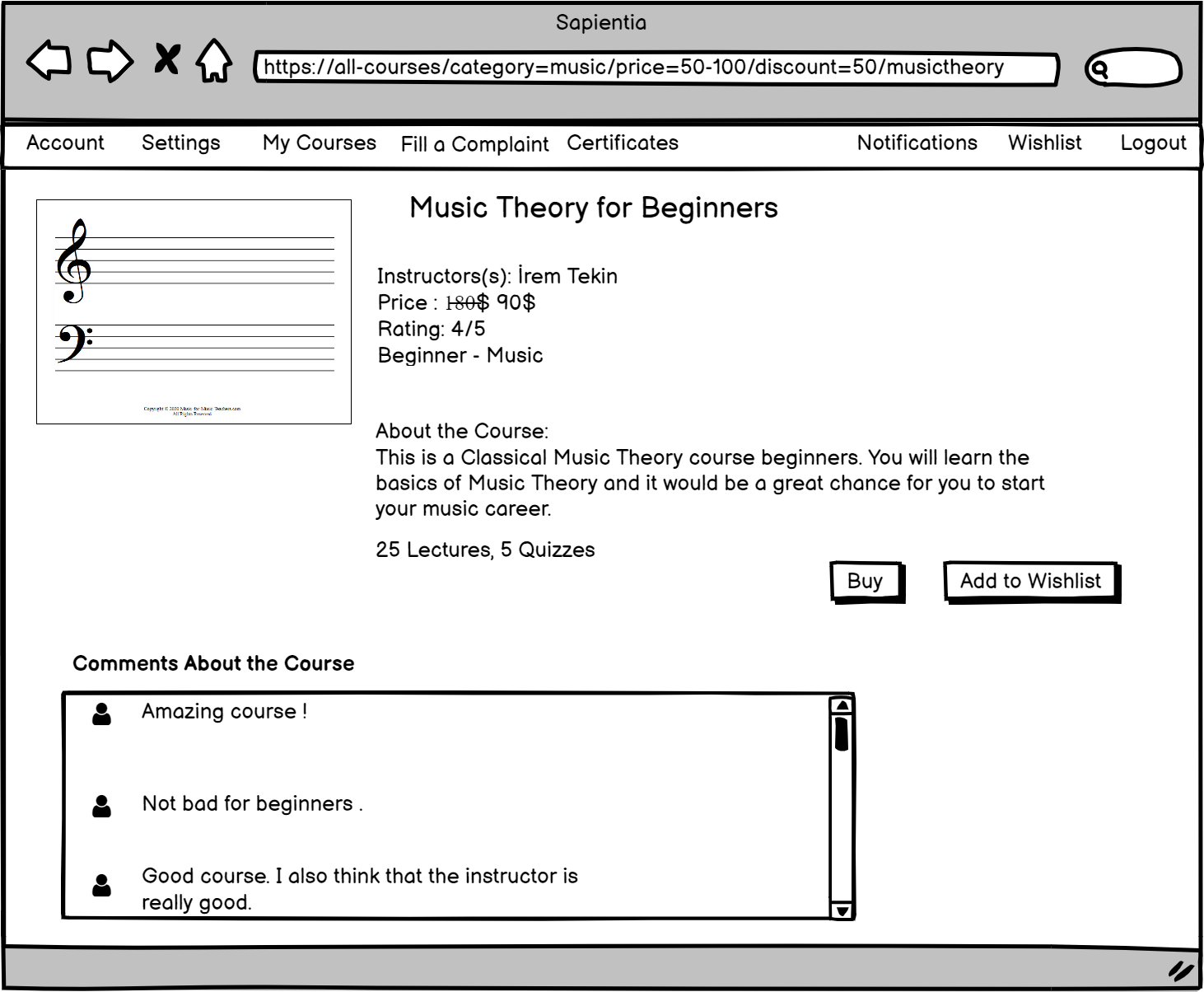
On Add to Wishlist button pressed:

DECLARE @current\_date date = getdate()

INSERT INTO Wishlist(SID, CID, date)

VALUES(@SID, @CID, @current\_date)

## **Course Information Page**



The users will be able to view information about the course if they press the **View** button shown in the right side of the course name in the main page. In this page, they will see the description of the course, price, overall rating, information such as quiz-lecture number of course's difficulty level and the comments from other students who finished the course.In addition, they can buy the course from this page and add it to their wishlist.

**Case:** In the above example, the user is viewing Music Theory for Beginner’s course’s page.

SELECT name, IID

FROM Teaches JOIN USING (IID) Instructor

WHERE CID = @CID

SELECT course\_photo,course\_name, old\_cost, cost, description, avg\_rating, name

FROM Course, Temp

WHERE CID = @CID

SELECT COUNT(\*)

FROM Lecture

WHERE CID = @CID

SELECT COUNT(\*)

FROM Quiz

WHERE CID = @CID

On Buy button pressed:

INSERT INTO ENROLLS(SID, CID, date)

VALUES (@SID, @CID, @date)

SELECT COUNT(\*)

FROM Enrolls

WHERE SID = @SID

Using this count, we will determine membership type with PHP and then update student table.

UPDATE Student

SET membership\_type = @type

On Add to Wishlist button pressed:

DECLARE @current\_date date = getdate()

INSERT INTO Wishlist(SID, CID, date)

VALUES(@SID, @CID, @current\_date)

## Wishlist



The students will be able to see their wishlist from the **Wishlist** button in the navigation bar, then they can either buy the course or remove it from the wishlist.

