**SQL PROJECT**

**Creating Database Model For Computer Game*: Catch The Fish***

Technion - Israel Institute of Technology

Continuing and Professional Education

Course: Big Data Analyst

Handed by: Haim Zimbovsky, Slava Bakman, Gabriel Halimi

Table of Contents

[1 Intro 3](#_Toc75719659)

[2 Database Model - ERD 5](#_Toc75719660)

[3 Tables Description 6](#_Toc75719661)

[3.1 tbl\_Users 6](#_Toc75719662)

[3.2 tbl\_Countries 7](#_Toc75719663)

[3.3 tbl\_Ganders 7](#_Toc75719664)

[3.4 tbl\_Fisherman 7](#_Toc75719665)

[3.5 tbl\_ScoreStatus 8](#_Toc75719666)

[3.6 tbl\_GameConfiguratin 8](#_Toc75719667)

[4 Custom Data Types 10](#_Toc75719668)

[5 User Define Functions (UDF) 11](#_Toc75719669)

[5.1 fn\_Rand 11](#_Toc75719670)

[5.2 fn\_drawFish 11](#_Toc75719671)

[5.3 fn\_updateFishermanRating 12](#_Toc75719672)

[5.4 fn\_ValidEmail 12](#_Toc75719673)

[5.5 fn\_checkAge 12](#_Toc75719674)

[6 Stored Procedures 13](#_Toc75719675)

[6.1 sp\_SelectFishermanID 13](#_Toc75719676)

[6.2 sp\_RunOneSet 13](#_Toc75719677)

[6.3 sp\_RunAlleSets 13](#_Toc75719678)

[6.4 sp\_RunFishermanGame 13](#_Toc75719679)

[6.5 Check Username - sp\_If\_username\_exist 14](#_Toc75719680)

[6.6 sp\_checkpassword 14](#_Toc75719681)

[6.7 Registration - sp\_UserAdd 15](#_Toc75719682)

[6.8 sp\_Login 16](#_Toc75719683)

[6.9 sp\_Logout 16](#_Toc75719684)

[7 Appendix 17](#_Toc75719685)

# Intro

The computer game “*Catch the Fish*” was made by BDA curse students. The main subject is to create a fully functional database model that stores all the data related to the game. The database is used to store information about users, game characteristics, and success rates (or loss). Furthermore, the database model enables proper initial registration procedures according to predefined settings.

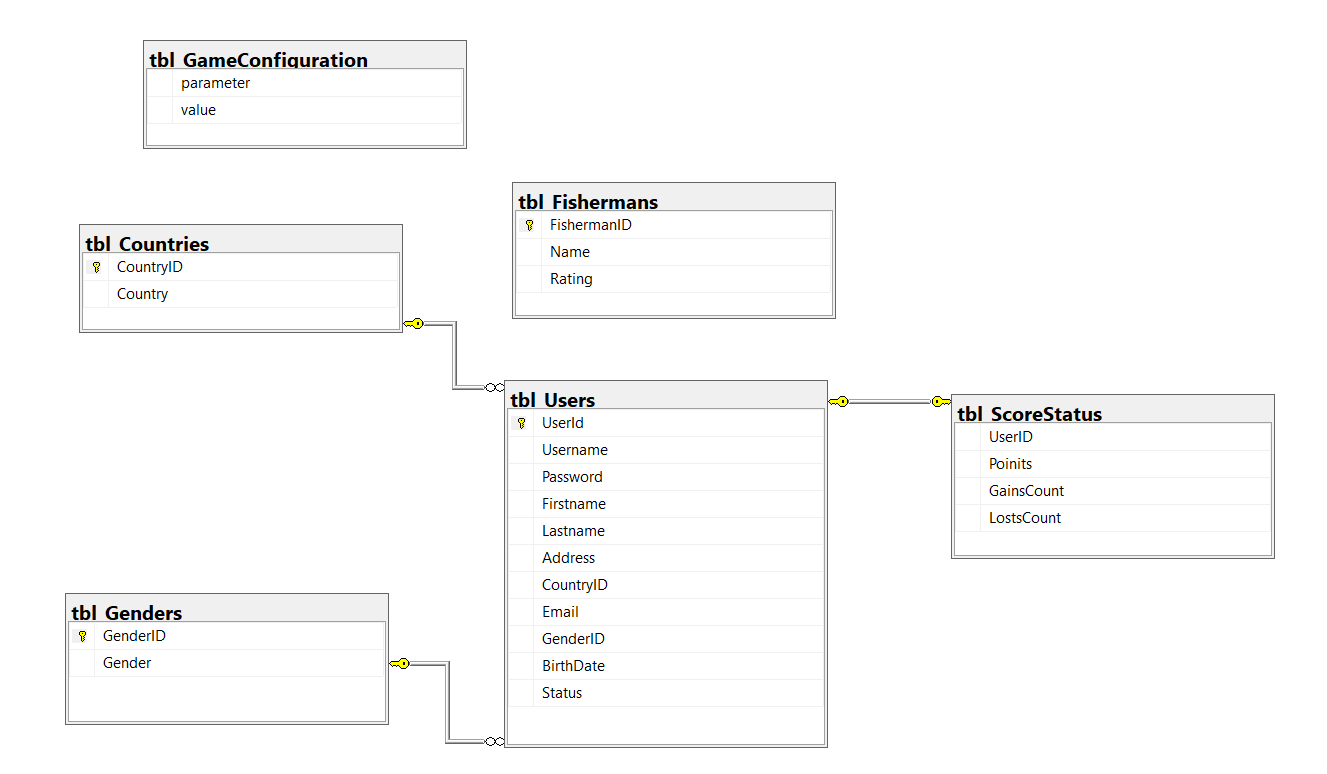
The final solution is reflected by implementation and testing the Integrity of all the necessary backstage processes by running and showing the game in SSMS, using the SQL language.

