Shakhawath Shah

COMP2311 Database Systems

Coursework 2

01/12/2021

Contents

Part A: Normalisation

For this section I will implement 1NF, 2NF and 3NF, to normalise the data given from the information snippet. I have included attributes which I have assumed to be relevant for each entity, in addition to the ones given by the source. Below I have started with a UNF, to which I will use as a basis to normalise the data.

UNF:

I have created the UNF table using the information provided by the source. I have included the attributes given for both Students and Quizzes, including the sample data for the first record in the table. At this stage there are repeated groups for Quiz that would be repeated for every student who takes a quiz, and there is also data which is not yet atomic, in the options attribute. I have identified these ahead of the 1NF stage.

Student Table

Stud_ID	Stud_name	Q_ID	Q_name	Q_Author	Q_Available	Q_Duration	Date_attempted	Question	Options

44	Duncan Hull	34	SQL	Peter Parker	Yes	60 Mins	22/11/2020	1.Which SQL statement is used to extract data from a database?
								2.Which SQL statement is used to insert data from a database?
								3.With SQL, how do you select all the records from a table named “Persons” where the values of the column “FirstName” is “Peter”
									INSERT NEW,
									INSERT INTO,
									ADD RECORD,
									ADD NEW,
									
									SELECT * FROM Persons WHERE FIRSTNAME <> ‘Peter’,
									SELECT [all] FROM Person WHERE FirstName = ‘Peter’,
									SELECT * FROM Persons WHERE FirstName = ‘Peter’
									SELECT [all] FROM Person WHERE
									FirstName Like ‘Peter’

1NF:

As mentioned above, there are repeated groups for Quiz, therefore following the rules of 1NF, I will move these attributes into a new relation called Quiz. As shown below I have taken all attributes out leaving, ID and name remaining within the Student relation with Stud_ID as the primary key. Quiz also includes Stud_ID to create a link between the two, and forms the compound key with Q_ID, by following the rules of 1NF. I have also split student name into first name and surname, making the values atomic.

After this step the Quiz table was still not in 1NF, so I repeated the process, removing repeated groups, questions, options and answer into a new relation, Questions. In this I took Q_ID and Question to form a new compound primary key within the relation. I also again split Author name into atomic values. Finally I repeated this step once more to make Questions into 1NF, also including Ques_ID in the questions table. This removed repeating groups options into a new relation with Ques_ID and option forming the compound primary key. Therefore for my final 1NF I had 4 relations, Student, Quiz, Questions, Options.

Student(**Stud_ID**, fname, sname)

Quiz(**Stud_ID, Q_ID**, Q_Name, A_fname, A_sname, Q_Available, Q_Duration, Date_Attempted)

Questions(**Q_ID,Ques_ID**, Question, Answer)

Options(**Ques_ID, Option**)

2NF:

For 2NF I checked the dependency of the attributes on the keys. For students, the name depends fully on the key so it can remain as is. However for Quiz the name, author, duration and options only depend on Q_ID and not Stud_ID as well therefore I made a separate relation called QuizDetails, containing all of these Quiz attributes.

The two attributes remaining in Quiz, are Q_Available and Date_Attempted, as they depend on both IDs. Q_Available is dependant on which quiz, as well as which student, as the availability may rely on a student already completing the quiz so may no longer be available for them. Date_Attempted is also based on when the student takes the quiz so fully dependant on the compound key.

Student(**Stud_ID**, fname, sname)

Quiz(**Stud_ID, Q_ID**, Q_Available, Date_Attempted)

QuizDetails(**Q_ID**, Q_Name, A_fname, A_sname, Q_Duration)

Questions(**Q_ID**, **Question_num**, Question, Answer)

Options(**Question_num, Option**)

3NF:

For this I have kept it the same as my 2NF. This is as I have checked that there is no transitive dependencies

Student(**Stud_ID**, fname, sname)

Quiz(**Stud_ID, Q_ID**, Q_Available, Date_Attempted)

QuizDetails(**Q_ID**, Q_Name, A_fname, A_sname, Q_Duration)

Questions(**Q_ID**, **Question_num**, Question, Answer)

Options(**Question_num, Option**)

Part B: Relational Schema

Below is the schema I have produced using the normalised data from part A. There are also few assumptions I have made resulting in new attributes and tables, that were not part of my normalised data. These include a new table for Staff, as well as storing username and password data for both user types.

