CSCE 367 - Database Management
Term Project: Quiz Database
Group: Hippopotamus
Company: Blaha Inc.
Date: 12 May 2014
PRINT SECTION 2, 3, NEW SQL, 5 testing

Table of Contents

Deliverable 5

Contact Information:

Brandon Horan horanbg@plu.edu (509) 720-3189
Peter W. Joyce joycepw@plu.edu (507) 990-3475
Nick Erickson ericksnc@plu.edu (425) 879-0708

Chair & Secretary Duties:

Deliverable 1
Chair: Brandon Horan
Secretary: Lori Lamm

Deliverable 3
Chair: Lori Lamm
Secretary: Brandon Horan

Deliverable 2
Chair: Peter W. Joyce
Secretary: Nick Erickson

Deliverable 4
Chair: Nick Erickson
Secretary: Peter W. Joyce

Deliverable 5
Chair: Peter W. Joyce
Secretary: Brandon Horan

Agendas & Minutes:

Agenda: (22 April 2014)
Meet with Dr. Blaha to discuss Deliverable 4
Assign tasks for Deliverable 5

Minutes: (22 April 2014)
Meeting began at 2:00pm, Group members present: Brandon, Nick and Peter
Name your Java classes Utility and TestUtility
Each do own use cases; Brandon do open/close DB
Meeting adjourned at 3:15pm

Agenda: (29 April 2014)
Try to finish JDBC
Assign tasks for Deliverable 5

Minutes: (29 April 2014)
Meeting began at 3:00pm, Group members present: Brandon, Peter
Lots of coding and debugging.
Plan on asking Nick about his code tomorrow in class.
Depending on where the project sits tomorrow in class, we may meet on Thursday to finalize.
Meeting adjourned at 7:20pm
Please upload your code into the shared Utilities java file in the google doc folder.

Agenda: (6 May 2014)
Assign/work on tasks for Deliverable 5

Minutes: (6 May 2014)
Meeting began at 2:05pm, Group members present: Brandon, Peter, Nick.
Discussed structure for convenience of web interface.
Will meet Thursday at 5ish until 7 to put together our HTML code.

Friday is for presentation
Use cases: Brandon - MC Question, login, logout, backup, restore
Peter - Check quiz grade, delete course
NIck - update contact info, find available questions
Meeting adjourned at 3:00pm
Please upload your code into the shared Utilities files in the google doc folder.

Agenda: (9 May 2014)
Work on tasks for Deliverable 5

Minutes: (9 May 2014)
Meeting began at 10:30am, Group members present: Brandon, Peter, Nick.
Worked and debugged on the use cases. Brandon took delete course use case.
Meeting adjourned at 3:00pm
Please upload your code into the shared Utilities files in the google doc folder.

Agenda: (12 May 2014)
Work on Deliverable 5--finish ALL programming.
Begin on presentation.

Minutes: (12 May 2014)
Meeting began at 10:30am, Group members present: Brandon, Nick.
Worked and debugged on the use cases.
Meeting adjourned at 11:30am
Please upload your code into the shared Utilities files in the google doc folder.

Agenda: (12 May 2014)
Work on Deliverable 5--finish ALL programming.
Assigned presentation for tomorrow's meeting. Make slides for your use cases.

Minutes: (12 May 2014)
Meeting began at 1:15pm, Group members present: Brandon, Nick, Peter.
Worked and debugged on the use cases.
Nick left at 6:00pm.
Meeting adjourned at 7:30pm when Brandon left.
Please upload your code into the shared Utilities files in the google doc folder.

Agenda: (13 May 2014)
Finish/compile all programming together. --do a last check for error checking
Finish testing for turn in.
Compile and practice our presentation.

Minutes: (13 May 2014)
Meeting began at 2:05pm, Group members present: Brandon, Peter, Nick
Nick arrived at about 2:30pm.
All programming finished.
Began presentation on the google drive.
Worked on finalizing the deliverable.
Peter left at 6:00pm
Meeting adjourned at 7:00pm when Nick left.

Agenda: (13 May 2014)
Compile and practice our presentation.

Minutes: (13 May 2014)
Meeting began at 10:20pm, Group members present: Brandon, Peter, Nick
Meeting adjourned at _____

Required Documents:

Informal Requirement Document

PLU Blaha Inc. Dr. Kenneth Blaha <Blahakd@plu.edu> 253-535-8702

Introduction

Dr. Blaha would like to have an online quiz application to monitor and maintain quizzes in the classes he teaches at PLU. The current way he deploys quizzes is ineffective because it has a lack of efficiency; he has to hand grade each and every quiz and keep his classes' quizzes organized while doing this. In addition, Dr. Blaha has to manually enter grades and calculate quiz statistics i.e. the average, in order to provide a good measure of how well the class is doing.

The database we propose would allow students to take quizzes online, eliminating the need for Dr. Blaha to do any grading. In addition, scores would be recorded, the average would be visible to the students as well as Dr. Blaha, and detailed solutions would be available for viewing after the quiz. This would increase efficiency by making less work for Dr. Blaha when giving a class a quiz, and would save class time because he would no longer need to discuss solutions in class. Finally, it would increase Dr. Blaha's organizational abilities because he would no longer have to have multiple piles of quizzes for different classes to sort through.

Data Requirements and Relationships

The data stored will belong to one of four entities: User, Course, Question and Quiz as described below.

Entities:

USER:

User_Name, Password, Name (First, Last, and middle initial), and email.

COURSE:

Course_Name, Course_Num

QUESTION:

Question_Number, Tags, Instructions, Question_Type, Solution, Point_Value, Prompt

QUIZ:

Quiz_Num, Quiz_Points, Start_Time, End_Time, Time_Limit

INSTRUCTOR:

No attributes

STUDENT:

Student_ID

FITB:

No attributes

TRUE_FALSE:

No attributes

MULT_CHOICE:

Answer_Option, Answer_Letter

MATCHING:

Description, Keywords

Relationships:

Enrolled_In is the relationship between a student and a particular course. It contains an attribute that displays the average quiz grade of that student for that course. Every student will be enrolled in a course and there will be many students per class.

Given_In is the relationship between a course and a particular quiz that is given in that course. Every quiz will be identified by the quiz number, and the course it is given in. Each quiz will also have a Class_Average for the average score on that quiz.

Question_Asked is the relationship that reflects which questions are asked on a particular quiz.

Course_Question is the relationship between a course and the questions in the database that can be asked for that course. Each question will be tagged with the courses the question would be relevant to be asked for.

Takes_Quiz is the relationship between a student and each quiz they take. Includes the score the student got for that quiz. Not every student will take every quiz, so score will be N/A if a quiz is uncompleted.

Student_Answer is the relationship that reflects a student's answer to a particular question on a particular quiz. It will store the Student's answer to the question, and the points the student earned on the question.

Functional Requirements:

An instructor must be able to update the database with a new quiz, having a quiz number, and the start and end times for the quiz. The instructor must be able to create, view, modify, and delete questions, and set question solutions. The instructor will also be able to set time limits for a quiz. The instructor must be able to query a single student's results; in addition, the instructor must be able to query the results for the entire class. This query will show an exhaustive list of students; in addition, there will be other features like average grade, and the ability to sort by best score.

Students will take quizzes and their results will populate the database. Students will be able to see question solutions and check their scores after the quiz. Students will also be able to query the class average for each quiz, as well as their average on quizzes given in a particular course.

Students will be able to sign up for a new account on the quiz database system; the instructor will be given a default instructor account. All users can login to the system, or select an option to reset their password if they have forgotten it. In addition, if a user has forgotten his/her username, then there will be an option to send the user's username to a valid email address in the database. If logged in, all users have the option to change the account password.

User Groups:

Instructor- The instructor will have full access to database functionality. The instructor will be the only person able to view, modify, create or delete questions; add or delete questions, view, create or delete quizzes. The instructor will also have access to a list of students' scores, sortable by last name and by score. The instructor would also be able to edit students' quiz scores in the event grade corrections are needed.

Student- Students are able to take quizzes, answer questions, view quiz time length, as well as view detailed information about each question on past quizzes (after the quiz is no longer available) including viewing quiz scores, average score, class average, question text, student answer, solutions, and point value.

EER Digram

Data Dictionary

Entities:

Entity:	**USER** is an entity that contains information about users of the database. It will store basic information such as username, password, name, etc.	
Attribute	**Description**	**Domain/Key/Null**
Name	Composite attribute that contains the first and last name, and middle initial of the student.	String/no/no

User_Name	The username of the student for login credentials, also the key identifying attribute for the entity.	String/yes/no
Password	The password of the student for login credentials. Must have at least three levels of complexity i.e. upper/lowercase letters, numbers, symbols.	String/no/no
Email	The email address of the student. This is unique.	String/no/no

Entity:	**STUDENT** is an entity that contains information about the students taking quizzes from this database. Each student will have a username and password, and a unique Student_ID number. Subset of User.	
Attribute	**Description**	**Domain/Key/Null**
Student_ID	The student ID number of the student, this is also a candidate key for the entity.	String/yes/no

Entity:	**INSTRUCTOR** is an entity that contains information about the instructor distributing quizzes from this database. The instructor will have a username and password that allows login and control of admin functions for the database. Subset of User.	
Attribute	**Description**	**Domain/Key/Null**
N/A	N/A	N/A

Entity:	**COURSE** is the entity type that stores information about a particular course(class) that an Instructor is teaching.	
Attribute	**Description**	**Domain/Key/Null**
Course_Name	The name of the course quizzes are administered for.	String/yes/no
Course_Num	The ID number of the course being instructed. Part of the composite attribute of Class that is the key attribute.	Integer/yes/no

Entity:	**QUIZ** Quizzes are given in a course, every quiz will be identified by the quiz number, and the course it is given in.	
Attribute	**Description**	**Domain/Key/Null**
Quiz_Points	The total points possible on the quiz, a derived attribute that totals the point values of each question asked on a quiz.	Integer/no/yes
Start_Time	The beginning of the time the quiz was available.	DATETIME/no/no
End_Time	The last point in time the quiz was available.	DATETIME/no/no

Time_Limit	The amount of time students have to take a quiz after they open it.	Integer/no/yes
Quiz_Num	The number of quiz, denotes the order in which a quiz is given in a course.	Integer/yes/no

Entity:	**QUESTION** is the entity type that stores information about individual questions that can be pulled upon to form quizzes.	
Attribute	**Description**	**Domain/Key/Null**
Question_Number	The ID number of the question, no two questions in the database will have the same ID number making this a key attribute.	Integer/yes/no
Tags	The tags will be keywords/phrases that can be used to quickly search for a question/questions of a particular type.	String/no/yes

Question_Type	The type of the question being asked, i.e. multiple choice, matching, True/False, etc. Each question type will be represented by a string, and will be restricted to predefined question types.	enum('TF', 'MC', 'MA', 'FB')/no/no
Instructions	The specific instructions for the question.	String/no/yes
Solution	The solution to the prompted question, the domain will depend on the type of question. For example, True/False will be 'T' or 'F', and multiple choice will have 'A', 'B', or 'C', etc.	String/no/yes
Point_Value	The number of points a particular question is worth. TF - 1 pt, MC - 2 pts, FB - 2pts, MA - 2 pt.	Integer/no/no
Prompt	The text of the question.	String/no/yes

Entity:	**FITB** is the fill in the blank question entity type.	
Attribute	**Description**	**Domain/Key/Null**
N/A	N/A	N/A

Entity:	**TRUE_FALSE** is the true/false question entity type.	
Attribute	**Description**	**Domain/Key/Null**
N/A	N/A	N/A

Entity:	**MULT_CHOICE** is the multiple choice question entity type.	
Attribute	**Description**	**Domain/Key/Null**

Answer_Options	This is a list of possible answers for the question	List<String>/no/yes

Entity:	**MATCHING** is the matching question entity type.	
Attribute	**Description**	**Domain/Key/Null**
Description	This is a description that needs a word(s) matched to it.	String/yes/no
Keywords	This is a list of keywords that may be matched to the description.	List<String>/no/no

Relationships:

Relationship:	**Enrolled_In** is the relationship between a student and a particular course. It contains an attribute that displays the average quiz grade of that student for that course.	
Participating Entities	**Participation**	**Cardinality**
Student,	Student: Total	Student: (1,m)

Course	Course: Total	Course: (1,m)
Attributes	**Description**	**Domain/Null**
Avg_Quiz_Grade	The derived average percent of all of a student's quizzes in a particular course.	Double/yes

Relationship:	**Given_In** is the relationship between a course and a particular quiz that is given in that course.	
Participating Entities	**Participation**	**Cardinality**
Quiz, Course	Quiz: Total Course: Partial	Quiz: (1,m) Course: (0,1)
Attributes	**Description**	**Domain/Null**
Class_Average	The class average on a particular quiz, derived from the Quiz_Score of each student in the class that has taken the Quiz.	Integer/yes

Relationship:	**Question_Asked** the relationship that reflects which questions are asked on particular quiz.	
Participating Entities	**Participation**	**Cardinality**
Question, Quiz	Question: Partial Quiz: Partial	Question: (0,m) Quiz: (0,m)
Attributes	**Description**	**Domain/Null**

N/A	N/A	N/A

Relationship:	**Course_Question** is the relationship between a course and the questions in the database that can be asked for that course.	
Participating Entities	**Participation**	**Cardinality**
Question, Course	Question: Total Course: Partial	Question: (0,m) Course: (0,m)
Attributes	**Description**	**Domain/Null**
N/A	N/A	N/A

Relationship:	**Takes_Quiz** the relationship between a student and each quiz they take. Includes the score the student got for that quiz.	
Participating Entities	**Participation**	**Cardinality**
Student, Quiz	Student: Partial Quiz: Partial	Student: (0,m) Quiz: (0,m)
Attributes	**Description**	**Domain/Null**

Quiz_Score	The amount of points that a student received on a particular quiz in percentages, only after a student completes a quiz. This is derived from the answers that a student gave on the questions on the quiz.	Integer/no

Relationship:	**Student_Answer** the relationship that reflects a student's answer to a question on a particular quiz.	
Participating Entities	**Participation**	**Cardinality**
Question, Quiz, Student	Question: Partial Quiz: Partial Student: Partial	Question: (0,m) Quiz: (0,m) Student: (0,m)
Attributes	**Description**	**Domain/Null**
Student_Answer	The answer the student gave on a particular question for a particular quiz.	String/yes
Points_Earned	The number of points a student earned on a particular question on a particular quiz. The points earned on each problem are determined by comparing the student answer string to the question solution string.	Integer/no

User Groups:

Group	**Student** is the user group of students that will be using the database.
Function	**Description**
Take Quiz	A student will take a quiz that has been assigned by the instructor.
View Past Quizzes	A student will view past quizzes taken, including his/her scores, quiz solutions, and class quiz statistics.

Update Contact Information	A student will be able to update his/her contact information in the database.
Update password	A student will be able to change his/her password.

Group	**Instructor** is the instructor that will be using the database.
Function	**Description**
Manage Questions	The instructor will be able to create and delete questions, set question types and solutions.
Manage Quizzes	The instructor will be able to create and delete quizzes, assign quizzes to a course, add and delete questions on quizzes, and set quiz availability time and time limit.
Manage Quiz Results	The instructor will be able to view quiz results and modify scores as necessary.
Update Contact Information	A student will be able to update his/her contact information in the database.
Update password	A student will be able to change his/her password.

Sitemap

Use Case Scenarios**

Use Case 1)

Name: Create a MC Question (Insert)

Purpose: To create a new multiple choice question in the database to be asked on quizzes.

User Group: Instructor / Blaha

Input: Prompt, optionA, optionB, optionC,
(optionD, optionE) --optional, solution, course(s), tag(s)

Result: The new question will be displayed indicating a successful creation, and any courses specified will have a relationship made between them and the

question. Or an error message will be displayed stating why the operation failed.

Exceptions: Input error if the solution value does not match one of the available options, or prompt is null, etc. Database will prompt user to finish completing the question and highlight the invalid input.

Tables: **Input:** QUESTION(Prompt, Question_Type)
Result: QUESTION(Question_num, Prompt, Solution, Question_Type), MC(Question_Num), MC_Ans_Option(Question_NumAnswer_Option, Answer_Letter)

SQL: Insert into Question(Question_Type, Solution, Prompt)
VALUES ('MC', '@Solution'**, "@Prompt") ***Here and in other places, the @ represents that this data is input by the user.*
Insert into MC
VALUES (@Question_Num)
Insert into MC_Ans_Options
VALUES (@Question_Num, '@Ans_Option', 'a')

Insert into MC_Ans_Options
VALUES (@Question_Num, '@Ans_Option', 'b')

Insert into MC_Ans_Options
VALUES (@Question_Num, '@Ans_Option', 'c') --Note: Similar insert statements are optional for d and e answer options.

Insert into Course_Question
VALUES(@Course_Num, @Question_Num) --Note: There may be multiple tuples of this form, depending on the number of courses input as relevant courses.

Insert into Tags
VALUES(@Question_Num, "@Tag") --Note: There may be multiple tuples of this

form, depending on the number of tags specified.

Testing:

Test 1: Insert

BEFORE INSERTION:

Question table:

Question_N...	Question_Type	Solution	Prompt
1	TF	F	In Java, the statement 'print "I like Java"' displays to the screen...
2	MC	C	A standard ER digram cannot contain information about _____...
3	MA	Attribute	The name given for each column header in the relational data...
4	FB	Heap	The _____ data structure is typically used to implement a ...
5	TF	T	In the relational DB model, attributes are always atomic.
6	MC	A	State all integrity constraints violated by the following update o...
7	MA	while	In Java, what type of loop can always be used for loop operati...
8	FB	Array	A(n) _____ is a useful data structure when handling a list ...
9	TF	T	Select true for bonus points on this quiz!
10	MC	A	A Stack contains these methods:
11	MA	ArrayList	This type of list has faster lookup times.
12	FB	LinkedList	Between ArrayList and LinkedList, _____ is faster when ad...
13	MA	Multivalued	When mapping ER diagrams to the relational DB model, we a...

MC table:

Question_N...
2
6
10

MC_Ans_Options table:

Question_N...	Answer_Option	Answer_Letter
2	Cardinality of relationships	a
2	How many instances of an entity there are in the database	c
2	Participation of relationships	b
2	Super-entity to Sub-entity relationships	d
6	All 4	d
6	D, EI	c
6	D, EI, RI	e
6	K, RI	a
6	K, RI, D	b
10	add, delete, isEmpty, top	c
10	push, pop, empty, peek	a
10	push, pop, isEmpty, peek	b

INSERTION STATEMENTS:

```
Insert into Question(Question_Type, Solution, Prompt)
Values ('MC', 'C', 'Which is the superior operating system?)

Insert into MC Values (14)

Insert into MC_Ans_Options
Values(14, "Windows 8", a)

Insert into MC_Ans_Options
Values(14, "MacOS 10.9", b)

Insert into MC_Ans_Options
Values(14, "Linux", c)
```

AFTER EXECUTION:

Question table:

Question_N...	Question_Type	Solution	Prompt
1	TF	F	In Java, the statement 'print "I like Java"' displays to the screen...
2	MC	C	A standard ER digram cannot contain information about ____...
3	MA	Attribute	The name given for each column header in the relational data...
4	FB	Heap	The _____ data structure is typically used to implement a ...
5	TF	T	In the relational DB model, attributes are always atomic.
6	MC	A	State all integrity constraints violated by the following update o...
7	MA	while	In Java, what type of loop can always be used for loop operati...
8	FB	Array	A(n) _____ is a useful data structure when handling a list ...
9	TF	T	Select true for bonus points on this quiz!
10	MC	A	A Stack contains these methods:
11	MA	ArrayList	This type of list has faster lookup times.
12	FB	LinkedList	Between ArrayList and LinkedList, _____ is faster when ad...
13	MA	Multivalued	When mapping ER diagrams to the relational DB model, we a...
14	MC	C	Which is the superior operating system?

MC table:

MC_Ans_Options table:

Test 1 is exactly what we would expect. This test was successful. Everything inserted correctly.

Test 2: Invalid Question Type

BEFORE INSERTION

Question table:

Question_N...	Question_Type	Solution	Prompt
1	TF	F	In Java, the statement 'print "I like Java"' displays to the screen...
2	MC	C	A standard ER digram cannot contain information about ____...
3	MA	Attribute	The name given for each column header in the relational data...
4	FB	Heap	The _____ data structure is typically used to implement a ...
5	TF	T	In the relational DB model, attributes are always atomic.
6	MC	A	State all integrity constraints violated by the following update o...
7	MA	while	In Java, what type of loop can always be used for loop operati...
8	FB	Array	A(n) _____ is a useful data structure when handling a list ...
9	TF	T	Select true for bonus points on this quiz!
10	MC	A	A Stack contains these methods:
11	MA	ArrayList	This type of list has faster lookup times.
12	FB	LinkedList	Between ArrayList and LinkedList, _____ is faster when ad...
13	MA	Multivalued	When mapping ER diagrams to the relational DB model, we a...
14	MC	C	Which is the superior operating system?

INSERTION STATEMENTS:

```
Insert into Question(Question_Type, Solution, Prompt)
Values ('SA', "I like Computer Science", "Write, 'I like Computer Science' to get extra credit."
```

AFTER EXECUTION:

```
Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 2
SQL State:  42000
ErrorCode: 1064
```

Test 2 was exactly what we would expect. An invalid insertion statement would result in this type of error.

Use Case 2)

Name: Update Contact Information

Purpose: Changes a particular student's contact information to new information input by the user. We will assume the student is logged in.

User Group: Student

Input: At least one of the following: email, F_Name, M_init, L_Name.

Result: The information that was input for modification will replace the old information that was in the database, i.e. the contact information, after modification, for the user updating contact information will be the combination of unchanged attributes of those listed in the possible inputs and data that was input.

Exceptions: If the user enters data that is the same as what it already is, nothing will happen. If the user inputs an invalid email address, the database will ask for a valid email address. If the user inputs multiple characters for the middle initial, only the first character will be saved.

Tables: **Input:** At least one of the following: STUDENT(email), STUDENT(F_Name), STUDENT(M_init), STUDENT(L_Name).
Result: STUDENT(email, F_Name, M_init, L_Name)

SQL: UPDATE Student(

SET STUDENT(email)=@email --there will be a list of statements of the form STUDENT(field), where each field is one of the above fields; email is only an example.
WHERE STUDENT(User_Name)=@User_Name);

Testing:

Test 1: Valid update

BEFORE UPDATE:

Student table:

User_Name	Password	email	F_Name	M_Init	L_Name	Student_ID
AllLanguagesAreFun	Qapla'!!	wishRomulanHadMoreDialets@plu.edu	Nyota		Uhura	00000013
BellaTakesQuizzes	BellaHatesThePhantom42	SinghsSuck@plu.edu	Bella	A	Lithya	00000006
dbuser001	abc123..	dbuser001@plu.edu	Don	B	User	00000000
dbuser002	abc123..	dbuser002@plu.edu	Dan	B	Usee	00000001
dbuser003	IlikeCSCE367	dbuser003@plu.edu	Dirk	B	Underscore	00000002
Defiantismyship	IreallyhatedQo'noS	KlingonAmbassador@plu.edu	Worf		HouseOfMartok	34567890
Dyingsucked	IHateLor123	cantBeatMeAtPoker@plu.edu	Data		IsSmart	87654321
Enterpriseallthewaу	NCC-1701D	Engage!999@plu.edu	Jean	L	Picard	23456789
frankisawesome	ManFrankIsAwesome!	frankisawesome@plu.edu	Frank	I	Awesome	00000005
HarryPotter	IlikeGinny456	harrypotter@plu.edu	Harry	J	Potter	00000008
Hermoineiscool	HarryPotter_IsStupid	GrangerJW@plu.edu	Hermoine	J	Granger	00000010
IAmCaptainKirk	Wishe3ewasinthestates	KirkJamesT@plu.edu	James	T	Kirk	00000011
IamNumber1	RomulanAle2135	IamNumber1@plu.edu	William	T	Riker	98765432
jordanjames	chicken_Bob	jordanjames@plu.edu	Jordan	L	James	00000003
mickey456	zbcdefg1234567!@#$%^&	mickeyAla@plu.edu	Mickey	F	Ala	00000004
RealPerson	No_This_Isn't	ImARealPerson@plu.edu	Real	A	Person	13658739
RonlikesCSCE367	It'sjustaface	RonbWeasley@plu.edu	Ronald	B	Weasley	00000009
SinghLegacy	death2ThePhantom	Singh_Brotherhood@plu.edu	Singh	B	Legacy	00000007
Spock.Enoughsaid	VulcansRock!	longlive&prosper@plu.edu	Mister		Spock	00000012
SuluLikesCSCE367	YesSir!999	wishIWasCaptain@plu.edu	Hikaru		Sulu	12345678

UPDATE STATEMENTS:

```
UPDATE Student
SET email = 'dbuser0001@plu.edu', F_Name = 'Donald'
WHERE Student_ID = 00000000
```

AFTER EXECUTION:

Student table:

User_Name	Password	email	F_Name	M_Init	L_Name	Student_ID
AllLanguagesAreFun	Qapla'!!	wishRomulanHadMoreDialets@plu.edu	Nyota		Uhura	00000013
BellaTakesQuizzes	BellaHatesThePhantom42	SinghsSuck@plu.edu	Bella	A	Lithya	00000006
dbuser001	abc123..	dbuser0001@plu.edu	Donald	B	User	00000000
dbuser002	abc123..	dbuser002@plu.edu	Dan	B	Usee	00000001
dbuser003	IlikeCSCE367	dbuser003@plu.edu	Dirk	B	Underscore	00000002
Defiantismyship	IreallyhatedQo'noS	KlingonAmbassador@plu.edu	Worf		HouseOfMartok	34567890
Dyingsucked	IHateLor123	cantBeatMeAtPoker@plu.edu	Data		IsSmart	87654321
Enterpriseallitheway	NCC-1701D	Engage!999@plu.edu	Jean	L	Picard	23456789
frankisawesome	ManFrankIsAwesome!	frankisawesome@plu.edu	Frank	I	Awesome	00000005
HarryPotter	IlikeGinny456	harrypotter@plu.edu	Harry	J	Potter	00000008
Hermoineiscool	HarryPotter_IsStupid	GrangerJW@plu.edu	Hermoine	J	Granger	00000010
IAmCaptainKirk	Wishe3ewasinthestates	KirkJamesT@plu.edu	James	T	Kirk	00000011
IamNumber1	RomulanAle2135	IamNumber1@plu.edu	William	T	Riker	98765432
jordanjames	chicken_Bob	jordanjames@plu.edu	Jordan	L	James	00000003
mickey456	zbcdefg1234567!@#$%^&	mickeyAla@plu.edu	Mickey	F	Ala	00000004
RealPerson	No_This_Isn't	ImARealPerson@plu.edu	Real	A	Person	13658739
RonlikesCSCE367	It'sjustaface	RonbWeasley@plu.edu	Ronald	B	Weasley	00000009
SinghLegacy	death2ThePhantom	Singh_Brotherhood@plu.edu	Singh	B	Legacy	00000007
Spock.Enoughsaid	VulcansRock!	longlive&prosper@plu.edu	Mister		Spock	00000012
SuluLikesCSCE367	YesSir!999	wishIWasCaptain@plu.edu	Hikaru		Sulu	12345678

Test 1 was exactly what we would expect. The user's email and first name were successfully changed.

