# **Assignment 1: Design and Create App/Game Database**

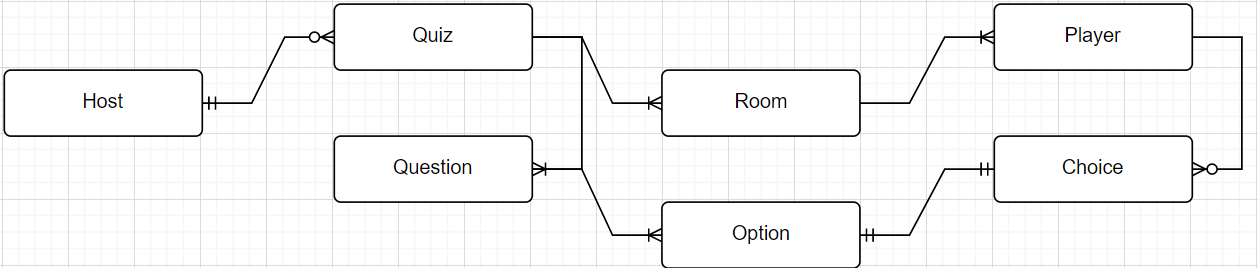
## **1)** **Name of App/Game**

*Kahoot – an online quiz platform that is easily accessible by browser. It allows teachers to create quizzes and host free sessions for multiple students in real time.*

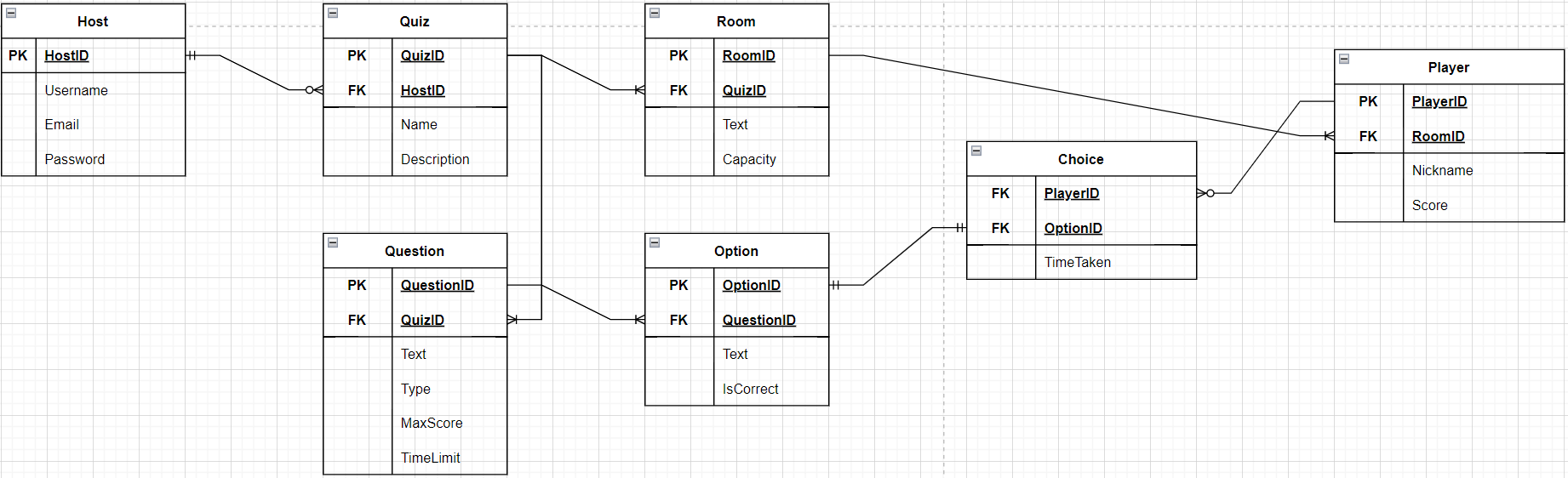
## **2)** **Description of App Functions / Game Play**