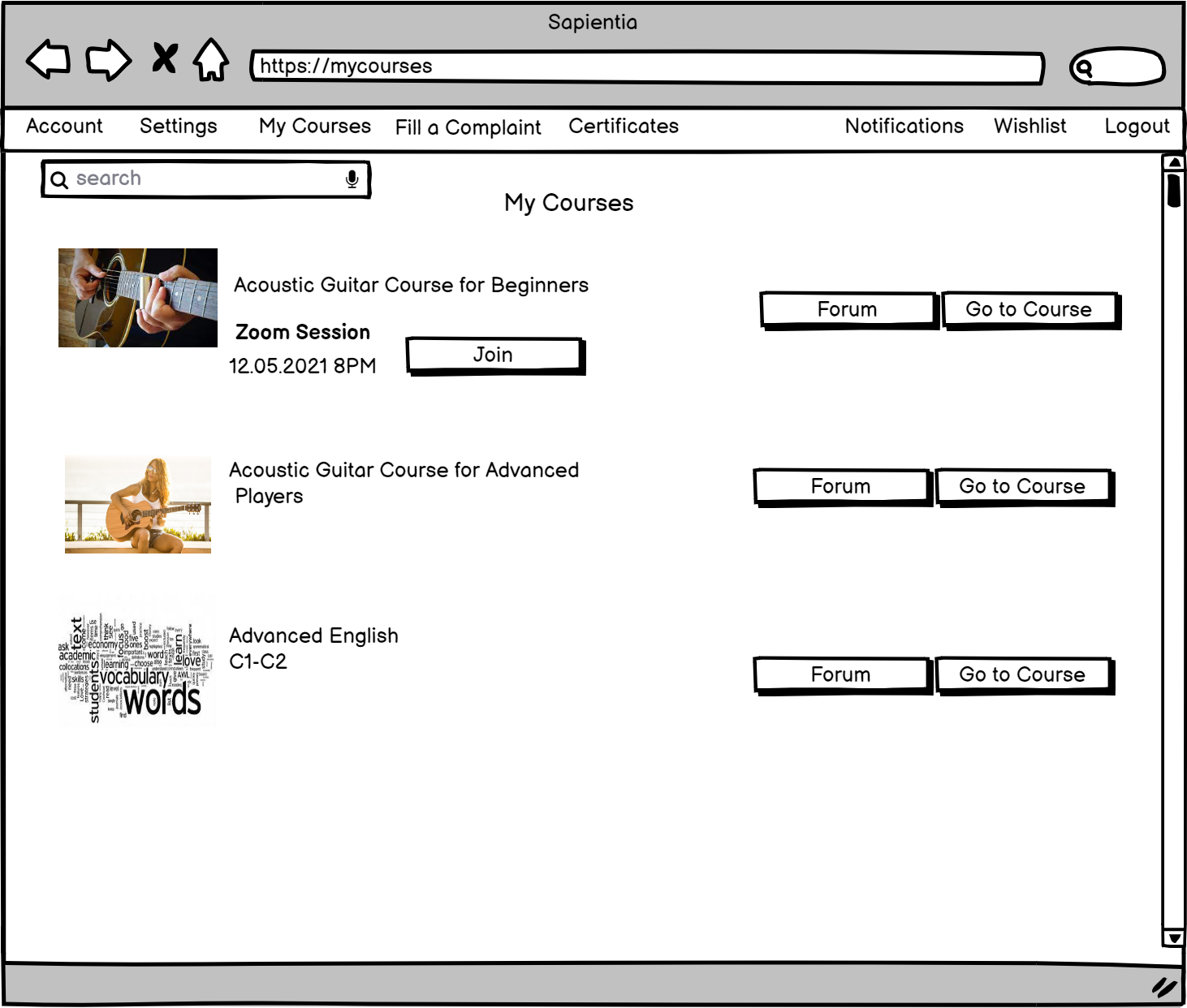
**Case**: In the above example the student added three courses to his/her wishlist.

SELECT course\_name, course\_photo, cost, category, level, name

FROM Course FULL OUTER JOIN Discount NATURAL JOIN Instructor NATURAL JOIN Wishlist

WHERE SID = @SID

## My Courses



The users will be able to see the courses that they purchased from the **My Courses** button in the navigation bar. On that page, they will be able to see the Q&A section from the Forum button and go to the course’s own page to continue watching the course.

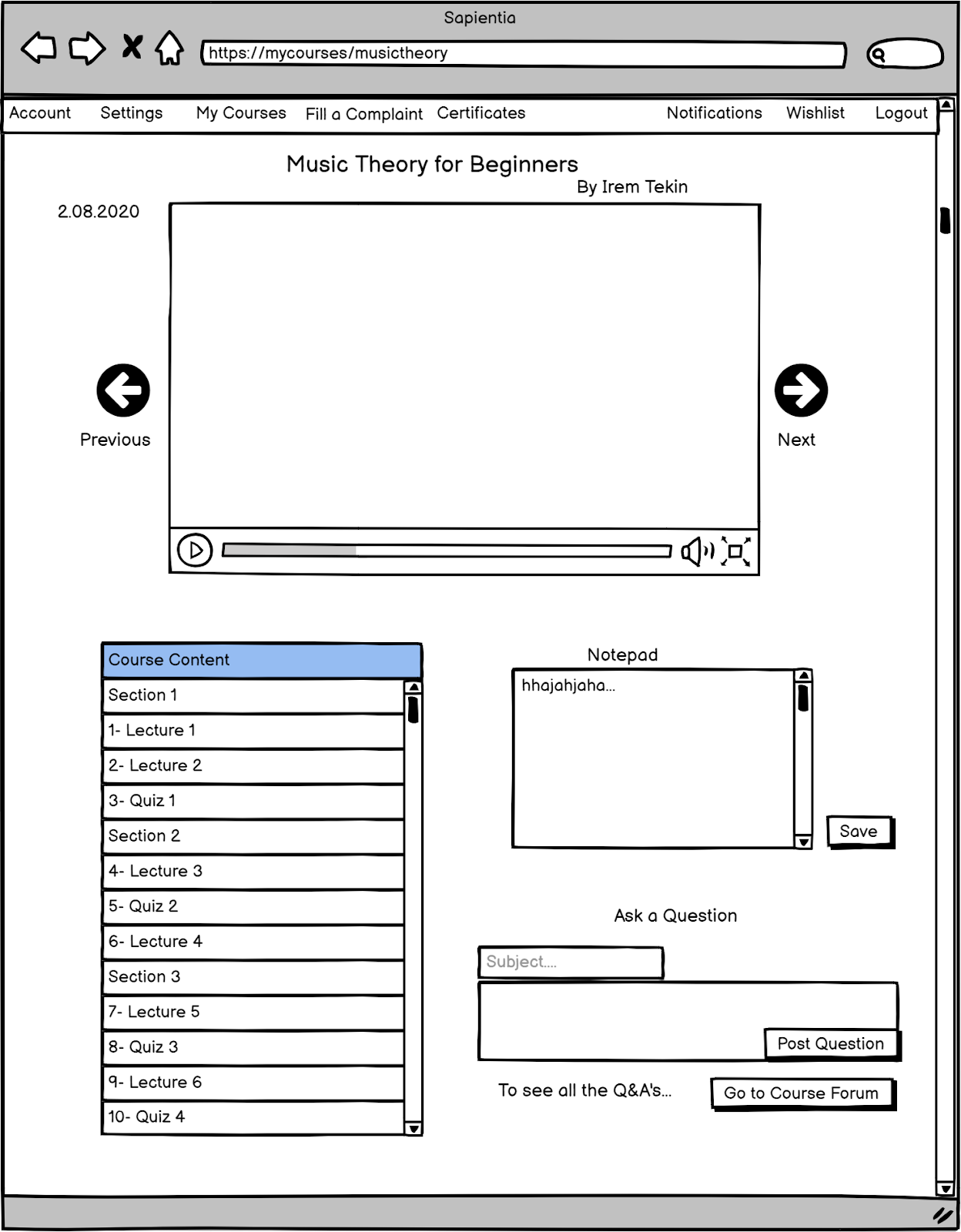
**Case** : In the above example the student purchased 3 courses. The first course has a Zoom Q&A meeting conducted by the instructor where the student can join the meeting by the Join button and will be directed to Zoom.

SELECT course\_name, course\_photo, invite\_link, start\_date

FROM Course NATURAL JOIN Enrolls FULL OUTER JOIN Zoom\_Session

WHERE SID = @SID

## Course - Lecture



As shown in the **left corner**, course content is categorized by sections and under each section, there are quizzes and lectures. The students will also be able to ask a question to the forum and add notes to their notepad while watching a lecture.