Test 2: Duplicate email

BEFORE UPDATE:

Student table:

User_Name	Password	email	F_Name	M_Init	L_Name	Student_ID
AllLanguagesAreFun	Qapla'!!	wishRomulanHadMoreDialets@plu.edu	Nyota		Uhura	00000013
BellaTakesQuizzes	BellaHatesThePhantom42	SinghsSuck@plu.edu	Bella	A	Lithya	00000006
dbuser001	abc123..	dbuser0001@plu.edu	Donald	B	User	00000000
dbuser002	abc123..	dbuser002@plu.edu	Dan	B	Usee	00000001
dbuser003	IlikeCSCE367	dbuser003@plu.edu	Dirk	B	Underscore	00000002
Defiantismyship	IreallyhatedQo'noS	KlingonAmbassador@plu.edu	Worf		HouseOfMartok	34567890
Dyingsucked	IHateLor123	cantBeatMeAtPoker@plu.edu	Data		IsSmart	87654321
Enterpriseallitheway	NCC-1701D	Engage!999@plu.edu	Jean	L	Picard	23456789
frankisawesome	ManFrankIsAwesome!	frankisawesome@plu.edu	Frank	I	Awesome	00000005
HarryPotter	IlikeGinny456	harrypotter@plu.edu	Harry	J	Potter	00000008
Hermoineiscool	HarryPotter_IsStupid	GrangerJW@plu.edu	Hermoine	J	Granger	00000010
IAmCaptainKirk	Wishe3ewasinthestates	KirkJamesT@plu.edu	James	T	Kirk	00000011
IamNumber1	RomulanAle2135	IamNumber1@plu.edu	William	T	Riker	98765432
jordanjames	chicken_Bob	jordanjames@plu.edu	Jordan	L	James	00000003
mickey456	zbcdefg1234567!@#$%^&	mickeyAla@plu.edu	Mickey	F	Ala	00000004
RealPerson	No_This_Isn't	ImARealPerson@plu.edu	Real	A	Person	13658739
RonlikesCSCE367	It'sjustaface	RonbWeasley@plu.edu	Ronald	B	Weasley	00000009
SinghLegacy	death2ThePhantom	Singh_Brotherhood@plu.edu	Singh	B	Legacy	00000007
Spock.Enoughsaid	VulcansRock!	longlive&prosper@plu.edu	Mister		Spock	00000012
SuluLikesCSCE367	YesSir!999	wishIWasCaptain@plu.edu	Hikaru		Sulu	12345678

UPDATE STATEMENTS:

```
UPDATE Student
Set email = 'dbuser0001@plu.edu'
WHERE Student_ID = 00000001
```

AFTER EXECUTION:

```
Error: Duplicate entry 'dbuser0001@plu.edu' for key 'email'
SQL State: 23000
ErrorCode: 1062
```

Test 2 was exactly what we would expect. As email is unique, there cannot be more than one entry with the same email.

Use Case 3)

Name: Check Quiz Grade

Purpose: Finds the student's grade for a quiz and shows the class average on the quiz for comparison.

User Group: Student

Input: Quiz_num, Course_num

Result: The student's score on the quiz out of the total score, the percentage the student completed successfully on the quiz, class average on the quiz are all displayed.

Exceptions: A student can not access this feature on a particular quiz until after the quiz is no longer available. If the student chooses a quiz that s/he has not taken, or a quiz that does not exist, no result will be displayed and the database will prompt the user to choose another quiz.

Tables: **Input:** QUIZ(quiz_num), COURSE(course_num), STUDENT(Student_ID)
Result: QUIZ(quiz_num, quiz_points), COURSE(course_num) GIVEN_IN(class_average), TAKES_QUIZ(quiz_score)

SQL:

1. SQL gets name and student score

```
(Select concat(F_name,' ',L_name)as name, sum(PT_value) as score
From Student_Answer as sa, Student as s, Question as q, Question_Type as qt
Where sa.question_num=q.question_num and q.question_type=qt.question_type and
q.solution=sa.student_answer and sa.Student_ID=s.Student_ID and sa.quiz_num=1 and
sa.course_num=144 group by s.User_Name)
```

2. SQL gets total points possible and then classScore and classPossible to derive class average.

```
Select total, sum(score), sum(total)
From (Select sa.Student_ID, sum(PT_value) as total
        From Student_Answer as sa, Student as s, Question as q, Question_Type as qt
        Where sa.question_num=q.question_num and q.question_type=qt.question_type
and sa.Student_ID=s.Student_ID and sa.quiz_num=@quiz_num and
sa.course_num=@course_num group by s.Student_ID) as a,
        (Select sa.Student_ID, sum(PT_value) as score
        From Student_Answer as sa, Student as s, Question as q, Question_Type as qt
        Where sa.question_num=q.question_num and q.question_type=qt.question_type
and q.solution=sa.student_answer and sa.Student_ID=s.Student_ID and
sa.quiz_num=@quiz_num and sa.course_num=@course_num group by s.Student_ID) as b
where a.student_ID=b.student_ID
```

3. SQL to check check the Course List for a user, instructor will see all courses, a student will see courses they are enrolled in.

```
Select course_num
From Enrolled_In
Where Student_ID=@StudentID *leave this line out for instructor
```

4. SQL to check the available Quiz List for a course

```
Select quiz_num
From Quiz
Where course_num=@course_num
```

Test 1: Total Points

BEFORE REPORT

Student table:

User_Name	Password	email	F_Name	M_Init	L_Name	Student_ID
AllLanguagesAreFun	Qapla'!!	wishRomulanHadMoreDialets@plu.edu	Nyota		Uhura	00000013
BellaTakesQuizzes	BellaHatesThePhantom42	SinghsSuck@plu.edu	Bella	A	Lithya	00000006
dbuser001	abc123..	dbuser0001@plu.edu	Donald	B	User	00000000
dbuser002	abc123..	dbuser002@plu.edu	Dan	B	Usee	00000001
dbuser003	IlikeCSCE367	dbuser003@plu.edu	Dirk	B	Underscore	00000002
Defiantismyship	IreallyhatedQo'noS	KlingonAmbassador@plu.edu	Worf		HouseOfMartok	34567890
Dyingsucked	IHateLor123	cantBeatMeAtPoker@plu.edu	Data		IsSmart	87654321
Enterpriseallthewaay	NCC-1701D	Engage!999@plu.edu	Jean	L	Picard	23456789
frankisawesome	ManFrankIsAwesome!	frankisawesome@plu.edu	Frank	I	Awesome	00000005
HarryPotter	IlikeGinny456	harrypotter@plu.edu	Harry	J	Potter	00000008
Hermoineiscool	HarryPotter_IsStupid	GrangerJW@plu.edu	Hermoine	J	Granger	00000010
IAmCaptainKirk	Wishe3ewasinthestates	KirkJamesT@plu.edu	James	T	Kirk	00000011
IamNumber1	RomulanAle2135	IamNumber1@plu.edu	William	T	Riker	98765432
jordanjames	chicken_Bob	jordanjames@plu.edu	Jordan	L	James	00000003
mickey456	zbcdefg1234567!@#$%^&	mickeyAla@plu.edu	Mickey	F	Ala	00000004
RealPerson	No_This_Isn't	ImARealPerson@plu.edu	Real	A	Person	13658739
RonlikesCSCE367	It'sjustaface	RonbWeasley@plu.edu	Ronald	B	Weasley	00000009
SinghLegacy	death2ThePhantom	Singh_Brotherhood@plu.edu	Singh	B	Legacy	00000007
Spock.Enoughsaid	VulcansRock!	longlive&prosper@plu.edu	Mister		Spock	00000012
SuluLikesCSCE367	YesSir!999	wishIWasCaptain@plu.edu	Hikaru		Sulu	12345678

Question table:

Question_N...	Question_Type	Solution	Prompt
1	TF	F	In Java, the statement 'print "I like Java"' displays to the screen...
2	MC	C	A standard ER digram cannot contain information about _____...
3	MA	Attribute	The name given for each column header in the relational data...
4	FB	Heap	The _____ data structure is typically used to implement a ...
5	TF	T	In the relational DB model, attributes are always atomic.
6	MC	A	State all integrity constraints violated by the following update o...
7	MA	while	In Java, what type of loop can always be used for loop operati...
8	FB	Array	A(n) _____ is a useful data structure when handling a list ...
9	TF	T	Select true for bonus points on this quiz!
10	MC	A	A Stack contains these methods:
11	MA	ArrayList	This type of list has faster lookup times.
12	FB	LinkedList	Between ArrayList and LinkedList, _____ is faster when ad...
13	MA	Multivalued	When mapping ER diagrams to the relational DB model, we a...
14	MC	C	Which is the superior operating system?

Student_Answer table:

Question_Type table:

QUERY STATEMENTS:

AFTER EXECUTION:

This test was successful. There were 3 total possible points on this quiz.

Test 2: Student Score

BEFORE REPORT

Student table:

User_Name	Password	email	F_Name	M_Init	L_Name	Student_ID
AllLanguagesAreFun	Qapla'!!	wishRomulanHadMoreDialets@plu.edu	Nyota		Uhura	00000013
BellaTakesQuizzes	BellaHatesThePhantom42	SinghsSuck@plu.edu	Bella	A	Lithya	00000006
dbuser001	abc123..	dbuser0001@plu.edu	Donald	B	User	00000000
dbuser002	abc123..	dbuser002@plu.edu	Dan	B	Usee	00000001
dbuser003	IlikeCSCE367	dbuser003@plu.edu	Dirk	B	Underscore	00000002
Defiantismyship	IreallyhatedQo'noS	KlingonAmbassador@plu.edu	Worf		HouseOfMartok	34567890
Dyingsucked	IHateLor123	cantBeatMeAtPoker@plu.edu	Data		IsSmart	87654321
Enterpriseallitheway	NCC-1701D	Engage!999@plu.edu	Jean	L	Picard	23456789
frankisawesome	ManFrankIsAwesome!	frankisawesome@plu.edu	Frank	I	Awesome	00000005
HarryPotter	IlikeGinny456	harrypotter@plu.edu	Harry	J	Potter	00000008
Hermoineiscool	HarryPotter_IsStupid	GrangerJW@plu.edu	Hermoine	J	Granger	00000010
IAmCaptainKirk	Wishe3ewasinthestates	KirkJamesT@plu.edu	James	T	Kirk	00000011
IamNumber1	RomulanAle2135	IamNumber1@plu.edu	William	T	Riker	98765432
jordanjames	chicken_Bob	jordanjames@plu.edu	Jordan	L	James	00000003
mickey456	zbcdefg1234567!@#$%^&	mickeyAla@plu.edu	Mickey	F	Ala	00000004
RealPerson	No_This_Isn't	ImARealPerson@plu.edu	Real	A	Person	13658739
RonlikesCSCE367	It'sjustaface	RonbWeasley@plu.edu	Ronald	B	Weasley	00000009
SinghLegacy	death2ThePhantom	Singh_Brotherhood@plu.edu	Singh	B	Legacy	00000007
Spock.Enoughsaid	VulcansRock!	longlive&prosper@plu.edu	Mister		Spock	00000012
SuluLikesCSCE367	YesSir!999	wishIWasCaptain@plu.edu	Hikaru		Sulu	12345678

Question table:

Question_N...	Question_Type	Solution	Prompt
1	TF	F	In Java, the statement 'print "I like Java"' displays to the screen...
2	MC	C	A standard ER digram cannot contain information about ____...
3	MA	Attribute	The name given for each column header in the relational data...
4	FB	Heap	The _____ data structure is typically used to implement a ...
5	TF	T	In the relational DB model, attributes are always atomic.
6	MC	A	State all integrity constraints violated by the following update o...
7	MA	while	In Java, what type of loop can always be used for loop operati...
8	FB	Array	A(n) _____ is a useful data structure when handling a list ...
9	TF	T	Select true for bonus points on this quiz!
10	MC	A	A Stack contains these methods:
11	MA	ArrayList	This type of list has faster lookup times.
12	FB	LinkedList	Between ArrayList and LinkedList, _____ is faster when ad...
13	MA	Multivalued	When mapping ER diagrams to the relational DB model, we a...
14	MC	C	Which is the superior operating system?

Student_Answer table:

Question_Type table:

QUERY STATEMENTS:

AFTER EXECUTION

This test was successful. It accurately reflected the student's score on the quiz.

Test 3:

BEFORE REPORT:

Student Score (derived table) table:

QUERY STATEMENTS:

AFTER EXECUTION:

This test was successful. The quiz average is displayed clearly and is checked against student scores manually.

4)

Name: Delete course

Purpose: Deletes a course from the roster.

User Group: Instructor / Blaha

Input: Course_Num

Result: The specific course will be deleted. All quiz scores will be deleted, as well as all students enrolled in the course who are not also enrolled in other courses. If desired, all quizzes from the course are also deleted. All questions are unaffected.

Exceptions: If there is at least one student in the course, the course cannot be removed unless all students have been removed first.

Tables: **Input:** COURSE(Course_Num)
Result: ENROLLED_IN(Student_ID, Course_Num), STUDENT_ANSWER(Student_Answer, Student_ID, Quiz_Num, Course_Num, Question_Num), COURSE_QUESTION(Course_Num, Question_Num), TAKES_QUIZ(Quiz_Num, Course_Num, Student_ID), COURSE(Course_Num, Course_Name), QUIZ(Quiz_Num, Course_Num, Start_Time, End_Time, Time_Limit), STUDENT(User_Name, Password, email, F_Name, M_Init, L_Name, Student_ID)

SQL:

```
Delete from Student where Student_ID in
        (Select Student_ID
         From Enrolled_In
         Where Course_Num = @Course_Num and Student_ID not in
                (Select Student_ID
                 From Enrolled_In
                 Where Course_Num <> @Course_Num)
        )
Delete from Course where Course_Num = @Course_Num
```

Testing*:

Test 1: Valid delete

BEFORE DELETION:

Student table:

**Testing for this use case was done when there were only 13 questions in the database.*

User_Name	Password	email	F_Name	M_Init	L_Name	Student_ID
AllLanguagesAreFun	Qapla'!!	wishRomulanHadMoreDialets@plu.edu	Nyota		Uhura	00000013
BellaTakesQuizzes	BellaHatesThePhantom42	SinghsSuck@plu.edu	Bella	A	Lithya	00000006
dbuser001	abc123..	dbuser0001@plu.edu	Donald	B	User	00000000
dbuser002	abc123..	dbuser002@plu.edu	Dan	B	Usee	00000001
dbuser003	IlikeCSCE367	dbuser003@plu.edu	Dirk	B	Underscore	00000002
Defiantismyship	IreallyhatedQo'noS	KlingonAmbassador@plu.edu	Worf		HouseOfMartok	34567890
Dyingsucked	IHateLor123	cantBeatMeAtPoker@plu.edu	Data		IsSmart	87654321
Enterpriseallitheway	NCC-1701D	Engage!999@plu.edu	Jean	L	Picard	23456789
frankisawesome	ManFrankIsAwesome!	frankisawesome@plu.edu	Frank	I	Awesome	00000005
HarryPotter	IlikeGinny456	harrypotter@plu.edu	Harry	J	Potter	00000008
Hermoineiscool	HarryPotter_IsStupid	GrangerJW@plu.edu	Hermoine	J	Granger	00000010
IAmCaptainKirk	Wishe3ewasinthestates	KirkJamesT@plu.edu	James	T	Kirk	00000011
IamNumber1	RomulanAle2135	IamNumber1@plu.edu	William	T	Riker	98765432
jordanjames	chicken_Bob	jordanjames@plu.edu	Jordan	L	James	00000003
mickey456	zbcdefg1234567!@#$%^&	mickeyAla@plu.edu	Mickey	F	Ala	00000004
RealPerson	No_This_Isn't	ImARealPerson@plu.edu	Real	A	Person	13658739
RonlikesCSCE367	It'sjustaface	RonbWeasley@plu.edu	Ronald	B	Weasley	00000009
SinghLegacy	death2ThePhantom	Singh_Brotherhood@plu.edu	Singh	B	Legacy	00000007
Spock.Enoughsaid	VulcansRock!	longlive&prosper@plu.edu	Mister		Spock	00000012
SuluLikesCSCE367	YesSir!999	wishIWasCaptain@plu.edu	Hikaru		Sulu	12345678

Course table:

Enrolled_In table:

Student_Answer table:

Course_Question table:

Quiz table:

Takes_Quiz table:

DELETION STATEMENTS:

AFTER EXECUTION:

Student table:

Course table:

Enrolled_In table:

Student_Answer table:

Course_Question table:

Quiz_table:

Takes_Quiz table:

This test was successful. We expected any reference to 144 students to disappear, and they have. The only tricky part was deleting students that were enrolled in no other classes than 144, which is why there were 2 statements.

Test 2: Invalid Course Number

BEFORE DELETION:

**Testing for this use case was done when there were only 13 questions in the database.*

Student table:

User_Name	Password	email	F_Name	M_Init	L_Name	Student_ID
AllLanguagesAreFun	Qapla'!!	wishRomulanHadMoreDialets@plu.edu	Nyota		Uhura	00000013
BellaTakesQuizzes	BellaHatesThePhantom42	SinghsSuck@plu.edu	Bella	A	Lithya	00000006
dbuser001	abc123..	dbuser0001@plu.edu	Donald	B	User	00000000
dbuser002	abc123..	dbuser002@plu.edu	Dan	B	Usee	00000001
dbuser003	IlikeCSCE367	dbuser003@plu.edu	Dirk	B	Underscore	00000002
Defiantismyship	IreallyhatedQo'noS	KlingonAmbassador@plu.edu	Worf		HouseOfMartok	34567890
Dyingsucked	IHateLor123	cantBeatMeAtPoker@plu.edu	Data		IsSmart	87654321
Enterpriseallitheway	NCC-1701D	Engage!999@plu.edu	Jean	L	Picard	23456789
frankisawesome	ManFrankIsAwesome!	frankisawesome@plu.edu	Frank	I	Awesome	00000005
HarryPotter	IlikeGinny456	harrypotter@plu.edu	Harry	J	Potter	00000008
Hermoineiscool	HarryPotter_IsStupid	GrangerJW@plu.edu	Hermoine	J	Granger	00000010
IAmCaptainKirk	Wishe3ewasinthestates	KirkJamesT@plu.edu	James	T	Kirk	00000011
IamNumber1	RomulanAle2135	IamNumber1@plu.edu	William	T	Riker	98765432
jordanjames	chicken_Bob	jordanjames@plu.edu	Jordan	L	James	00000003
mickey456	zbcdefg1234567!@#$%^&	mickeyAla@plu.edu	Mickey	F	Ala	00000004
RealPerson	No_This_Isn't	ImARealPerson@plu.edu	Real	A	Person	13658739
RonlikesCSCE367	It'sjustaface	RonbWeasley@plu.edu	Ronald	B	Weasley	00000009
SinghLegacy	death2ThePhantom	Singh_Brotherhood@plu.edu	Singh	B	Legacy	00000007
Spock.Enoughsaid	VulcansRock!	longlive&prosper@plu.edu	Mister		Spock	00000012
SuluLikesCSCE367	YesSir!999	wishIWasCaptain@plu.edu	Hikaru		Sulu	12345678

Course table:

Enrolled_In table:

Student_Answer table:

Course_Question table:

Quiz table:

Takes_Quiz table:

DELETION STATEMENTS:

AFTER EXECUTION:

Student table:

User_Name	Password	email	F_Name	M_Init	L_Name	Student_ID
AllLanguagesAreFun	Qapla'!!	wishRomulanHadMoreDialets@plu.edu	Nyota		Uhura	00000013
BellaTakesQuizzes	BellaHatesThePhantom42	SinghsSuck@plu.edu	Bella	A	Lithya	00000006
dbuser001	abc123..	dbuser0001@plu.edu	Donald	B	User	00000000
dbuser002	abc123..	dbuser002@plu.edu	Dan	B	Usee	00000001
dbuser003	IlikeCSCE367	dbuser003@plu.edu	Dirk	B	Underscore	00000002
Defiantismyship	IreallyhatedQo'noS	KlingonAmbassador@plu.edu	Worf		HouseOfMartok	34567890
Dyingsucked	IHateLor123	cantBeatMeAtPoker@plu.edu	Data		IsSmart	87654321
Enterpriseallthewaу	NCC-1701D	Engage!999@plu.edu	Jean	L	Picard	23456789
frankisawesome	ManFrankIsAwesome!	frankisawesome@plu.edu	Frank	I	Awesome	00000005
HarryPotter	IlikeGinny456	harrypotter@plu.edu	Harry	J	Potter	00000008
Hermoineiscool	HarryPotter_IsStupid	GrangerJW@plu.edu	Hermoine	J	Granger	00000010
IAmCaptainKirk	Wishe3ewasinthestates	KirkJamesT@plu.edu	James	T	Kirk	00000011
IamNumber1	RomulanAle2135	IamNumber1@plu.edu	William	T	Riker	98765432
jordanjames	chicken_Bob	jordanjames@plu.edu	Jordan	L	James	00000003
mickey456	zbcdefg1234567!@#$%^&	mickeyAla@plu.edu	Mickey	F	Ala	00000004
RealPerson	No_This_Isn't	ImARealPerson@plu.edu	Real	A	Person	13658739
RonlikesCSCE367	It'sjustaface	RonbWeasley@plu.edu	Ronald	B	Weasley	00000009
SinghLegacy	death2ThePhantom	Singh_Brotherhood@plu.edu	Singh	B	Legacy	00000007
Spock.Enoughsaid	VulcansRock!	longlive&prosper@plu.edu	Mister		Spock	00000012
SuluLikesCSCE367	YesSir!999	wishIWasCaptain@plu.edu	Hikaru		Sulu	12345678

Course table:

Enrolled_In table:

Student_Answer table:

Course_Question table:

Quiz table:

Takes_Quiz table:

This test was successful. There is nothing illegal about telling the database to delete a course that does not exist; it is simply that nothing happens upon execution of these statements, as the tables shown demonstrate.

5)

Name: Find Available Questions

Purpose: Provides the Instructor with a list of questions that are available to be asked on a quiz, based on the courses they are relevant for and tags the instructor has defined.

User Group: Instructor / Blaha

Input: Course_Num, Tag(s)

Result: A list of the Questions that match the course and any additional tags if present

Exceptions: If the course number does not match a course in the database a list of possible courses will be displayed with an error message. If the combination of course number and optional tags returns no questions, an error message will display the attempted search terms, and allow for an additional search attempts.

Tables: **Input:** COURSE(Course_Num), TAGS(tag)+*

**The + by TAGS(tag) indicates that the instructor could add multiple tags at once if desired.*

Result: QUESTION(Question_Num, Prompt)

SQL:

```
SELECT Question_Type, Prompt
FROM Question as Q join Tags on Q.Question_Num =
Tags.Question_Num, Course_Question as CQ
WHERE Q.Question_Num = CQ.Question_Num and (Tags.tag =
"@Tag" or Tags.tag = "@Tag") and
Course_Num = @Course_Num        --Note: The number of tags will vary
                                depending on the number of tags
                                input, each tag separated by an or.
                                The Java code will enforce this.
```

Testing:

Test 1: Select tags that don't have questions associated with the input course number.