*The quiz is hosted by a teacher and students can enter the* room *via a unique* ***room identifier*** *code. They are prompted to enter a* ***nickname*** *to allow the teacher to identify them. The students are then presented with a question, with multiple options**where choosing the* ***correct option*** *will earn them points. Speed is a factor as well, the shorter the* ***time****, the higher the* ***points*** *gained. This is calculated with the formula* ( 1 - (( {response time} / {question timer} ) / 2 )) {points possible}*). At the end of the quiz, there is a leader board, where the podium is shown as a form of encouragement to the students. The teacher then gets a report of how each student is performing. The report will show the scores of each student alongside their* ***choice*** *history. This allows the teacher to apply reinforced learning to students based on their results. The report is a summary of the data recorded during the session, and thus is just a storage medium of the session.*

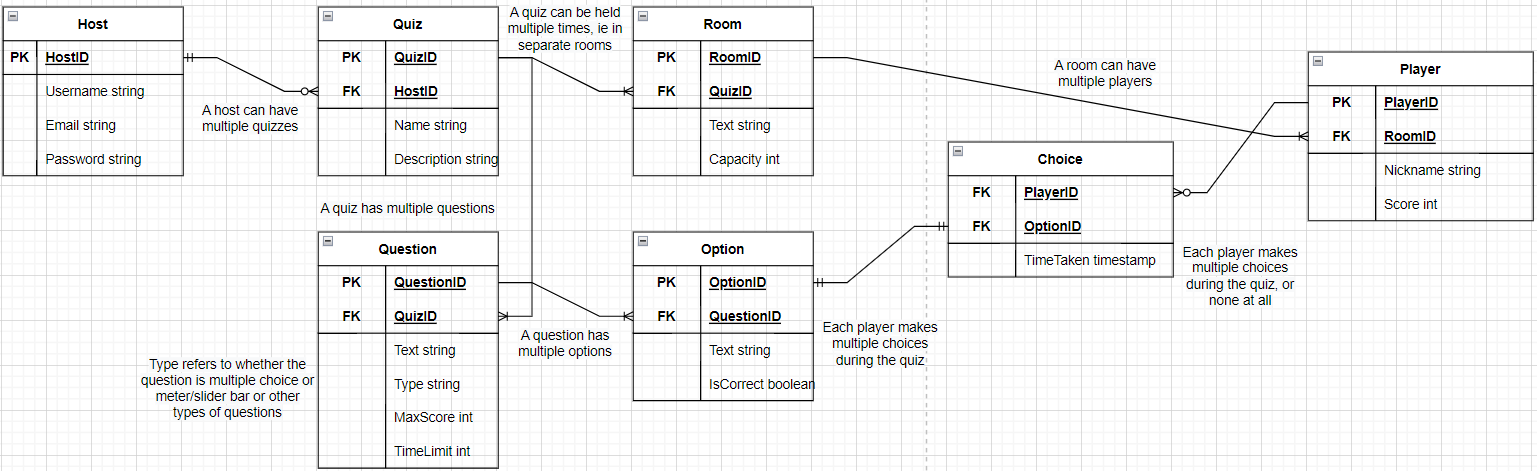
### Conceptual model

****

### Logical model

****

### Physical model

****

## **3)** **App/Game Data Elements**

| Host | **Host ID** | Username | Email | Password |  |
| --- | --- | --- | --- | --- | --- |
| Player | **Player ID** | **Room ID** | Nickname | Score |  |
| Quiz | **Quiz ID** | Name | Description |  |  |
| Question | **Question ID** | Question Name | Description | MaxScore | TimeLimit |
| Option | **Option ID** | **Question ID** | Text | isCorrect |  |
| Choice | **Option ID** | **Player ID** | TimeTaken |  |  |
| Room | **Room ID** | **Quiz ID** | Text | Capacity |  |

## **4)** **Entity-Relation Diagram (Data Model(s))**

### Data Dictionary / Sample Data / DDL / DML / Results Screenshots

This section documents the following: a) Data Dictionary, b) Sample Data, c) Data Definition Language, d) Screenshot of DDL results, e) Data Manipulation Language to add the sample data, f) Screenshot of Insert Results

