Golf Tracking App: Hole in One

Jing-Chian Teng

Boston University

CS 669: Database Design and Implementation for Business

Edward Matthews

12.15.2023

Abstract

I want to build a mobile app system and computer system that tracks every shot every golfer hits in every course, not only in a stimulator but also in a real course. I want to name the system “Hole in One”. When everyone uses the system, they can view the history of other

golfers and the conditions like wind speed and weather to make the ideal shot. Typically, we are familiar with the golf course based on the experience, but it will take a lot of time to be an ace on the golf course. When we have the database from other golfers or maybe the PGA golfers, we can spend less time to be more familiar with the course.

Conceptual Design

Use cases and Fields:

1. Download the app or use the website.
2. Create an account.
3. The system will ask the user to enter their bio-information.
4. The system will ask the users to allow the system to use the camera to record every shot they made to track the shot and the location.

Field	What it stores	Why it's needed
Account name	Summary of the account	In case the user has more than one account. Also display on email
First name	First name of the user	displaying the person's name on screens and addressing them when sending them emails or other communications.

Last name	Last name of the user	displaying the person's name on screens and addressing them when sending them emails or other communications.
Customer Since	The date the account was created	To give discount to loyalty customers
Bio-information	The user gender and height	To categorize the different users
Email address	the way to contact the user	To inform some important news to the user

Field	What it stores	Why it's needed
Play date	The date the user play	To track the weather condition
Shot distance	Shot from the users	To track the hit distance from the user
club	Club used by the user	To track the chosen club
wind	Wind speed on the course	To track the wind speed

Summary and Reflection:

My system is for Hole in One which records every shot from the users and the weather conditions on the course, and it records across all states. Typically, when we play on a golf

course, we only can record our shots and our friends. However, using Hole in One can provide the history from the user, which is a big amount, and give the best strategy for every course.

To improve the system, I need plenty of users to expand the database in every course because every golf course has different conditions, and every golfer will hit the ball at a different location. Also, bio-information also plays a big role in golf. I think I have to store all kinds of data.

I am excited to develop this project and publish my Hole in One in the future because I am interested in golf, but playing golf is a high-cost sport especially if we want to be an expert. I hope this system can reduce the cost of playing golf and become a more affordable sport for other people. User feedback is important to the system.

Logical Design

Structure Rule:

Installation and sign-up:

1. Download the app or use the website.
2. Create an account.
3. The system will ask the user to enter their bio-information.
4. The system will ask the user to allow the system to have the location.
5. The system will ask the users to allow the system to use the camera to record every shot they made to track the shot and the location.

Golf course tracking:

1. The weather on the golf course.
2. The player selected the golf club and the distance the player hit.

Rules:

1. Every golf shot hit a ball with a club; each club may be used many times.
2. Every golf shot only appears in a course; each course may have many golf shots.
3. Every golf shot only appears in an account; the account may be associated with many shots.

Define:

1. I created the rule because no one can hit a ball with multiple clubs at once. Also, we can use a club multiple times. For example, there are some Par 4 and Par 5 holes in a golf course, and we will use a driver multiple times. However, some clubs may not be used when a golfer plays. For example, when we never hit the ball into the bunker, we don't need the sand wedge.
2. Every golfer can only go to a golf course each time. And of course, there are different golfers playing at the golf course on the same day.
3. This rule indicates that every golf shot is linked with an account. And of course, each account can be associated with many shots since the golfer swings a lot of shots.

Summary and Reflection:

My system is for Hole in One which records every shot from the users and the weather conditions on the course, and it records across all states. Typically, when we play on a golf course, we only can record our shots and our friends. However, using Hole in One can provide the history from the user, which is a big amount, and give the best strategy for every course.

To improve the system, I need plenty of users to expand the database in every course because every golf course has different conditions, and every golfer will hit the ball at a different

location. Also, bio-information also plays a big role in golf. I think I have to store all kinds of data.

I am excited to develop this project and publish my Hole in One in the future because I am interested in golf, but playing golf is a high-cost sport especially if we want to be an expert. I hope this system can reduce the cost of playing golf and become a more affordable sport for other people. User feedback is important to the system.