BEFORE REPORT

Question table:

Tags table:

Course_Question table:

QUERY STATEMENTS:

AFTER EXECUTION:

This test was successful. We expected that even though the ER tag was there, that only 270 questions would be ‘available’.

Test 2: Include the ER diagram questions from test 1.

BEFORE REPORT

Question table:

Tags table:

Course_Question table:

QUERY STATEMENTS:

AFTER EXECUTION:

This test was successful. The result included all of the results of test 1 in addition to questions having an ER diagrams tag.

Relational Model

Instructor

Field Name	Description	Domain	PK	Constraints
User_Name	The user's username for the database.	VARCHAR(30)	PK	CHECK
Password	The user's password for a given username.	VARCHAR(30)		NOT NULL, CHECK
email	The user's email address.	VARCHAR	K1	CHECK, UNIQUE, NOT NULL
F_Name	The user's first name	VARCHAR		NOT NULL
M_init	The user's middle initial	CHAR(1)		
L_Name	The user's last name	VARCHAR		NOT NULL

Description:
This relation serves the purpose of storing the instructor's basic information. INSTRUCTOR is a sub-entity the USER entity represented using option 8B as described in lecture notes Week5 - 2.pdf on slide 2. <http://www.cs.plu.edu/courses/csce367/current/lectures/Week5-2.pdf>

Foreign keys:
N/A

Constraints:
The CHECK constraint on User_Name is to ensure that the username is at least 8 characters in length.
The CHECK constraint on password is to ensure that there are at least three levels of complexity in a user's password.
The CHECK constraint on email is to force that an email address must take a valid form.

Functional Dependencies:
User_Name → Password, email, F_Name, M_init, L_Name
email → User_Name, Password, F_Name, M_init, L_Name

Student

Field Name	Description	Domain	PK	Constraints
User_Name	The user's username for the database.	VARCHAR(30)	PK	CHECK
Password	The user's password for a given username.	VARCHAR(30)		NOT NULL, CHECK
email	The user's email address.	VARCHAR	K1	CHECK, UNIQUE, NOT NULL
F_Name	The user's first name	VARCHAR		NOT NULL
M_init	The user's middle initial	CHAR(1)		
L_Name	The user's last name	VARCHAR		NOT NULL
Student_ID	The student's ID number.	CHAR(8)	K2	CHECK

Description:
This relation serves the purpose of storing student's basic information. STUDENT is a sub-entity the USER entity represented using option 8B as described in lecture notes Week5 - 2.pdf on slide 2. <http://www.cs.plu.edu/courses/csce367/current/lectures/Week5-2.pdf>

Foreign keys:
N/A

Constraints:
The CHECK constraint on User_Name is to ensure that the username is at least 8 characters in length.
The CHECK constraint on password is to ensure that there are at least three levels of complexity in a user's password.
The CHECK constraint on email is to force that an email address must take a valid form.

Functional Dependencies:
User_Name → Password, email, F_Name, M_init, L_Name
email → User_Name, Password, F_Name, M_init, L_Name

Course

Field Name	Description	Domain	PK	Constraints
Course_Num	The course number for a course.	SMALLINT	PK	CHECK
Course_Name	The name of the course.	VARCHAR	K1	UNIQUE

Description:
This relation serves the purpose of storing course information for active courses.

Foreign keys:
N/A

Constraints:
The CHECK constraint on Course_Num is to verify that the course number is positive.

Functional Dependencies:
Course_Num → Course_Name
Course_Name → Course_Num

Question

Field Name	Description	Domain	PK	Constraints
Question_Num	The unique question number.	INT auto_increment	PK	CHECK
Solution	The solution to the question.	VARCHAR		CHECK
Question_Type	The type-defining attribute for question.	ENUM('TF', 'MC', 'MA', 'FB')	FK	NOT NULL
Prompt	The text of the question.	VARCHAR		CHECK

Description:
This relation serves the purpose of storing questions that have been created; questions will be added to quizzes at the discretion of the instructor. The FITB, MC, Matching, True_False ⊂ Question was represented using option 8A as described in lecture notes Week5 - 2.pdf on slide 2.

<http://www.cs.plu.edu/courses/csce367/current/lectures/Week5-2.pdf> This choice was made because 8A works in every circumstance and avoids having null values for questions that do not have attributes that other questions do.

Foreign keys:
foreign key Question(Question_Type) references Question_Type(Question_Type)

Constraints:
The CHECK constraint on Question_Num is to verify that the question number is positive.
The CHECK constraint on Solution is to enforce that the solution has to have (legal) value before the question can be added to a quiz. Legal means the solution must not be null, and must match up with the type of question being asked.

Functional Dependencies:
Question_Num → Solution, Question_Type, Instructions

Question_Type

Field Name	Description	Domain	PK	Constraints
Question_Type	The type-defining attribute for question.	ENUM('TF', 'MC', 'MA', 'FB')	PK	
Pt_Value	How many points this type of question is worth.	TINYINT		CHECK
Instructions	Specific instructions for this type of question.	VARCHAR		NOT NULL

Description:
This relation serves the purpose of identifying the type of question that is at hand, and enforcing basic attributes of the question type; namely, the point value and default instructions for a question of that particular type. This is the type-defining attribute of a question.

Foreign keys:
N/A

Constraints:
The CHECK constraint on Pt_Value is to ensure that the individual question types have the following point values: TF (1pt), MA (2pts), FB (2pts), MC (2pts)

Functional Dependencies:
Question_Type → Pt_Value, Instructions

FITB

Field Name	Description	Domain	PK	Constraints
Question_Num	The unique question number.	INT	PK, FK	CHECK

Description:
This relation serves the purpose of distinguishing fill in the blank questions from other question types. The FITB ⊂ Question was represented using option 8A as described in lecture notes Week5 - 2.pdf on slide 2. <http://www.cs.plu.edu/courses/csce367/current/lectures/Week5-2.pdf> This choice was made because 8A works in every circumstance and avoids having null values for questions that do not have attributes that other questions do.

Foreign keys:
foreign key FITB(Question_Num) references Question(Question_Num)

Constraints:
The CHECK constraint on Question_Num is to verify that the question number is positive.
The CHECK constraint on Question_Type is to verify that the type takes one of the following forms: TF; MC; MA; FB.

Functional Dependencies:
N/A

MC

Field Name	Description	Domain	PK	Constraints
Question_Num	The unique question number.	INT	PK, FK	CHECK

Description:
This relation serves the purpose of distinguishing multiple choice questions from other question types. The MC ⊂ Question was represented using option 8A as described in lecture notes Week5 - 2.pdf on slide 2. <http://www.cs.plu.edu/courses/csce367/current/lectures/Week5-2.pdf> This choice was made because 8A works in every circumstance and avoids having null values for questions that do not have attributes that other questions do.

Foreign keys:
foreign key MC(Question_Num) references Question(Question_Num)

Constraints:
The CHECK constraint on Question_Num is to verify that the question number is positive.

Functional Dependencies:
N/A

Matching

Field Name	Description	Domain	PK	Constraints
Question_Num	The unique question number.	INT	PK, FK	CHECK

Description:
This relation serves the purpose of distinguishing matching questions from other question types. The Matching ⊂ Question was represented using option 8A as described in lecture notes Week5 - 2.pdf on slide 2. <http://www.cs.plu.edu/courses/csce367/current/lectures/Week5-2.pdf> This choice was made because 8A works in every circumstance and avoids having null values for questions that do not have attributes that other questions do.

Foreign keys:
foreign key Matching(Question_Num) references Question(Question_Num)

Constraints:
The CHECK constraint on Question_Num is to verify that the question number is positive.

Functional Dependencies:
N/A

True_False

Field Name	Description	Domain	PK	Constraints
Question_Num	The unique question number.	INT	PK, FK	CHECK

Description:
This relation serves the purpose of distinguishing True/False questions from other question types. The True_False ⊂ Question was represented using option 8A as described in lecture notes Week5 - 2.pdf on slide 2. <http://www.cs.plu.edu/courses/csce367/current/lectures/Week5-2.pdf> This choice was made because 8A works in every circumstance and avoids having null values for questions that do not have attributes that other questions do.

Foreign keys:
foreign key MC(Question_Num) references Question(Question_Num)

Constraints:
The CHECK constraint on Question_Num is to verify that the question number is positive.

Functional Dependencies:
N/A

Quiz

Field Name	Description	Domain	PK	Constraints
Quiz_Num	The number of the Quiz for the course.	TINYINT	PK	CHECK
Course_Num	The Identifying number for the course	SMALLINT	PK, FK	NOT NULL
Start_Time	The time the quiz was started	DATETIME		CHECK
End_Time	The time the quiz was ended	DATETIME		CHECK
Time_Limit	The time limit for taking the quiz	TIME		CHECK

Description:
This relation serves the purpose of storing information for all quizzes that have been created by the instructor.

Foreign keys:
foreign key Quiz(Course_Num) references Course(Course_Num)

Constraints:
The CHECK constraint on Quiz_Num is to verify that it is positive.
The CHECK constraint on T_Lim is to enforce that T_Lim cannot be null when the quiz is assigned to a course.

The CHECK constraint on Start_Time is to enforce that Start_Time cannot be null when the quiz is assigned to a course.
The CHECK constraint on End_Time is to enforce that the end time of the quiz's availability is at least T_Lim after Start_Time, and that End_Time cannot be null when the quiz is assigned to a course.
The CHECK constraint on Course_Num is to verify that the course number is positive.

Functional Dependencies:
Quiz_Num, Course_Num → Start_Time, T_Lim, End_Time

Enrolled_In

Field Name	Description	Domain	PK	Constraints
Student_ID	The student's ID number.	CHAR(8)	PK, FK	CHECK
Course_Num	The course number for a course.	SMALLINT	PK, FK	CHECK

Description:
This relation serves the purpose of storing which students are enrolled in which courses, for the purpose of restricting which quizzes a student can take: i.e. a student can only take quizzes for a course s/he is in.

Foreign keys:
foreign key Enrolled_In(Student_ID) references Student(Student_ID)
foreign key Enrolled_In(Course_Num) references Course(Course_Num)

Constraints:
The CHECK constraint on Student_ID is to verify that the student's ID is an 8 digit identifier.
The CHECK constraint on Course_Num is to verify that the course number is positive.

Functional Dependencies:
N/A

Takes_Quiz

Field Name	Description	Domain	PK	Constraints
Quiz_Num	The identifying Quiz ID number.	INT	PK, FK	CHECK
Course_Num	The Identifying number for the course	SMALLINT	PK, FK	CHECK
Student_ID	The ID number of the student taking the quiz.	CHAR(8)	PK, FK	CHECK

Description:
This relation serves the purpose of storing which students took what quizzes for a particular course.

Foreign Keys:
foreign Key Takes_Quiz(Quiz_Num, Course_Num) references Quiz(Quiz_Num, Course_Num)
foreign Key Takes_Quiz(Student_ID) references Student(Student_ID)

Constraints:
The CHECK constraint on Quiz_Num is to verify that it is positive.
The CHECK constraint on Course_Num is to verify that the course number is positive.
The CHECK constraint on Student_ID is to verify that the student's ID is an 8 digit identifier.

Functional Dependencies:
N/A

Question_Asked

Field Name	Description	Domain	PK	Constraints
Question_Num	The unique question number.	INT	PK, FK	CHECK
Quiz_Num	The number of the Quiz for the course.	TINYINT	PK, FK	CHECK
Course_Num	The Identifying number for the course.	SMALLINT	PK, FK	CHECK

Description:
This relation serves the purpose of storing the questions that were asked on a particular quiz in a particular course.

Foreign Keys:
foreign key Question_Asked(Quiz_Num, Course_Num) references Quiz(Quiz_Num, Course_Num)
foreign key Question_Asked(Question_Num) references Question(Question_Num)

Constraints:
The CHECK constraint on Course_Num is to verify that the course number is positive.
The CHECK constraint on Quiz_Num is to verify that it is positive.
The CHECK constraint on Question_Num is to verify that the question number is positive.

Functional Dependencies:
N/A

Course_Question

Field Name	Description	Domain	PK	Constraints
Course_Num	The Identifying number for the course.	SMALLINT	PK, FK	CHECK
Question_Num	The unique question ID number.	TINYINT	PK, FK	CHECK

Description:
This represents the relationship between a course and the questions in the database that can be asked for that course.

Foreign Keys:
foreign key Course_Question(Course_Num) references Course(Course_Num)
foreign key Course_Question(Question_Num) references Question(Question_Num)

Constraints:
The CHECK constraint on Course_Num is to verify that the course number is positive.
The CHECK constraint on Question_Num is to verify that the question number is positive.

Functional Dependencies:
N/A

Tags

Field Name	Description	Domain	PK	Constraints
Question_Num	The unique question ID number.	INT	PK, FK	CHECK
Tag	Keyword that describes question content.	VARCHAR(32)	PK	CHECK

Description:
This relation serves the purpose of storing which questions may be relevant to ask on a particular quiz. A tag could be a keyword/keywords for a particular subject matter, or anything else that may help identify the content of the question.

Foreign Keys:
foreign key Tags(Question_Num) references Question(Question_Num)

Constraints:
The CHECK constraint on Question_Num is to verify that the question number is positive.
The CHECK constraint on Tag is to enforce that a tag cannot be an existing course number.

Functional Dependencies:
Question_Num → Tag

MC_Ans_Options

Field Name	Description	Domain	PK	Constraints
Question_Num	The unique question number.	INT	PK, FK	CHECK
Answer_Option	A possible answer option for this multiple choice question	VARCHAR		CHECK
Answer_Letter	The letter corresponding to the answer option	ENUM('a','b','c','d','e')		CHECK

Description:
This relation serves the purpose of storing possible answers that will be provided on a quiz for a particular multiple choice question.

Foreign keys:
foreign key MC_Ans_Options(Question_Num) references Question(Question_Num)

Constraints:
The CHECK constraint on Question_Num is to verify that the question number is positive.
The CHECK constraint on Answer_Option is to enforce that there must between 3 and 5 answer options before the question can be added to a quiz.
The CHECK constraint on Answer_Letter is to enforce that d and e cannot be used as valid answer letter unless there are 4 and 5 answer options, respectively.

Functional Dependencies:
Question_Num → Answer_Option

Matching_Keywords

Field Name	Description	Domain	PK	Constraints
Question_Num	The unique question ID number.	INT	PK, FK	CHECK
Keyword	The word which will be match to the descriptions.	VARCHAR(32)	PK	

Description:
This relation serves the purpose of storing what keywords will be provided on a particular matching question.

Foreign Keys:
foreign key Matching_Keywords(Question_Num) references Question(Question_Num)

Constraints:
The CHECK constraint on Question_Num is to verify that the question number is positive.

Functional Dependencies:
Question_Num → Keyword

Student_Answer

Field Name	Description	Domain	PK	Constraints
Student_Answer	The answer a student gave on a question.	VARCHAR		
Student_ID	The student's ID number	CHAR(8)	PK, FK	CHECK
Quiz_Num	The quiz number	TINYINT	PK, FK	CHECK
Course_Num	The course number.	SMALLINT	PK, FK	CHECK
Question_Num	The unique question number.	INT	PK, FK	CHECK

Description:
This relation serves the purpose of storing the answer a student gave on each question on a particular quiz in a particular course.

Foreign keys:
foreign key Student_Answer(Student_ID, Course_Num) references Enrolled_In(Student_ID, Course_Num)
foreign key Student_Answer(Quiz_Num, Course_Num) references Quiz(Quiz_Num, Course_Num)
foreign key Student_Answer(Question_Num) references Question(Question_Num)

Constraints:
The CHECK constraint on Student_ID is to verify that the student's ID is an 8 digit identifier.
The CHECK constraint on Question_Num is to verify that the question number is positive.
The CHECK constraint on Quiz_Num is to verify that it is positive.
The CHECK constraint on Course_Num is to verify that the course number is positive.

Functional Dependencies:
Student_ID, Quiz_Num, Course_Num, Question_Num → Student_Answer

---

Table Creation SQL

Table Name	SQL Statement	Notes
Instructor	CREATE TABLE Instructor(User_Name VARCHAR(30), Password CHAR(41) NOT NULL, email VARCHAR(255) NOT NULL UNIQUE, F_Name VARCHAR(255) NOT NULL, M_Init CHAR(1), L_Name VARCHAR(255) NOT NULL, PRIMARY KEY (User_Name));	CHECK constraint on User_Name, Password, email.
Student	CREATE TABLE Student (User_Name VARCHAR(30), Password CHAR(41), email VARCHAR(255) UNIQUE, F_Name VARCHAR(255), M_init CHAR(1), L_Name VARCHAR(255), Student_ID CHAR(8) UNIQUE, PRIMARY KEY (User_Name));	CHECK constraint on User_Name, Password, Email
Course	CREATE TABLE Course (Course_Num SMALLINT, Course_Name VARCHAR(255) UNIQUE, PRIMARY KEY (Course_Num));	CHECK constraint on Course_Num.

Table Name	SQL Statement	Notes
Enrolled_In	CREATE TABLE Enrolled_In(Student_ID CHAR(8), Course_Num SMALLINT, PRIMARY KEY (Student_ID, Course_Num), FOREIGN KEY (Student_ID) references Student(Student_ID) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (Course_Num) references Course(Course_Num) ON DELETE CASCADE ON UPDATE CASCADE);	Check constraints on Student_ID, Course_Num. Student and Course tables must be created before Enrolled_In table.
Quiz	CREATE TABLE Quiz(Quiz_Num TINYINT, Course_Num SMALLINT, Start_Time DATETIME, End_Time DATETIME, Time_Limit TIME, PRIMARY KEY (Quiz_Num, Course_Num), FOREIGN KEY (Course_Num) references Course (Course_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraint on Quiz_Num and Course_Num. Must create Course table before creating Quiz table.
Question_Type	CREATE TABLE Question_Type(Question_Type ENUM('TF', 'MC', 'MA', 'FB'), Pt_Value TINYINT, Instructions VARCHAR(255) NOT NULL, PRIMARY KEY (Question_Type));	CHECK constraints on Question_Type.

Table Name	SQL Statement	Notes
Question	CREATE TABLE Question (Question_Num INT auto_increment, Question_Type ENUM('TF', 'MC', 'MA', 'FB') NOT NULL, Solution VARCHAR(255), Prompt VARCHAR(255), PRIMARY KEY (Question_Num), FOREIGN KEY (Question_Type) references Question_Type(Question_Type) ON DELETE CASCADE ON UPDATE CASCADE);	Question_Num, Solution, and Prompt have CHECK constraints. Question-type table must be created before Question table.
Course_Question	CREATE TABLE Course_Question(Course_Num SMALLINT, Question_Num INT, PRIMARY KEY (Course_Num, Question_Num), FOREIGN KEY (Course_Num) references Course(Course_Num) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (Question_Num) references Question(Question_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraints on Course_Num, Question_Num. Must create Course table and Question table before creating Course_Question table.
Tags	CREATE TABLE Tags(Question_Num INT, Tag VARCHAR(32), PRIMARY KEY (Question_Num, Tag), FOREIGN KEY (Question_Num) references Question(Question_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraints on Question_Num, Tag. Question table must be created before Tags table.

Table Name	SQL Statement	Notes
FITB	CREATE TABLE FITB(Question_Num INT, PRIMARY KEY (Question_Num), FOREIGN KEY (Question_Num) references Question (Question_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraint on Question_Num. Question table must be created before FITB
MC	CREATE TABLE MC(Question_Num INT, PRIMARY KEY (Question_Num), FOREIGN KEY (Question_Num) references Question(Question_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraint on Question_Num. Question table must be created before MC table.
MC_Ans_Options	CREATE TABLE MC_Ans_Options(Question_Num INT, Answer_Option VARCHAR(255), Answer_Letter ENUM('A','B','C','D','E'), PRIMARY KEY (Question_Num, Answer_Option), FOREIGN KEY (Question_Num) references MC (Question_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraints on Question_Num, Answer_Option. Must create Question table before MC_Ans_Options table.
Matching	CREATE TABLE Matching(Question_Num INT, PRIMARY KEY (Question_Num), FOREIGN KEY (Question_Num) references Question(Question_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraint on Question_Num. Must create Question table before Matching table.

Table Name	SQL Statement	Notes
Matching_Keywords	CREATE TABLE Matching_Keywords(Question_Num INT, Keyword VARCHAR(30), PRIMARY KEY (Question_Num, Keyword), FOREIGN KEY (Question_Num) references Matching(Question_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraint on Keyword.
True_False	CREATE TABLE True_False(Question_Num INT, PRIMARY KEY (Question_Num), FOREIGN KEY (Question_Num) references Question (Question_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraint on Question_Num. Must create Question table before True_False table.
Question_Asked	CREATE TABLE Question_Asked(Question_Num INT, Quiz_Num TINYINT, Course_Num SMALLINT, PRIMARY KEY (Question_Num, Quiz_Num, Course_Num), FOREIGN KEY (Question_Num) references Question (Question_Num) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (Quiz_Num, Course_Num) references Quiz (Quiz_Num, Course_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraints on Question_Num, Quiz_Num, Course_Num. Must create Question table and Quiz table before creating Question_Asked table.

Table Name	SQL Statement	Notes
Student_Answer	CREATE TABLE Student_Answer(Student_Answer VARCHAR(255), Student_ID CHAR(8), Quiz_Num TINYINT, Course_Num SMALLINT, Question_Num INT, PRIMARY KEY (Student_ID, Quiz_Num, Course_Num, Question_Num), FOREIGN KEY (Student_ID, Course_Num) references Enrolled_In(Student_ID, Course_Num) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (Quiz_Num, Course_Num) references Quiz(Quiz_Num, Course_Num) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (Question_Num) references Question(Question_Num) ON DELETE CASCADE ON UPDATE CASCADE);	CHECK constraints on Student_ID, Quiz_Num, Course_Num, Question_Num. The Student, Quiz, and Question tables must be created before the Student_Answer table.

Note that we have also created backup tables. These tables are named as follows: tablename_Backup, where tablename is the particular table name that the backup corresponds to.
The only difference in the SQL for table creation with these tables is that there is no foreign key statements.

Prototype

● Authentication & Authorization

Introduction: We have 3 user groups for this database--the admin, the instructor, and the students. The admin has access to all database functionality, but the update contact information is slightly different for the admin. The instructor has access to all database functions except for the Backup/Restore DB functions--these are admin-only functions. Students have access to only limited database functionality. The check quiz grade is slightly different for students than it is for the instructor or admin.

Authentication: Our login page looks as follows:

Following are a few tests of the authentication process:

Test1:*Valid username and password*
Input:

Note : Password = abc123..
Result:

Test2: *Invalid username*
Input:

Note: This test holds true regardless of what the input password is.
Result:

Test3: *Invalid password*
Input:

Note: password = kenb
Result:

Authorization: Admins have access to all websites
Instructors have access to all websites except for backup/restore DB.
Students only have access to update contact information and check quiz grade.
Test1: *Instructor login*
Input:

Note: password = Torturethemwithquizzes!
Result:

A successful student login was shown above in authentication test 1.
A successful admin login contains the following input:
Username = hippo367
Password = hippo_367.DB!
Test2: *A student typing in the url of a page that students are unauthorized for*
Input: *Assume the student is logged in:*

Result:

- ## Create New Multiple Choice Question

Introduction: Both the admin and the instructor have access to this page. This use case creates a new Multiple Choice Question and inserts into the tables Question, MC, MC_Ans_Options, and possibly Course_Question & Tags.

Test1: *No input*
Input:

Result:

Test2: *No option d; no solution selected.*
Input:

Result:

Test3*: Valid question creation.*
Input:

Before Addition:

Result:

- **Update Contact Information**

When a non-PLU email is used, no update is made

Name fields also have length restrictions.

- **Check Quiz Grade**

First the user will arrive at a course selection which looks like this:

<u>Test1:</u> Course without quizzes
Input: 371
Result:

Here you see the quiz list was not populated because course 371 has no quizzes in the database. Because of this you cannot select a quiz number, the page will not crash, instead it will wait for the user to select a different course number and then load the appropriate quizzes.

Test2: ***Course with quizzes***

Input: 367

Result:

Here you see a course that has quizzes. After the Instructor selects the desired quiz number the page he see's a report of the class stats for that quiz:

Here you see The Instructor's quiz report with each student's quiz report hyperlinked to their name.

Here you see the Student's report after the Instructor clicks on one of the hyperlinks. If a student was accessing checkQuizGrade the instructor page is not displayed and instead the student see's the student report associated with their username.

- ## Delete Course

Introduction: The delete course affects a number of tables, and hence error checking was very intensive. Following is what the delete course page looks like when first loaded.

Note that if no course is selected and the select courses button is pressed, nothing will happen.
Test1: *2 courses: 1 with students enrolled in it, the other not.*

Note that selecting Delete Anyway if neither yes nor no is selected does nothing, and selecting no and pressing delete anyway forwards to the menu page.