#### Tb\_hosts

| **Table** | **Description** | | | | |
| --- | --- | --- | --- | --- | --- |
| tb\_hosts | Table storing details of host accounts | | | | |
| **Key Type** | **Mandatory** | **Column Name** | **Data Type** | **Size** | **Remarks (e.g. FK Reference, default value)** |
| PK | Auto-Increment | HostID | int | Default |  |
|  | True | Username | Varchar | 100 |  |
|  | True | Password | Varchar | 32 |  |
|  | True | Email | Varchar | 255 |  |

| **Username** | **Password** | **Email** |
| --- | --- | --- |
| “SchoolTeacher” | “Password1” | “SchoolTeacher@gmail.com” |
| “Lecturer” | “Password2” | “Lecturer@gmail.com” |
| “TuitionTeacher” | “Password3” | “TuitionTeacher@gmail.com” |

| Table Name: **tb\_hosts** |
| --- |
| DDL:  CREATE TABLE tb\_hosts (  HostID int AUTO\_INCREMENT,  Username varchar(100) NOT NULL,  Password varchar(32) NOT NULL,  Email varchar(255) NOT NULL,  PRIMARY KEY(HostID)  ); |
| Output of Structure: |
| DML to populate data:  INSERT INTO *tb\_hosts* (*Username, Password, Email*)  VALUES *(“SchoolTeacher”, “Password1”, “SchoolTeacher@gmail.com”),*  *(“Lecturer”, “Password2”, “Lecturer@gmail.com”),*  *(“TuitionTeacher”, “Password3”, “TuitionTeacher@gmail.com”)*; |
| Output of Sample Data: |

#### Tb\_quizes

| **Table** | **Description** | | | | |
| --- | --- | --- | --- | --- | --- |
| tb\_quizes | Table storing details of created quizes | | | | |
| **Key Type** | **Mandatory** | **Column Name** | **Data Type** | **Size** | **Remarks (e.g. FK Reference, default value)** |
| PK | Auto-Increment | QuizID | int | Default |  |
| FK | True | HostID | int | Default | FK reference ID from tb\_hosts |
|  | True | Name | Varchar | 32 |  |
|  | True | Description | Varchar | 255 |  |

| **HostID** | **Name** | **Description** |
| --- | --- | --- |
| 1 | SchoolQuiz | Quiz created by School Teacher |
| 2 | LectureQuiz | Quiz created by Lecturer |
| 3 | TuitionQuiz | Quiz created by Tuition Teacher |

| Table Name: **tb\_quizes** |
| --- |
| DDL:  CREATE TABLE tb\_quizes (  QuizID int AUTO\_INCREMENT,  HostID int NOT NULL,  Name varchar(32) NOT NULL,  Description varchar(255) NOT NULL,  PRIMARY KEY(QuizID),  FOREIGN KEY(HostID) REFERENCES tb\_hosts(HostID)  ); |
| Output of Structure: |
| DML to populate data:  INSERT INTO *tb\_quizes* (*HostID, Name, Description*)  VALUES *(1, “SchoolQuiz”, “Quiz created by School Teacher”),*  *(2, “LectureQuiz”, “Quiz created by Lecturer”),*  *(3, “TuitionQuiz”, “Quiz created by Tuition Teacher”)*; |
| Output of Sample Data: |

#### Tb\_questions

| **Table** | **Description** | | | | |
| --- | --- | --- | --- | --- | --- |
| tb\_questions | Table storing details of questions | | | | |
| **Key Type** | **Mandatory** | **Column Name** | **Data Type** | **Size** | **Remarks (e.g. FK Reference, default value)** |
| PK | Auto-Increment | QuestionID | Int | Default |  |
| FK | True | QuizID | Int | Default | FK reference from tb\_quiz |
|  | True | Text | Varchar | 100 |  |
|  | True | Type | Varchar | 32 |  |
|  | True | Maxscore | Int | Default |  |
|  | True | TimeLimit | Int | Default | Assumes in milliseconds |

| **QuizID** | **Text** | **Type** | **MaxScore** | **TimeLimit** |
| --- | --- | --- | --- | --- |
| 1 | How many continents are there in the world? | MCQ | 2000 | 30000 |
| 1 | Is Taiwan a country? | True/False | 1000 | 15000 |
| 1 | How many oceans are there in the world? | MCQ | 2000 | 30000 |