Physical Design

Use cases and Fields (New):

1. Download the app or use the website.
2. Ask the user to choose a free account or paid account.
3. The user selects the type of account, and the system will ask the user to enter their bio-information and create it in the database.
4. The system will ask the users to allow the system to use the camera to record every shot they made to track the shot and the location.
5. When the user uses the simulator to play, the system will connect with the compatible simulator and upload the data to Hole in One.

Tracking use case (new):

1. The user visits a golf course and plays for 18 holes.
2. The Hole in One detects that the game is made and record all information in the database as whether the game is indoor simulator or outdoor course.

Structure Rule (new):

Installation and sign-up:

1. Download the app or use the website.

2. Create an account.
3. The system will ask the user to enter their bio-information.
4. The system will ask the user to allow the system to have the location.
5. The system will ask the users to allow the system to use the camera to record every shot they made to track the shot and the location.
6. An account is free or paid.
7. The game is indoor, or outdoor.

Summary and Reflection (new):

My database is for a mobile app named Hole in One which records shots made across all golf courses, making it for any shot history. Typically, golfers don't record their every shot at every course. However, Hole in One can provide every detail for every shot including distance, weather, and wind. The database can support every golfer analyzing their shots across all courses.

The structural database rules and conceptual ERD for my database design contain entities of Course, Shot, Club, Account, and the relationships between them. The design contains a hierarchy of Shot/Outdoor course and Shot/Indoor Simulator to reflect the two primary ways people play golf. The design also contains a hierarchy of Account/Paid Account and Account/ Free Account to reflect the fact that people can sign up for a free account or paid account for Hole in One. The DBMS physical ERD contains the same entities and relationships and uses the best practice of synthetic keys.

From iteration 2 to iteration 3, I made some updates to all my categories. I think the update can improve my future work because it gives me a clear track to follow. Also, I learn a lot from the contents since there are a lot of examples. I am very excited to do the project. However,

I want to avoid making my database too complex in future weeks, so I am trying to be cautious about the relationships of my entities, but I am still very positive about my project.

Implementation

Account:

Attributes	Datatype	Reason
Username	Varchar(64)	Every account needs a username to login to the account. I allow usernames to be up to 64 characters.
Age	Date	Every customer can enter their birthday to get some discount or prize during the birth month.
Gender	Char(1)	There are two types of gender – Male and Female. When entering gender, it is more easy to predict the shot.
email	Varchar (64)	This is the email address of the account holder. 255 characters should be a safe upper bound.
Password	Varchar (64)	Every account has a password. It will be stored in encrypted text format in the database. 255 characters should be a safe limit to store encrypted text.
Name	Varchar (64)	This is the name of the account holder; 255 characters should be a safe upper bound.

Account Type	Char(1)	There are two types of accounts – free and paid. This attribute is the subtype discriminator indicating which it is.

Paid Account:

RenewalDate	Date	This is the date on which the account needs to be renewed with a new payment.

Course:

Course Name	Varchar(64)	Every Course has a name which acts like the identifier for the product when people are looking it up. I allow for up to 64 characters.
Location	Varchar(64)	Every Course is in a location. I allow for up to 64 characters.
Type	Char(1)	There are two types of golf courses – indoor simulator and outdoor golf course
Difficulty Level	Varchar(64)	When the golfer visits the golf course, they can know how hard the course is. I use numbers to distinguish the level. Smaller number means easier.

Par Score	Integer	Every course has a par score. When a golfer plays at the course, they can see if they hit higher or lower than par.