This takes the user to the following page:

Notice that at any time during this process the user may choose to deselect courses or select additional courses.
After selecting Delete selected course(s), the following occurs:

Before deletion:
Course table:

Course_Question table:

Enrolled_In table:

Student_Answer table (for 270 and 320 ONLY):

Quiz table:

After Deletion:

Course table:

Course_Question table:

Enrolled_In table:

Student_Answer table (for 270 and 320 ONLY):

Quiz table:

- **Find Available Questions for adding to a Quiz**

When a class and tags are selected, all questions related to that class that apply to at least one of the tags are included. If there are no questions in that class that apply to those tags, a message is displayed informing you of the lack of questions.

- ## Backup/Restore the Database

Introduction: The backup and restore methods were created specifically for the purpose of making the deletion process be less worrisome. Since the delete course affects so many tables, we wanted a quick way to restore data from a previous backed up state.

Test1: *Restore the Database after the successful delete course test shown (Test1).*
We begin at the admin menu page:

The before restoration status is the same as the after deletion status for the delete course Test1.
Selecting Restore DB, we see the following:

The after restoration status is the same as the before status for the delete course Test1.

Test2: *Start with an incomplete backup table and then backup the database.*

Before backup:

Notice the True_False table:

As with the restoration, we begin at the menu page and click backup DB.
Upon selection we see that:

After backup:

Code

Utilities.java:

```
package deliverable;
/*
 * Group: Hippo
 * Assignment: Deliverable 5
 * Due Date: 12 May 2014
 */

import java.io.File;
import java.io.FileNotFoundException;
import java.math.BigInteger;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
```

```
import java.util.Scanner;

public class Utility {

    /**
     * This class contains methods to implement our 5 use cases, as well as to log in to the database.
     * An additional feature we have added is the ability to back up our database and restore it to a previous,
backed up, state.
     * The primary reason for doing so is to help make restoring data after a deletion easier.
     * In addition, we have added the ability to populate the database from the java code, rather than having
to do it manually in SQuirreL--to save time.
     */

    Scanner keyboard = new Scanner(System.in);
    public Connection conn = null;
    private ArrayList<ArrayList<String>> tableDescriptions;
    private ArrayList<String> tableNames;
    public String username = "";
    public boolean badUsername = false;
    public boolean badPassword = false;
    public boolean noMCQuestionText = false;
    public boolean longMCQuestionText = false;
    public boolean noMC_ReqAns_Options = false;
    public boolean badMC_nReqAns_Options = false;
    public boolean longMC_Ans_Options = false;
    public boolean noMC_Soln = false;
    public boolean invalidSoln = false;
    public boolean notUniqueMC_Ans_Options = false;
    public boolean badConnection = false;
    public String[] courseNumDelete;
    public boolean redirect = false;
    public boolean web = false;
    public boolean noUsernameUpdateContactInfo = false;
    public boolean longMInitUpdateContactInfo = false;
    public boolean longFnameUpdateContactInfo = false;
    public boolean longLnameUpdateContactInfo = false;
    public boolean longEmailUpdateContactInfo = false;
    public boolean invalidEmailUpdateContactInfo = false;
    public boolean noCourseSelectedAvailableQuestions = false;
    private BigInteger p =  new BigInteger("170141183460469231731687303715884105727");
    private BigInteger q = new
BigInteger("136300555243466607821742128462127993362710278088105335847 3");
    private ArrayList<BigInteger> ms;
    private ArrayList<BigInteger> ns;
    private ArrayList<BigInteger> mults;
    private ArrayList<BigInteger> rs;
    public BigInteger n;
    public BigInteger k;
```

```
public BigInteger c;
public BigInteger encodedPassword;
public String quizNum;
public String courseNum;


/**
 * The constructor called by the web interface.
 */
public Utility(){
        n = p.multiply(q);
        tableNames = new ArrayList<String>();
        ms = new ArrayList<BigInteger>();
        ns = new ArrayList<BigInteger>();
        rs = new ArrayList<BigInteger>();
        mults = new ArrayList<BigInteger>();
        k = new BigInteger("5");
        tableNames.add("Instructor");
        tableNames.add("Student");
        tableNames.add("Course");
        tableNames.add("Enrolled_In");
        tableNames.add("Quiz");
        tableNames.add("Question_Type");
        tableNames.add("Question");
        tableNames.add("Course_Question");
        tableNames.add("Tags");
        tableNames.add("FITB");
        tableNames.add("MC");
        tableNames.add("MC_Ans_Options");
        tableNames.add("Matching");
        tableNames.add("Matching_Keywords");
        tableNames.add("True_False");
        tableNames.add("Question_Asked");
        tableNames.add("Student_Answer");
}

/**
 * The Utility constructor
 * Initializes the tables to the tables in the file filename
 * @param the filename containing the names of the tables in the database
 */
public Utility(String filename){
        n = p.multiply(q);
        ms = new ArrayList<BigInteger>();
        ns = new ArrayList<BigInteger>();
        rs = new ArrayList<BigInteger>();
        mults = new ArrayList<BigInteger>();
        k = new BigInteger("5");
```

```
		tableDescriptions = new ArrayList<ArrayList<String>>();
		tableNames = new ArrayList<String>();
		ArrayList<String> tablesFromFile = new ArrayList<String>(); //the temp list before I parse it into
table names and descriptions.
		try {
			Scanner infile = new Scanner(new File(filename));
			while(infile.hasNext()){
				tablesFromFile.add(infile.nextLine());
			}
			//parse each table description
			for(int i = 0; i < tablesFromFile.size(); i++){
				Scanner table = new Scanner(tablesFromFile.get(i));
				table.useDelimiter(","); //I HAVE TO LOOK UP THE DELIMITER STUFF AGAIN
				String tableName = table.next();
				tableNames.add(tableName);
				ArrayList<String> description = new ArrayList<String>();
				description.add(tableName);
				while(table.hasNext()){
					description.add(table.next());
				}
				tableDescriptions.add(description);
				table.close();
			}
			infile.close();
		} catch (FileNotFoundException e) {
			System.out.println("Can't find file '" + filename + "'.");
			System.out.println(e.getMessage());
		}
	}

	/**
	 * open method that opens the DB with the user name, password, and dbName given as input.
	 *
	 * @param username is a String that is the DB account username
	 * @param password is a String that is the password the account
	 * @param dbName is name of the database that will be the active db
	 */
	public void openDB() {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.out.println("problem with Driver: " + e.getMessage());
			//e.printStackTrace();
		}

		try {
			String url = "jdbc:mysql://zoe.cs.plu.edu:3306/";
			Scanner keyboard = new Scanner(System.in);
```

```
                System.out.println("Enter database name (all DBs are on zoe.cs.plu.edu): ");
                url += keyboard.nextLine();
                System.out.println("Enter username: ");
                String username = keyboard.nextLine();
                System.out.println("Enter password: ");
                String password = keyboard.nextLine();
                conn = DriverManager.getConnection(url, username, password);
        } catch (SQLException e) {
                System.out.println("problem connecting to MySQL: Please re-run testUtility");
                System.exit(1);
        }
}

/***
 * Overload the open method that opens
 * the DB with the user name, password, and dbName given as input.
 *
 * @param username is a String that is the DB account username
 * @param password is a String that is the password the account
 * @param dbName is name of the database that will be the active db
 */
public void openDB(String user, String pass, String dbName) {
        try {
                Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
                System.out.println("problem with Driver: " + e.getMessage());
                //e.printStackTrace();
        }
        try {
                String url = "jdbc:mysql://zoe.cs.plu.edu:3306/" + dbName;
                conn = DriverManager.getConnection(url, user, pass);
        } catch (SQLException e) {
                System.out.println("problem connecting to MySQL: " + e.getMessage());
        }

}

/**
 * Use case 1:
 * Name: Create a MC Question (Insert)
 * Purpose: To create a new multiple choice question in the database to be asked on quizzes.
 * User Group: Instructor / Blaha
 * Input: Prompt, optionA, optionB, optionC, (optionD, optionE), solution --optional course(s), tag(s)
 * Result: The new question will be displayed indicating a successful creation, and
 * any courses specified will have a relationship made between them and the
 * question. Or an error message will be displayed stating why the operation failed.
 * Exceptions: Input error if the solution value does not match one of the available
 * options, or prompt is null, etc. Database will prompt user to finish
```

```
     * completing the question and highlight the invalid input.
     */
    public void createMCQuestion(){
            System.out.println("Enter Question prompt: ");
            String prompt = keyboard.nextLine();
            prompt = accomodateForProblemChars(prompt);
            int numAnswers = 0;
            String[] Ans_Options = new String[5];
            for(int i = 0; i<Ans_Options.length; i++){
                    Ans_Options[i] = "";
            }
            System.out.println("Enter answer options for this quesiton (up to 5). Please enter only 1 answer
option at a time.");
            while(numAnswers < 3){
                    boolean unique = false;
                    System.out.print("Enter ");
                    switch(numAnswers){
                    case 0: System.out.print("first ");
                    break;
                    case 1: System.out.print("second ");
                    break;
                    case 2: System.out.print("third ");
                    break;
                    }
                    System.out.println("answer option: ");
                    Ans_Options[numAnswers] = keyboard.nextLine();
                    if(Ans_Options[numAnswers].equals(""))
                            System.out.println("Must input a value for an answer option.");
                    else{
                            unique = unique(Ans_Options, numAnswers);
                            if(unique){
                                    numAnswers++;
                            }
                            else System.out.println("Must input unique values for answer options.");
                    }
            }
            while(numAnswers < 5){
                    System.out.print("Enter ");
                    switch(numAnswers){
                    case 3: System.out.print("fourth ");
                    break;
                    case 4: System.out.print("fifth and final ");
                    break;
                    }
                    System.out.println("answer option (If you do not desire to have another answer option,
simply press enter.): ");
                    Ans_Options[numAnswers] = keyboard.nextLine();
                    if(!Ans_Options[numAnswers].equals("")){
```

```
                        if(unique(Ans_Options, numAnswers))
                                numAnswers++;
                        else System.out.println("Must input unique values for answer options.");
                }
                else break;
        }
        System.out.println("Here is the question entered: ");
        System.out.println(prompt);
        for(int i = 0; i< numAnswers; i++){
                switch(i){
                case 0: System.out.print("a) ");
                break;
                case 1: System.out.print("b) ");
                break;
                case 2: System.out.print("c) ");
                break;
                case 3: System.out.print("d) ");
                break;
                case 4: System.out.print("e) ");
                break;
                }
                System.out.println(Ans_Options[i]);
        }
        String soln = "z";
        switch(numAnswers){
        case 3: while(soln.charAt(0) != 'a' && soln.charAt(0) != 'b' && soln.charAt(0) != 'c'){
                System.out.println("Enter the letter corresponding to the answer option that is the solution
to the question: ");
                soln = keyboard.nextLine();
                soln = soln.toLowerCase();
                if(soln.charAt(0) != 'a' && soln.charAt(0) != 'b' && soln.charAt(0) != 'c')
                        System.out.println("Please select a valid letter option.");
        }
        break;
        case 4: while(soln.charAt(0) != 'a' && soln.charAt(0) != 'b' && soln.charAt(0) != 'c' &&
soln.charAt(0) != 'd'){
                System.out.println("Enter the letter corresponding to the answer option that is the solution
to the question: ");
                soln = keyboard.nextLine();
                soln = soln.toLowerCase();
                if(soln.charAt(0) != 'a' && soln.charAt(0) != 'b' && soln.charAt(0) != 'c' && soln.charAt(0) !=
'd')
                        System.out.println("Please select a valid letter option.");
        }
        break;
        case 5: while(soln.charAt(0) != 'a' && soln.charAt(0) != 'b' && soln.charAt(0) != 'c' &&
soln.charAt(0) != 'd' && soln.charAt(0) != 'e'){
```

```
                System.out.println("Enter the letter corresponding to the answer option that is the solution to the question: ");
                soln = keyboard.nextLine();
                soln = soln.toLowerCase();
                if(soln.charAt(0) != 'a' && soln.charAt(0) != 'b' && soln.charAt(0) != 'c' && soln.charAt(0) != 'd' && soln.charAt(0) != 'e')
                        System.out.println("Please select a valid letter option.");
        }
        break;
        }

        //Now we have the values from the user. Next we insert the question into question.
        soln = soln.toUpperCase();
        String sql = "Insert into Question(Question_Type, Solution, Prompt) Values('MC', " + "'" + soln + "'" + ", " + "'" + prompt + "'" + ")";
        Statement stm;
        PreparedStatement pstm;
        try {
                pstm = conn.prepareStatement(sql);
                pstm.executeUpdate(sql);
        } catch (SQLException e) {
                System.out.println("Problem with SQL statement: " + sql);
                System.out.println(e.getMessage());
        }

        //Now we insert the question number into MC
        //Before we can do this, we need to get the question number from the database. To do this, we need all 3 nonprimary fields.
        //So we get the Q_Type, the prompt, and the solution
        int q_num = questionNum("MC", soln, prompt);
        sql = "Insert into MC Values(" + q_num + ")";

        try {
                pstm = conn.prepareStatement(sql);
                pstm.executeUpdate(sql);
        } catch (SQLException e) {
                System.out.println("Problem with SQL statement: " + sql);
                System.out.println(e.getMessage());
        }

        //Now finally we insert the answer options to the MC_Ans_Option table
        for(int i = 0; i<numAnswers; i++){
                sql = "Insert into MC_Ans_Options Values(" + q_num + ", '" + Ans_Options[i] + "', '";
                switch(i){
                case 0: sql += "a')";
                break;
                case 1: sql += "b')";
                break;
```

```
                case 2: sql += "c')";
                break;
                case 3: sql += "d')";
                break;
                case 4: sql += "e')";
                break;
                }
                try {
                        stm = conn.createStatement();
                        stm.execute(sql);
                } catch (SQLException e) {
                        System.out.println("Problem with SQL statement: " + sql);
                        System.out.println(e.getMessage());
                }
        }
        //courses
        ArrayList<Integer> coursesRelated = new ArrayList<Integer>();
        String course = "-";
        System.out.println("If desired, enter a course number(s) that are related to this question. If not,
simply press enter. Please enter only one course number at a time.");
        while(!course.equals("")){
                System.out.println("Enter course number.");
                course = keyboard.nextLine();
                if(isCourse(course)){
                        coursesRelated.add(Integer.parseInt(course));
                }
        }
        for(int i = 0; i < coursesRelated.size(); i++){
                sql = "Insert into Course_Question Values('" + coursesRelated.get(i) + "', '" + q_num + "')";
                try {
                        stm = conn.createStatement();
                        stm.executeUpdate(sql);
                } catch (SQLException e) {
                        System.out.println("Problem with SQl Statement: " + sql);
                        System.out.println(e.getMessage());
                }
        }
        //tags
        ArrayList<String> selectedTags = getTags();
        for(int i = 0; i < selectedTags.size(); i++){
                sql = "Insert into Tags Values('" + q_num + "', '" + selectedTags.get(i) + "')";
                try {
                        stm = conn.createStatement();
                        stm.executeUpdate(sql);
                } catch (SQLException e) {
                        System.out.println("Problem with SQL Statement: " + sql);
                        System.out.println(e.getMessage());
                }
```

```
		}
	}

	/**
	 * This is the createMCQuestion use case utilized by the web interface.
	 * @param prompt
	 * @param Ans_Options
	 * @param soln
	 * @param numAnswers
	 * @param relatedCourses
	 * @param tags
	 */
	public void createMCQuestion(String prompt, String[] Ans_Options, String soln, int numAnswers,
ArrayList<String> coursesRelated, ArrayList<String> selectedTags){
		for(int i = 0; i < Ans_Options.length; i++){
			System.out.println("Ans_Options[" + i + "] = " + Ans_Options[i]);
		}
		System.out.println("prompt = " + prompt);
		System.out.println("soln = " + soln);
		String sql = "Insert into Question(Question_Type, Solution, Prompt) Values('MC', " + "'" + soln + "'"
+ ", " + "'" + prompt + "'" + ")";
		Statement stm;
		PreparedStatement pstm;
		try {
			pstm = conn.prepareStatement(sql);
			pstm.executeUpdate(sql);
		} catch (SQLException e) {
			System.out.println("Problem with SQL statement: " + sql);
			System.out.println(e.getMessage());
		}
		int q_num = questionNum("MC", soln, prompt);
		sql = "Insert into MC Values(" + q_num + ")";

		try {
			pstm = conn.prepareStatement(sql);
			pstm.executeUpdate(sql);
		} catch (SQLException e) {
			System.out.println("Problem with SQL statement: " + sql);
			System.out.println(e.getMessage());
		}

		//Now finally we insert the answer options to the MC_Ans_Option table
		for(int i = 0; i<numAnswers; i++){
			sql = "Insert into MC_Ans_Options Values(" + q_num + ", '" + Ans_Options[i] + "', '";
			switch(i){
			case 0: sql += "a')";
			break;
			case 1: sql += "b')";
```

```
                break;
                case 2: sql += "c')";
                break;
                case 3: sql += "d')";
                break;
                case 4: sql += "e')";
                break;
                }
                try {
                        stm = conn.createStatement();
                        stm.execute(sql);
                } catch (SQLException e) {
                        System.out.println("Problem with SQL statement: " + sql);
                        System.out.println(e.getMessage());
                }
        }

        for(int i = 0; i < coursesRelated.size(); i++){
                sql = "Insert into Course_Question Values('" + coursesRelated.get(i) + "', '" + q_num + "')";
                try {
                        stm = conn.createStatement();
                        stm.executeUpdate(sql);
                } catch (SQLException e) {
                        System.out.println("Problem with SQl Statement: " + sql);
                        System.out.println(e.getMessage());
                }
        }
        for(int i = 0; i < selectedTags.size(); i++){
                sql = "Insert into Tags Values('" + q_num + "', '" + selectedTags.get(i) + "')";
                try {
                        stm = conn.createStatement();
                        stm.executeUpdate(sql);
                } catch (SQLException e) {
                        System.out.println("Problem with SQL Statement: " + sql);
                        System.out.println(e.getMessage());
                }
        }
  }


  /**
   * Use case 2:
   * Name: Update Contact Information
   * Purpose: Changes a particular student's contact information to new information input by the user. We
will assume the student is logged in.
   * User Group: Student
   * Input: At least one of the following: email, F_Name, M_init, L_Name.
   * Result: The information that was input for modification will replace the old
```

```
    * information that was in the database, i.e. the contact information, after modification, for the user
updating contact information will be the combination of unchanged attributes of those listed in the possible
inputs and data that was input.
    * Exceptions: If the user enters data that is the same as what it already is, nothing will
    * happen. If the user inputs an invalid email address, the database will ask for a valid email address. If
the user inputs multiple characters for the middle initial, only the first character will be saved.
    */
   public void updateContactInformation(){

        String email = "";
        String fname = "";
        String minit = "";
        String lname = "";
        String temp ="";
        String id="";
        ResultSet r=null;
        java.sql.ResultSetMetaData mdata;


        System.out.println("please input std ID: ");
        id=keyboard.nextLine();

        //Get current values to display and to load back in if user does not change the field
        String sql="Select Email, F_name, M_init, L_name\n"+
                        "From Student\n"+
                        "Where student_Id='"+id+"'";

        Statement stm;
        try {
                stm = conn.createStatement();
                r = stm.executeQuery(sql);
                mdata = r.getMetaData();

                //Prints current attribute values and gets user input handling exceptions
                System.out.print("Input updated attribute or press Enter to leave unchanged \n");
                while( r.next() ) {
                        while(true){
                                System.out.println( r.getString( 1 ) + "\nUpdate Email? :");
                                temp = keyboard.nextLine();
                                if(temp.length()==0){
                                        email = r.getString( 1 );
                                        break;
                                }else
if(temp.length()>8&&temp.substring(temp.length()-8).equalsIgnoreCase("@plu.edu")){
                                        email = temp;
                                        break;
                                }
                                else{
```

```
                                        System.out.println("Please input valid @plu.edu email");
                                }
                        }
                        System.out.println( r.getString( 2 ) + "\nUpdate First Name? :");
                        temp = keyboard.nextLine();
                        if(temp.length()==0){
                                fname = r.getString( 2 );
                        }else{
                                System.out.println("new fname= "+temp);
                                fname = temp;
                        }
                        fname = accomodateForProblemChars(fname);
                        System.out.println( r.getString( 3 ) + "\nUpdate Middle Initial? :");
                        temp = keyboard.nextLine();
                        if(temp.length()==0){
                                minit = r.getString( 3 );
                        }else{
                                minit = temp.charAt(0)+"";
                        }
                        System.out.print( r.getString( 4 ) + "\nUpdate Last Name? :");
                        temp = keyboard.nextLine();
                        if(temp.length()==0){
                                lname = r.getString( 4 );
                        }else{
                                lname = temp;
                        }
                        lname = accomodateForProblemChars(lname);


                        sql="Update Student\n"+
                                        "Set email='"+email+"', F_name='"+fname+"', M_init='"+minit+"',
L_name='"+lname+"'\n"+
                                        "Where student_Id='"+id+"'";

                        PreparedStatement pstm = conn.prepareStatement(sql);
                        pstm.executeUpdate();

                        System.out.println("update complete");


                }
        } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
  }

  /**
```

```
    * This is the update contact info called by the web interface.
    * @param email
    * @param fname
    * @param minit
    * @param lname
    */
   public void updateContactInformation(String username_, String email, String fname, String minit, String
lname){
          String sql="Update ";
          if(user_Type(username_).equals("Student"))
                sql += "Student ";
          else{
                sql+= "Instructor ";
          }
          Statement stm;
          try {
                sql+="Set email='"+email+"', F_name='"+fname+"', M_init='"+minit+"',
L_name='"+lname+"' "+
                            "Where User_name='"+username_+"'";
                stm = conn.createStatement();
                stm.executeUpdate(sql);

          }catch(SQLException e){
                System.out.println("Problem with SQL Staement: " + sql);
                System.out.println(e.getMessage());
          }
   }

   /**
    * Use case 3:
    * Name: Check Quiz Grade
    * Purpose: Finds the student's grade for a quiz and shows the class average on the quiz for
comparison.
    * User Group: Student
    * Input: Quiz_num, Course_num
    * Result: The student's score on the quiz out of the total score, the percentage the
    * student completed successfully on the quiz, class average on the quiz are all displayed.
    * Exceptions: A student can not access this feature on a particular quiz until after the
    * quiz is no longer available. If the student chooses a quiz that s/he has not taken, or a quiz that does
not exist, no result will be displayed and the database will prompt the user to choose another quiz.
    */
   public String checkQuizGrade(){
          //String Student_ID, int Quiz_Num, int Course_Num
          String student_ID="";
          int quiz_num=0;
          int course_num=0;
          int studentScore=0;
          int studentPossible=0;
```

```
		int classScore=0;
		int classPossible=0;
		String quizNumber = "";
		String courseNumber = "";
		boolean badIn=false;
		//
		System.out.println("Please input student ID: ");
		student_ID=keyboard.nextLine();
		System.out.println("please input quiz number: ");
		quizNumber = keyboard.nextLine();
		while(!isInt(quizNumber)){
			System.out.println("Quiz number must be a positive integer.");
			System.out.println("please input quiz number: ");
			quizNumber = keyboard.nextLine();
		}
		quiz_num = Integer.parseInt(quizNumber);
		System.out.println("please input course number: ");
		courseNumber = keyboard.nextLine();
		while(!isInt(courseNumber)){
			System.out.println("Course number must be a positive integer.");
			courseNumber = keyboard.nextLine();
		}
		course_num = Integer.parseInt(courseNumber);
		//Note to Peter: The update was done by Brandon. The only exception I handled was the case that
the user did not enter an integer value.
		ResultSet r=null;
		java.sql.ResultSetMetaData mdata;
		String headerOut="";
		String dataOut="";

		String sql="(Select concat(F_name,' ',L_name)as name, sum(PT_value) as score\n"+
				"From Student_Answer as sa, Student as s, Question as q, Question_Type as
qt\n"+
				"Where sa.question_num=q.question_num and q.question_type=qt.question_type
and q.solution=sa.student_answer and sa.Student_ID=s.Student_ID and sa.quiz_num="+quiz_num+" and
sa.course_num="+course_num+" and s.Student_ID='"+student_ID+"')\n";

		Statement stm;
		try {
			stm = conn.createStatement();
			r = stm.executeQuery(sql);
			mdata = r.getMetaData();
			headerOut +=mdata.getColumnName(1) +"\t\t";
			headerOut +=mdata.getColumnName(2) +"\t";

			while( r.next() ) {
				dataOut+= r.getString(1)+"\t";
				dataOut+= r.getString(2)+"\t";
```

```
                studentScore= r.getInt(2);
            }

            sql="Select total, sum(score), sum(total) \n"+
                    "From (Select sa.Student_ID, sum(PT_value) as total \n"+
                    "From Student_Answer as sa, Student as s, Question as q,
Question_Type as qt \n"+
                    "Where sa.question_num=q.question_num and
q.question_type=qt.question_type and sa.Student_ID=s.Student_ID and sa.quiz_num="+quiz_num+" and
sa.course_num="+course_num+" group by s.Student_ID) as a, \n"+
                    "(Select sa.Student_ID, sum(PT_value) as score \n"+
                    "From Student_Answer as sa, Student as s, Question as q,
Question_Type as qt \n"+
                    "Where sa.question_num=q.question_num and
q.question_type=qt.question_type and q.solution=sa.student_answer and sa.Student_ID=s.Student_ID and
sa.quiz_num="+quiz_num+" and sa.course_num="+course_num+" group by s.Student_ID) as b \n"+
                    "where a.student_ID=b.student_ID";

            r = stm.executeQuery(sql);
            mdata = r.getMetaData();
            headerOut +=mdata.getColumnName(1) +"\t";
            headerOut += "%\t";
            headerOut +="Class Avg\t";

            while( r.next() ) {
                dataOut+= r.getString(1)+"\t";
                studentPossible=r.getInt(1);
                classScore= r.getInt(2);
                classPossible= r.getInt(3);
                if(r.getString(1)==null||!isInt(r.getString(2))||!isInt(r.getString(3))){
                    System.out.println("Non exsistant input values");
                    badIn=true;
                }
            }



            dataOut+= String.format("%.2f", (double)studentScore/studentPossible*100)+"\t";
            dataOut+= String.format("%.2f", (double)classScore/classPossible*100)+"\t";


        } catch (SQLException e1) {
            e1.printStackTrace();
        }
        if(!badIn){
            System.out.println(headerOut+"\n"+dataOut);
            return headerOut+"\n"+dataOut;
        }else{
```

```
                return null;
        }
    }

    /**
     * This is the Check Quiz Grade method called by the web interface.
     * @param user
     * @param quiz
     * @param course
     * @return
     */
    public ArrayList<String> checkQuizGrade(String user, String quiz, String course){
        ArrayList<String> courses = new ArrayList<String>();
        ArrayList<String> quizes = new ArrayList<String>();
        ArrayList<String> names = new ArrayList<String>();
        ArrayList<String> header = new ArrayList<String>();
        ArrayList<Integer> stdScores = new ArrayList<Integer>();
        ArrayList<String> StudentUserNames = new ArrayList<String>();
        String userType=user_Type(user);
        String userName=user;
        String studentID="";
        String sql="";
        int quiz_num=Integer.parseInt(quiz);
        int course_num=Integer.parseInt(course);
        int i=0;
        int quizTotal=0;
        int classScore=0;
        int classPossible=0;
        double classAvg=0;
        double studentPercent=0;

        ResultSet r=null;
        Statement stm=null;

        sql="Select total, sum(score), sum(total) \n"+
                        "From (Select sa.Student_ID, sum(PT_value) as total \n"+
                        "From Student_Answer as sa, Student as s, Question as q, Question_Type as qt
\n"+
                        "Where sa.question_num=q.question_num and q.question_type=qt.question_type
and sa.Student_ID=s.Student_ID and sa.quiz_num="+quiz_num+" and sa.course_num="+course_num+"
group by s.Student_ID) as a, \n"+
                        "(Select sa.Student_ID, sum(PT_value) as score \n"+
                        "From Student_Answer as sa, Student as s, Question as q, Question_Type as qt
\n"+
                        "Where sa.question_num=q.question_num and q.question_type=qt.question_type
and q.solution=sa.student_answer and sa.Student_ID=s.Student_ID and sa.quiz_num="+quiz_num+" and
sa.course_num="+course_num+" group by s.Student_ID) as b \n"+
                        "where a.student_ID=b.student_ID";
```

```
		try{
			stm = conn.createStatement();
			r = stm.executeQuery(sql);

		while( r.next() ) {
			quizTotal=r.getInt(1);
			classScore=r.getInt(2);
			classPossible=r.getInt(3);
		}
		} catch (SQLException e1) {
			e1.printStackTrace();
		}
		classAvg=((double)classScore/classPossible*100);

		if(userType.equals("Student")){
			sql="(Select s.User_Name, concat(F_name,' ',L_name)as name, sum(PT_value) as
score\n"+
					"From Student_Answer as sa, Student as s, Question as q,
Question_Type as qt\n"+
					"Where sa.question_num=q.question_num and
q.question_type=qt.question_type and q.solution=sa.student_answer and sa.Student_ID=s.Student_ID and
sa.quiz_num="+quiz_num+" and sa.course_num="+course_num+" and s.User_Name='"+userName+"')\n";
		}else{
			sql="(Select s.User_Name, concat(F_name,' ',L_name)as name, sum(PT_value) as
score\n"+
					"From Student_Answer as sa, Student as s, Question as q,
Question_Type as qt\n"+
					"Where sa.question_num=q.question_num and
q.question_type=qt.question_type and q.solution=sa.student_answer and sa.Student_ID=s.Student_ID and
sa.quiz_num="+quiz_num+" and sa.course_num="+course_num+" group by s.Student_ID)\n";
		}
		try{
			stm = conn.createStatement();
			r = stm.executeQuery(sql);

		while( r.next() ) {
			StudentUserNames.add(r.getString(1));
			names.add(r.getString(2));
			stdScores.add(r.getInt(3));
		}
		} catch (SQLException e1) {
			e1.printStackTrace();
		}

		if(userType.equals("Student")){
			studentPercent=((stdScores.get(0)/(double)quizTotal)*100);
			System.out.println("studentPercent: "+studentPercent);
			header.add("Course");
```

```
            header.add("Quiz");
            header.add("Score");
            header.add("Total");
            header.add("Percent");
            header.add("ClassAVG");
            header.add(""+course_num);
            header.add(""+quiz_num);
            header.add(""+stdScores.get(0));
            header.add(""+quizTotal);
            header.add(""+String.format("%.2f",studentPercent));
            header.add(""+String.format("%.2f",classAvg));
}
else{
            header.add("Course");
            header.add("Quiz");
            header.add("Total");
            header.add("ClassAVG");
            header.add(""+course_num);
            header.add(""+quiz_num);
            header.add(""+quizTotal);
            header.add(""+String.format("%.2f",classAvg));
            header.add("Name");
            header.add("Score");
            header.add("Total");
            header.add("Percent");
            for(i=0; i < names.size();i++){
                        header.add(StudentUserNames.get(i));
                        header.add(names.get(i));
                        header.add(""+stdScores.get(i));
                        header.add(""+quizTotal);
                        header.add(""+String.format("%.2f",((double)stdScores.get(i)/quizTotal)*100));
            }
}

if(userType.equals("Student")){
            for(i=0;i<header.size();i++){
                        System.out.print(header.get(i)+"\t");
                        if(((i+1)%6)==0){
                                    System.out.println();
                        }
            }
}
else{
            for(i=0;i<header.size();i++){
                        System.out.print(header.get(i)+"\t");
                        if(((i+1)%4)==0){
                                    System.out.println();
                        }
```

```
                }
            }


            return header;
    }

     public ArrayList<ArrayList<String>> showQuizReport(String username, String quiz, String course){
            ArrayList<ArrayList<String>> report= new ArrayList<ArrayList<String>>();
            ArrayList<String> temp= new ArrayList<String>();
            ArrayList<String> temp1= new ArrayList<String>();
            ResultSet r=null;
           Statement stm=null;
            String studentID=getStudentID(username);
            String sql="Select sa.Question_num, Question_Type, Prompt, Solution, Student_Answer \n"+
                                "From Question as q, Student_Answer as sa \n"+
                                "Where q.Question_Num=sa.Question_Num and sa.Quiz_Num="+quiz+" and
sa.Course_Num="+course+" and sa. student_ID="'+studentID+"'\n ";

            try{
                        stm = conn.createStatement();
                        r = stm.executeQuery(sql);

                    while( r.next() ) {
                        ArrayList<String> temp2= new ArrayList<String>();
                        temp2.add(r.getString(1));
                        temp2.add(r.getString(2));
                        temp2.add(r.getString(3));
                        temp2.add(r.getString(4));
                        temp2.add(r.getString(5));
                        report.add(temp2);
                    }
                    } catch (SQLException e1) {
                        e1.printStackTrace();
                    }

            System.out.println("report size: "+report.size());
            for(int i=0;i<report.size();i++){
                temp=report.get(i);
                System.out.println("Temp get 1: "+temp.get(1));
                if(temp.get(1).equals("MC")){
                    temp1=getMultipleChoiceOptions(temp.get(0));
                    for(int j=0;j<temp1.size();j++){
                        temp.add(temp1.get(j));
                    }
                    report.set(i,temp);
                }
                else if(temp.get(1).equals("MA")){
```

```
                        temp1=getMatchingKeywords(temp.get(0));
                        for(int j=0;j<temp1.size();j++){
                                temp.add(temp1.get(j));
                        }
                        report.set(i,temp);
                }



        }
        for(int i=0;i<report.size();i++){
                for(int j=0;j<(report.get(i)).size();j++){
                        System.out.print(report.get(i).get(j));
                }
                System.out.println();

        }
        return report;
    }

        /**
         * Use case 4:
         * Name: Delete course
         * Purpose: Deletes a course from the roster.
         * User Group: Instructor / Blaha
         * Input: Course_Num
         * Result: The specific course will be deleted. All quiz scores will be deleted, as well
         * as all students enrolled in the course who are not also enrolled in other courses. If desired, all
quizzes from the course are also deleted. All questions are unaffected.
         * Exceptions: If there is at least one student in the course, the course cannot be removed unless
all students have been removed first.
         */
        public void deleteCourse(){
                String Course = "";
                System.out.println("Enter Course Number to delete: ");
                Course = keyboard.nextLine();
                /*
                 * First we handle exceptions
                 */
                ResultSet r = null;
                String sql = "";
                Statement stm;
                //if the course number doesn't exist in the database
                try{
                        if(!isCourse(Course))
                                System.out.println("Course number '" + Course + "' does not exist in the
database!");
                        else{//check if students in class
```

```
                                int Course_Num = (int) Integer.parseInt(Course);
                                sql = "Select Student_ID From Enrolled_In Where Course_Num = '" +
Course_Num + "'";
                                stm = conn.createStatement();
                                r = stm.executeQuery(sql);
                                if(r.next()){//if there are students in the class
                                        System.out.println("Cannot delete course '" + Course_Num + "'
because there are students enrolled in that course.");
                                        System.out.println("Delete students from course '"+ Course_Num
+ "' and also delete the course (y/n)?");
                                        String deleteAnyway = keyboard.nextLine();
                                        deleteAnyway = deleteAnyway.toLowerCase();
                                        if(deleteAnyway.charAt(0) == 'y'){
                                                System.out.println("Are you sure you want to delete
course '"+ Course_Num + "' (y/n)?");
                                                String finalAnswer = keyboard.nextLine();
                                                finalAnswer = finalAnswer.toLowerCase();
                                                if(finalAnswer.charAt(0) == 'y'){
                                                        backupDatabase();
                                                        sql = "Delete from Student where Student_ID in "
                                                                        +     "(Select Student_ID "
                                                                        +     "From Enrolled_In "
                                                                        +     "Where Course_Num = '" +
Course_Num + "' and Student_ID not in "
                                                                        +
                                                "(Select Student_ID "
                                                                        +
                                                 "From Enrolled_In "
                                                                        +
                                                 "Where Course_Num <> '" + Course_Num + "')"
                                                                        +     ")";
                                                        stm = conn.createStatement(); //these
statements delete the course
                                                        stm.execute(sql);
                                                        sql = "Delete from Course where Course_Num =
'" + Course_Num + "'";
                                                        stm = conn.createStatement();
                                                        stm.execute(sql);
                                                }
                                        }
                                }
                                else {//there are no students enrolled in the course
                                        System.out.println("Are you sure you want to delete course '"+
Course_Num + "' (y/n)?");
                                        String finalAnswer = keyboard.nextLine();
                                        finalAnswer = finalAnswer.toLowerCase();
                                        if(finalAnswer.charAt(0) == 'y'){
                                                backupDatabase();
```

```
                                                sql = "Delete from Course where Course_Num = '" +
Course_Num + "'";
                                                stm = conn.createStatement();
                                                r = stm.executeQuery(sql); //this deletes the course
                                        }
                                }
                        }
                        //keyboard.close();
                } catch (SQLException e) {
                        System.out.println("Problem with sql statement: " + sql);
                        System.out.println(e.getMessage());
                }
        }

        /**
         * The delete course called by the web interface.
         * @param courseNums
         */
        public void deleteCourse(String[] courseNums){
                for(int i = 0; i < courseNums.length; i++){
                        deleteACourse(courseNums[i]);
                }
        }

        /**
         * A helper method that deletes a single course.  Called indirectly from the Web Interface.
         * @param courseNum
         */
        private void deleteACourse(String courseNum){
                String sql;
                Statement stm;
                sql = "Delete from Student where Student_ID in "
                                +     "(Select Student_ID "
                                +     "From Enrolled_In "
                                +     "Where Course_Num = '" + courseNum + "' and Student_ID not in "
                                +
        "(Select Student_ID "
                                +
                 "From Enrolled_In "
                                +
                 "Where Course_Num <> '" + courseNum + "')"
                                +    ")";
                try {
                        stm = conn.createStatement();
                        stm.execute(sql);
                } catch (SQLException e) {
                        System.out.println("Problem with SQL Statement: " + sql);
                        System.out.println(e.getMessage());
```

```
		} //these statements delete the course
		sql = "Delete from Course where Course_Num = '" + courseNum + "'";
		try {
			stm = conn.createStatement();
			stm.execute(sql);
		} catch (SQLException e) {
			System.out.println("Problem with SQL Statement: " + sql);
			System.out.println(e.getMessage());
		}
	}

	/**
	 * Use case 5:
	 * Name: Find Available Questions
	 * Purpose: Provides the Instructor with a list of questions that are available to be asked on a quiz, based on the courses they are relevant for and tags the instructor has defined.
	 * User Group: Instructor / Blaha
	 * Input: Course_Num, Tag(s)
	 * Result: A list of the Questions that match the course and any additional tags if present
	 * Exceptions: If the course number does not match a course in the database a list of possible courses will be displayed with an error message. If the combination of course number and optional tags returns no questions, an error message will display the attempted search terms, and allow for an additional search attempts.
	 */
	public void findAvailableQuestions(){
		ArrayList<String> courses = new ArrayList<String>();
		ArrayList<String> existingTags = new ArrayList<String>();
		//print courses horizontally with commas
		String sql = "Select Course_Num from Course";
		Statement stm;
		ResultSet r = null;
		try {
			stm = conn.createStatement();
			r = stm.executeQuery(sql);
		} catch (SQLException e) {
			System.out.println("Problem with SQL Statement: " + sql);
			System.out.println(e.getMessage());
		}
		System.out.print("Active Courses: ");
		try {
			while(r.next()){
				System.out.print(r.getString(1) + ", ");
				courses.add(r.getString(1));
			}
			System.out.println();
		} catch (SQLException e) {
			System.out.println("Problem with getString");
			System.out.println(e.getMessage());
```

```
		}

		//print tags same way

		existingTags = existingTags();

		//get course number and tags
		String courseNum = "";

		while(!isInt(courseNum) || !courseIsInUseCase5(courseNum, courses)){
			System.out.println("Enter course number: ");
			courseNum = keyboard.nextLine();
			if(!isInt(courseNum))
				System.out.println("Enter an integer value for a course number.");
			if(!courseIsInUseCase5(courseNum, courses) && !courseNum.equals(""))
				System.out.println("Course number entered does not exist.");
		}
		int course_Num = Integer.parseInt(courseNum);

		ArrayList<String> selectedTags = getTags();
		//Find available questions
		sql =   "SELECT distinct Question_Type, Prompt "
				+ "FROM Question as Q join Tags on Q.Question_Num =
Tags.Question_Num, Course_Question as CQ "
				+ "WHERE Q.Question_Num = CQ.Question_Num and Course_Num = '"
+ course_Num + "'";
		//THE TAGS PART ONLY EXISTS IF THERE ARE TAGS
		if(selectedTags.size() > 0){
			sql += " and (";
			for(int j = 0; j<selectedTags.size(); j++){
				selectedTags.set(j,accomodateForProblemChars(selectedTags.get(j)));
				sql += "Tags.tag = '" + selectedTags.get(j) + "'";
				if(j != selectedTags.size() - 1)
					sql += " or ";
			}
			sql += ")";
		}

		try {
			stm = conn.createStatement();
			r = stm.executeQuery(sql);
		} catch (SQLException e) {
			System.out.println("Problem with SQL Statement: " + sql);
			System.out.println(e.getMessage());
		}

		//print course num, tags, and available questions
```

```
                System.out.println("Questions that may be added to a quiz for Course Number: " +
course_Num);
                if(selectedTags.size() > 0){
                        System.out.print("and tag(s): ");
                        for(int j = 0; j<selectedTags.size() - 1; j++){
                                System.out.print(selectedTags.get(j) + ", ");
                        }
                        System.out.println(selectedTags.get(selectedTags.size() - 1));
                }
                System.out.println("\nQuestion Type\t\tPrompt");
                try {
                        while(r.next()){
                                switch(r.getString(1)){
                                case "TF": System.out.print("True/False:\t\t");
                                break;
                                case "MC": System.out.print("Multiple Choice:\t");
                                break;
                                case "MA": System.out.print("Matching:\t\t");
                                break;
                                case "FB": System.out.print("Fill In The Blank:\t");
                                break;
                                }
                                System.out.println(r.getString(2));
                        }
                } catch (SQLException e) {
                        System.out.println("Problem getting data from ResultSet.");
                        System.out.println(e.getMessage());
                }
        }

        /**
         * This is the findAvailableQuestions method called by the web interface.
         * @param course_Num
         * @param selectedTags
         * @return
         */
        public ResultSet findAvailableQuestions(String course_Num, String[] selectedTags){
                String sql ="";
                Statement stm;
                ResultSet r = null;
                sql =   "SELECT distinct Q.Question_Num, Question_Type, Prompt "
                                + "FROM Question as Q join Tags on Q.Question_Num =
Tags.Question_Num, Course_Question as CQ "
                                + "WHERE Q.Question_Num = CQ.Question_Num and Course_Num = '"
+ course_Num + "'";
                //THE TAGS PART ONLY EXISTS IF THERE ARE TAGS
                if(selectedTags.length > 0){
                        sql += " and (";
```

```
                        for(int j = 0; j<selectedTags.length; j++){
                                selectedTags[j] = accomodateForProblemChars(selectedTags[j]);
                                sql += "Tags.tag = '" + selectedTags[j] + "'";
                                if(j != selectedTags.length - 1)
                                        sql += " or ";
                        }
                        sql += ")";
                }

                try {
                        stm = conn.createStatement();
                        r = stm.executeQuery(sql);
                } catch (SQLException e) {
                        System.out.println("Problem with SQL Statement: " + sql);
                        System.out.println(e.getMessage());
                }
                return r;
        }

        /**
         * This method populates the database from text files.
         */
        /*
    public void populateDatabase(){
        //first we delete everything in the database.
        for(int i = 0; i<tableNames.size(); i++){
                ResultSet r = null;
                String sql = "delete from " + tableNames.get(i);
                Statement stm;
                try {
                        stm = conn.createStatement();
                        stm.execute(sql);
                } catch (SQLException e) {
                        System.out.println("Problem with SQL statement: " + sql);
                        System.out.println(e.getMessage());
                }
        }
        //Then we insert everything from the files.
        for(int i = 0; i<tableNames.size(); i++){
                int passwordLoc = -1;
                String filename = tableNames.get(i) + ".txt";
                try {
                        Scanner infile = new Scanner(new File(filename));
                        infile.useDelimiter(";|\\r?\\n");
                        ResultSet r = null;
                        //TODO parse the values
                        int numAtts = tableDescriptions.get(i).size() - 1; //minus 1 because the first is the
table name
```

```
                        while(infile.hasNext()){ //parses a file
                                String header = infile.nextLine(); //skips the header line
                                //CHECKS FOR PASSWORD FIELD
                                //if passwordLoc = -1, there is no password field. If it is some positive
value, that is the index of the password
                                Scanner head = new Scanner(header);
                                head.useDelimiter(",");
                                int index = 0;
                                while(head.hasNext()){
                                        String s = head.next();
                                        s = s.toLowerCase();
                                        if(s.equals("password"))
                                                passwordLoc = index;
                                        index ++;
                                }
                                /**
         * THIS IS WHERE THE PARSE LINE GOES
         */
        /*
                                String valuesString = parseLine(numAtts, passwordLoc, infile);
                                if(infile.hasNext()){
                                        //System.out.println("valuesString before add: " + valuesString);
                                        String lastToken = infile.next();
                                        char[] last = lastToken.toCharArray();
                                        if(last[last.length-1] == '\n')
                                                lastToken = lastToken.substring(0,lastToken.length() -
1);
                                        valuesString += "'" + lastToken + "'" ; //avoids adding an extra
comma to the end of valuesString
                                        //System.out.println("valuesString after add: " + valuesString);
                                }

                                String sql = "insert into " + tableNames.get(i) + " Values(" + valuesString
+ ")";
                                try {
                                        Statement stm = conn.createStatement();
                                        stm.execute(sql);
                                } catch (SQLException e) {
                                        System.out.println("Problem with SQL statement: " + sql);
                                        System.out.println(e.getMessage());
                                }
                        }
                        infile.close();
                } catch (FileNotFoundException e) {
                        System.out.println("File '" + filename + "' not found.");
                        System.out.println(e.getMessage());
                }
                passwordLoc = -1;
```

```
	}
}

/**
	 * This is a private helper method that parses a line of the input file when populating the database
	 */
	/*
private String parseLine(int numAtts, int passwordLoc, Scanner infile){
	String valuesString = "";
	for(int j = 0; j< numAtts - 1; j++){ //parses a row of the file
		//System.out.println("valuesString before add: " + valuesString);
		String s = infile.next();
		if(j == passwordLoc){
			s = parseWithPassword(s);
		}
		else{
			s = parse(s);
		}
		valuesString += "'" +  s + "'" +  ", ";
		//System.out.println("valuesString after add: " + valuesString);
	}
	return valuesString;
}

private String parseWithPassword(String s){
	//I am at a password attribute. The SQL must look like: Password('insert password here')
	String toAdd = "Password(";

	return s;
}

private String parse(String s){

	return s;
}

private String accomodateForProblemChars(String s){
	//CHECK FOR PROBLEM CHARACTERS (just ' for now):
	int indexLastProblem = -1;
	for(int t = 0; t<s.length(); t++){
		if(s.charAt(t) == '\''){
			String temp = s;
			temp = s.substring(indexLastProblem + 1, t) + ;
		}
	}
	return s;
}*/
```

```
	/**
	 * This method backs up the database
	 */
	public void backupDatabase(){
		for(int i = 0; i<tableNames.size(); i++){
			ResultSet r = null;
			String sql = "delete from " + tableNames.get(i) + "_Backup";
			Statement stm;
			try {
				stm = conn.createStatement();
				stm.executeUpdate(sql);
				sql = "insert into " + tableNames.get(i) + "_Backup" + " Select * from " +
tableNames.get(i);
				stm = conn.createStatement();
				stm.executeUpdate(sql);
			} catch (SQLException e) {
				System.out.println("Problem with SQL statement: " + sql);
				System.out.println(e.getMessage());
			}
		}
	}

	/**
	 * This method restores the database to a backed up state.
	 */
	public void restoreDatabase(){
		for(int i = 0; i<tableNames.size(); i++){
			ResultSet r = null;
			String sql = "delete from " + tableNames.get(i);
			Statement stm;
			try {
				stm = conn.createStatement();
				stm.executeUpdate(sql);
			} catch (SQLException e) {
				System.out.println("Problem with SQL statement: " + sql);
				System.out.println(e.getMessage());
			}
		}
		for(int i = 0; i< tableNames.size(); i++){
			ResultSet r = null;
			String sql = "insert into " + tableNames.get(i) + " Select * from " +
tableNames.get(i) + "_Backup";
			Statement stm;
			try {
				stm = conn.createStatement();
				stm.executeUpdate(sql);
			} catch (SQLException e) {
				System.out.println("Problem with SQL statement: " + sql);
```

```
                        System.out.println(e.getMessage());
                }
        }
}

/**
 * Close the connection to the DB
 */
public void closeDB() {
        try {
                keyboard.close();
                conn.close();
                conn = null;
        } catch (SQLException e) {
                e.printStackTrace();
        }
}

//String helper methods

public boolean isDouble(String s){
        boolean deci = false;
        boolean minusSign = false;
        char[] string = s.toCharArray();
        for(int i = 0; i<string.length; i++){
                if (i == 0){
                        if(string[0] != '-' && !isDigit(string[0]))
                                return false;
                }
                if(string[i] == '.' && deci == true)
                        return false;
                if(string[i] =='.')
                        deci = true;
                if(!isDigit(string[i]))
                        return false;
        }
        return true;
}

public boolean isInt(String s){
        char[] string = s.toCharArray();
        for(int i = 0; i<string.length; i++){
                if (i == 0){
                        if(string[0] != '-' && !isDigit(string[0]))
                                return false;
                }
                if(!isDigit(string[i]))
                        return false;
```

```
		}
		return true;
	}

	public boolean isDigit(char c){
		if(c == '1' || c == '2' || c == '3' || c == '4' || c == '5' || c == '6' || c == '7' || c == '8' || c == '9' || c
== '0')
			return true;
		return false;
	}

	private boolean courseIsInUseCase5(String s, ArrayList<String> courses){
		for(int i = 0; i<courses.size(); i++){
			if(s.equals(courses.get(i)))
				return true;
		}
		return false;
	}

	private boolean tagsIsInUseCase5(String s, ArrayList<String> tags){
		for(int i = 0; i<tags.size(); i++){
			if(s.equalsIgnoreCase(tags.get(i)))
				return true;
		}
		return false;
	}

	public String accomodateForProblemChars(String s){
		//CHECK FOR PROBLEM CHARACTERS (just ' for now): <<<GENIUES
		String temp = s;
		int indexLastProblem = -1;
		for(int t = 0; t<s.length(); t++){
			if(s.charAt(t) == '\''){
				temp = s.substring(indexLastProblem + 1, t) + "\'" +  s.substring(t,
s.length());
			}
		}
		return temp;
	}

	private boolean isCourse(String Course){
		if(!isInt(Course)){
			return false;
		}
		if(Course.equals(""))
			return false;
		int Course_Num = Integer.parseInt(Course);
		ResultSet r = null;
```

```
		String sql = "Select Course_Num	From Course";
		try{
			Statement stm = conn.createStatement();
			r = stm.executeQuery(sql);
			while(r.next()){
				if(Integer.parseInt(r.getString(1)) == Course_Num)
					return true;
			}
			return false;
		} catch(SQLException e){
			System.out.println("Problem with SQL Statement: " + sql);
			System.out.println(e.getMessage());
		}
		return false;
	}

	public ResultSet listCourses(){
		ResultSet r = null;
		String sql = "Select Course_Num	From Course";
		try{
			Statement stm = conn.createStatement();
			r = stm.executeQuery(sql);
		} catch(SQLException e){
			System.out.println("Problem with SQL Statement: " + sql);
			System.out.println(e.getMessage());
		}
		return r;
	}

	private int questionNum(String type, String soln, String prompt){
		ResultSet r = null;
		Statement stm;
		String sql =  "Select Question_Num "
				+ "From Question "
				+ "Where Question_Type = '" + type + "' and Solution = '" + soln + "' and
Prompt = '" + prompt + "'";
		try {
			stm = conn.createStatement();
			r = stm.executeQuery(sql);
		} catch (SQLException e) {
			System.out.println("Problem with SQL statement: " + sql);
			System.out.println(e.getMessage());
		}

		String Question_Num = "";
		try {
			r.next();
			Question_Num = r.getString(1);
```

```
		} catch (SQLException e) {
			System.out.println("Problem getting question number.");
			System.out.println(e.getMessage());
		}
		int q_num = Integer.parseInt(Question_Num);
		return q_num;
	}

	private ArrayList<String> getTags(){
		ArrayList<String> existingTags = existingTags();
		ArrayList<String> selectedTags = new ArrayList<String>();
		System.out.println("Enter tags (if desired). Please enter only 1 tag at a time. Simply press
enter for no tags: ");
		String tag = "-";
		while(!tag.equals("")){
			System.out.println("Enter tag: ");
			tag = keyboard.nextLine();
			if(tagsIsInUseCase5(tag, existingTags))
				selectedTags.add(tag);
			else if(tag.equals(""))
				break;
			else {
				System.out.println("Enter existing tag or simply press enter to finish
entering tags.");
				tag = "-";
			}
		}
		return selectedTags;
	}

	public ArrayList<String> existingTags(){
		String sql;
		ResultSet r = null;
		Statement stm;
		ArrayList<String> existingTags = new ArrayList<String>();
		sql = "Select distinct Tag from Tags";
		try {
			stm = conn.createStatement();
			r = stm.executeQuery(sql);
		} catch (SQLException e) {
			System.out.println("Problem with SQL Statement: " + sql);
			System.out.println(e.getMessage());
		}
		if(!web){
			System.out.print("Active Tags: ");
		}
		try {
			while(r.next()){
```

```
                                System.out.print(r.getString(1) + ", ");
                                existingTags.add(r.getString(1));
                        }
                        System.out.println();
                } catch (SQLException e) {
                        System.out.println("Problem with getString");
                        System.out.println(e.getMessage());
                }
                return existingTags;
        }

        /**
         * ONLY FOR USE CASE 1 works for command line input only
         * @param Ans_Options
         * @param numAnswers
         * @return
         */
        public boolean unique(String[] Ans_Options, int numAnswers){
                for(int i = 0; i < numAnswers; i++){
                        if(Ans_Options[numAnswers].equals(Ans_Options[i]))
                                return false;
                }
                return true;
        }

        /**
         * ONLY FOR USE CASE 1 works for both inputs
         * @param Ans_Options
         * @param numAnswers
         * @return
         */
        public boolean uniqueWeb(String[] Ans_Options, int numAnswers){
                for(int i = 0; i < Ans_Options.length; i++){
                        for(int j = i + 1; j < Ans_Options.length; j++){
                                if(Ans_Options[i].equalsIgnoreCase(Ans_Options[j]) &&
!Ans_Options[i].equals(""))
                                        return false;
                        }
                }
                return true;
        }

        public String user_Type(String username){
                //check if admin
                if(username.equals("hippo367"))
                        return "Admin";
                //check if Instructor
                String sql = "Select User_Name from Instructor";
```

```
		Statement stm;
		ResultSet r = null;
		try {
			stm = conn.createStatement();
			r = stm.executeQuery(sql);
		} catch (SQLException e) {
			System.out.println("Problem with SQL Statement: " + sql);
			System.out.println(e.getMessage());
		}
		if(!(r == null)){
			try {
				while(r.next()){
					if(username.equals(r.getString(1)))
						return "Instructor";
				}
			} catch (SQLException e) {
				e.printStackTrace();
			}
		}
		//check if Student
		sql = "Select User_Name from Student";
		try {
			stm = conn.createStatement();
			r = stm.executeQuery(sql);
		} catch (SQLException e) {
			System.out.println("Problem with SQL Statement: " + sql);
			System.out.println(e.getMessage());
		}
		if(!(r == null)){
			try {
				while(r.next()){
					if(username.equals(r.getString(1)))
						return "Student";
				}
			} catch (SQLException e) {
				e.printStackTrace();
			}
		}
		return "Invalid";
	}

	public boolean verifyPassword(String username, String password, String user_Type){
		//String password = decodePassword(encodedPassword);
		//and change to BigInteger encodedPassword when doing that.
		boolean verified = false;
		if(!user_Type.equals("Instructor") && !user_Type.equals("Student") &&
!user_Type.equals("Admin")){
			System.out.println("Invalid user_Type: " + user_Type);
```

```
                verified =  false;
            }//If it's an admin, the check is simple:
            else if(password.equals("hippo_367.DB!") && user_Type.equals("Admin") &&
username.equals("hippo367")){
                verified = true;
            }
            else if(user_Type.equals("Admin")){
                verified = false;
            }
            else{    //check for instructor and student
                String sql = "";
                ResultSet r = null;
                Statement stm;
                if(user_Type.equals("Instructor")) //first I insert a temp user to encode the given
password
                    sql = "Insert into Instructor Values('temp', Password('" + password + "'),
'temp@plu.edu', 'temp', 't', 'temp')";
                else sql = "Insert into Student Values('temp', Password('" + password + "'),
'temp@plu.edu', 'temp', 't', 'temp', '99999999')";
                try {
                    stm = conn.createStatement();
                    stm.executeUpdate(sql);
                } catch (SQLException e) {
                    System.out.println("Problem with SQL Statement: " + sql);
                    System.out.println(e.getMessage());
                }
                //then I get the password of the user I hope to verify the password for
                String dbPassword = "";
                sql = "Select password from " + user_Type + " where User_Name = '" +
username + "'";
                try {
                    stm = conn.createStatement();
                    r = stm.executeQuery(sql);
                } catch (SQLException e) {
                    System.out.println("Problem with SQL Statement: " + sql);
                    System.out.println(e.getMessage());
                }
                if(!(r == null)){
                    try {
                        r.next();
                        dbPassword = r.getString(1);
                    } catch (SQLException e) {
                        System.out.println("Problem with getting password from
ResultSet.");
                        System.out.println(e.getMessage());
                    }
                }
                //then I get the password of the temp I inserted
```

```
                        String tempPassword = "";
                        //then I get the password of the temp user (ie what the dbPassword should match)
                        sql = "Select password from " + user_Type + " where User_Name = 'temp'";
                        try {
                                stm = conn.createStatement();
                                r = stm.executeQuery(sql);
                        } catch (SQLException e) {
                                System.out.println("Problem with SQL Statement: " + sql);
                                System.out.println(e.getMessage());
                        }
                        if(!(r == null)){
                                try {
                                        r.next();
                                        tempPassword = r.getString(1);
                                } catch (SQLException e) {
                                        System.out.println("Problem with getting password from
ResultSet.");
                                        System.out.println(e.getMessage());
                                }
                        }
                        //finally I see if the 2 passwords are equal
                        if(tempPassword.equals(dbPassword)){
                                if(!tempPassword.equals(""))
                                        verified = true;
                        }
                        //before I return, I delete the temp entry
                        sql = "delete from " + user_Type + " where User_Name = 'temp'";
                        try {
                                stm = conn.createStatement();
                                stm.executeUpdate(sql);
                        } catch (SQLException e) {
                                System.out.println("Problem with SQL Statement: " + sql);
                                System.out.println(e.getMessage());
                        }
                }
                return verified;
        }

        /**
         * For use case 1. Determines if the user selected a solution that has no answer option.
         * @param Ans_Options
         * @param solution
         * @return
         */
        public boolean validSoln(String[] Ans_Options, String solution){
                char soln = solution.toLowerCase().charAt(0);
                switch(soln){
                case 'a': if(Ans_Options[0].equals("")){
```

```
                return false;}
        break;
        case 'b':if(Ans_Options[1].equals("")){
                return false;}
        break;
        case 'c':if(Ans_Options[2].equals("")){
                return false;}
        break;
        case 'd':if(Ans_Options[3].equals("")){
                return false;}
        break;
        case 'e':if(Ans_Options[4].equals("")){
                return false;}
        break;
        }
        return true;
}

public ArrayList<String> checkQuizList(String course){
        ArrayList<String> quizes = new ArrayList<String>();
        String course_num=course;
        String sql="";
        ResultSet r=null;
        Statement stm=null;

        sql="select quiz_num\n From Quiz\n Where course_num="+course_num;
        try{
                stm = conn.createStatement();
                r = stm.executeQuery(sql);
                //System.out.println("TEST QUIZLIST");
                while( r.next() ) {
                        quizes.add(r.getString(1));
                        //System.out.println(r.getString(1));
                }
        } catch (SQLException e1) {
                e1.printStackTrace();
        }
        //quizes now contains a selection of each quiz offered in that class
        return quizes;
}

public ArrayList<String> checkQuizCourseList(String userName){
        String type=user_Type(userName);
        ArrayList<String> courses = new ArrayList<String>();
        ResultSet r=null;
        Statement stm=null;
        String sql="";
        String studentID="";
```

```
		if(type.equals("Student")){

			sql="select student_ID\n from Student\n where User_Name='"+userName+"'";
			try{
				stm = conn.createStatement();
				r = stm.executeQuery(sql);

				while( r.next() ) {
					studentID=r.getString(1);
				}
			} catch (SQLException e1) {
				e1.printStackTrace();
			}

			sql="select course_num\n From Enrolled_In\n Where
Student_ID='"+studentID+"'";
		}
		else{
			sql="Select course_num\n From Course";
		}
		try{
			stm = conn.createStatement();
			r = stm.executeQuery(sql);

			while( r.next() ) {
				courses.add(r.getString(1));
			}
		} catch (SQLException e1) {
			e1.printStackTrace();
		}
		// courses now has all available classes to be displayed in a dropdown selection
		return courses;
	}

	/**
	 * This method encodes a password via an RSA algorithm.
	 * @param password
	 * @return
	 */
	public BigInteger encodePassword(String password){
		//first we take the password and convert it to a string which is a sequence of integers (a
semi-ASCII form)
		String intFormMessage = "";
		char[] message = password.toCharArray();
		for(int i = 0; i < message.length; i++){
			intFormMessage += CharsToInt(message[i]);
		}
```

```
		//intFormMessage is m.
		BigInteger m = new BigInteger(intFormMessage);
		return m.modPow(k, n);
	}

	/**
	 * This method decodes a password encoded with the RSA algorithm used.
	 * @param password
	 * @return
	 */
	public String decodePassword(BigInteger c){
		//I need r.
		//If I did it right, r is the following
		BigInteger r = new
BigInteger("92761351101769894533398925185642408837930080032056461818138495256949672215516197969046650324269");
		BigInteger decoded = c.modPow(r, n);
		String decodedString = decoded.toString();
		char[] decodedchars = decodedString.toCharArray();
		System.out.println("decoded password = " + decodedString);
		for(int i = 0; i < decodedchars.length; i++){
			System.out.print(decodedchars[i]);
		}
		System.out.println();
		System.out.println(decodedString.charAt(0));
		String password = "";
		for(int i = 0; i < decodedchars.length; i=i+2){
			String token = decodedchars[i] + "" + decodedchars[i+1] + "";
			System.out.println("token = " + token);
			password += intToChars(token);
		}
		return password;
	}

	private BigInteger gcd(BigInteger m, BigInteger n){
		BigInteger mModn = m.mod(n);
		if(mModn == BigInteger.ZERO)
			return n;
		else{
			BigInteger mult = m.divide(m.subtract(mModn));
			ms.add(m);
			ns.add(n);
			mults.add(mult);
			rs.add(mModn);
			return gcd(n, mModn);
		}
	}
```

```
	/**This code came from:
	 *
https://code.google.com/a/eclipselabs.org/p/cryptocodes/source/browse/trunk/Crypto/src/gcd.java?r=8
	 */
	//*************THIS IS THE XGCD**************//
	/*********************************************************
	 *	A short Java program to find the GCD of two integers.
	 * The "error checking" part of the code is significantly
	 * longer than the actual code to find the GCD itself.
	 *
	 *	The program should be invoked with two integer
	 * arguments like
	 *
	 *	java GCD 230 64
	 *
	 * The error checking consists of verifying there are at
	 * least two (integer) arguments (and that these are the
	 * first arguments given to the program), and that the
	 * two inputs are not both 0, as gcd(0,0) is not defined.
	 *
	 *   Runnig time:
	 * Since this is really just a  straightforward
	 * implementation of the Extended Euclidean
	 * algorithm, as described in the lecture notes, the
	 * running time of this algorithm is O(log (max ({a, b})).
	 *********************************************************/
	private BigInteger GCD_(BigInteger[] args)
	{
		BigInteger a=BigInteger.ZERO, b=BigInteger.ZERO, d;
		BigInteger sign_a = BigInteger.ONE, sign_b = BigInteger.ONE;   //  These are used to
handle cases when a < 0
		//  and/or b < 0, as the extendedEuclid function
		//  assumes that a and b are nonnegative.
		BigInteger[] ans = new BigInteger[3];

		/*  Check for errors, see if user gave enough arguments, etc.  */

		if (args.length < 2)  {
			System.out.println("\n   Please input two arguments!\n");
			System.exit(1);
		}
		else if (args.length > 0)  {
			try {
				a = args[0];
			}
			catch (NumberFormatException e) {
				System.err.println("\n   Arguments must be integers!\n");
				System.exit(1);
```

```
                }
                try {
                        b = args[1];
                }
                catch (NumberFormatException e) {
                        System.err.println("\n  Arguments must be integers!\n");
                        System.exit(1);
                }
        }  /*  end of "if" block to catch input errors  */


        /*  Check for input of two zeros, and print error if that's that case  */

        if ((a.equals(BigInteger.ZERO)) && (b.equals(BigInteger.ZERO))) {
                System.out.println("\n  Oops, both arguments are zero!  No GCD of zero!\n");
                System.exit(1);
        }

        /*  If a and/or b is negative, then track this information for later output, because
    the extendedEuclid function expects nonnegative input.  */
        if ( a.min(BigInteger.ZERO).equals(a) ) //a < 0
        {
                sign_a = BigInteger.ONE.negate();
                a = a.abs();
        }
        if ( b.min(BigInteger.ZERO).equals(b) ) //b < 0
        {
                sign_b = BigInteger.ONE.negate();
                b = b.abs();
        }

        /*  Find the answer and print it out  */
        ans = xgcd(a,b);
        System.out.println("\n   gcd(" + a.multiply(sign_a) + "," + b.multiply(sign_b) +") = " +
ans[0]);
        System.out.print("    " + ans[0] + " = (" + sign_a.multiply(ans[1]) + ") (" + sign_a.multiply(a)
+")");
        System.out.println(" + (" + sign_b.multiply(ans[2]) + ") (" + sign_b.multiply(b) + ")\n");
        return sign_b.multiply(ans[1]);
    }

    private BigInteger[] xgcd(BigInteger a, BigInteger b){
        BigInteger[] ans = new BigInteger[3];
        BigInteger q;

        if (b.equals(BigInteger.ZERO))  {  /*  If b = 0, then we're done...  */
                ans[0] = a;
                ans[1] = BigInteger.ONE;
```

```
                ans[2] = BigInteger.ZERO;
            }
            else
            {       /*  Otherwise, make a recursive function call  */
                q = a.divide(b);
                ans = xgcd(b, a.mod(b));
                BigInteger temp = ans[1].subtract(ans[2].multiply(q));
                ans[1] = ans[2];
                ans[2] = temp;
            }

            return ans;
        }
        /*  end of "main" program  */
        //********************END OF XGCD************************//

        //FOR THIS EULER PHI, WE WILL ASSUME THAT IT COMES FROM 2 PRIME NUMBERS, P
AND Q
        private BigInteger euler_phi(BigInteger p, BigInteger q) {
            return p.add(BigInteger.ONE.negate()).multiply(q.add(BigInteger.ONE.negate()));
        }

        private char intToChars(String n){
            char c = ';';
            switch (n){
            case "10": c = 'k';
            break;
            case "11": c = 'l';
            break;
            case "12": c = 'm';
            break;
            case "13": c = 'n';
            break;
            case "14": c = 'o';
            break;
            case "15": c = 'p';
            break;
            case "16": c = 'q';
            break;
            case "17": c = 'r';
            break;
            case "18": c = 's';
            break;
            case "19": c = 't';
            break;
            case "20": c = 'u';
            break;
            case "21": c = 'v';
```

```
break;
case "22": c = 'w';
break;
case "23": c = 'x';
break;
case "24": c = 'y';
break;
case "25": c = 'z';
break;
case "26": c = 'a';
break;
case "27": c = 'b';
break;
case "28": c = 'c';
break;
case "29": c = 'd';
break;
case "30": c = 'e';
break;
case "31": c = 'f';
break;
case "32": c = 'g';
break;
case "33": c = '!';
break;
case "34": c = '\"';
break;
case "35": c = '#';
break;
case "36": c = '$';
break;
case "37": c = '%';
break;
case "38": c = '&';
break;
case "39": c = '\';
break;
case "40": c = '(';
break;
case "41": c = ')';
break;
case "42": c = '*';
break;
case "43": c = '+';
break;
case "44": c = '-';
break;
case "45": c = '.';
```

```
break;
case "46": c = '/';
break;
case "47": c = '0';
break;
case "48": c = '1';
break;
case "49": c = '2';
break;
case "50": c = '3';
break;
case "51": c = '4';
break;
case "52": c = '5';
break;
case "53": c = '6';
break;
case "54": c = '7';
break;
case "55": c = '8';
break;
case "56": c = '9';
break;
case "57": c = ':';
break;
case "58": c = ';';
break;
case "59": c = '<';
break;
case "60": c = '=';
break;
case "61": c = '>';
break;
case "62": c = '?';
break;
case "63": c = '@';
break;
case "64": c = 'A';
break;
case "65": c ='B';
break;
case "66": c = 'C';
break;
case "67": c = 'D';
break;
case "68": c = 'E';
break;
case "69": c = 'F';
```

```
break;
case "70": c = 'G';
break;
case "71": c = 'H';
break;
case "72": c = 'I';
break;
case "73": c = 'J';
break;
case "74": c = 'K';
break;
case "75": c = 'L';
break;
case "76": c = 'M';
break;
case "77": c = 'N';
break;
case "78": c = 'O';
break;
case "79": c = 'P';
break;
case "80": c = 'Q';
break;
case "81": c = 'R';
break;
case "82": c = 'S';
break;
case "83": c = 'T';
break;
case "84": c = 'U';
break;
case "85": c = 'V';
break;
case "86": c = 'W';
break;
case "87": c = 'X';
break;
case "88": c = 'Y';
break;
case "89": c = 'Z';
break;
case "90": c = '[';
break;
case "91": c = '\\';
break;
case "92": c = ']';
break;
case "93": c = '^';
```

```
            break;
            case "94": c = '_';
            break;
            case "95": c = '~';
            break;
            case "96": c = '`';
            break;
            case "97": c = 'h';
            break;
            case "98": c = 'i';
            break;
            case "99": c = 'j';
            break;
            }
            return c;
}

private String CharsToInt(char c){
            String s = "-1";
            switch (c){
            case 'a': s = "26";
            break;
            case 'b': s ="27";
            break;
            case 'c': s = "28";
            break;
            case 'd': s = "29";
            break;
            case 'e': s = "30";
            break;
            case 'f': s = "31";
            break;
            case 'g': s = "32";
            break;
            case 'h': s = "97";
            break;
            case 'i': s = "98";
            break;
            case 'j': s = "99";
            break;
            case 'k': s = "10";
            break;
            case 'l': s = "11";
            break;
            case 'm': s = "12";
            break;
            case 'n': s = "13";
            break;
```

```
case 'o': s = "14";
break;
case 'p': s = "15";
break;
case 'q': s = "16";
break;
case 'r': s = "17";
break;
case 's': s = "18";
break;
case 't': s = "19";
break;
case 'u': s = "20";
break;
case 'v': s = "21";
break;
case 'w': s = "22";
break;
case 'x': s = "23";
break;
case 'y': s = "24";
break;
case 'z': s = "25";
break;
case '!': s = "33";
break;
case \'": s = "34";
break;
case '#': s = "35";
break;
case '$': s = "36";
break;
case '%': s = "37";
break;
case '&': s = "38";
break;
case \': s = "39";
break;
case '(': s = "40";
break;
case ')': s = "41";
break;
case '*': s = "42";
break;
case '+': s = "43";
break;
case '-': s = "44";
break;
```

```
case '.': s = "45";
break;
case '/': s = "46";
break;
case '0': s = "47";
break;
case '1': s = "48";
break;
case '2': s = "49";
break;
case '3': s = "50";
break;
case '4': s = "51";
break;
case '5': s = "52";
break;
case '6': s = "53";
break;
case '7': s = "54";
break;
case '8': s = "55";
break;
case '9': s = "56";
break;
case ':': s = "57";
break;
case ';': s = "58";
break;
case '<': s = "59";
break;
case '=': s = "60";
break;
case '>': s = "61";
break;
case '?': s = "62";
break;
case '@': s = "63";
break;
case 'A': s = "64";
break;
case 'B': s ="65";
break;
case 'C': s = "66";
break;
case 'D': s = "67";
break;
case 'E': s = "68";
break;
```

```
case 'F': s = "69";
break;
case 'G': s = "70";
break;
case 'H': s = "71";
break;
case 'I': s = "72";
break;
case 'J': s = "73";
break;
case 'K': s = "74";
break;
case 'L': s = "75";
break;
case 'M': s = "76";
break;
case 'N': s = "77";
break;
case 'O': s = "78";
break;
case 'P': s = "79";
break;
case 'Q': s = "80";
break;
case 'R': s = "81";
break;
case 'S': s = "82";
break;
case 'T': s = "83";
break;
case 'U': s = "84";
break;
case 'V': s = "85";
break;
case 'W': s = "86";
break;
case 'X': s = "87";
break;
case 'Y': s = "88";
break;
case 'Z': s = "89";
break;
case '[': s = "90";
break;
case \\': s = "91";
break;
case ']': s = "92";
break;
```

```
                case '^': s = "93";
                break;
                case '_': s = "94";
                break;
                case '~': s = "95";
                break;
                case '`': s = "96";
                break;
                }
                return s;
        }

        public Question getQuestion(String Question_Num, String Question_Type, String prompt){
                Question q;
                //Based off the question type, get the answer and answer options.
                //regardless I need the solution and tags
                ResultSet r = null;
                String soln = "";
                String sql = "Select Solution from Question where Question_Num = '" + Question_Num +
"'";
                Statement stm;
                try {
                        stm = conn.createStatement();
                        r = stm.executeQuery(sql);
                        r.next();
                        soln = r.getString(1);
                } catch (SQLException e) {
                        System.out.println("Problem with SQL Statement: " + sql);
                        System.out.println(e.getMessage());
                }

                sql = "Select Tag from Tags where Question_Num = '" + Question_Num + "'";
                ArrayList<String> tags_ = new ArrayList<String>();
                String[] tags = null;
                try {
                        stm = conn.createStatement();
                        r = stm.executeQuery(sql);
                        while(r.next()){
                                tags_.add(r.getString(1));
                        }
                        tags = new String[tags_.size()];
                        for(int i = 0; i < tags_.size(); i++){
                                tags[i] = tags_.get(i);
                        }
                } catch (SQLException e) {
                        System.out.println("Problem with SQL Statement: " + sql);
                        System.out.println(e.getMessage());
                }
```

```
		if(Question_Type.equals("TF")){
			//TF question
			q = new TF(Question_Type, prompt, soln, tags);
		}
		else if(Question_Type.equals("MA")){
			//MA question
			String Description = prompt;
			sql = "Select Keyword from Matching_Keywords Where Question_Num = '" +
Question_Num + "'";
			ArrayList<String> keywords_ = new ArrayList<String>();
			try {
				stm = conn.createStatement();
				r = stm.executeQuery(sql);
				while(r.next()){
					keywords_.add(r.getString(1));
				}
			} catch (SQLException e) {
				System.out.println("Problem with SQL Statement: " + sql);
				System.out.println(e.getMessage());
			}
			String[] keywords = new String[keywords_.size()];
			for(int i = 0; i < keywords.length; i++){
				keywords[i] = keywords_.get(i);
			}
			q = new Matching(Question_Type, Description, soln, keywords, tags);
		}
		else if(Question_Type.equals("FB")){
			//FB question
			q = new FITB(Question_Type, prompt, soln, tags);
		}
		else{
			//MC question
			sql = "Select Answer_Letter, Answer_Option from MC_Ans_Options Where
Question_Num = '" + Question_Num + "'";
			ArrayList<String[]> Answer_Options = new ArrayList<String[]>();
			try {
				stm = conn.createStatement();
				r = stm.executeQuery(sql);
				while(r.next()){
					String[] anAnswer = new String[2];
					anAnswer[0] = r.getString(1);
					anAnswer[1] = r.getString(2);
					Answer_Options.add(anAnswer);
				}
			} catch (SQLException e) {
				System.out.println("Problem with SQL Statement: " + sql);
				System.out.println(e.getMessage());
```

```
                }
                q = new MC(Question_Type, prompt, soln, Answer_Options, tags);
            }
            return q;
        }



        public String getStudentID(String username){
            String s="";
            ResultSet r=null;
            Statement stm=null;
            String sql="Select Student_ID\n From Student\n Where User_Name='"+username+"'";

            try{
                stm = conn.createStatement();
                r = stm.executeQuery(sql);

                while( r.next() ) {
                    s=(r.getString(1));
                }
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
            return s;
        }

        public ArrayList<String> getMatchingKeywords(String q_num){
            ArrayList<String> s= new ArrayList<String>();
            ResultSet r=null;
            Statement stm=null;
            String sql="Select Keyword\n From Matching_Keywords\n Where
Question_Num="+q_num;

            try{
                stm = conn.createStatement();
                r = stm.executeQuery(sql);

                while( r.next() ) {
                    s.add(r.getString(1));
                }
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
            return s;
        }

        public ArrayList<String> getMultipleChoiceOptions(String q_num){
```

```
                ArrayList<String> s= new ArrayList<String>();
                ResultSet r=null;
                Statement stm=null;
                String sql="Select Answer_Letter, Answer_Option\n From MC_Ans_Options\n Where
Question_Num="+q_num+"\n    Order by Answer_Letter";
                try{
                        stm = conn.createStatement();
                        r = stm.executeQuery(sql);

                        while( r.next() ) {
                                s.add(r.getString(1));
                                s.add(r.getString(2));
                        }
                } catch (SQLException e1) {
                        e1.printStackTrace();
                }
                return s;
        }
  }
```