| Table Name: **tb\_questions** |
| --- |
| DDL:  CREATE TABLE tb\_questions (  QuestionID int AUTO\_INCREMENT,  QuizID int NOT NULL,  Text varchar(100) NOT NULL,  Type varchar(32) NOT NULL,  MaxScore int NOT NULL,  TimeLimit int NOT NULL,  PRIMARY KEY(QuestionID),  FOREIGN KEY(QuizID) REFERENCES tb\_quizes(QuizID)  ); |
| Output of Structure: |
| DML to populate data:  INSERT INTO *tb\_questions* (*QuizID, Text, Type, MaxScore, TimeLimit*)  VALUES *(1, “How many continents are there in the world?”, “MCQ”, 2000, 30000),*  *(1, “Is Taiwan a country?”, “True/False”, 1000, 15000),*  *(1, “How many oceans are there in the world?”, “MCQ”, 2000, 30000)*; |
| Output of Sample Data: |

#### Tb\_options

| **Table** | **Description** | | | | |
| --- | --- | --- | --- | --- | --- |
| tb\_options | Table storing details of options in a question | | | | |
| **Key Type** | **Mandatory** | **Column Name** | **Data Type** | **Size** | **Remarks (e.g. FK Reference, default value)** |
| PK | Auto-Increment | OptionID | int | Default |  |
| FK | True | QuestionID | int | Default | FK reference of tb\_questions |
|  | True | Text | Varchar | 100 |  |
|  | True | IsCorrect | Boolean | Default |  |

| **QuestionID** | **Text** | **IsCorrect** |
| --- | --- | --- |
| 1 | 5 | False |
| 1 | 6 | False |
| 1 | 7 | True |
| 1 | 8 | False |
| 2 | False | False |
| 2 | True | True |
| 3 | 5 | True |
| 3 | 6 | False |
| 3 | 7 | False |
| 3 | 8 | False |

| Table Name: **tb\_options** |
| --- |
| DDL:  CREATE TABLE tb\_options (  OptionID int AUTO\_INCREMENT,  QuestionID int NOT NULL,  Text varchar(255) NOT NULL,  IsCorrect boolean NOT NULL,  PRIMARY KEY(OptionID),  FOREIGN KEY(QuestionID) REFERENCES tb\_questions(QuestionID)  ); |
| Output of Structure: |
| DML to populate data:  INSERT INTO *tb\_options* (*QuestionID, Text, IsCorrect*)  VALUES *(1, “5”, False), (1, “6”, False), (1, “7”, True), (1, “8”, False),*  *(2, “False”, False), (2, “True”, True),*  *(3, “5”, True), (3, “6”, False), (3, “7”, False), (3, “8”, False)*; |
| Output of Sample Data: |

#### 

#### Tb\_rooms

| **Table** | **Description** | | | | |
| --- | --- | --- | --- | --- | --- |
| tb\_rooms | Table storing details of rooms | | | | |
| **Key Type** | **Mandatory** | **Column Name** | **Data Type** | **Size** | **Remarks (e.g. FK Reference, default value)** |
| PK | Auto-Increment | RoomID | int | 7 | Should have 7 random digits |
| FK | True | QuizID | int | Default | FK reference of tb\_quizes |
|  | False | Capacity | Varchar | 255 | Default value of 0 |

| **QuizID** |
| --- |
| 1 |
| 2 |
| 3 |

| Table Name: **tb\_rooms** |
| --- |
| DDL:  CREATE TABLE tb\_rooms (  RoomID int(7) AUTO\_INCREMENT,  QuizID int NOT NULL,  Capacity int DEFAULT 0,  PRIMARY KEY(RoomID),  FOREIGN KEY(QuizID) REFERENCES tb\_quizes(QuizID)  ); |
| Output of Structure: |
| DML to populate data:  INSERT INTO *tb\_rooms* (*QuizID*)  VALUES *(1), (2), (3)*; |
| Output of Sample Data: |