I have also included the foreign key constraints for the tables, citing which table the foreign keys are referenced from, as well as the update and delete constraints. The update and delete constraints will ensure that when a parent key is updated or deleted, the children are updated accordingly in any tables where the parent key is referenced.

Student(<u>Stud_ID</u>, fname, sname, password)

StudentInfo(<u>Stud_ID, Q_ID</u>, Q_Available, Date_Attempted, score)

- FK Stud_ID → Student(Stud_ID) ON UPDATE CASCADE ON DELETE CASCADE

QuizDetails(<u>Q_ID,</u> Staff_ID, Q_Name, Q_Duration, totalScore)

- FK Staff_ID → Staff(Staff_ID) ON UPDATE CASCADE ON DELETE CASCADE

Questions(<u>Q_ID, Question_num</u>, Question, Answer)

- FK → Q_ID QuizDetails(Q_ID) ON UPDATE CASCADE ON DELETE CASCADE

Options(<u>Question_num, Option</u>, option_chosen, correct)

- FK Question_num → Questions(Question_num) ON UPDATE CASCADE ON DELETE CASCADE

Staff(<u>Staff_ID</u>, fname, sname, password)

Part C: Implementation of design

Below I have included the physical implementation of my schema showing the SQL queries used to create the tables for my application and the database setup. The queries show the data types I have used for the attributes of each entity as well as the primary and foreign keys for each. I have also include the referential actions for the foreign keys to take, for update and delete actions made by the user.

Create Table Queries:

```
CREATE TABLE IF NOT EXISTS student(

  stud_ID varChar (100) NOT NULL,

  fname varChar (100) NOT NULL,

  sname varChar (100) NOT NULL,

  pass varChar (100) NOT NULL,

  PRIMARY KEY(stud_ID)

);


CREATE TABLE IF NOT EXISTS staff(

  staff_ID varChar (100) NOT NULL,

  fname varChar (100) NOT NULL,

  sname varChar (100) NOT NULL,

  pass varChar (100) NOT NULL,

  PRIMARY KEY(staff_ID)

);


CREATE TABLE IF NOT EXISTS quizDetails(

  quiz_ID int UNSIGNED NOT NULL AUTO_INCREMENT,

  staff_ID varChar (100) NOT NULL,

  quiz_name varChar (100),

  duration smallint,

  total_score smallint,

  PRIMARY KEY(quiz_ID, staff_ID),

  FOREIGN KEY(staff_ID) REFERENCES staff(staff_ID) ON UPDATE CASCADE ON DELETE CASCADE

);
```

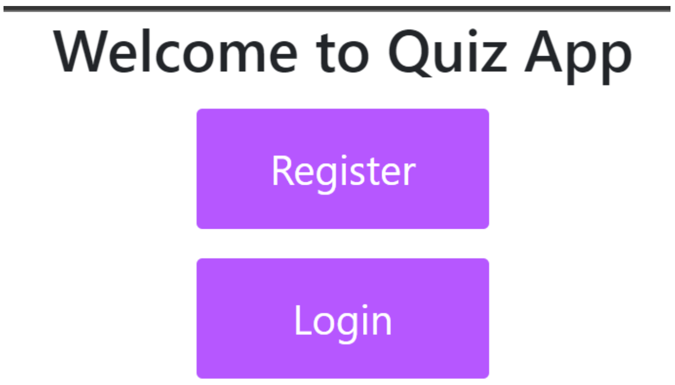
```
CREATE TABLE IF NOT EXISTS studentInfo(
    stud_ID varChar (100) NOT NULL,
    quiz_ID int UNSIGNED NOT NULL,
    available varChar (10),
    date_attempted datetime,
    score smallint,
    PRIMARY KEY(stud_ID, quiz_ID),
    FOREIGN KEY (stud_ID) REFERENCES student(stud_ID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (quiz_ID) REFERENCES quizDetails(quiz_ID) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS questions(
    question_num int NOT NULL AUTO_INCREMENT,
    quiz_ID int UNSIGNED NOT NULL,
    question varChar (100),
    PRIMARY KEY(question_num, quiz_ID),
    FOREIGN KEY(quiz_ID) REFERENCES quizDetails(quiz_ID) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS options(
    question_option varChar(100) NOT NULL,
    quiz_ID int UNSIGNED NOT NULL,
    question_num int NOT NULL,
    option_chosen varChar(100),
    correct boolean,
    PRIMARY KEY(question_option, quiz_ID, question_num),
    FOREIGN KEY(quiz_ID) REFERENCES quizDetails(quiz_ID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(question_num) REFERENCES questions(question_num) ON UPDATE CASCADE ON DELETE
CASCADE
);
```