TestUtility.java

```
package deliverable;

import java.math.BigInteger;
import java.util.Scanner;

public class TestUtility {

   public static void main(String[] args) {

                Utility u = new Utility();
                String username = "hippo367";
                String password = "hippo_367.DB!";
                String dbName = "hippo367_2014";
                //connect to the database
                //u.openDB(username, password, dbName);
                //TEST FOR ENCODE/DECODE:
                //password = "abc123..";
                //String decoded = u.decodePassword(u.encodePassword(password));
                //System.out.println("password decoded = " + password);
                //END TESTFOR ENCODE/DECODE
                //TEST FOR verifyPassword
                //username = "kenb";
                //password = "Torturethemwithquizzes!";
                //username = "dbuser001";
```

```
		//password = "abc123..";
		BigInteger encodedPassword = u.encodePassword(password);
		System.out.println("encoded password = " + encodedPassword);
		//System.out.println(username + " verifyPassword = " + u.verifyPassword(username,
encodedPassword, "Admin"));
		//END TEST
		//TEST FOR WEBUNIQUE:
		String[] Ans_Options = new String[3];
		Ans_Options[0] = "alpha";
		Ans_Options[1] = "beta";
		Ans_Options[2] = "gamma";
		int numAnswers = 3;
		System.out.println("uniqueWeb = " + u.uniqueWeb(Ans_Options, numAnswers));
		//END TEST FOR WEBUNIQUE
		Scanner menu = new Scanner(System.in);
		System.out.println("Select a number option for connecting to the database: ");
		System.out.println("1)	Default");
		System.out.println("2)	Enter login information");
		int connect = menu.nextInt();
		if(connect == 1){
			u.openDB(username, password, dbName);
		}
		else u.openDB();

		int n = 0;
		//Scanner key = new Scanner(System.in);
		while(n != -1){
			System.out.println("Select an option (-1 to quit): ");
			System.out.println("1) Create A Multiple Choice Question");
			System.out.println("2) Update Contact Information");
			System.out.println("3) Check Quiz Grade");
			System.out.println("4) Delete Course");
			System.out.println("5) Find Available Questions");
			System.out.println("6) Backup Database");
			System.out.println("7) Restore Database");
			if(menu.hasNext())
				n = menu.nextInt();
			if(n == 1){
				u.createMCQuestion();
			}
			else if(n == 2){
				u.updateContactInformation();
			}
			else if(n == 3){
				u.checkQuizList("144");
				u.checkQuizGrade();
			}
			else if(n == 4){
```

```
                        u.deleteCourse();
                    }
                    else if(n == 5){
                        u.findAvailableQuestions();
                    }
                    else if(n == 6){
                        u.backupDatabase();
                    }
                    else if(n == 7){
                        u.restoreDatabase();
                    }
                }
                //key.close();
                u.closeDB();
                menu.close();

    }


}
```