#### Tb\_players

| **Table** | **Description** | | | | |
| --- | --- | --- | --- | --- | --- |
| tb\_players | Table storing details of players | | | | |
| **Key Type** | **Mandatory** | **Column Name** | **Data Type** | **Size** | **Remarks (e.g. FK Reference, default value)** |
| PK | Auto-Increment | PlayerID | int | Default |  |
| FK | True | RoomID | int | 7 | FK reference of tb\_rooms |
|  | True | Nickname | Varchar | 32 |  |
|  | False | Score | int | Default | Default value of 0 |

| **RoomID** | **Nickname** |
| --- | --- |
| 1 | Tom |
| 1 | Harry |
| 1 | John |

| Table Name: **tb\_players** |
| --- |
| DDL:  CREATE TABLE tb\_players (  PlayerID int AUTO\_INCREMENT,  RoomID int(7) NOT NULL,  Nickname varchar(32) NOT NULL,  Score int DEFAULT 0,  PRIMARY KEY(PlayerID),  FOREIGN KEY(RoomID) REFERENCES tb\_rooms(RoomID)  ); |
| Output of Structure: |
| DML to populate data:  INSERT INTO *tb\_players* (RoomID, Nickname)  VALUES *(1, “Tom”), (1, “Harry”), (1, “John”)*; |
| Output of Sample Data: |

#### Tb\_choices

| **Table** | **Description** | | | | |
| --- | --- | --- | --- | --- | --- |
| tb\_choices | Table storing details of player choices | | | | |
| **Key Type** | **Mandatory** | **Column Name** | **Data Type** | **Size** | **Remarks (e.g. FK Reference, default value)** |
| FK | True | PlayerID | int | Default | FK reference from tb\_players |
| FK | True | OptionID | int | Default | FK reference from tb\_options |
|  | True | TimeTaken | int | Default | Assumes its in milliseconds |

| **PlayerID** | **OptionID** | **TimeTaken** |
| --- | --- | --- |
| 1 | 1 | 3000 |
| 1 | 5 | 1000 |
| 1 | 7 | 5000 |
| 2 | 2 | 10000 |
| 2 | 6 | 10000 |
| 2 | 8 | 10000 |
| 3 | 3 | 1000 |
| 3 | 6 | 1000 |
| 3 | 7 | 1000 |

| Table Name: **tb\_choices** |
| --- |
| DDL:  CREATE TABLE tb\_choices (  PlayerID int NOT NULL,  OptionID int NOT NULL,  TimeTaken int NOT NULL,  FOREIGN KEY(PlayerID) REFERENCES tb\_players(PlayerID),  FOREIGN KEY(OptionID) REFERENCES tb\_options(OptionID)  ); |
| Output of Structure: |
| DML to populate data:  INSERT INTO *tb\_choices* (*PlayerID, OptionID, TimeTaken*)  VALUES *(1, 1, 3000), (1, 5, 1000), (1, 7, 5000),*  *(2, 2, 10000), (2, 6, 10000), (2, 8, 10000),*  *(3, 3, 1000), (3, 6, 1000), (3, 7, 1000)*; |
| Output of Sample Data: |

## **5)** **SQL for App/Game Functions**

This section documents all the necessary SQL to perform data retrieval or update when the App/Game Functions are executed. (Update SQL-Functions mapping.xls)

Guidelines

| S/N | Function Name | (Name of the game function, e.g. user login, buy weapon, join party) |
| --- | --- | --- |
| 0 | Tables Affected | (the tables that will be used by the function, can be more than one) |
|  | Inputs | (what are the inputs sent by the client) |
|  | Outputs | (what will be the response or data returned from the query?) |
|  | SQL Statement | [the actual SQL Statement to be used using the samples input] |
|  | Remarks | [other remarks or describe any other assumptions or explain how to the statement works] |
|  | Example | [give ONE example of the actual SQL statements, and variations if there is any] |

### Create new host account

| S/N | App/Game Function | *Register New User* |
| --- | --- | --- |
| 1 | Tables Affected | tb\_hosts |
|  | Inputs | Username1 , Password1, Email1 |
|  | Outputs | Returns how many rows added |
|  | SQL Statement | Insert into tb\_hosts (Username, Password, Email) values (Username1, Password1, Email1); |
|  | Remarks | HostID will be auto-generated as Primary key, hence no input required |
|  | Example | Insert into tb\_hosts (Username, Password, Email) values (*“SchoolTeacher”, “Password1”, “SchoolTeacher@gmail.com”)*; |