In order to make the best programmability and getting the best performance, we used SQL tables, Stored Procedures, Functions, Views, T-SQL statements, and every command that use in our need.

**Game Description**:

* There is 3 fish swimming in the sea. The fish’s amount is the same as the fisherman’s amount. Each fisherman owner of only one fish.
* There are 10 rounds, in each round, the fish can swim randomly forward. The rounds amount is configurable.
* In each round, the player can see the movement of each fish, and the fish itself.
* The fish swim ends as soon as one of the fish is caught in a cage - the fish is caught in the 10th round, randomly.
* The player selects a fisherman from an existing list.
* The game is built on randomness that in each play a round fisherman catches a fish.
* The player who chooses the name of the fisherman who captured the fish, wins and gets a score. However, if the player chooses a losing fisherman, he gets a negative score.

# Database Model - ERD

The following is an Entity Relationship Diagram:

**Our model contains two kinds of tables:**

1. Changing tables: they will be updated/changed throughout the relevant process. The database within will change depending on the input and the output by users' actions, and the results of their score. This kind of data table is to be able to analyze and monitor the history of the game.
2. Permanent tables: Will not change regarding user's use. These are only auxiliary tables to normalize the workspace.

# Tables Description

To create and design the needed tables for the database model, we used a Data Definition Language (DDL) and a Data Manipulation Language (DML).

In the following description you can see the tables we created for the database model. You can see the column names of each table, the data type of the column and its constraints.

## tbl\_Users

|  |  |  |
| --- | --- | --- |
| tbl\_Users | | |
| **Column** | **Data Type** | **Constrains** |
| **UserId** | **Int** | **PK Identity (1,1)** |
| Username | nvarchar(40) | Uniq, not null |
| Firstname | nvarchar(40) | Not null |
| Lastname | nvarchar(40) | Not null |
| Address | nvarchar(40) |  |
| CountryID | int | FK |
| Email | nvarchar(40) | Uniq, not null |
| GenderID | int | FK |
| BirthDate | Date |  |
| Status | bit | Default 0 |

* The table contains all of the users’ registration information.
* That table has a Foreign Key relationship to tbl\_Countrie and to tbl\_Ganders.
* Status refers to login status of the user. This is a Boolean expression, if the user did a login to the game the status will be 1. If the user logout of the game the status will be 0.

## tbl\_Countries

|  |  |  |
| --- | --- | --- |
| tbl\_Countries | | |
| **Column** | **Data Type** | **Constrains** |
| **CountryID** | **Int** | **PK Idntity (1,1)** |
| Country | Nvarchar(40) | Uniq |

* This is a Permanent table. The table contains the name of the country that the user selects in the registration processes.
* That table has a Foreign Key relationship to “Tbl\_Users”.

## tbl\_Ganders

|  |  |  |
| --- | --- | --- |
| tbl\_Genders | | |
| **Column** | **Data Type** | **Constrains** |
| **GenderID** | **int** | **PK Identity (1,1)** |
| Gender | nvarchar(10) | Uniq, not null |

* This is a Permanent table. The table contains the gander option in the registration process, male or female.
* That table has a Foreign Key relationship to “Tbl\_Users”.

## tbl\_Fisherman

|  |  |  |
| --- | --- | --- |
| tbl\_Fisherman | | |
| **Column** | **Data Type** | **Constrains** |
| **FishermanID** | **Int** | **PK Identity (1,1)** |
| Name | Nvarchar(40) | Not null |
| Rating | Int | Default 0 |

* This is Permanent table. The table contains the names of the fishermen the user can choose to play the game.
* There are three fishermen whose name is predefined in the game.
* The user must choose the name of the fisherman who will catch the fish.
* Each time game is played, another fisherman wins randomly. That is possible by using a function that we have created - “fn\_Rand” (see description function No.1).
* The rating column refers to the rating that the fisherman accumulates according to the number of times he catches the fish.