Question.java

```
package deliverable;

public class Question {

    public String[] tags() {
        return tags;
    }

    public String soln() {
        return soln;
    }

    public String Question_Type() {
        return Question_Type;
    }

    public String Prompt() {
        return prompt;
    }

    private String[] tags;
    private String soln;
    private String Question_Type;
```

```
    protected String prompt;

    public Question(String Question_Type_, String prompt_, String soln_, String[] tags_){
            Question_Type = Question_Type_;
            prompt = prompt_;
            tags = tags_;
            soln = soln_;
    }

}
```

MC.java

```
package deliverable;

import java.util.ArrayList;

public class MC extends Question {

    private ArrayList<String[]> Answer_Options;

    public MC(String Question_Type_, String prompt_, String soln_,
                    String[] tags_) {
            super(Question_Type_, prompt_, soln_, tags_);
            // TODO Auto-generated constructor stub
    }

    public MC(String Question_Type_, String prompt_, String soln_, ArrayList<String[]> Ans_Options_,
String[] tags_){
            super(Question_Type_, prompt_, soln_, tags_);
            Answer_Options = Ans_Options_;
    }

    public ArrayList<String[]> Ans_Options(){
            return Answer_Options;
    }

}
```

Matching.java

```
package deliverable;

import java.util.HashMap;

public class Matching extends Question {
```

```
    private String[] Keywords;


    public Matching(String Question_Type_, String prompt_, String soln_, String[] tags_) {
            super(Question_Type_, prompt_, soln_, tags_);
    }

    public Matching(String Question_Type_, String prompt_, String soln_, String[] Keywords_, String[] tags_)
{
            super(Question_Type_, prompt_, soln_, tags_);
            Keywords = Keywords_;
    }

    public String[] Keywords(){
            return Keywords;
    }
    //NOTE THAT THE DESCRIPTION IS THE PROMPT

    public String Description() {
            return prompt;
    }

}
```

FITB.java

```
package deliverable;

public class FITB extends Question {

    public FITB(String Question_Type_, String prompt_, String soln_, String[] tags_) {
            super(Question_Type_, prompt_, soln_, tags_);
            // TODO Auto-generated constructor stub
    }

}
```

TF.java

```
package deliverable;

public class TF extends Question {
    public TF(String Question_Type_, String prompt_, String soln_, String[] tags_) {
            super(Question_Type_, prompt_, soln_, tags_);
```

```
        // TODO Auto-generated constructor stub
    }


}
```

actuallyDeleteCourse.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login Page</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<h1>Login</h1>
<p> </p>
<%u.openDB("hippo367", "hippo_367.DB!", "hippo367_2014");%><br><%
if(u.badUsername){ %>
<div style="color:#FF0000">
<b>Username not found.</b></div>
<%}
if(u.badPassword){%>
<div style="color:#FF0000">
<b>Password does not match username.</b></div>
<%}
if(u.badConnection){%>
<div style="color:#FF0000">
<b>Problem connecting to the database.</b></div>
<%}%>
<p> </p>
<form action="open.jsp" method="post">
<table>
<tr> <td> Username: </td> <td> <input type="text" name="user" value="" size="10"> </td> </tr>
<tr> <td> Password: </td> <td> <input type="password" name="pw" value="" size="10"> </td> </tr>
</table>
<input type="submit" value="Login">
</form>
<p> </p>
<b>Forgot username?</b> <br>
```

```
<b>Forgot password?</b>
</body>
</html>
```

backupDB.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Backup Database</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<h1>Backup the Database</h1>
<%if(u.conn == null){%>
   u.conn
<jsp:forward page="index.jsp" />
<%}
if(!u.user_Type(u.username).equals("Admin")){
   %><jsp:forward page="menu.jsp" />
   <%}
u.backupDatabase();
%>
<table>
<tr> <td> <a href="menu.jsp">Home</a> </td><td> </td> <td> <a href="logout.jsp">Logout</a> </td>
</tr>
</table>
Backup successful!
</body>
</html>
```

checkQuizGradeReport.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page import="java.sql.*" %>
<%@ page import="java.util.ArrayList" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Check Quiz Grade</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<table>
<tr> <td> <a href="menu.jsp">Home</a> </td><td> </td> <td> <a href="logout.jsp">Logout</a> </td>
</tr>
</table>
<%
String reportUSERNAME=u.username;
if(request.getParameter("username")!=null&&!(u.username.equals(request.getParameter("username")))){
    %><a
href="checkQuizGradeReport.jsp?course_num=<%out.print(request.getParameter("course_num"));%>&quiz
_num=<%out.print(request.getParameter("quiz_num"));%>&username=<%out.print(u.username);%>">Back
to Class List</a><%
}%>

<%if(u.conn == null){%>
<jsp:forward page="index.jsp" />
<%}%>



<%

ArrayList<String> a= new ArrayList<String>();
if(request.getParameter("username")!=null){
    a=u.checkQuizGrade(request.getParameter("username"), request.getParameter("quiz_num"),
request.getParameter("course_num"));
    reportUSERNAME=request.getParameter("username");



}


else{
    a=u.checkQuizGrade(u.username, request.getParameter("quiz_num"),
request.getParameter("course_num"));
}%>
<h1>
<%if(u.user_Type(reportUSERNAME).equals("Student")){
```

```
   %>Student<%
}else{
   %>Instructor<%
}%>
Quiz Grade</h1>
<%ArrayList<ArrayList<String>> s= new ArrayList<ArrayList<String>>();%>

<table table style width=400px;>
<tr>
<%if(u.user_Type(reportUSERNAME).equals("Student")){
   for(int i=0;i<a.size();i++){
         if(i<=5){%>
         <td style="border: 1px solid black;"><b><%out.print(a.get(i));%></b>
   <%}
         else{%>
         <td style="border: 1px solid black;"> <%out.print(a.get(i));%>
   <%}
         if(((i+1)%6)==0){%>
               <tr><%
         }
   }
   s=u.showQuizReport(reportUSERNAME,
request.getParameter("quiz_num"),request.getParameter("course_num"));
   %>
   </tr></table>
   <h2>Quiz Questions:</h2>
   <p><font style="background-color: #00FF00">Green = Corrent Student Answer.</font>
   <font style="background-color: #FF0000">Red = Incorrent Student Answer.</font>
   <font style="background-color: #FFFF00">Yellow = Corrent Solution to a missed question.</font></p>


   <%
   for(int i=0;i<s.size();i++){
         if(s.get(i).get(1).equals("MC")){
               out.println((i+1)+". "+s.get(i).get(2));%>
               <table><tr><td>
               <%
               for(int j=5;j<s.get(i).size();j+=2){
                     if(s.get(i).get(3).equals(s.get(i).get(4))&&s.get(i).get(3).equals(s.get(i).get(j))){
                           %><font style="background-color: #00FF00"><%
                           out.println(s.get(i).get(j)+")");%>
                           <%out.println(s.get(i).get(j+1));%>
                           <tr><td></font><%
                     }
                     else
if(!(s.get(i).get(3).equals(s.get(i).get(4)))&&s.get(i).get(3).equals(s.get(i).get(j))){
                                 %><font style="background-color: #FF0000"><%
                                       out.println(s.get(i).get(j)+")");
```

```
                                        out.println(s.get(i).get(j+1));%>
                                        <tr><td></font><%
                        }
                        else
if(!(s.get(i).get(3).equals(s.get(i).get(4)))&&s.get(i).get(4).equals(s.get(i).get(j))){
                                %><font style="background-color: #FFFF00"><%
                                        out.println(s.get(i).get(j+")");%>
                                        <%out.println(s.get(i).get(j+1));%>
                                        <tr><td></font><%
                        }else{
                                out.println(s.get(i).get(j+")");%>
                                <%out.println(s.get(i).get(j+1));%>
                                <tr><td><%
                        }
                }%>
                </td></tr></table>
                <br><%
        }
        else if(s.get(i).get(1).equals("TF")){
                out.println((i+1)+". "+s.get(i).get(2));
                %><br><%
                if(s.get(i).get(3).equals(s.get(i).get(4))){
                        %><font style="background-color: #00FF00"><%
                        out.println("Answer: "+s.get(i).get(4));
                        %></font><br><%
                }
                else{
                        %><font style="background-color: #FF0000"><%
                        out.println("Answer: "+s.get(i).get(4));
                        %></font><br><font style="background-color: #FFFF00"><%
                        out.println("Solution: "+s.get(i).get(3));
                        %></font><%
                        }
                %><br><%
        }
        else if(s.get(i).get(1).equals("FB")){
                out.println((i+1)+". "+s.get(i).get(2));
                %><br><%
                if(s.get(i).get(3).equals(s.get(i).get(4))){
                        %><font style="background-color: #00FF00"><%
                        out.println("Answer: "+s.get(i).get(4));
                        %></font><br><%
                }
                else{
                        %><font style="background-color: #FF0000"><%
                        out.println("Answer: "+s.get(i).get(4));
                        %></font><br><font style="background-color: #FFFF00"><%
                        out.println("Solution: "+s.get(i).get(3));
```

```
                        %></font><%
                }
                %><br><%
        }
        else{//donothing for MA as of yet
        }
}

ArrayList<String> k= new ArrayList<String>();
ArrayList<String> d= new ArrayList<String>();
for(int i=0;i<s.size();i++){
        if(s.get(i).get(1).equals("MA")){
                d.add(s.get(i).get(2));
                for(int j=5;j<s.get(i).size();j++){
                        k.add(s.get(i).get(j));
                }
        }
}
if(d.size()!=0&&k.size()!=0){

        %><TABLE WIDTH="100%">
        <tr><td width="100px"><b>Keywords</b></td><td><b>Descriptions</b></td></tr>
        <tr><td><table width="100px" border="0" cellspacing="0" cellpadding="1">
        <%for(int i=0;i<k.size();i++){
                %><tr><td width="100px"><%
                out.print(k.get(i));
                }
        %></table></td><td><table width="80%" border="0" cellspacing="0" cellpadding="1"><%
        for(int i=0;i<d.size();i++){
                %><tr><td height=30px><%
                out.println(i+1+". "+d.get(i));
                %></td></tr><%
                }
        %></table></td></table><%
        int j=1;
        for(int i=0;i<s.size();i++){
                if(s.get(i).get(1).equals("MA")){


                        if(s.get(i).get(4).equals(s.get(i).get(3))){
                                %> <font style="background-color: #00FF00"> <%
                                        out.println(j+". "+s.get(i).get(4));
                                        %></font><br><%
                        }
                        else{
                                %><font style="background-color: #FF0000"><%
                                        out.println(j+". "+s.get(i).get(4));
```

```
                                                %></font><br><font style="background-color:
#FFFF00"><%
                                                out.println(j+". "+s.get(i).get(3));
                                                %></font><%
                                }
                                j++;

                                }
                        }
                }

}else{
        for(int i=0;i<a.size();i++){
                if(i<=3||(i<=11&&i>=8)){%>
                        <td style="border: 1px solid black;"><b><%out.print(a.get(i));%></b>
                <%}
                else if(i>11&&(i-12)%5==0){%>
                        <%
                }
                else if(i>11&&(i-12)%5==1){%>
                <td style="border: 1px solid black;"><a
href="checkQuizGradeReport.jsp?course_num=<%out.print(request.getParameter("course_num"));%>&quiz
_num=<%out.print(request.getParameter("quiz_num"));%>&username=<%out.print(a.get(i-1));%>">
<%out.print(a.get(i));%></a><%
                }
                else{%>
                <td style="border: 1px solid black;"> <%out.print(a.get(i));
                }
                if(i<13){
                if(((i+1)%4)==0){
                        if((i==7)){%>
                                </table><br><table table style width=400px;>
                                <%}%>
                                <tr>
                        <%}
                }
                else{
                        if((i-12)%5==0){%>
                                <tr>
                        <%
                        }
                }

 }
 }%>
</table>
</body>
</html>
```

checkQuizGradeSelect.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page import="java.sql.*" %>
<%@ page import="java.util.ArrayList" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Check Quiz Grade</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<table>
<tr> <td> <a href="menu.jsp">Home</a> </td><td> </td> <td> <a href="logout.jsp">Logout</a> </td>
</tr>
</table>
<h1>Check Quiz Grade</h1>
<%if(u.conn == null){%>
   u.conn
<jsp:forward page="index.jsp" />
<%}
ArrayList<String> b = new ArrayList<String>();
if(request.getParameter("course_num")!=null){
   b=u.checkQuizList(request.getParameter("course_num"));
}
%>
<h5>Required fields marked by an<b><font color="red">*</font>.</b></h5>

<table>
<tr> <td>Select Course Number: <font color="red">*</font>
<tr> <td>
<%if(request.getParameter("course_num")==null||b.size()==0) {%>
   <form action="checkQuizGradeSelect.jsp" method="post">
<%}else{ %>
   <form action="checkQuizGradeReport.jsp" method="post">
<%} %>
<select name="course_num">
<% ArrayList<String> a=u.checkQuizCourseList(u.username);
for(int i=0;i<a.size();i++){%>
```

```
   <%if(request.getParameter("course_num") != null &&
request.getParameter("course_num").equals(a.get(i))){ %>
   <option selected="selected" value=<%=a.get(i)%>><%=a.get(i)%></option>
   <%}else if(request.getParameter("course_num") == null){
          %>
          <option value=<%=a.get(i)%>><%=a.get(i)%></option>
          <%
   }else{
          ArrayList<String> c=u.checkQuizList(a.get(i));
          if((c.size()!=0)){
                 %>
                 <option value=<%=a.get(i)%>><%=a.get(i)%></option>
                 <%
          }

   } %>
   <%} if(request.getParameter("course_num")==null||b.size()==0){ %>
          <input type="submit" value="Select">
          <%} %>
</select>
<tr>
<%if(request.getParameter("course_num")!=null){ %>
<tr><td>Select Quiz Number: <font color="red">*</font>
<tr><td>
<select name="quiz_num">
<%
for(int i=0;i<b.size();i++){%>
   <option value=<%=b.get(i)%>><%=b.get(i)%></option><%
   }%>
<input type="submit" value="Select">
</select>
</form>
<%} %>
</table>
```

```
</body>
</html>
```

createMCQuestion.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page import="java.sql.*" %>
<%@ page import="java.util.ArrayList" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Create A New Multiple Choice Question</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<table>
<tr> <td> <a href="menu.jsp">Home</a> </td><td> </td> <td> <a href="logout.jsp">Logout</a> </td>
</tr>
</table>
<h1>New Multiple Choice Question</h1>
<h5>Required fields marked by an<b><font color="red">*</font>.</b></h5>
<%if(u.conn == null){%>
   u.conn
<jsp:forward page="index.jsp" />
<%}%>
<%if(u.user_Type(u.username).equals("Student")){
   %><jsp:forward page="menu.jsp" />
   <%}
if(u.noMCQuestionText){%>
   <b><font color="red">Must enter Question text.</font></b><br>
<%}
if(u.longMCQuestionText){%>
   <b><font color="red">Question must be no longer than 255 characters.</font></b><br>
<%}
if(u.noMC_ReqAns_Options){%>
<b><font color="red">Must enter at least options a, b, and c.</font></b><br>
<%}
if(u.badMC_nReqAns_Options){%>
<b><font color="red">Must enter option d before entering option e.</font></b><br>
<%}
```

```
if(u.noMC_Soln){%>
<b><font color="red">Must select a solution for the question.</font></b><br>
<%}
if(u.longMC_Ans_Options){%>
<b><font color="red">Answer option must be no longer than 255 characters.</font></b><br>
<%}
if(u.invalidSoln){%>
<b><font color="red">Must select solution from entered answer options.</font></b><br>
<%}
if(u.notUniqueMC_Ans_Options){%>
<b><font color="red">Answer options must be unique.</font></b><br>
<%}
   u.noMCQuestionText = false;
   u.longMCQuestionText = false;
   u.noMC_ReqAns_Options = false;
   u.badMC_nReqAns_Options = false;
   u.longMC_Ans_Options = false;
   u.noMC_Soln = false;
   u.invalidSoln = false;
   u.notUniqueMC_Ans_Options = false;
   %>
<form action="InsertMCQuestion.jsp" method="post">
<b>Question prompt<font color="red">*</font>:</b><br><textarea name="prompt" rows="5"
cols="60"></textarea> <br>
<p></p>
<b>Enter answer options for the question and select the answer which is the solution<font
color="red">*</font>:</b><br>
            <input type="radio" name="soln" value="A">a)<b><font color="red">*</font></b><input
type="text" name="a" value="" size="30"><br />
            <input type="radio" name="soln" value="B">b)<b><font color="red">*</font></b><input
type="text" name="b" value="" size="30"><br />
            <input type="radio" name="soln" value="C">c)<b><font color="red">*</font></b><input
type="text" name="c" value="" size="30"><br />
            <input type="radio" name="soln" value="D">d) <input type="text" name="d" value=""
size="30"><br />
            <input type="radio" name="soln" value="E">e) <input type="text" name="e" value=""
size="30"><br />
<p> </p>
<b>Courses Related to this question</b><br>
<% try{
   ResultSet r = u.listCourses();
   while(r.next()) {%>
         <input type="checkbox" name="courseNum" value="<%= r.getString(1)
%>"><%=r.getString(1)%><br />
   <% }
} catch(SQLException e){
   System.out.println("Something went wrong.");
}%>
```

```
<p> </p>
<b>Associated Tags</b><br>
<%
   ArrayList<String> existingTags = u.existingTags();
   for(int i = 0; i < existingTags.size(); i++) {%>
          <input type="checkbox" name="tag" value="<%= existingTags.get(i) %>"><%=
existingTags.get(i) %><br />
   <% }%>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

deleteCourse.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
          pageEncoding="ISO-8859-1"%>
          <%@ page import="java.sql.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Delete Course</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<%if(u.conn == null){%>
   u.conn
<jsp:forward page="index.jsp" />
<%}%>
<%if(u.user_Type(u.username).equals("Student")){
   %><jsp:forward page="menu.jsp" />
   <%}%>
<table>
<tr> <td> <a href="menu.jsp">Home</a> </td><td> </td> <td> <a href="logout.jsp">Logout</a> </td>
</tr>
</table>
<p> </p>

<%int numCoursesSelected;
if(request.getParameterValues("courseNum") == null){
   numCoursesSelected = 0;
}
```

```
else numCoursesSelected = request.getParameterValues("courseNum").length;
if(request.getParameter("delAnyway")!= null && request.getParameter("delAnyway").equals("No")){%>
   <jsp:forward page="menu.jsp" />
<%}
if(request.getParameter("deleteCourses") != null){%>
   <jsp:forward page="actuallyDeleteCourse.jsp" />
<%}