**Case:** In the above example, the student is at Section **2** and watching **Lecture 3**. The student can press the next and see the next course content which is **Quiz 2** in this example. However, after the video is finished , the course will automatically skip to the next content.

    SELECT lecture\_content, content\_num, date

FROM Lecture

WHERE CID = @CID AND content\_num = @content\_num

    SELECT content\_num, title

    FROM Lecture

    SELECT content\_num, title

    FROM Quiz

    SELECT text

    FROM Note NATURAL JOIN Student NATURAL JOIN Lecture

    WHERE CID = @CID AND content\_num = @content\_num AND SID = @SID

    Input: @note\_text, @text, @subject\_name

iii. Create notes on lectures (Visible only to user)

    On Save button pressed:

    INSERT INTO Note(SID, content\_num, text)

    VALUES(@SID,@content\_num,@note\_text)

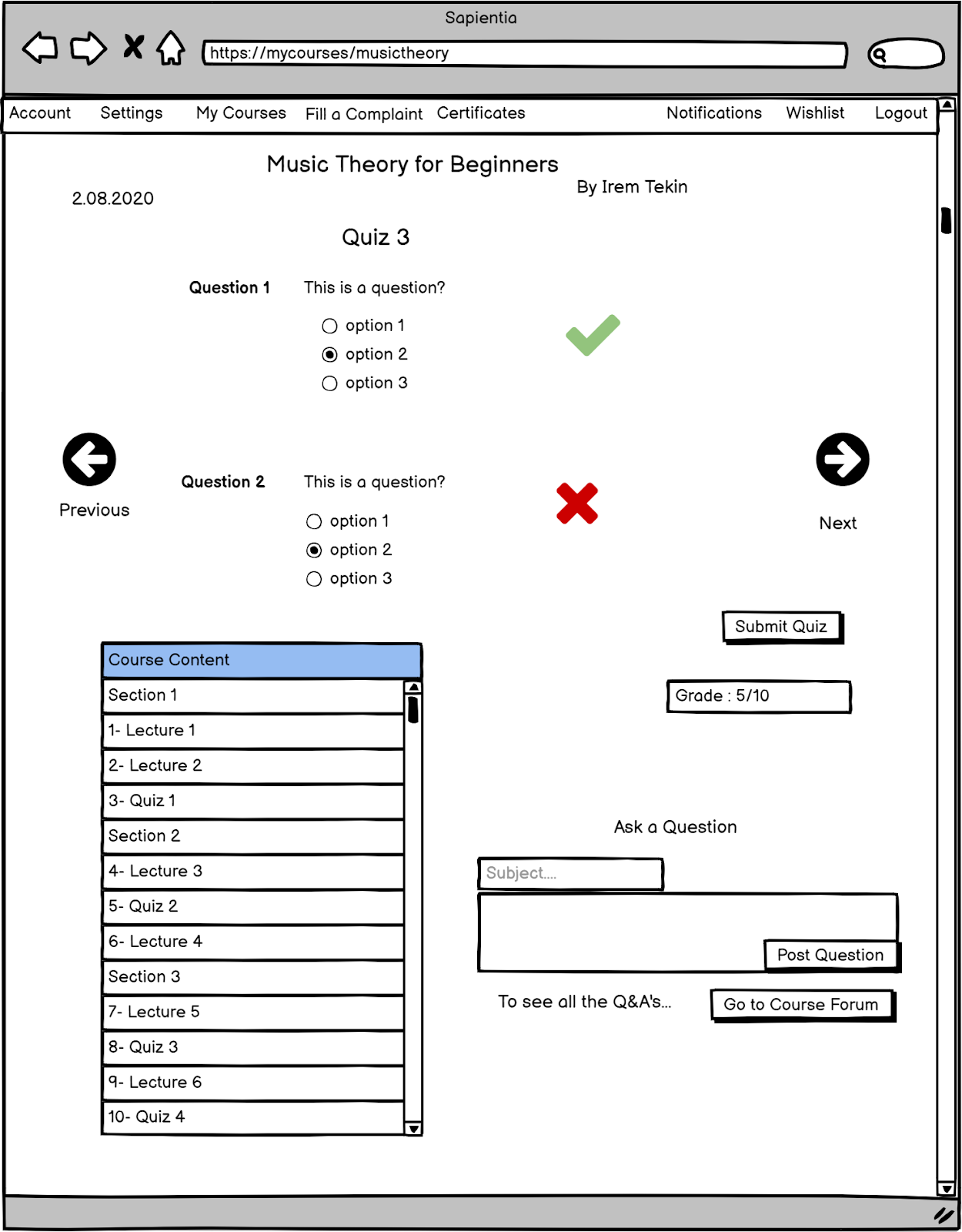
    On Post Question button pressed:

DECLARE @current\_date date = getdate()

INSERT INTO QnA\_Entry\_Student(CID, SID, subject\_name, text, date)

VALUES(@CID, @SID, @subject\_name, @text, @current\_date)

## Course- Quiz



There will be quizzes in a course as well as lectures. As shown in the **left corner**, course content is categorized by sections and under each section, there are quizzes and lectures. The students will also be able to ask a question to the forum while they are solving the quiz.

**Case:** In the above example, the student is at **Section 3** and solving **Quiz 3**. The first question is answered correctly and the second is false. After solving the quiz, the student can press the next and see the next course content which is **Lecture 6** in this example.

Inputs: @subject\_name, @answer, @text

SELECT date, name, title, question\_num, question\_text, choice1, choice2, choice3

FROM Quiz\_Question NATURAL JOIN Quiz NATURAL JOIN Course

WHERE CID = @CID AND content\_num = @content\_num

On Submit Quiz button pressed

INSERT INTO Take\_Quiz\_Question(SID, CID, content\_num, question\_num, answer, isTrue)

VALUES (@SID, @CID, @content\_num, @question\_num, @answer, @isTrue)

INSERT INTO Take\_Quiz(SID, CID, content\_num, grade)

VALUES(@SID, @CID, @content\_num, @grade)