Hole:

Hole ID	Varchar(64)	Every hole has an ID which acts like the identifier for the hole when people are looking it up. I allow for up to 64 characters.
Par	Integer	Every hole has a par score. When a golfer plays at the course, they can see if they hit higher or lower than par.
Stroke Count	Integer	To count the total stroke of the hole
Fairway Hit	Boolean	To see if the golfer hit on the fairway or not
Green in regulation	Boolean	To see if the golfer hit on the green in regulation or not
Putt	Integer	To count how many putt does the golfer shot

Primary keys:

Pk column	Description
Hole.Hole ID	This is the Pk of Hole table
Address.AddressID	This is the pk of Address table
State.StateID	This is the pk of State table

Course.CourseID	This is the pk of Course table
Shot.ShotID	This is the pk of Shot table
ClubGolfShotLink. ClubGolfShotLinkID	This is the pk of ClubGolfShotLink table
Club.ClubID	This is the pk of Club table
Account.AccountID	This is the pk of Account table
Indoorsimulator. CourseID	This is the pk of Indoorsimulator table
OutDoorCourse. CourseID	This is the pk of OutDoorCourse table
Free_account. AccountID	This is the pk of Free_account table
Paid_account. AccountID	This is the pk of Paid_account table

Foreign Key:

FK column	Unique	Description
Address.StateID	Not Unique	This one is not unique because there are a lot of addresses in the same state.
Hole.CourseID	Not Unique	There are a lot of holes in the same course.
Course.AddressID	Unique	There is only a course at the address.
Course.ShotID	Not Unique	The course has a lot of shots.
Shot.CourseID	Unique	There is only one shot in the course.

Shot.AccountID	Unique	The shot is in an account.
ClubGolfShotLink. ClubID	Not Unique	Same Club can be used to shoot a lot of times.
ClubGolfShotLink. ShotID	Unique	Every shot can only link with a club.
Indoorsimulator. CourseID	Not Unique	Indoorsimulator can link with plenty of CourseIDs.
OutDoorCourse. CourseID	Not Unique	OutDoorCourse can link with plenty of CourseIDs.
Free_account. AccountID	Not Unique	Free accounts can link to a lot of AccountIDs.
Paid_account. AccountID	Not Unique	Paid accounts can link to a lot of AccountIDs.

Summary:

My database is for a mobile app named Hole in One which records shots made across all golf courses, making it for any shot history. Typically, golfers don't record their every shot at every course. However, Hole in One can provide every detail for every shot including distance, weather, and wind. The database can support every golfer analyzing their shots across all courses.

The structural database rules and conceptual ERD for my database design contain entities of Course, Shot, Club, Account, and the relationships between them. The design contains a hierarchy of Course/Outdoor course and Course/Indoor Simulator to reflect the two primary ways people play golf. The design also contains a hierarchy of Account/Paid Account and

Account/ Free Account to reflect the fact that people can sign up for a free account or paid account for Hole in One. The DBMS physical ERD contains the same entities and relationships and uses the best practice of synthetic keys.

The SQL script used for creating all tables precisely matches the specifications outlined in the physical ERD provided by the DBMS. To enhance database access speed, essential indexes have been established, with their details included in a separate index script.

Constructing the tables and columns necessary for my database is an exciting process. My design, once theoretical, is now realized in a tangible, contemporary database environment. I'm eager to populate it with data and begin querying.

The structure of my database appears to align well with SQL. The true test of its effectiveness will come in the next phase, as I start inputting data and fully utilizing the database.

Implementation II

Stored Procedure Execution:

I name the stored procedure “AddFreeAccount”, and give it parameters that correspond to the Account and FreeAccount tables. In the procedure, I don’t need a parameter because CreatedDate is always the current date and using the GETDATE() function in SQL Server. Since this procedure is always for a free account, I do not use a parameter for AccountType, but hardcode the character “F”.

For adding the free account, I created a fictional person named John Doe with his account information. I nested the stored procedure call between transaction control statements to ensure the transaction is committed.

Trigger Creation and Use:

1. `CREATE TRIGGER UpdateLastReservationDateOnUpdate`:

- `CREATE TRIGGER` is the command used to create a new trigger.

- `UpdateLastReservationDateOnUpdate` is the name you're giving to the trigger, which describes its purpose.

2. `AFTER UPDATE ON Reservation`:

  - `AFTER UPDATE` specifies that the trigger should run after any `UPDATE` operation is performed on a table.