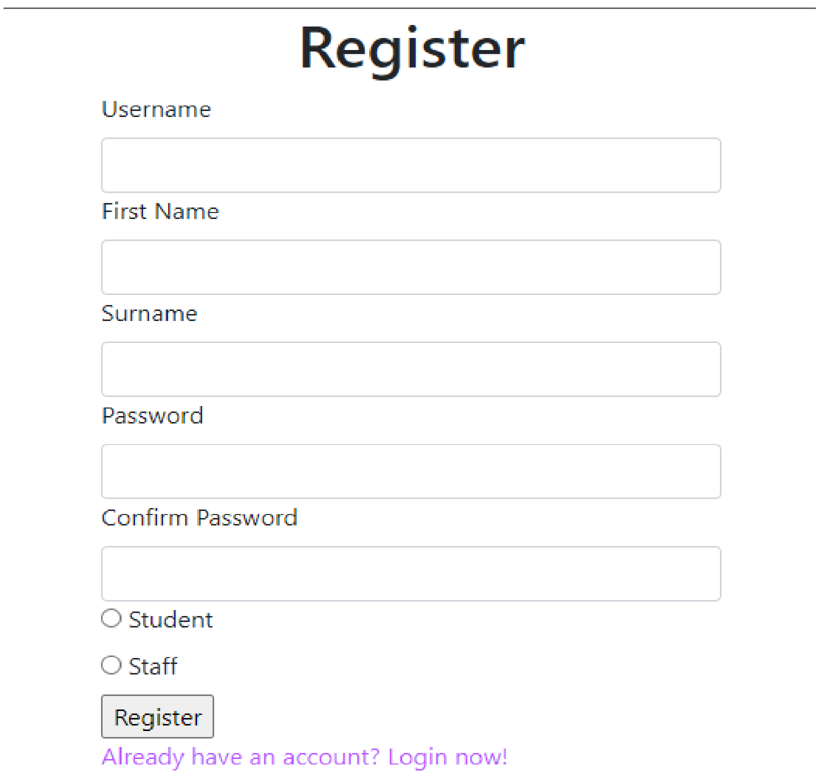
Part D: Application

For this section I will briefly go through the application explaining each page, and how the page works both for the user and behind the scenes. I have successfully implemented by application allowing students and staff members to use the functions they have access to.

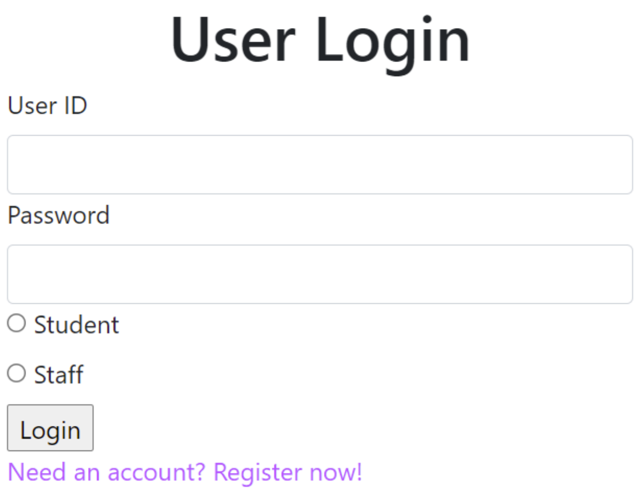
Registration

On start-up the home page gives the user the option to either login or register a new account. If they choose to register they will be taken to the register page, which requires them to choose a username, provide their name and a password for the account. They then choose the type of user they are either a student or a staff member. Once registered it will redirect the user to the login page to successfully login with their new account. There is also an option to login is the user already has an account which will redirect them to login rather than registering again.