SELECT grade

FROM Take\_Quiz

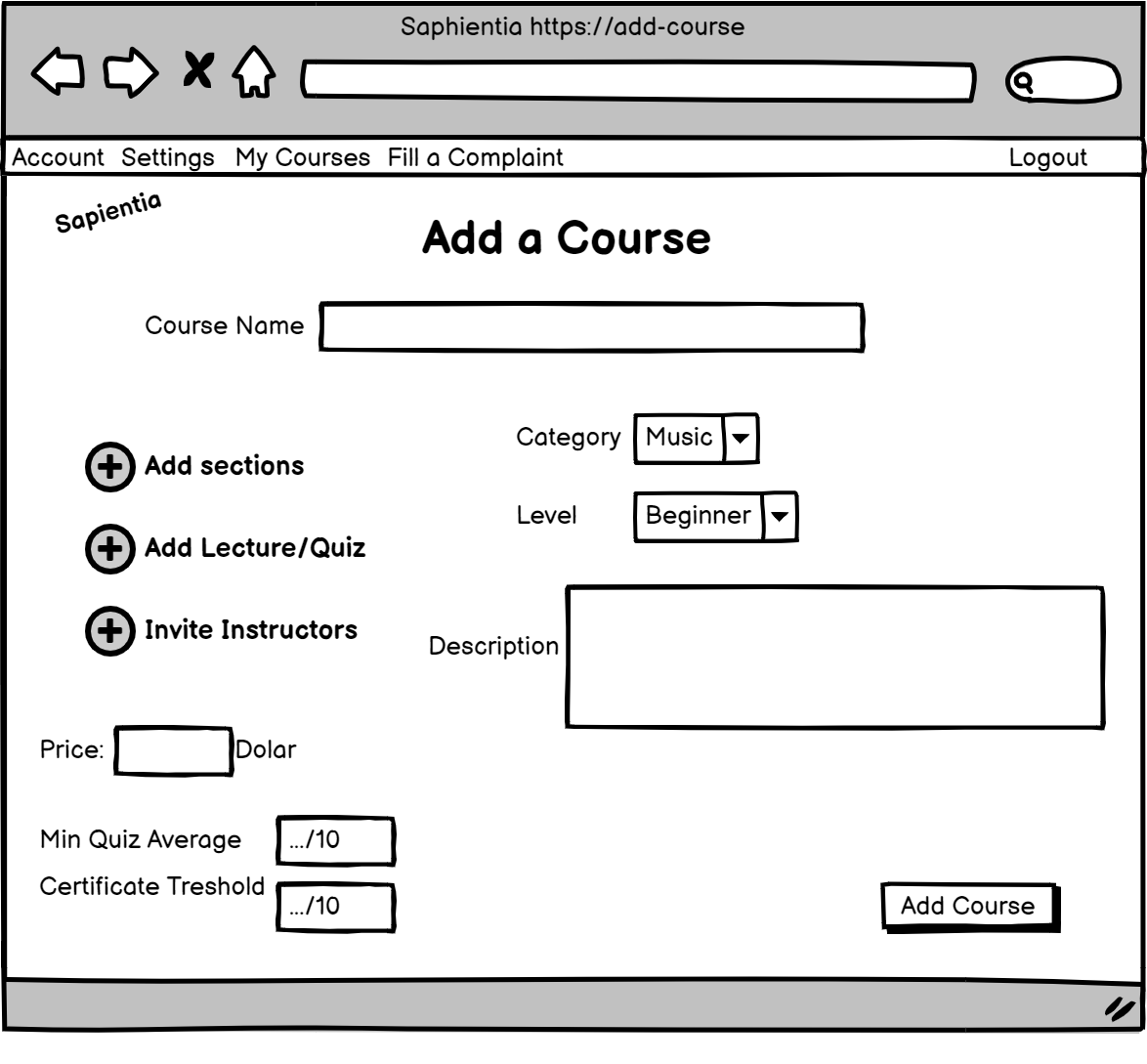
WHERE SID = @SID AND CID = @CID AND content\_num = @content\_num

DECLARE @current\_date date = getdate()

INSERT INTO QnA\_Entry\_Student(CID, SID, subject\_name, text, date)

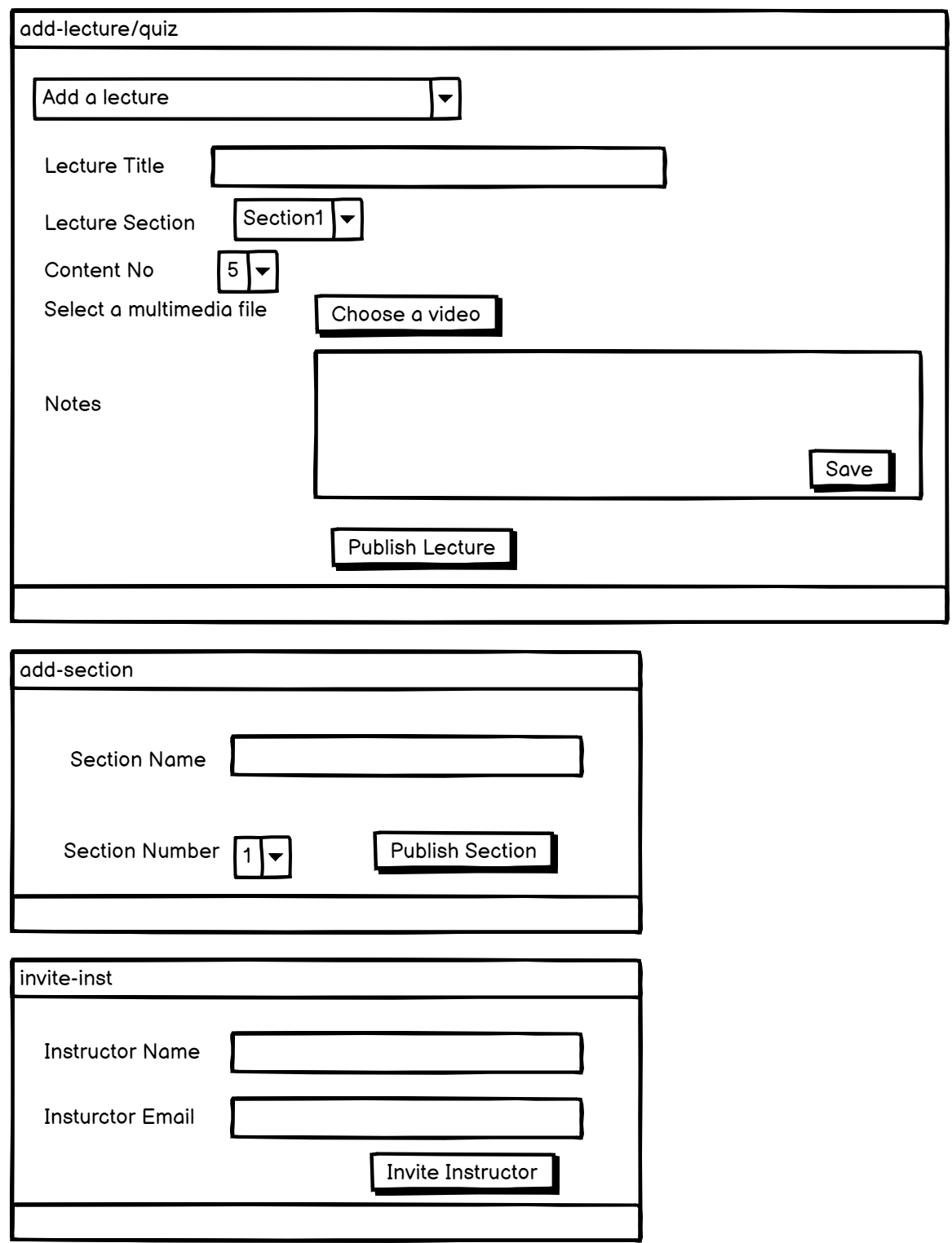
VALUES(@CID, @SID, @subject\_name, @text, @current\_date)

## Add a Course



An instructor can add a course from the **Add Course Page .** From this page, the instructor can determine the sections of the course,the price, minimum quiz average for a student to receive a certificate, lectures or quizzes to the course, add description, choose the category or difficulty level of the course. This page is shown in order to explain additional quiz and invite instructor feature. SQL statements for this page is not given because of this, SQL statements for quiz feature are given in the following parts.