## tbl\_ScoreStatus

|  |  |  |
| --- | --- | --- |
| tbl\_ScoreStatus | | |
| **Column** | **Data Type** | **Constrains** |
| **UserID** | **int** | **FK** |
| Points | int | default 1000 |
| GainsCount | int | default 0 |
| LostsCount | int | default 0 |

* This table is dynamic, the variables updating according to the score of the users in the game.
* The user gets a bonus of 1,000 points in-game registration. If a user wins, his points will be credited with +100 extra points. However, if he loses by choosing the fisherman who did not catch the fish, 100 reduce from his balance score.

## tbl\_GameConfiguratin

|  |  |  |
| --- | --- | --- |
| tbl\_GameConfiguration | | |
| **Column** | **Data Type** | **Constrains** |
| parameter | nvarchar(20) | not null |
| value | nvarchar(10) | not null |

* This table stores the parameters in the game. The table allows creating the visibility of the game.
* There are three parameters in that table:
  + Number of rounds in each play – 10 rounds (“sets” as it defined in the table).
  + Left window border - 1
  + Right window border – 60

# Custom Data Types

We created a Data Type called ct\_FishLocation to store the location of each fish in each round in the game. The fish’s location changes regards to game flow (see full syntax in appendix).

The custom data type has table definition:

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constrains** |
| FishNumber | int | unique |
| Location | int | default 0 |

# User Define Functions (UDF)

The following section explains the functions we created for the database model. In the description of each function will be present the purpose and the functionality of that function. Also, the input and the output of the function. The full syntax of all the functions described below, will show in a script attached to that pepper.

## fn\_Rand

1. * 1. To simplify the syntax of the function we created a View as an auxiliary object that contain a “rand”.
     2. Input: Void
     3. Output: A random number between 0 and 1.
     4. See full syntax in appendix.
     5. The uses of the fn\_Rand function as the following:

* To create a randomness process of the winning fisherman who catches the fish every time the game is played. That is happening in the 10th round of the game. Then the fish, whom captured, also randomly, will appear in a cage.
* To create a random movement of the three fish in the sea in each round.
* When the user trying to register with an existing user name, it will print as a massage with a suggestion for a new random user name.
  + 1. Input: tow values -
* Upper, refers to the higher of the two numbers that need to enter into the function.
* Lower, Upper, refers to the lower of the two numbers that need to enter into the function.
  + 1. Output: random number between the upper and lower values.
    2. See full syntax in appendix

## fn\_drawFish

1. * 1. This function created to draw the fish symbol in the game.
     2. Input: Void
     3. Output: ' ><> '
     4. See full syntax in appendix.

## fn\_updateFishermanRating

1. * 1. The functionality of that function is to update the rating of the fisherman in the fisherman table (tbl\_Fisherman), according to the results after each play made by the user.
     2. Input: fishermanID
     3. Output: The new rating of the wining fisherman.
     4. See full syntax in appendix

## fn\_ValidEmail

1. * 1. The purpose of creating this function is to make sure that the user registered with a correct mail address.
     2. Input: the email the user enters at the time of registration.
     3. Output: Boolean expression:

* 0 = Valid
* 1 = Invalid
  + 1. See full syntax in appendix.

## fn\_checkAge

1. * 1. The purpose of creating this function is to enforce the specified minimum age (13) for which the user allowed to register.
     2. As part of the functionality of the function, it taken into account the birth dates that are on the edges, in monthly terms. For example; if the user typing a date of birth and it has appeared he 12.8 years old, that is to say he, born at the end of the month. The engine will not let him to proceed the registration process to the game.
     3. Input: birthdate of the user
     4. Output:

* 0 = Valid (Over 13)
* 1 = Invalid (Under 13)
  + 1. See full syntax in appendix.

# Stored Procedures

The stored Procedures allows the user to play the game and run all of the backstage parameters of the game. In addition, the Procedures allows making all the changes in the database dooring the game and its registration process.

## sp\_SelectFishermanID

1. * 1. The procedure presents the rating’s information of each fisherman.
     2. Regards to this information the user can select a fishermanID.
     3. Input: Void
     4. Output: Void
     5. See full syntax in appendix.