Login

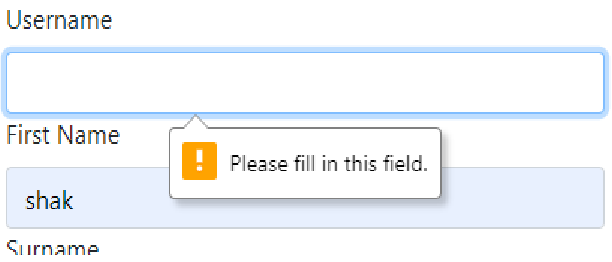


When the user chooses to login, they will be asked to enter their username and password only. This will then check if the username exists in the database and check whether the password matches to that username. Again the user is asked which user type they are so the necessary checks are made within the database.

Validation

Validation within the application is important as it ensures the users correctly use the application and enter data correctly when asked. I have implemented a few validation types, to check user input with the database, as well as checking inputs are entered as intended, I will show and explain them below.

The most basic validation in the application is ensuring certain input fields are filled in before the user can submit to either login or register.



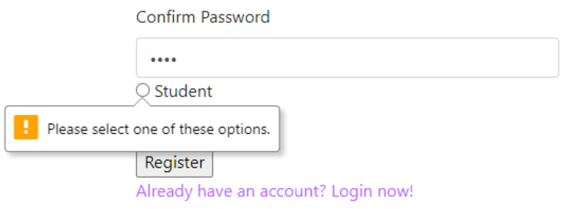
Username

First Name

shak

Please fill in this field.

Surname



Confirm Password

••••

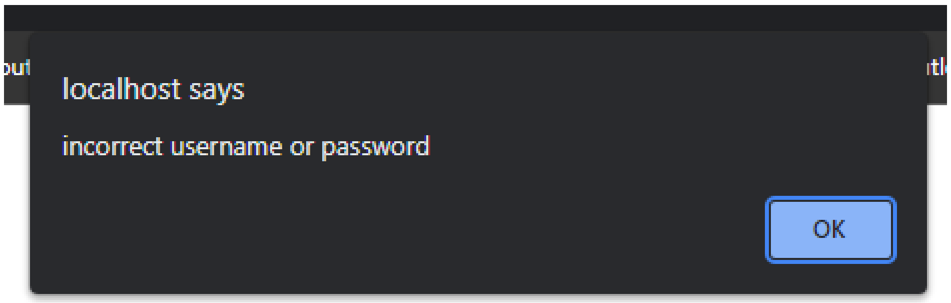
Student

Please select one of these options.

Register

Already have an account? Login now!

There is also validation which checks if the user correctly entered the password for their login. This alert notifies the user if their details are incorrect.

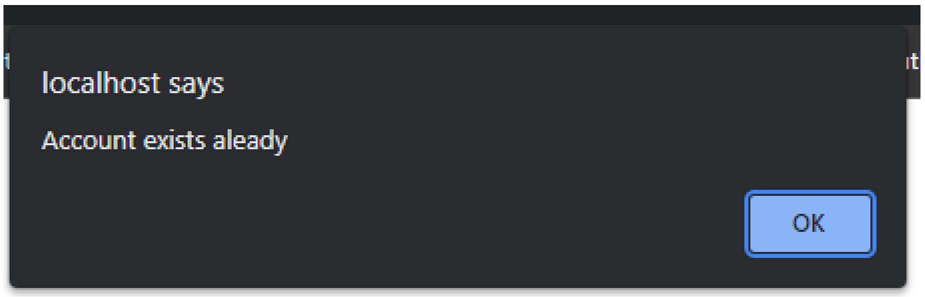


localhost says

incorrect username or password

OK

I have also created the same alert for the user if they try to register a new account with the same username. This lets the user know this account already exists and they should either enter a new username or login with the account they already have.