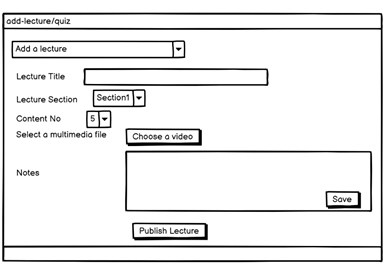
**Add Section / Invite Another Instructor**



While adding a course, as an additional feature, the instructor will be able to add sections to the course and add one more instructor to the course. This page doesn’t have an SQL statement now because it is actually related to the publish course part (second part in functionality document). This page is shown in order to represent additional functionality invite instructor.

**Case**: In order to add an instructor to a course the instructor needs to send an invitation to the other instructor by using his/her name and email. To add a section to the course, the name of the section and the number of the section is needed.

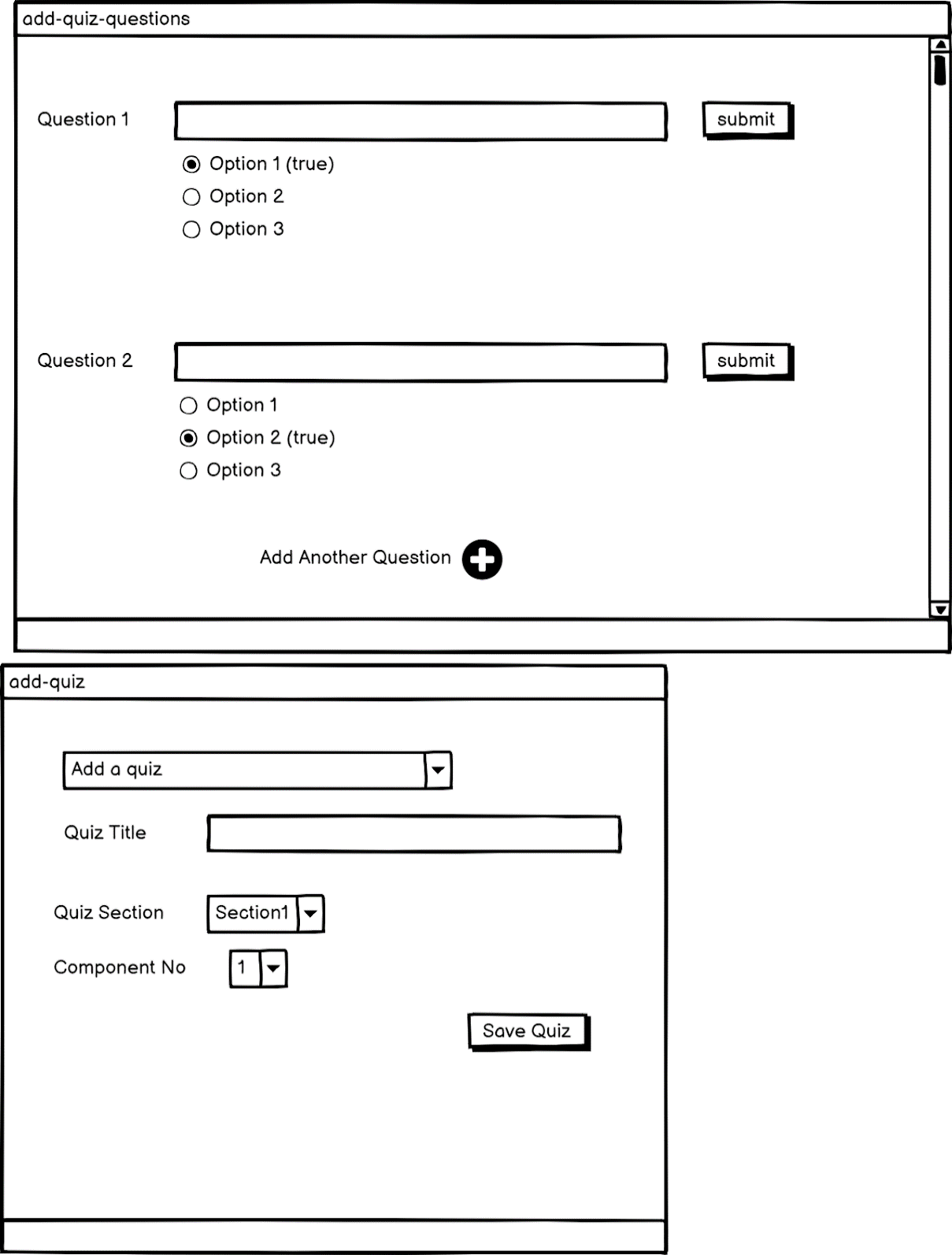
**Add Lecture**



The instructor can either add a lecture or a quiz to the course while adding the course or editing the course. This page doesn’t have an SQL statement now because it is actually related to the publish course part (second part in functionality document).

**Case**: In the example above, the instructor adds a lecture to the course. In order to add a   lecture, the instructor needs to give a title, section and a component number to lecture. Component number indicates the order of the lecture among other lectures and quizzes. Then, the instructor needs to upload a video file and add additional notes to the lecture.

**Add a Quiz**



While adding or editing a course, instructors will be able to add quizzes to their course. First, they will choose whether they want to add a quiz or a lecture to their course. Then, they will give a title to their quiz, choose the section of the quiz and indicate the component number of the quiz. The component number shows the order of the quiz in the lecture/quiz list of the course.

Then they will be directed to the Edit Quiz Page where they can write as many questions as they want.

**Case**: In the above example adding a quiz to a course is shown.

Input: @title, @section, @content\_num

On Save Quiz button pressed:

DECLARE @current\_date date = getdate()

INSERT INTO Quiz (title, section, CID, content\_num, date)

VALUES (@title, @section, @CID, @content\_num, @current\_date)