### Login existing account

| **S/N** | **App/Game Function** | *Login existing account* |
| --- | --- | --- |
| **2** | Tables Affected | tb\_hosts |
|  | Inputs | Username2 , Password2 |
|  | Outputs | User record or NULL (if invalid user or wrong password) |
|  | SQL Statement | Select \* from tb\_hosts  where Username = Username2  And Password = Password2; |
|  | Remarks | Assumes that the account exists before sql statement is called |
|  | Example | Select \* from tb\_users where Username = “SchoolTeacher” And Password = “Password1”; |

### Create new quiz

| **S/N** | **App/Game Function** | *Create new quiz* |
| --- | --- | --- |
| **3** | Tables Affected | tb\_quiz |
|  | Inputs | HostID1, Name1, Description1 |
|  | Outputs | Returns how many rows added |
|  | SQL Statement | Insert into tb\_quiz (HostID, Name, Description) values (HostID1, Name1, Description1) |
|  | Remarks | Assumes a valid HostID is given - account exists before sql statement is called.  QuizID will be auto-generated as Primary key, hence no input required. |
|  | Example | Insert into tb\_quiz (HostID, Name, Description) values (1, “FirstQuiz”, “The first quiz created by SchoolTeacher”); |

### Create new question

| **S/N** | **App/Game Function** | *Create a new question* |
| --- | --- | --- |
| **2** | Tables Affected | Tb\_questions, tb\_options |
|  | Inputs | QuizID1, Text1, Type1, MaxScore1, TimeLimit1  Based on type, ie MCQ or true/false question, number of options are different  Assuming MCQ,  OptionText1, IsCorrect1  OptionText2, IsCorrect2  OptionText3, IsCorrect3  OptionText4, IsCorrect4 |
|  | Outputs | Returns how many rows added |
|  | SQL Statement | DELIMITER &&  CREATE PROCEDURE create\_question (IN QuizID1 int, IN Text1 varchar(100), IN Type1 varchar(32), IN MaxScore1 int, IN TimeLimit1 int,  IN OptionText1 varchar(100), IN IsCorrect1 boolean,  IN OptionText2 varchar(100), IN IsCorrect2 boolean,  IN OptionText3 varchar(100), IN IsCorrect3 boolean,  IN OptionText4 varchar(100), IN IsCorrect4 boolean)  BEGIN  DECLARE QuestionID1 int;  Insert into tb\_questions (QuizID, Text, Type, MaxScore, TimeLimit) values (QuizID1, Text1, Type1, MaxScore1, TimeLimit1);  SET QuestionID1 = LAST\_INSERT\_ID();  Insert into tb\_options (QuestionID, Text, IsCorrect) values  (QuestionID1, OptionText1, IsCorrect1),  (QuestionID1, OptionText2, IsCorrect2),  (QuestionID1, OptionText3, IsCorrect3),  (QuestionID1, OptionText4, IsCorrect4);  END &&  DELIMITER ; |
|  | Remarks | Assumes that the quiz exists before sql statement is called.  Question ID will be auto-generated as Primary key, hence no input required.  Assumes that creation of the question is done without edits, else the statement would require a specific ID of the question instead of the last created question. |
|  | Example | CALL create\_question(0, “How many continents are there in the world?”, “MCQ”, 2000, 30000,  “5”, False,  “6”, False,  “7”, True,  “8”, False); |

### Create new room with existing quiz

| **S/N** | **App/Game Function** | *Create new room with existing quiz* |
| --- | --- | --- |
| **2** | Tables Affected | tb\_rooms |
|  | Inputs | QuizID1, HostID1 |
|  | Outputs | Returns how many rows added |
|  | SQL Statement | Insert into tb\_rooms (QuizID, HostID) values (QuizID1, HostID1); |
|  | Remarks | Assumes that both account and quiz exist before sql statement is called.  Capacity is updated whenever a new player joins the session (as shown below).  RoomID will be auto-generated as Primary key, hence no input required. |
|  | Example | Insert into tb\_rooms (QuizID, HostID) values (1, 1); |