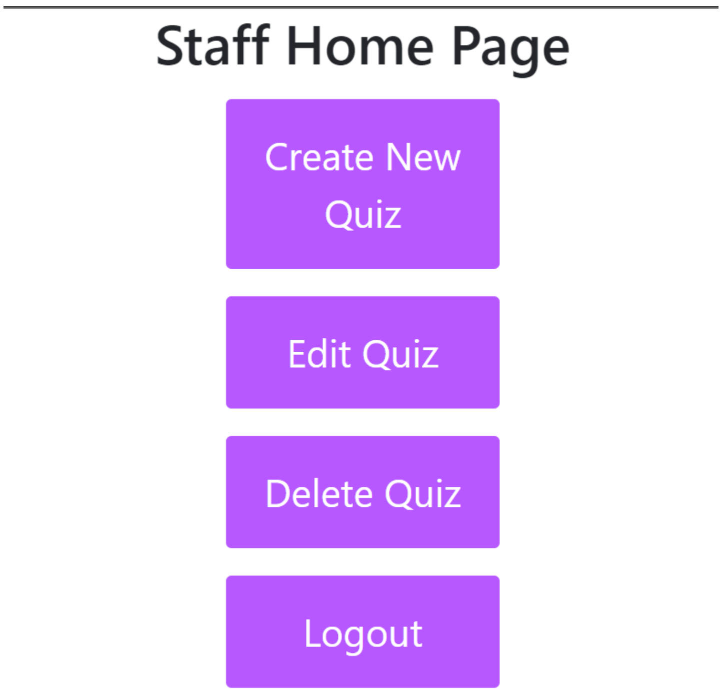
localhost says

Account exists aleady

OK

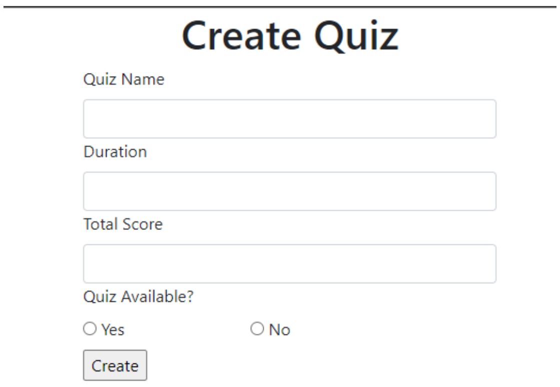
Staff

Once a staff member has logged in they will be shown the home page which gives them a range of choices as shown below. The user therefore chooses what they want to do.



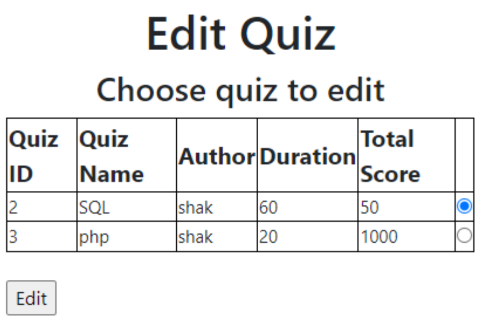
Create Quiz

For a staff member to create a new quiz they must first give the quiz its name, duration and score, as well as making it available for students to take. Once they press the button to create the quiz is successfully created within the database. Two attributes quiz ID and staff ID (the author) are automatically stored with the data, but the user does not see this.



Edit Quiz

If the user wishes to edit a quiz they will be given the list of quizzes they have created, and its associated details. The user can then select which to edit from the table given, by pressing the radio button next to each quiz.



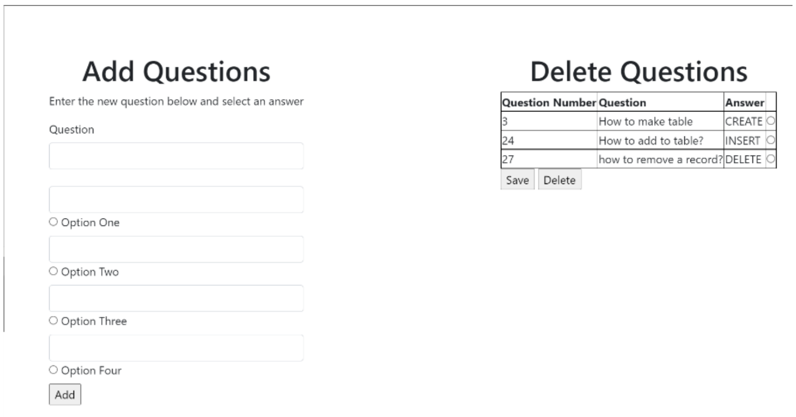
Edit Quiz

Choose quiz to edit

Quiz ID	Quiz Name	Author	Duration	Total Score	
2	SQL	shak	60	50	●
3	php	shak	20	1000	○

Once they have chosen a specific quiz to edit they will be given to options to add new questions, or delete existing questions from the quiz. As shown below there is a form ready for the user to enter the new question, its options and select the correct answer from the radio buttons given.

After they submit and add the question the table on the right hand side of the screen will update including the new question, and its answer. The user can then decide to select a question from the table to delete which will remove the question from the database for this quiz. The user can also save which will save the quiz in its current state and return back to the home page.