**Add question**

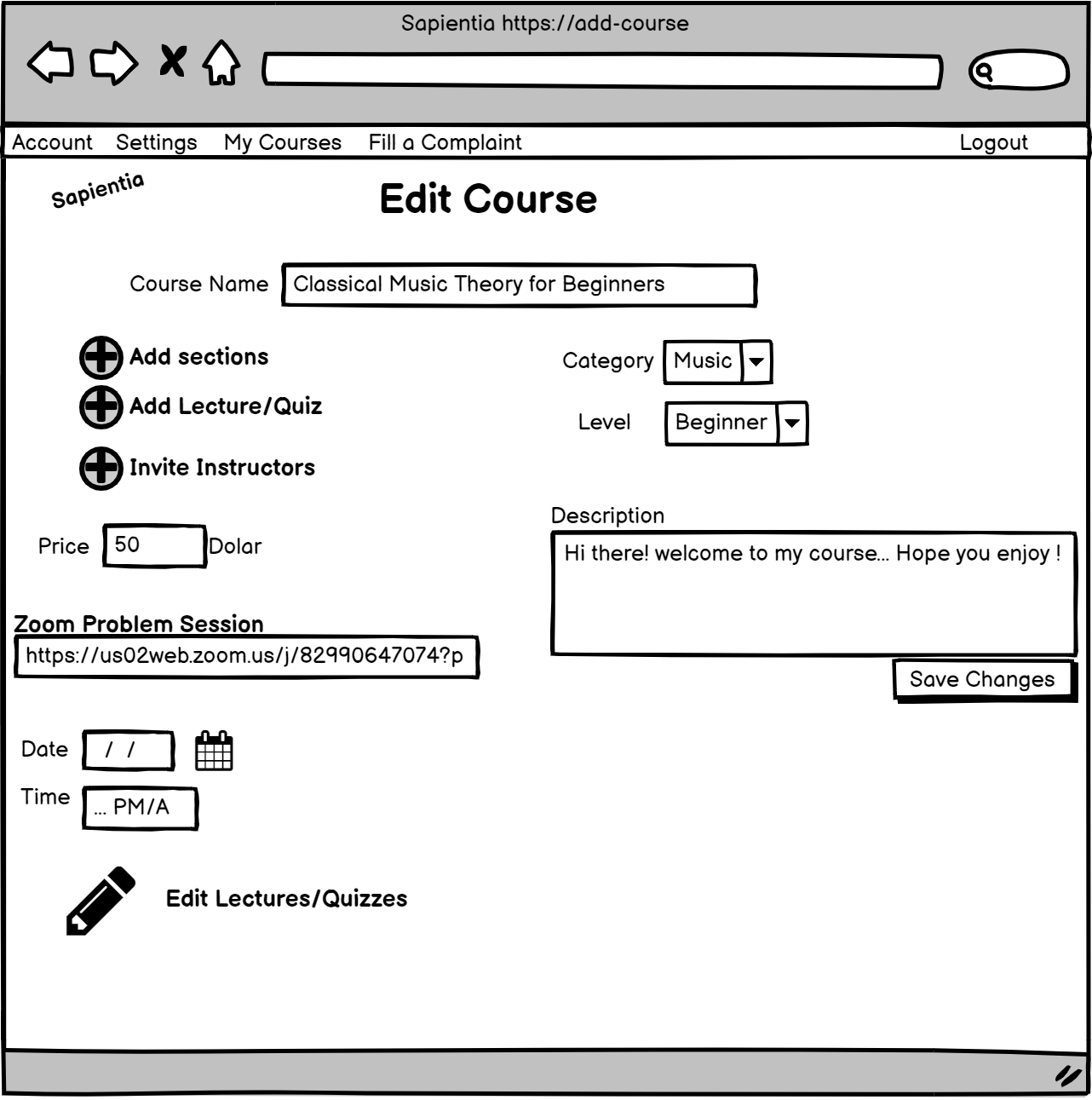
Input: @question\_text, @choice1, @choice2, @choice3, @answer

On Save button pressed:

INSERT INTO Quiz\_Question (CID, content\_num, question\_num, question\_text, choice1, choice2, choice3, answer)

VALUES ( @CID,@content\_num @question\_num, @question\_text, @choice1, @choice2, @choice3, @answer)

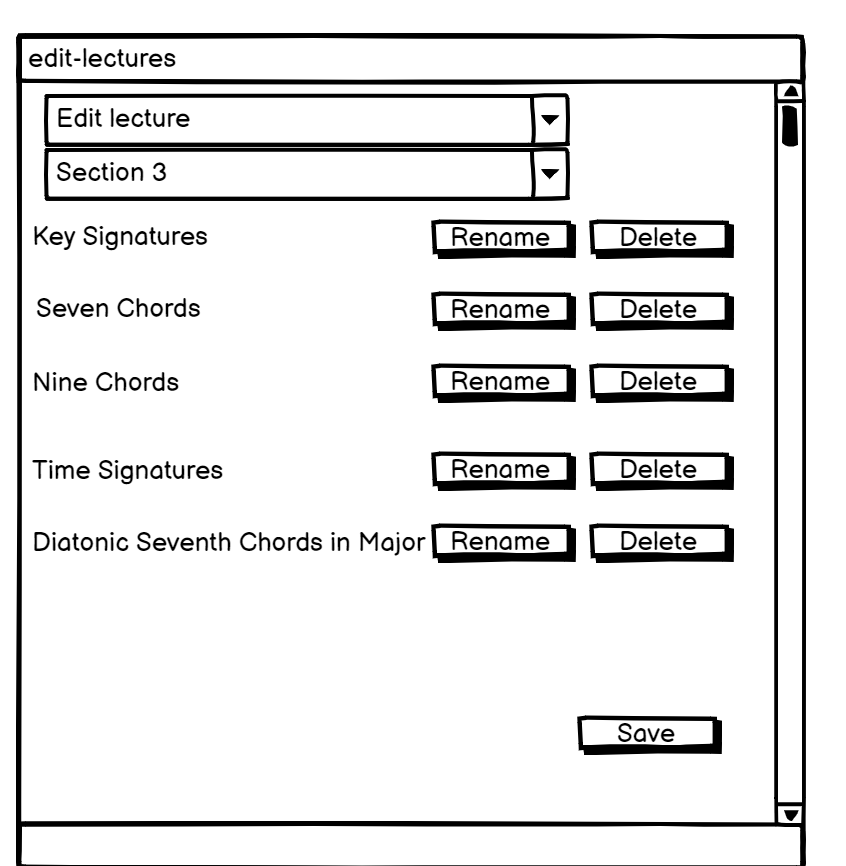
**Edit a Course**



When instructors press the **Edit Lectures/Quizzes** button in the edit course page, they will be directed to the **edit page** where they can choose to edit either quizzes or lectures from the combo box in the top right corner. This page doesn’t have an SQL statement now because it is actually related to the publish course part (second part in functionality document).

**Case:** In the example above, the instructor wants to edit the quizzes in a course where the section of quiz is Section 2. The instructor can edit a quiz by deleting/adding questions to it, changing the questions or quiz’ section . The instructor can also delete the quizzes from a section.