String[] theCourses = request.getParameterValues("courseNum");
if(theCourses == null)
   theCourses = new String[0];
try{
   ResultSet r = u.listCourses();%>
   <form action="deleteCourse.jsp" method="post">
<%%>
   <b>Select a course(s) to delete<font color="red">*</font>.</b><br>
   <%for(int i = 0; i < theCourses.length; i++){%>
                  <input type="checkbox" name="courseNum" value="<%= theCourses[i]
%>"checked><%=theCourses[i]%><br />
                  <%
         } //this prints the courses that were selected%>
         <%//Now print the courses that weren't selected
         while(r.next()){
                  boolean checked = false;
                  for(int i = 0; i < theCourses.length; i++){
                           if(r.getString(1).equals(theCourses[i])){
                                    checked = true;
                           }
                  }
                  if(!checked){%>
                           <input type="checkbox" name="courseNum" value="<%= r.getString(1)
%>"><%=r.getString(1)%><br /><%
                  }
         }
}catch(SQLException e){
   System.out.println("Something went wrong.");
}%>
<input type="submit" value="Select Courses"><br /><p> </p>
<%boolean needDelAnyway = false;
for(int i = 0; i < theCourses.length; i++){
         ResultSet s = null;
         String name = "y/n" + i;
         String sql = "Select Student_ID From Enrolled_In Where Course_Num = '" + theCourses[i] + "'";
         Statement stm = u.conn.createStatement();
         s = stm.executeQuery(sql);%><%
         if(s.next()){
                  needDelAnyway = true;
                  %><b>Students are enrolled in <%=theCourses[i]%>.</b><br /><%
```

```
        }
    }%>
    <%boolean gotoActuallyDelete = false; %>
<form<%if(request.getParameterValues("courseNum") != null && (request.getParameter("delAnyway") !=
null || !needDelAnyway)){%> action="actuallyDeleteCorurse.jsp"<%gotoActuallyDelete =
true;}else{%>action="deleteCourse.jsp"<%}%> method="post">    <p> </p>
<%if(request.getParameter("courseNum") != null){
                        if(needDelAnyway && request.getParameter("delAnyway") == null){%>
                        <b>Delete anyway?<font color="red">*</font></b><br />
                        <input type="radio" name="delAnyway" value="Yes">Yes<br />
                        <input type="radio" name="delAnyway" value="No">No<br />
                        <p> </p>
                        <input type="submit" value="Delete Anyway">
                        <%}
                        else{ %>
                        Are you sure you want to delete these courses?<br />
                        <input type="submit" name="deleteCourses" value="Delete selected course(s).">
                        <%}
                }%>
                        </form>
</form>
</body>
</html>
```

findAvailableQuestions.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page import="java.sql.*" %>
<%@ page import="java.util.ArrayList" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Find Available Questions</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<table>
<tr> <td> <a href="menu.jsp">Home</a> </td><td> </td> <td> <a href="logout.jsp">Logout</a> </td>
</tr>
</table>
<h1>Find a Question</h1>
```

```
<h5>Please select a class and any number of tags</h5>
<%if(u.conn == null){%>
   u.conn
<jsp:forward page="index.jsp" />
<%}%>
<%if(u.user_Type(u.username).equals("Student")){
   %><jsp:forward page="menu.jsp" />
   <%}
if (u.noCourseSelectedAvailableQuestions){%>
   <b><font color="red">Must select a course.</font></b>
<%}
u.noCourseSelectedAvailableQuestions = false;
%>
Required fields marked by an<b><font color="red">*</font>.</b>
<form action="printAvailableQuestions.jsp" method="post">

<b>Related Courses<font color="red">*</font></b><br>
<% try{
   ResultSet r = u.listCourses();
   while(r.next()) {%>
         <input type="radio" name="courseNum" value="<%= r.getString(1) %>"><%=r.getString(1)%><br
/>
   <% }
} catch(SQLException e){
   System.out.println("Something went wrong");
}%>
<p> </p>
<b>Tags</b><br>
<%
   ArrayList<String> existingTags = u.existingTags();
   for(int i = 0; i < existingTags.size(); i++) {%>
         <input type="checkbox" name="tag" value="<%= existingTags.get(i) %>"><%=
existingTags.get(i) %><br />
   <% }%>
<input type="submit" value="Submit">


</form>
</body>
</html>
```

index.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
         pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login Page</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<h1>Login</h1>
<p> </p>
<%u.openDB("hippo367", "hippo_367.DB!", "hippo367_2014");%><br><%
if(u.badUsername){ %>
<div style="color:#FF0000">
<b>Username not found.</b></div>
<%}
if(u.badPassword){%>
<div style="color:#FF0000">
<b>Password does not match username.</b></div>
<%}
if(u.badConnection){%>
<div style="color:#FF0000">
<b>Problem connecting to the database.</b></div>
<%}%>
<p> </p>
<form action="open.jsp" method="post">
<table>
<tr> <td> Username: </td> <td> <input type="text" name="user" value="" size="10"> </td> </tr>
<tr> <td> Password: </td> <td> <input type="password" name="pw" value="" size="10"> </td> </tr>
</table>
<input type="submit" value="Login">
</form>
<p> </p>
<b>Forgot username?</b> <br>
<b>Forgot password?</b>
</body>
</html>
```

InsertMCQuestion.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page import="java.sql.*" %>
<%@ page import="java.util.ArrayList" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert MC Question</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<%if(u.conn == null){%>
   u.conn
<jsp:forward page="index.jsp" />
<%}%>
<%if(u.user_Type(u.username).equals("Student")){
   %><jsp:forward page="menu.jsp" />
   <%}%>
<table>
<tr> <td> <a href="menu.jsp">Home</a> </td><td> </td> <td> <a href="logout.jsp">Logout</a> </td>
</tr>
</table>
<%
   if(request.getParameter("a").equals("") || request.getParameter("b").equals("") ||
request.getParameter("c").equals("")){
         u.noMC_ReqAns_Options = true;
   }
   if(request.getParameter("d").equals("") && !request.getParameter("e").equals("")){
         u.badMC_nReqAns_Options = true;
   }
   if(!u.noMC_ReqAns_Options && !u.badMC_nReqAns_Options){
          if((request.getParameter("a").toCharArray().length > 255) ||
(request.getParameter("b").toCharArray().length > 255) || (request.getParameter("c").toCharArray().length >
255))
                   u.longMC_Ans_Options = true;
          if(!request.getParameter("d").equals("")){
                   if(request.getParameter("d").toCharArray().length > 255){
                            u.longMC_Ans_Options = true;
                   }
          }
          if(!request.getParameter("e").equals("")){
                   if(request.getParameter("e").toCharArray().length > 255){
                            u.longMC_Ans_Options = true;
                   }
          }
   }
   if(request.getParameter("prompt").equals("")){
         u.noMCQuestionText = true;
   }
```

```
    else if(request.getParameter("prompt").toCharArray().length > 255){
        u.longMCQuestionText = true;
    }
    if(request.getParameter("soln") == null){
        u.noMC_Soln = true;
    }%>
<h1>Insertion Status:</h1>
<%
String prompt = request.getParameter("prompt");
System.out.println("prompt = " + prompt);
            String[] Ans_Options = new String[5];
            for(int i = 0; i < Ans_Options.length; i++){
                    switch(i){
                    case 0:
                                    Ans_Options[i] =
u.accomodateForProblemChars(request.getParameter("a"));
                                    break;
                    case 1:
                                    Ans_Options[i] =
u.accomodateForProblemChars(request.getParameter("b"));
                                    break;
                    case 2:
                                    Ans_Options[i] =
u.accomodateForProblemChars(request.getParameter("c"));
                                    break;
                    case 3:
                                    Ans_Options[i] =
u.accomodateForProblemChars(request.getParameter("d"));
                                    break;
                    case 4:
                                    Ans_Options[i] =
u.accomodateForProblemChars(request.getParameter("e"));
                                    break;
                    }
            }
                    int numAnswers = 0;
                    for(int i = 0; i < 5; i++){
                            if(!Ans_Options[i].equals(""))
                                    numAnswers++;
                                    String soln = request.getParameter("soln");
                                    if(soln != null){
                                            if(i == 0 && Ans_Options[i].equals("") && soln.equals("A"))
                                                    u.invalidSoln = true;
                                            if(i == 1 && Ans_Options[i].equals("") && soln.equals("B"))
                                                    u.invalidSoln = true;
                                            if(i == 2 && Ans_Options[i].equals("") && soln.equals("C"))
                                                    u.invalidSoln = true;
                                            if(i == 3 && Ans_Options[i].equals("") && soln.equals("D"))
```

```
                                        u.invalidSoln = true;
                                    if(i == 4 && Ans_Options[i].equals("") && soln.equals("E"))
                                        u.invalidSoln = true;
                                }
                            if(Ans_Options[i].toCharArray().length > 255)
                                u.longMC_Ans_Options = true;
                        }
                        if(Ans_Options[3].equals("") && !Ans_Options[4].equals(""))
                            u.badMC_nReqAns_Options = true;
                        if(!u.uniqueWeb(Ans_Options, numAnswers))
                            u.notUniqueMC_Ans_Options = true;
                        if(u.noMC_ReqAns_Options || u.badMC_nReqAns_Options || u.longMC_Ans_Options ||
u.noMCQuestionText || u.longMCQuestionText || u.noMC_Soln || u.invalidSoln ||
u.notUniqueMC_Ans_Options){%>
                            <jsp:forward page="createMCQuestion.jsp" />
                    <%}
                     ArrayList<String> coursesRelated = new ArrayList<String>();
                     ArrayList<String> selectedTags = new ArrayList<String>();
                     String soln[] = request.getParameterValues("soln");
                     String[] courseParam = request.getParameterValues("courseNum");
                     String[] tagsParam = request.getParameterValues("tag");
                     if(courseParam == null)
                            courseParam = new String[0];
                     if(tagsParam == null)
                            tagsParam = new String[0];
                     for(int i = 0; i < courseParam.length; i++){
                            coursesRelated.add(courseParam[i]);
                     }
                     for(int i = 0; i < tagsParam.length; i++){
                            selectedTags.add(u.accomodateForProblemChars(tagsParam[i]));
                     }

                    prompt = u.accomodateForProblemChars(prompt);
                    soln[0] = u.accomodateForProblemChars(soln[0]);
                     u.createMCQuestion(prompt, Ans_Options, soln[0], numAnswers, coursesRelated,
selectedTags);
                            %>
                            You have successfully created a new question!
</body>
</html>
```

logout.jsp

```
<jsp:useBean id="u" class="deliverable.Utility" scope="session"></jsp:useBean>
<%@ page import="java.sql.*" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Logout</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<% u.closeDB();
if(u.conn == null){
u.username = "";%>
Logout successful!<br>
<a href="index.jsp">Login</a>
<%}
u.web = false; %>
</body>
</html>
```

menu.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Home page</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<h1><a href="menu.jsp">Home</a></h1>
<ul>
<%if(u.user_Type(u.username).equals("Instructor")||u.user_Type(u.username).equals("Admin")){
   %>
<li><a href="createMCQuestion.jsp">Create a new Multiple Choice Question</a> </li>
<li><a href="deleteCourse.jsp">Delete a Course</a></li>
<li><a href="findAvailableQuestions.jsp">Find Questions for a Quiz</a></li>
   <%}%>
<li><a href="updateContactInfo.jsp">Update Contact Information</a></li>
<li><a href="checkQuizGradeSelect.jsp">Check Quiz Grade</a></li>
```

```
<%if(u.user_Type(u.username).equals("Admin")){
   %>
<li><a href="backupDB.jsp">Backup DB</a></li>
<li><a href="restoreDB.jsp">Restore DB</a></li>
   <%}%>
<li><a href="logout.jsp">Logout</a></li>
</ul>
<%u.web = true; %>
</body>
</html>
```

open.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Logged in?</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<h1>Logged in?</h1>
<% u.badUsername = false;
u.badPassword = false;
u.badConnection = false;
   u.openDB("hippo367", "hippo_367.DB!", "hippo367_2014");
   String user = request.getParameter("user");
        String password = request.getParameter("pw");
        password = u.accomodateForProblemChars(password);%>
   conn =<%= u.conn%>

   <%if(u.conn == null){
         u.badConnection = true;%>
         <jsp:forward page="index.jsp" />
   <%}
        //Here I determine what type of user I have
   if(u.user_Type(user).equals("Admin")){
        if(u.verifyPassword(user, password, "Admin")){
              u.username = user;%>
              <jsp:forward page="menu.jsp" />
        <%}
```

```
            else{
                        u.badPassword = true;%>
                        <jsp:forward page="index.jsp" />
            <%}
    }else if(u.user_Type(user).equals("Instructor")){
            if(u.verifyPassword(user, password, "Instructor")){
                        u.username = user;%>
                         <jsp:forward page="menu.jsp" />
            <%}
            else{
                        u.badPassword = true;%>
                        <jsp:forward page="index.jsp" />
            <%}
    }else if(u.user_Type(user).equals("Student")){
            if(u.verifyPassword(user, password, "Student")){
                        u.username = user;%>
                         <jsp:forward page="menu.jsp" />
            <%}
            else{
                        u.badPassword = true;%>
                        <jsp:forward page="index.jsp" />
            <%}
    }else{
             u.badUsername = true;
                                %>
              <jsp:forward page="index.jsp" />
 <%} %>
</body>
</html>
```

performUpdateContactInfo.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page import="java.sql.*" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
          pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Update Contact Info</title>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<table>
```

```
<tr> <td> <a href="menu.jsp">Home</a> </td><td> </td> <td> <a href="logout.jsp">Logout</a> </td>
</tr>
</table>
<h1>Update Contact Info</h1>
<%if(u.conn == null){
    %>
    <jsp:forward page="index.jsp" />
<%} %>
<%  String user = u.username;
if(u.user_Type(u.username).equals("Admin"))
    user = request.getParameter("username");


if(user.equals("") && u.user_Type(u.username).equals("Admin")){
    u.noUsernameUpdateContactInfo = true;%>
<%
}
String email = request.getParameter("email");
String fname = request.getParameter("fName");
String minit = request.getParameter("mInit");
String lname = request.getParameter("lName");

if(!email.equals("") && email.length() > 255)
    u.longEmailUpdateContactInfo = true;
if(!email.equals("")){
    if(email.length()<=8 || !email.substring(email.length()-8).equalsIgnoreCase("@plu.edu"))
            u.invalidEmailUpdateContactInfo = true;

}
if(!fname.equals("") && fname.length() > 255)
    u.longFnameUpdateContactInfo = true;
if(!lname.equals("") && lname.length() > 255)
    u.longLnameUpdateContactInfo = true;
if(!minit.equals("") && minit.length() > 1)
    u.longMInitUpdateContactInfo = true;

if(u.noUsernameUpdateContactInfo || u.longEmailUpdateContactInfo || u.longFnameUpdateContactInfo ||
u.longLnameUpdateContactInfo || u.invalidEmailUpdateContactInfo || u.longMInitUpdateContactInfo){
    %>    <jsp:forward page="updateContactInfo.jsp" /><%
}

    ResultSet r=null;
String sql = "Select Email, F_Name, M_Init, L_Name From ";
if(u.user_Type(user).equals("Student")){
            sql+="Student ";
}
else{
    sql+= "Instructor ";
```

```
}
   sql += "Where User_Name='"+user+"'";
   Statement stm;
   try {
         stm = u.conn.createStatement();
         r = stm.executeQuery(sql);
         r.next();
         String oldEmail = r.getString( 1 );
         String oldFname = r.getString( 2 );
         String oldMinit = r.getString( 3 );
         String oldLname = r.getString( 4 );
         %><h5><b><%
         if(email.equals("") && fname.equals("") && minit.equals("") && lname.equals("")){%>
               No fields selected for update. No operation occured.<%
         }else{
               if(email == null || email.equals("")){
                     email = oldEmail;
               }if(fname == null || fname.equals("")){
                     fname = oldFname;
               }if(minit == null || minit.equals("")){
                     minit = oldMinit;
               }if(lname == null || lname.equals("")){
                     lname = oldLname;
               }
         u.updateContactInformation(user, email, fname, minit, lname);%>
         Update Successful!<%} %></b></h5><%
   }catch (SQLException e) {
         e.printStackTrace();
   }%>
</body>
</html>
```

printAvailableQuestions.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*" %>
<%@ page import="deliverable.Question" %>
<%@ page import="deliverable.FITB" %>
<%@ page import="deliverable.Matching" %>
<%@ page import="deliverable.MC" %>
<%@ page import="deliverable.TF" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
         pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Print Available Questions</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<%
ArrayList<String> selectedTags = new ArrayList<String>();


if(u.conn == null){%>
    u.conn
<jsp:forward page="index.jsp" />
<%}%>
<%if(u.user_Type(u.username).equals("Student")){
    %><jsp:forward page="menu.jsp" />
    <%

}
if(request.getParameter("courseNum") == null){
    u.noCourseSelectedAvailableQuestions = true;
    %><jsp:forward page="findAvailableQuestions.jsp" />
    <%
}%>

<table>
<tr> <td> <a href="menu.jsp">Home</a> </td>
<td> </td>
<td> <a href="logout.jsp">Logout</a> </td> </tr>
</table>

<br>

<%
String course = request.getParameter("courseNum");
String[] tags = request.getParameterValues("tag");



if (tags == null){
    tags = new String[0];
}else{
    for(int i = 0; i < tags.length; i++){
            tags[i] = u.accomodateForProblemChars(tags[i]);
    }
}
```

```
    if(course == null){
        u.noCourseSelectedAvailableQuestions = true;
        course = "";
        %><jsp:forward page="findAvailableQuestions.jsp" /><%
    }

    ResultSet r = u.findAvailableQuestions(course, tags);
        %>
            <h2>Questions Available For A Quiz</h2>
  <table>
      <tr>
          <b><td>Course: </td></b>
          <td><%=request.getParameter("courseNum")%></td>
      </tr>
      <%if(tags.length != 0){ %>
      <tr>
          <b><td>Tag(s): </td></b>
          <% for(int i = 0; i < tags.length; i++){
          %>
          <td><%=request.getParameterValues("tag")[i]%><%if(i<tags.length - 1){ %>, <%}
%></td>
          <%} %>
      </tr>
      <%} %>
  </table>
  <br>
  <br>
<%
ResultSet s = u.findAvailableQuestions(course, tags);
ArrayList<String> temp = new ArrayList<String>();
String r1 = "Test"; //r.getString(1);
String empty = "There are no questions that fit the selected criteria.";
ArrayList<Question> questions = new ArrayList<Question>();
if(!s.next()){ //if there are no questions matching the criteria.
  %><%=empty.toString()%><%
  }else {
  while(r.next()){//get the questions
      String Question_Type = r.getString(2);
      String prompt = r.getString(3);
      String Question_Num = r.getString(1);
      questions.add(u.getQuestion(Question_Num, Question_Type, prompt));
  }
  //Now print out the questions
  for(int i = 0; i < questions.size(); i++){
      %><b><%=i+1%>) <%
      if(questions.get(i) instanceof TF){
          TF q = (TF) questions.get(i);
```

```
%><%=q.Prompt()%><br></b>
<i>Answer: <%=q.soln()%></i><br><p></p><%
}
else if(questions.get(i) instanceof MC){
MC q = (MC) questions.get(i);
%><%=q.Prompt()%><br></b>
<%ArrayList<String[]> Answer_Options = q.Ans_Options();
ArrayList<String> AnswersOrdered = new ArrayList<String>();
for(int j = 0; j < Answer_Options.size(); j++){//loop for the number of answers
//j tells me which answer I want to find (eg j = 0 corresponds to finding a)
for(int k = 0; k < Answer_Options.size(); k++){
char c = Answer_Options.get(k)[0].toLowerCase().charAt(0);//c is the
answer letter. I want to know if it is the jth answer.
//eg if j = 0, I want to know if c is a.
char compareTo = '!';
switch(j){
case 0: compareTo = 'a';
break;
case 1: compareTo = 'b';
break;
case 2: compareTo = 'c';
break;
case 3: compareTo = 'd';
break;
case 4: compareTo = 'e';
break;
}
if(c == compareTo){
//then print the answer option and break from the inner loop.
switch(c){
case 'a': %>a) <%
break;
case 'b': %>b) <%
break;
case 'c': %>c) <%
break;
case 'd': %>d) <%
break;
case 'e': %>e) <%
break;
}
%><%=Answer_Options.get(j)[1]%><br><%
}
}
}
%><i>Solution: <%=q.soln()%></i><br><p></p><%
}
else if(questions.get(i) instanceof FITB){
```

```
                FITB q = (FITB) questions.get(i);
                %><%=q.Prompt()%><br></b>
                <i>Answer: <%=q.soln()%></i><br><p></p><%
        }
        else if(questions.get(i) instanceof Matching){
                Matching q = (Matching) questions.get(i);
                %>Matching<table style="width:300px">
                <tr><td><%=q.Description()%></td></b><table><%
                String[] keywords = q.Keywords();
                for(int j = 0; j < keywords.length; j++){
                        %><tr><td><%=keywords[j]%></td></tr><%
                }%></table></tr><%
                %></table><%
                %><i>Solution: <%=q.soln()%></i><br><p></p><%
        }
   }
}%>
<br>
<br>
<br>
Add questions to a quiz.


</body>
</html>
```

restoreDB.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Restore Database</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<h1>Restore the Database from the Backup</h1>
<%if(u.conn == null){%>
   u.conn
<jsp:forward page="index.jsp" />
<%}
```

```
if(!u.user_Type(u.username).equals("Admin")){
   %><jsp:forward page="menu.jsp" />
   <%}
u.restoreDatabase();
%>
<table>
<tr> <td> <a href="menu.jsp">Home</a> </td><td> </td> <td> <a href="logout.jsp">Logout</a> </td>
</tr>
</table>
Restoration successful!
</body>
</html>
```

updateContactInfo.jsp

```
<jsp:useBean id="u"  class="deliverable.Utility" scope="session" ></jsp:useBean>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Update Contact Info</title>
</head>
<body background="Background2.png">
<img border="0" src="PLUlogo.jpg" width="50" height="50" STYLE="position:absolute; TOP:25px;
RIGHT:25px;" >
<%if(u.conn == null){%>
<jsp:forward page="index.jsp" />
<%}%>
<table>
<tr> <td> <a href="menu.jsp">Home</a> </td><td> </td> <td> <a href="logout.jsp">Logout</a> </td>
</tr>
</table>
<h1>Update Contact Info</h1>
<h5>Enter new contact information</h5>
<%if(u.noUsernameUpdateContactInfo){ %>
   <b><font color="red">Must enter a username to update contact information.</font></b><br>
<%} %>
<%if(u.longEmailUpdateContactInfo){ %>
   <b><font color="red">Email cannot exceed 255 characters in length.</font></b><br>
<%} %>
<%if(u.longMInitUpdateContactInfo){ %>
   <b><font color="red">Middle Initial must be only 1 character in length.</font></b><br>
<%} %>
```

```
<%if(u.longFnameUpdateContactInfo){ %>
   <b><font color="red">First Name cannot exceed 255 characters in length.</font></b><br>
<%} %>
<%if(u.longLnameUpdateContactInfo){ %>
   <b><font color="red">Last Name cannot exceed 255 characters in length.</font></b><br>
<%} %>
<%if(u.invalidEmailUpdateContactInfo){ %>
   <b><font color="red">Must enter a valid PLU email address.</font></b><br>
<%} %>

<form action="performUpdateContactInfo.jsp" method="post">
<%if(u.user_Type(u.username).equals("Admin")){%>
<h5>Required fields marked by an<b><font color="red">*</font>.</b></h5>
<%} %>
<table>
<%u.noUsernameUpdateContactInfo = false;
  u.longMInitUpdateContactInfo = false;
  u.longFnameUpdateContactInfo = false;
  u.longLnameUpdateContactInfo = false;
  u.longEmailUpdateContactInfo = false;
  u.invalidEmailUpdateContactInfo = false;
if(u.user_Type(u.username).equals("Admin")){%>
<tr> <td> <b>Username<font color="red">*</font></b> </td> <td> <input type="text" name="username"
value="" size="30"> </td> </tr>
<%}
  else{
  String username = u.username;%>
  <%} %>
<tr> <td> <b>Email</b> </td> <td> <input type="text" name="email" value="" size="30"> </td> </tr>
<tr> <td> <b>First name</b> </td> <td> <input type="text" name="fName" value="" size="30"> </td>
</tr>
<tr> <td> <b>Middle Initial</b> </td> <td> <input type="text" name="mInit" value="" size="30"> </td>
</tr>
<tr> <td> <b>Last name</b> </td> <td> <input type="text" name="lName" value="" size="30"> </td> </tr>
</table>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Files

TableDescriptions

Instructor, User_Name, Password, email, F_Name, M_Init, L_Name
Student, User_Name, Password, email, F_Name, M_Init, L_Name, Student_ID
Course, Course_Num, Course_Name
Enrolled_In, Student_ID, Course_Num
Quiz, Quiz_Num, Course_Num, Start_Time, End_Time, Time_Limit
Question_Type, Question_Type, Pt_Value
Question, Question_Num, Question_Type, Solution, Prompt
Course_Question, Course_Num, Question_Num
Tags, Question_Num, Tag
FITB, Question_Num
MC, Question_Num
MC_Ans_Options, Question_Num, Answer_Option, Answer_Letter
Matching, Question_Num
Matching_Keywords, Question_Num, Keywords
True_False, Question_Num
Question_Asked, Question_Num, Quiz_Num, Course_Num
Student_Answer, Student_ID, Quiz_Num, Course_Num, Question_Num

Background2.png

PLUlogo.jpg