  - `ON Reservation` specifies that the table the trigger is monitoring for updates is the `Reservation` table.

3. `FOR EACH ROW`:

  - This indicates that the trigger should be executed once for each row that gets updated in the `Reservation` table.

4. `BEGIN ... END;`:

  -This defines the beginning and end of the trigger's action block.

  - Any SQL statements within this block are executed as part of the trigger.

5. `UPDATE Account SET LastReservationDate = NEW.ReserveDate WHERE AccountID = NEW.AccountID;`:

  - This is the SQL statement that is executed whenever the trigger is fired.

  - `UPDATE Account` means it will update rows in the `Account` table.

- `SET LastReservationDate = NEW.ReserveDate` sets the `LastReservationDate` field in the `Account` table to the value of `ReserveDate` from the newly updated row in the `Reservation` table. The `NEW` keyword refers to the state of the row in the `Reservation` table after it is updated.

- `WHERE AccountID = NEW.AccountID` limits the update to the row in the `Account` table where `AccountID` matches the `AccountID` of the updated row in the `Reservation` table.

ReserveID	PK of Reserve Table Varchar(12)
AccountID	FK of Reserve Table decimal(12)
CourseID	FK of Reserve Table decimal(12)
Reserve Date	This is the reserve day
Reserve Time	This is the reserve time
Status	To show the status is reservation confirmed, completed, or canceled.

Query identification and Explanation:

Question: How can we retrieve a list of all reservations made by a specific account, including the reservation date, time, and status?

Answer: This question is very useful based on Customer Service, Account Management, Data Analysis, and Reporting. For Customer Service, understanding a customer's reservation history can help staff provide personalized service. For example, if a customer call inquiring about their reservations, the staff can quickly look up all the details. For Account Management, the user's reservation history is valuable for account management purposes. It helps in understanding the frequency and patterns of a user's engagement with the service. For Data

Analysis, analyzing reservation data can help the business understand demand patterns and customer preferences, which can inform marketing strategies, resource allocation, and operational planning. For Reporting, Regular reports on reservation activity can be useful for management to assess the performance of the business over time.

- `SELECT` specifies the columns that we want to retrieve: `ReserveID`, `ReserveDate`, `ReserveTime`, and `Status`.

- `FROM Reservation` indicates that we are selecting these columns from the `Reservation` table.

- `WHERE AccountID = ?` is a filter condition that limits the results to rows where the `AccountID` column matches a specific account ID. The question mark `?` is a placeholder for the account ID you want to filter by. In a real query, you would replace the question mark with an actual account ID or use a parameterized query to pass the ID.

- `ORDER BY ReserveDate DESC, ReserveTime DESC` orders the results first by `ReserveDate` in descending order (most recent first) and then by `ReserveTime` in descending order. This means the most recent reservations will appear first in the result set.

Summary:

Compared to Iteration 4, I add a new table Reservation because reservation is also an important part in golf. For customers, making a reservation guarantees access to a service or space at a specified time, eliminating wait times and ensuring a more organized and stress-free experience. Businesses, on the other hand, rely on reservations for efficient resource management and capacity planning. They enable the anticipation of demand, allowing for the

strategic allocation of staff, resources, and inventory, which in turn leads to improved operational efficiency. After adding the Reservation Table, I think my database design is more completed because this database can also be extended to business providers.

After building this database, I recognize it's just the beginning. The market is teeming with similar applications, providing a rich source of features and functionalities that I can learn from and possibly integrate into my own application. Drawing inspiration from existing solutions, I can refine and enhance my application, tailoring it to meet specific user needs that may not be fully addressed by current offerings. Once I feel confident that my application has reached a level of maturity and stands out in the market, I plan to publish it on platforms like Google Play or the App Store, marking the start of my entrepreneurial journey.

Creating a mature application is an arduous and demanding endeavor, requiring a deep understanding of user experience, technical robustness, and market trends. Nevertheless, I am committed to this project. With perseverance, attention to detail, and a willingness to continuously learn and adapt, I am optimistic about developing an application that not only meets users' needs but also exceeds their expectations. By staying user-focused and innovation-driven, I aim to build an application that delivers value and perhaps even sets new standards in its category.