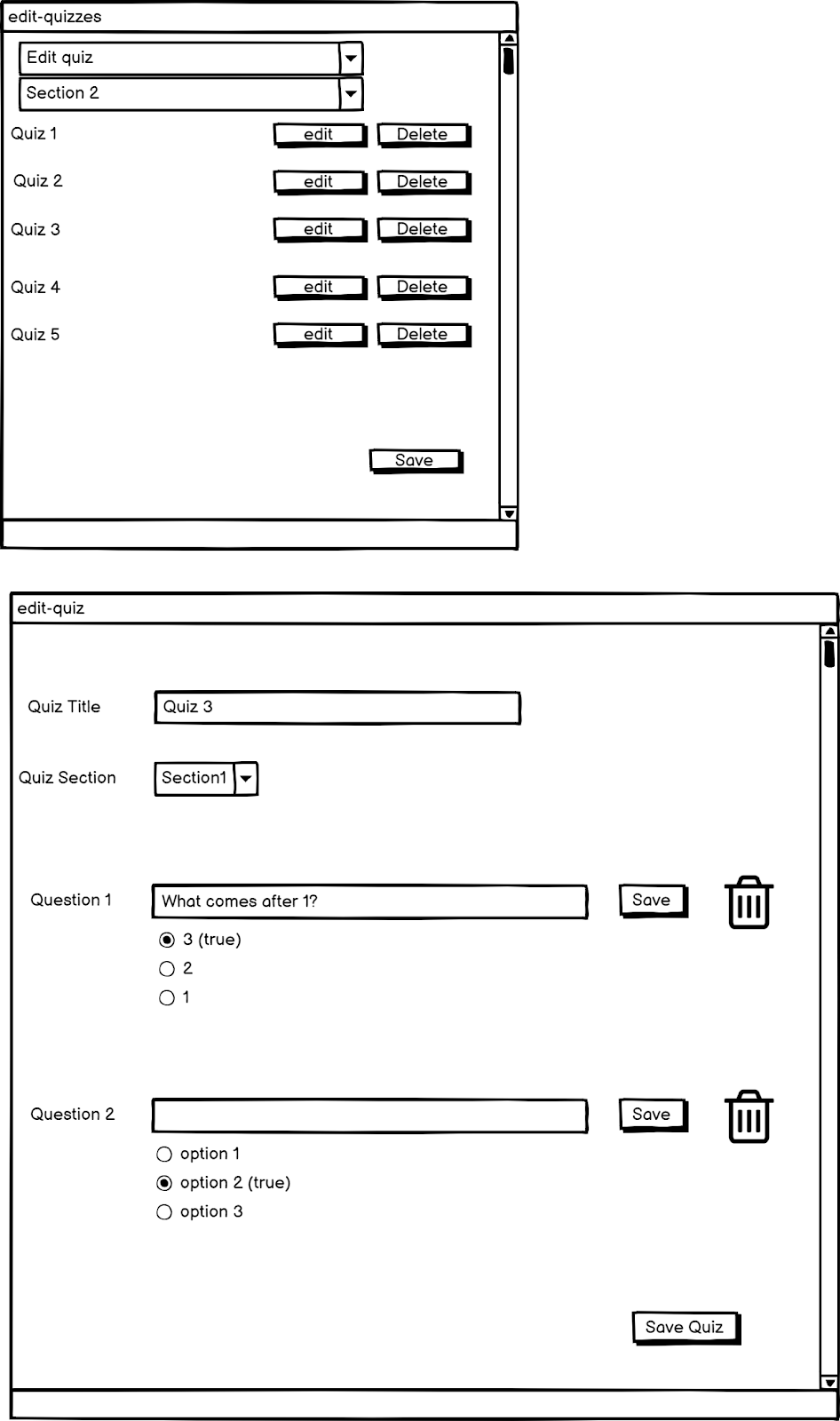
**Edit Lectures**



When instructors press the **Edit Lectures/Quizzes** button in the Edit Course Page, they will be directed to the **edit page** where they can choose to edit either quizzes or lectures from the combo box in the top right corner.

**Case:** In the example above, the instructor wants to edit lectures by either deleting them or renaming them. In the case of changing the lecture video, the instructor needs to first delete the video, then add a video from the Edit Course Page.

**Edit Quizzes**



When instructors press the **Edit Lectures/Quizzes** button in the edit course page, they will be directed to the **edit page** where they can choose to edit either quizzes or lectures from the combo box in the top right corner.

**Case:** In the example above, the instructor wants to edit the quizzes in a course where the section of quiz is Section 2. The instructor can edit a quiz by deleting/adding questions, changing the questions or quiz’s section. The instructor can also delete the quizzes from a section.

Input: @section, @title, @question\_text, @choice1, @choice2, @choice3, @answer

SELECT title

FROM Quiz

WHERE CID = @CID AND section = @section

Delete button pressed:

DELETE FROM Quiz

WHERE CID = @CID AND content\_num = @content\_num

**Edit specific quiz page**

On Save Quiz button pressed:

UPDATE Quiz

SET title = @title, section = @section

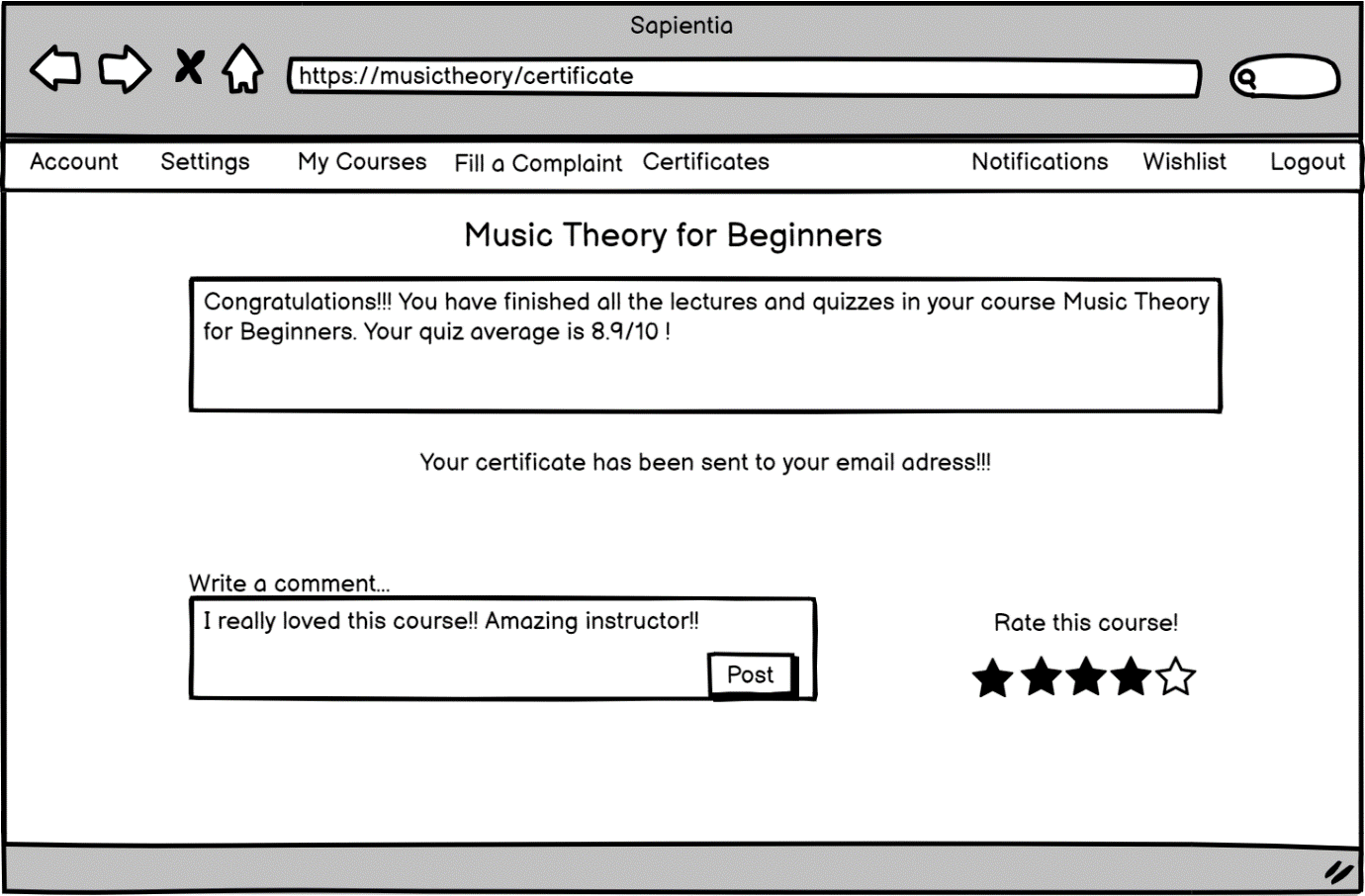
WHERE CID = @CID AND content\_num = @content\_num