Add Questions

Enter the new question below and select an answer

Question

○ Option One

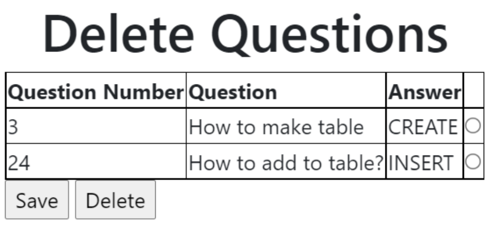
○ Option Two

○ Option Three

○ Option Four

Delete Questions

Question Number	Question	Answer	
3	How to make table	CREATE	○
24	How to add to table?	INSERT	○
27	how to remove a record?	DELETE	○



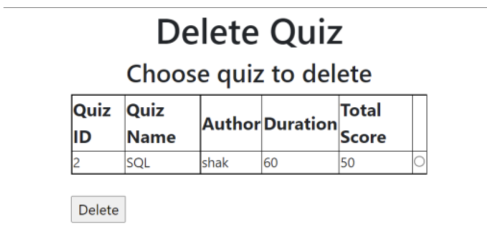
Delete Questions

Question Number	Question	Answer	
3	How to make table	CREATE	○
24	How to add to table?	INSERT	○

Figure 1 After Deletion of question

Delete Quiz

Delete quiz page works similar to delete questions, the user selects the quiz and presses the button to delete



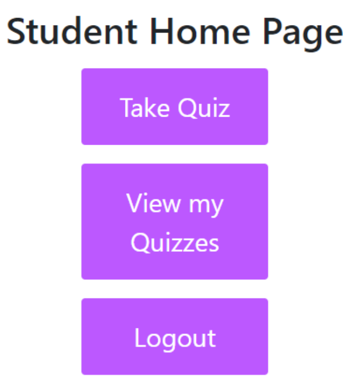
Delete Quiz

Choose quiz to delete

Quiz ID	Quiz Name	Author	Duration	Total Score	
2	SQL	shak	60	50	○

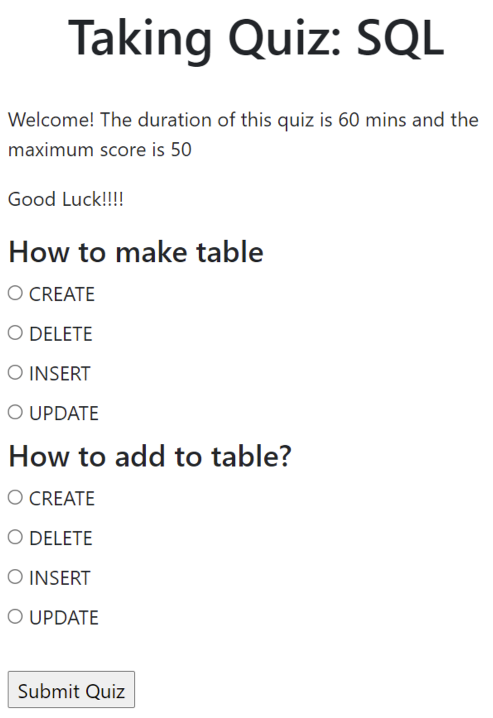
Student

The student home page is very similar to the staff page, however students are instead able to take a quiz set by a staff member, and also view all past quizzes they have taken, and the scores they have received.



Take quiz

If the student decides to take a new quiz they are taken to this page showing all the questions they can answer. They are given the quiz details that they need to know and can answer and submit the quiz. Once they submit the user is shown their score and how many questions they got correct as well as the date they attempted the quiz. The user can then return home to choose another quiz if they wish. This data is also stored in the database tracking the users score and other quiz details associated with the attempt



Taking Quiz: SQL

Welcome! The duration of this quiz is 60 mins and the maximum score is 50

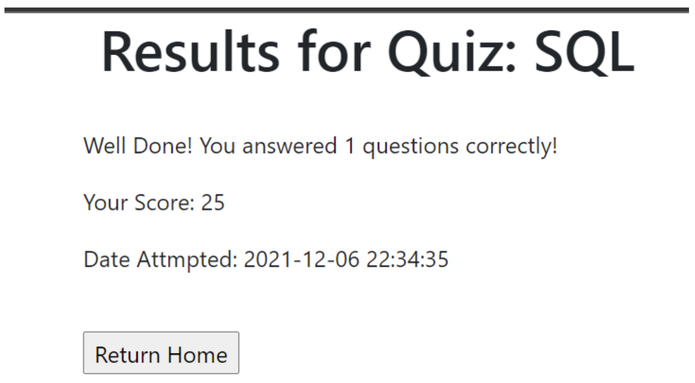
Good Luck!!!!