### Creating new player

| **S/N** | **App/Game Function** | *Creating new player* |
| --- | --- | --- |
| **2** | Tables Affected | Tb\_players, tb\_rooms |
|  | Inputs | RoomID1, Nickname1 |
|  | Outputs | Returns how many rows added, returns how many rows affected |
|  | SQL Statement | DELIMITER &&  CREATE PROCEDURE create\_player(IN RoomID1 int, IN Nickname1 varchar(32))  BEGIN  DECLARE room\_capacity int;  Insert into tb\_players (RoomID, Nickname) values (RoomID1, Nickname1);  Select capacity as room\_capacity from tb\_rooms Where RoomID = RoomID1;  SET room\_capacity = room\_capacity + 1;  Update tb\_rooms Set capacity = room\_capacity Where RoomID = RoomID1;  END &&  DELIMITER ; |
|  | Remarks | Assumes that the room exists before sql statement is called.  Score is generated after every question is answered  PlayerID will be auto-generated as Primary key, hence no input required. |
|  | Example | CALL create\_player(1, “Tom”);  CALL create\_player(1, “Harry”);  CALL create\_player(1, “John”); |

### Player answers question

| **S/N** | **App/Game Function** | *Player answers question* |
| --- | --- | --- |
| **2** | Tables Affected | Tb\_choices, tb\_players |
|  | Inputs | PlayerID1, OptionID1, TimeTaken1 |
|  | Outputs | Returns how many rows are added in tb\_choice.  No change in tb\_player if the answer is wrong. |
|  | SQL Statement | DELIMITER &&  CREATE PROCEDURE player\_choice(IN PlayerID1 int, IN OptionID1 int, IN TimeTaken1 int, OUT bool\_result boolean)  BEGIN  DECLARE calculated\_score, player\_score, time\_limit, max\_score int;  Insert into tb\_choices (PlayerID, OptionID, TimeTaken) values (PlayerID1, OptionID1, TimeTaken1);  Select IsCorrect into bool\_result from tb\_options Where OptionID = OptionID1;  IF bool\_result = True THEN  Select TimeLimit into time\_limit from tb\_questions Inner join tb\_options  On tb\_questions.QuestionID = tb\_options.QuestionID  where OptionID = OptionID1;  Select MaxScore into max\_score from tb\_questions Inner join tb\_options  On tb\_questions.QuestionID = tb\_options.QuestionID  where OptionID = OptionID1;  SET calculated\_score = (1-TimeTaken1/time\_limit/2)\*max\_score*;*  Select Score into player\_score from tb\_players where PlayerID = PlayerID1;  SET calculated\_score = calculated\_score + player\_score;  Update tb\_players Set Score = CalculatedScore Where PlayerID = PlayerID1;  END IF;  END &&  DELIMITER; |
|  | Remarks | Assumes that the player exists before sql statement is called.  Assumes that the option exists before sql statement is called. |
|  | Example | player\_choice(1, 1, 3000, OUT bool\_result boolean)  Select bool\_result;  player\_choice(1, 5, 1000, OUT bool\_result boolean)  Select bool\_result;  player\_choice(1, 7, 5000, OUT bool\_result boolean)  Select bool\_result;  player\_choice(2, 2, 10000, OUT bool\_result boolean)  Select bool\_result;  player\_choice(2, 6, 10000, OUT bool\_result boolean)  Select bool\_result;  player\_choice(2, 8, 10000, OUT bool\_result boolean)  Select bool\_result;  player\_choice(3, 3, 1000, OUT bool\_result boolean)  Select bool\_result;  player\_choice(3, 6, 1000, OUT bool\_result boolean)  Select bool\_result;  player\_choice(3, 7, 1000, OUT bool\_result boolean)  Select bool\_result; |

### Creating leaderboard

| **S/N** | **App/Game Function** | *Creating Leaderboard* |
| --- | --- | --- |
| **2** | Tables Affected | -nil- the leaderboard is a temporary table created after each session is over |
|  | Inputs | roomID, podium\_count |
|  | Outputs | List of players sorted on based on their scores, where the top “podium\_count” is displayed |
|  | SQL Statement | DELIMITER &&  CREATE PROCEDURE create\_leaderboard(IN RoomID1 int, IN podium\_count int)  BEGIN  Select \* from tb\_players  Where RoomID = RoomID  Order by score DESC  Limit podium\_count;  END &&  DELIMITER ; |
|  | Remarks | Assumes that the room exists before sql statement is called.  Assumes that there are more players in the room than the “podium\_count” |
|  | Example | CALL create\_leaderboard(1, 3); |