Figures

Figure 1:

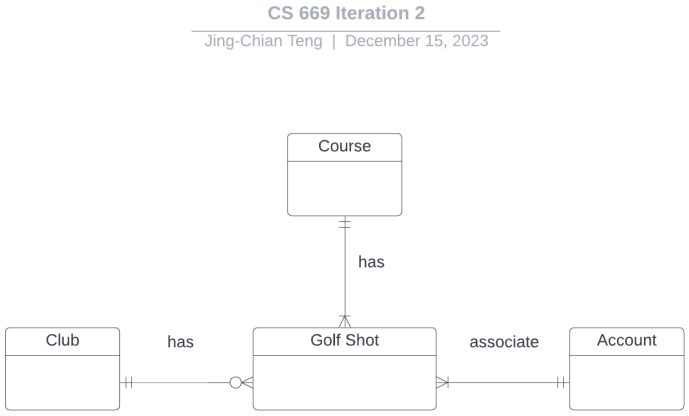


Figure 2:

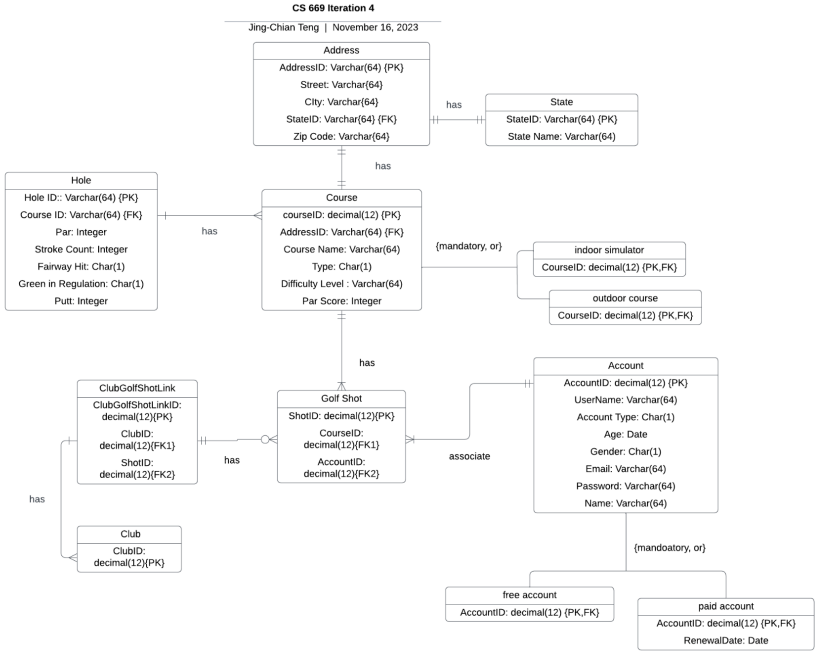


Figure 3:

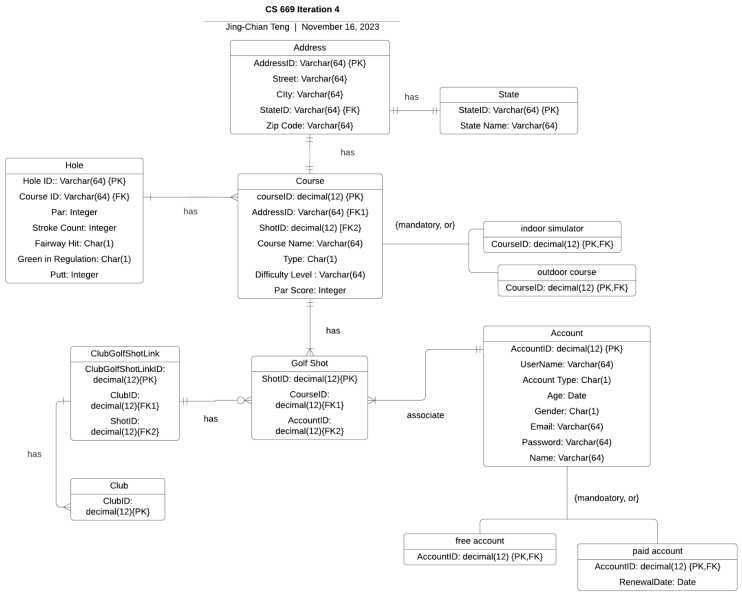
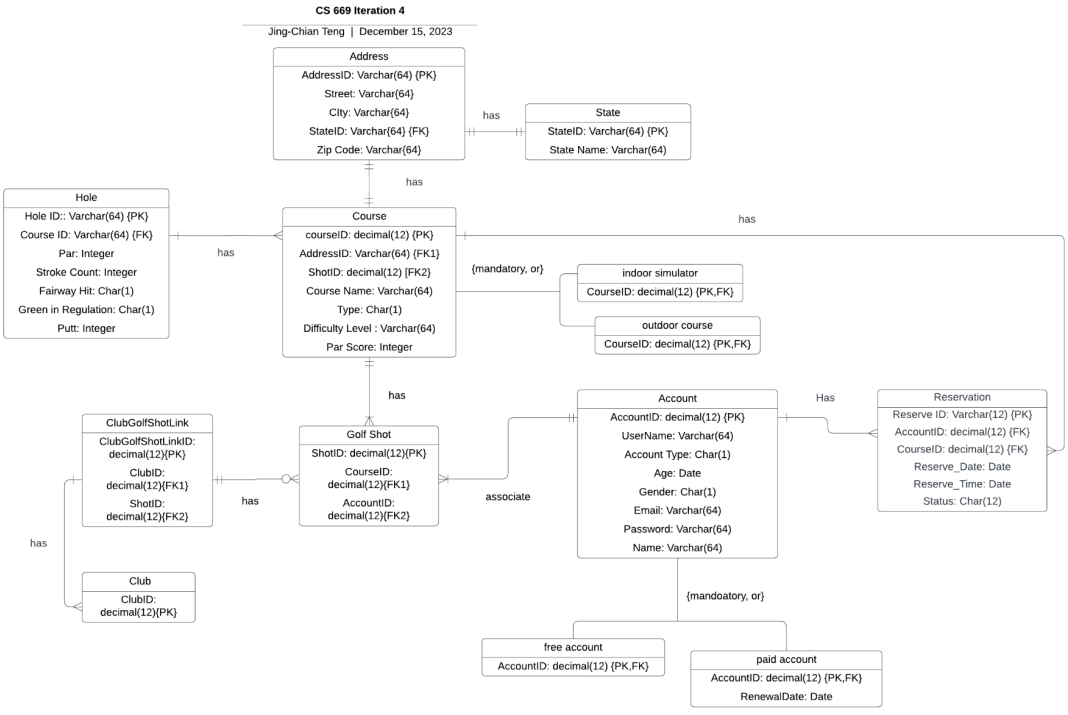


Figure 4:



SQL query (Iteration 4):

```
create Table State(
StateID Varchar(64) Not Null primary key,
State Name Varchar(64) Not null);

create Table Address(
AddressID Varchar(64) primary key not null,
Street Varchar(64) Not null,
City Varchar(64) Not Null,
StateID Varchar(64) not null,
Zip code Varchar(64) not null,
foreign key(StateID) references State(StateID));

Create Table Course(
CourseID decimal(12) Primary key Not null,
AddressID Varchar(64) Not null,
ShotID decimal(12) Not Null,
Course Name Varchar(64) Not Null,
Type Char(1) Not Null,
Difficulty Level Varchar(64) Not Null,
Par Score Integer Not Null,
foreign key(AddressID) references Address(AddressID)
foreign key(ShotID) references Shot(ShotID));

create Table Hole(
HoleID varchar(64) Not null primary key,
CourseID decimal(64)Not null,
Par Integer Not null,
Stroke Count integer Not null,
Fairway hit Chr(1) Not null,
Green_in_Regulation char(1) Not null,
Putt integer Not null,
foreign key(CourseID) references Course(CourseID));

Create Table indoorsimulator(
CourseID decimal(12)primary key Not null,
foreign key(CourseID) references Course(CourseID));
```

```
Create Table OutdoorCourse(
CourseID decimal(12) primary key Not null,
foreign key(CourseID) references Course(CourseID));

Create Table Account(
AccountID decimal(12) Primary key not null,
UserName varchar(64) not null,
Account Type char(1) not null,
Age Date not null,
Gender Char(1) not null,
Email varchar(64) not null,
Password Varchar(64) not null,
Name Varchar(64)not null);

Create Table Shot(
ShotID decimal(12) primary key not null,
CourseID decimal(12) not null,
AccountID decimal(12) not null,
foreign key(CourseID) references Course(CourseID));

create Table Club(
ClubID Varchar(64) primary key not null);

create table ClubGolfShotLink(
ClubGolfShotLinkID decimal(12) primary key not null,
ClubID VArchar(64) not null,
ShotID decimal(12) not null,
foreign key(ClubID) references Club(ClubID),
foreign key(ShotID) references Shot(ShotID));

create table free_account(
AccountID decimal(12) primary key not null,
foreign key(AccountID) references Account(AccountID));

create table paid_account(
AccountID decimal(12) primary key not null,
RenewalDate date not null,
foreign key(AccountID) references Account(AccountID));
```

```
create index AddressStateIdx on Address(StateID);
create index HoleCourseIdx on Hole(CourseID);
create unique index CourseAddressIdx on Course(AddressID);
create index CourseShotIdx on Course(ShotID);
create unique index ShotCourseIdx on Shot(CourseID);
create unique index ShotAccountIdx on Shot(AccountID);
create index ClubGolfShotLinkClubIdx on ClubGolfShotLink(ClubID);
create unique index ClubGolfShotLinkShotIdx on ClubGolfShotLink(ShotID);
create index IndoorsimulatorCourseIdx on Indoorsimulator(CourseID);
create index OutDoorCourseCourseIdx on OutDoorCourse(CourseID);
create index Free_accountAccountIdx on Free_account(AccountID);
create index Paid_accountAccountIdx on Paid_account(AccountID);

```

SQL query (Iteration 5):

```
CREATE PROCEDURE AddFreeAccount @AccountID decimal(12), @UserName varchar(64), @Account_Type char(1),
@Age Date, @Gender Char(1), @Email varchar(64), @Email varchar(64), @Password Varchar(64), @Name Varchar(64))
AS BEGIN
  INSERT INTO Account(AccountID, UserName, Account_Type, Age, Gender, Email, Password,Name)
  VALUES(@AccountID, @UserName, @Account_Type, @Age, @Gender, @Email, @Password,Name);
  INSERT INTO free_account(AccountID)
  VALUES(AccountID);
END;

BEGIN Transaction;
Execute AddFreeAccount 1, 'john_doe', 'F', '1980-01-01','M', 'johndoe@email.com', pwd12345, 'John Doe';
COMMIT Transaction AddFreeAccount;

```

```
Create Table Reservation(
ReserveID Varchar(12) primary key not null,
AccountID decimal(12) not null,
CourseID decimal(12) not null,
Reserve_Date Date not null,
Reserve_Time Date not null,
Status Char(12) not null,
foreign key(AccountID) references Account(AccountID)
foreign key(CourseID) references Course(CourseID));

CREATE TRIGGER ReservationTrigger
AFTER UPDATE ON Account
FOR EACH ROW
WHEN OLD.UserName != NEW.UserName
BEGIN
    INSERT INTO AccountLog (AccountID, ChangedColumn, OldValue, NewValue)
    VALUES (OLD.AccountID, 'UserName', OLD.UserName, NEW.UserName);
END;
```

```

SELECT ReserveID, ReserveDate, ReserveTime, Status
FROM Reservation
WHERE AccountID = 12345
ORDER BY ReserveDate DESC, ReserveTime DESC;

```