How to make table

○ CREATE
○ DELETE
○ INSERT
○ UPDATE

How to add to table?

○ CREATE
○ DELETE
○ INSERT
○ UPDATE



Results for Quiz: SQL

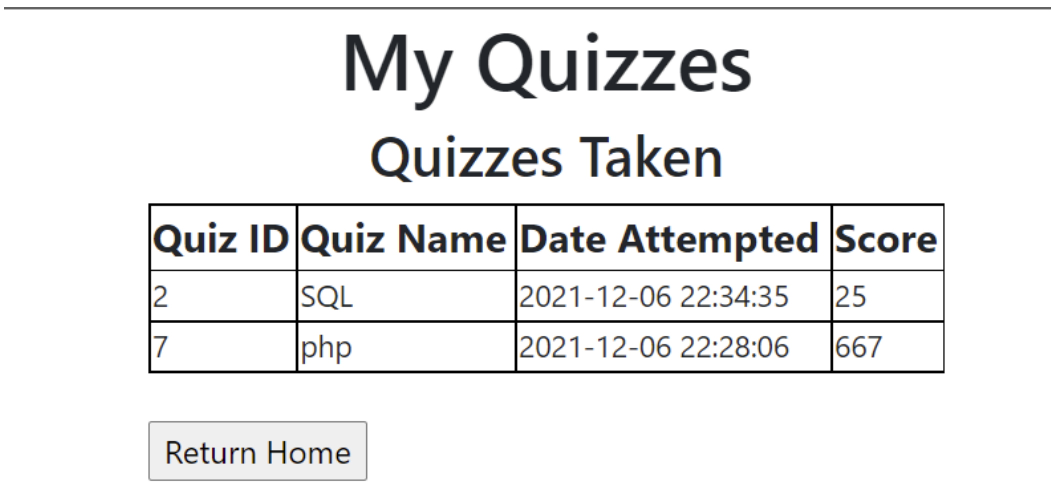
Well Done! You answered 1 questions correctly!

Your Score: 25

Date Attmpted: 2021-12-06 22:34:35

View quizzes

The view quiz page shows the user all quizzes they have previously taken and the score they have achieved. This data is obtained from the database and displayed to the user in a table.



My Quizzes

Quizzes Taken

Quiz ID	Quiz Name	Date Attempted	Score
2	SQL	2021-12-06 22:34:35	25
7	php	2021-12-06 22:28:06	667

Session/Logout

When either student or staff member wants to logout, they will be redirect back to the home page. If they wish to login again they will have to enter their details again. This is because once logged out the cookie session is destroyed losing all temporary data they had whilst logged in.

Part E:

Stored Procedure

```
DELIMITER ^

  CREATE PROCEDURE scoreLessThan40 ()

  BEGIN

  SELECT `quiz_name`, `studentInfo`.`score`, `student`.`fname`, `student`.`sname`

  FROM `quizDetails` INNER JOIN

  `studentInfo` ON `studentInfo`.`quiz_ID` = `quizDetails`.`quiz_ID`

  INNER JOIN

  `student` ON `studentInfo`.`stud_ID` = `student`.`stud_ID` WHERE `studentInfo`.`score` < '40';

  END ^

DELIMITER ;
```

Trigger

```
CREATE TABLE IF NOT EXISTS deleteQuiz(

    delete_ID int NOT NULL AUTO_INCREMENT,

    quiz_ID int UNSIGNED NOT NULL,

    staff_ID varChar (100) NOT NULL,

    time datetime NOT NULL,

    PRIMARY KEY(delete_ID),

    FOREIGN KEY (quiz_ID) REFERENCES quizDetails(quiz_ID)

);


DELIMITER ^

    CREATE TRIGGER delete_quiz

    AFTER DELETE ON quizDetails FOR EACH ROW

    SET time = NOW();

    BEGIN INSERT INTO delete_quiz (staff_ID, quiz_ID, time)

    VALUES (OLD.quiz_ID , OLD.quiz_ID, time)

    END ^

DELIMITER;
```