### 

### Delete Question

| **S/N** | **App/Game Function** | *Delete Question* |
| --- | --- | --- |
| **2** | Tables Affected | Tb\_questions |
|  | Inputs | QuestionID1 |
|  | Outputs | Shows number of rows affected. |
|  | SQL Statement | DELIMITER &&  CREATE PROCEDURE delete\_question(IN QuestionID1 int)  BEGIN  DELETE tb\_questions, tb\_options FROM tb\_questions  INNER JOIN tb\_options ON tb\_options.QuestionID = tb\_questions.QuestionID  WHERE tb\_questions.QuestionID = QuestionID1;  END &&  DELIMITER ; |
|  | Remarks | Assumes that the question and its relevant options exist before sql statement is called. |
|  | Example | CALL delete\_question(1);  CALL delete\_question(2);  CALL delete\_question(3); |

### 

### Delete Quiz

| **S/N** | **App/Game Function** | *Delete Quiz* |
| --- | --- | --- |
| **2** | Tables Affected | Tb\_quizes, tb\_questions, tb\_options |
|  | Inputs | QuizID1 |
|  | Outputs | Shows number of rows affected. |
|  | SQL Statement | DELIMITER &&  CREATE PROCEDURE delete\_quiz(IN QuizID1 int)  BEGIN  DELETE tb\_quizes, tb\_questions, tb\_options  FROM tb\_quizes  INNER JOIN tb\_questions ON tb\_questions.QuizID = tb\_quizes.QuizID  INNER JOIN tb\_options ON tb\_options.QuestionID = tb\_questions.QuestionID  WHERE tb\_quizes.QuizID = QuizID1;  END &&  DELIMITER ; |
|  | Remarks | Assumes that the quiz exists with relevant questions. |
|  | Example | CALL delete\_quiz(1); |

### 

### Delete Host Account

| **S/N** | **App/Game Function** | *Delete Host Account* |
| --- | --- | --- |
| **2** | Tables Affected | Tb\_hosts, tb\_quizes, tb\_questions, tb\_options |
|  | Inputs | HostID1 |
|  | Outputs | Shows number of rows affected. |
|  | SQL Statement | DELIMITER &&  CREATE PROCEDURE delete\_host(IN HostID1 int)  BEGIN  DELETE tb\_hosts, tb\_quizes, tb\_questions, tb\_options  FROM tb\_hosts  INNER JOIN tb\_quizes ON tb\_quizes.HostID = tb\_hosts.HostID  INNER JOIN tb\_questions ON tb\_questions.QuizID = tb\_quizes.QuizID  INNER JOIN tb\_options ON tb\_options.QuestionID = tb\_questions.QuestionID  WHERE tb\_quizes.QuizID = QuizID1;  END &&  DELIMITER ; |
|  | Remarks | Assumes that the host exists with relevant quizzes created. |
|  | Example | delete\_host(1); |

### 

## **6)** **Others**

E.g. explain what are the optimizations you have implemented to make the database more efficient

What have I improved

1. Previous tables were not normalized.
   1. Data was cluttered and hard to extract.
      1. Solution: normalized tables by splitting users into “hosts” and “players”.
         1. Allowed for players to have no login credentials and hosts to have login credentials and ability to create quizzes.
      2. Solution: normalized tables by splitting questions into “question”, “option” and “choice”.
         1. Allowed for questions to contain options and players to choose their answer.
      3. Solution: normalized tables by splitting session into “rooms” and “quizzes”
         1. Allowed for hosts to create quizzes, then creating rooms based on the quizzes, allows the players to interact in a room, without changing the original quiz.
2. Named variables had the same name
   1. Hard to reference named variables
      1. Solution: PID was used very often, thus changed into “tablename”ID, where it was easier to see which ID it was referencing
         1. Allowed for tables with multiple foreign keys to be readable, i.e. in the choice table, playerID and optionID were both called PID in their respective tables, thus it was easier to call it an ID of its table name.