On Save button pressed:

UPDATE Quiz\_Question

SET question\_text = @question\_text, choice1 = @choice1, choice2 = @choice2, choice3 = @choice3, answer = @answer

## Certificate



In order to finish a course student must watch all the lectures and have a quiz average above the threshold specified by the instructors. Then, they will receive a certificate from the website indicating that they finished the course. In addition, they will be able to rate the course and leave a comment about the course.

**Case**: In the above example, the student finished the course with the quiz average of 8.9/10, about to leave a comment to the course and rated the course as 4/5.

Inputs: @student\_rating

SELECT COUNT(\*)

FROM Lecture

WHERE CID = @CID

SELECT COUNT(\*)

FROM Quiz

WHERE CID = @CID

SELECT COUNT(\*)

FROM Take\_Lecture NATURAL JOIN Student

WHERE CID = @CID AND isCompleted = ‘true’

SELECT AVG(grade)

FROM Take\_Quiz NATURAL JOIN Student

WHERE CID = @CID

SELECT quiz\_threshold

FROM Course

WHERE CID = @CID

UPDATE Enrolls

SET rating = @student\_rating

WHERE SID = @SID AND CID = @CID

SELECT rating

FROM Course

WHERE CID = @CID

WITH Temp(avg\_rating) as

(SELECT CID, AVG(rating)

FROM Enrolls

GROUP BY CID

HAVING CID = @CID),

DECLARE @avg  NUMERIC(2,1)

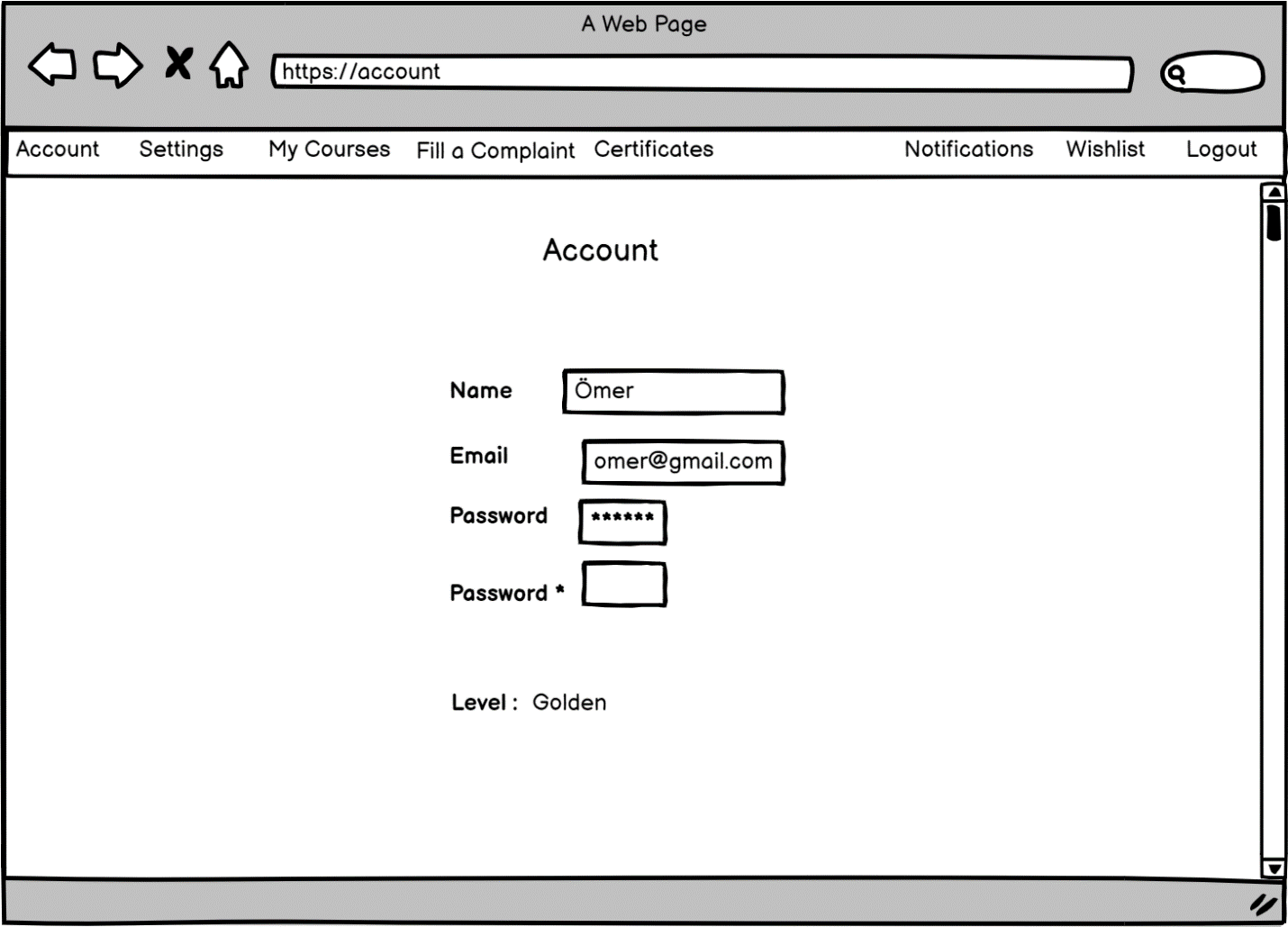
SELECT @avg = avg\_rating

FROM Temp

UPDATE Course

SET rating = @avg

## Student Account



A student is a bronze member by default. Then will be upgraded to silver and golden member if he /she purchases 10 or 20 courses respectively.

A student can reach his/her account by pressing to the Account button in the navigation bar. In there, information about the student is listed.

**Case:** In the example above, the student is a golden member which means he has purchased 25 courses.

SELECT ( name, e\_mail, password, membership\_type )

FROM Student

WHERE SID = @SID