## sp\_RunOneSet

1. * 1. The purpose of this procedure is to run one round out of 10 in single game.
     2. Input:

* Table form, from of the Custom Data Type - *ct\_FishLocation.*
* Set type:
* 0 – general – all the rounds 1-9 (except round 10).
* 1 – final– the last round in single game, round number 10.
  + 1. Output:
* When set type is 0, shows one round with a random fish location in the sea.
* When set type is 1, shows the final round with the result of the winning fisherman and the fish he captures (out of three existing fish), framed as caught in a cage.
  + 1. See full syntax in appendix.

## sp\_RunAlleSets

* + 1. The purpose of this procedure is to run all of the rounds in a single game at ones, from the first round till the last.
    2. Input: Void
    3. Output: The FishermanID winner.
    4. See full syntax in appendix.

## sp\_RunFishermanGame

* 1. 1. The purpose of this procedure is to execute the game from A to Z.
     2. This is the main Stored Procedure of the game engine.
     3. It's also updating the final result in the game for individual user.
     4. Additionally, the procedure runs a multiple test:
* Test whether the user login or logout.
* Test whether the user have enough point to play in the game.
* Test of the final result in the game and printing the gained score (+100/-100).
  + 1. Input: Tow parameters -
* UserDI of the user plying at that moment.
* The FishermanID selected by that user.
  + 1. Output: Void
    2. See full syntax in appendix.

## Check Username - sp\_If\_username\_exist

* + 1. This procedure is checks whether the user name that the player typed already exists in the database.
    2. Input: username
    3. Output:
* 0 = Valid (Exists)
* 1 = Invalid (Not exists)
  + 1. See full syntax in appendix .

## sp\_checkpassword

* 1. 1. The purpose is to check if the password that the user is trying to set for his account meets the defined conditions of a strong password that allowing to registration the game.
     2. The defined conditions for a strong password:
     3. Password length cannot be less than 7 characters.
     4. The password must contain a combination of lowercase letters, capital letters and numbers.
     5. A strong password must contain at least one lowercase letter, at least capital letter and at least one number.
     6. The password cannot be the same as the username.
     7. The password cannot be the word “password”.
     8. Input: Password
     9. Output:
* 0 = Valid (Strong password)
* 1 = Invalid (The password didn’t past one of the conditions above)
  + 1. See full syntax in appendix.

## Registration - sp\_UserAdd

* 1. 1. This is the procedure to execute in order to registering into the “Catch The Fish” game. The procedure checks that the user inserted correct data that meet the defined conditions off all the parameters in the database model, that are needed for the registering. As a result, the functionality of this Stored Procedure is to run multiple tests:
* Check username – This test is configured that if the user trying to registered with already exist username, the system will offer him an alternate user name, using the random function (fn\_Rand) that implemented here.
* Check Password
* Check Email
* Check Minimum Age
  + 1. Input: all the parameters as the following:
* Username
* Password
* First name
* Last name
* Address
* Country
* Email
* Gender
* Birthdate
  + 1. Output:
* The procedure output is to feels in all the data from the parameters above into the users table (tbl\_Users).
* Additionally, this procedure feels in the relevant data into the score table.
* See full syntax in appendix.

## sp\_Login

* + 1. The purpose of the sp\_login is to change the user status in the users table (tbl\_Users), when the user login into the game.
    2. Input: Username
    3. Output: Changing the “status” column in the users table to “1”.
    4. See full syntax in appendix.

## sp\_Logout

* 1. 1. The purpose of the sp\_logout is to change the user status in the users table (tbl\_Users), when the user logout from the game.
     2. Input: Username
     3. Output: Changing the “status” column in the users table to “0”.
     4. See full syntax in appendix.

# Appendix

|  |
| --- |
| create database FishingGameDB go  use FishingGameDB go   -- Tables  -- tbl\_Fishermans create table tbl\_Fishermans (  FishermanID int Identity (1,1) Primary Key,  [Name] nvarchar(40) not null,  Rating int default 0 )  insert into tbl\_Fishermans (Name) values ('Avi') insert into tbl\_Fishermans (Name) values ('Moshe') insert into tbl\_Fishermans (Name) values ('Shir')      -- tbl\_Countries create table tbl\_Countries (  CountryID integer Identity (1,1) Primary Key,  Country nvarchar(40) unique )  insert into tbl\_Countries values('France') insert into tbl\_Countries values('Germany') insert into tbl\_Countries values('Ireland') insert into tbl\_Countries values('Italy') insert into tbl\_Countries values('Mexico') insert into tbl\_Countries values('Norway') insert into tbl\_Countries values('Poland') insert into tbl\_Countries values('Portugal') insert into tbl\_Countries values('Spain') insert into tbl\_Countries values('Sweden') insert into tbl\_Countries values('Switzerland') insert into tbl\_Countries values('UK') insert into tbl\_Countries values('USA') insert into tbl\_Countries values('Venezuela') insert into tbl\_Countries values('Israel')    -- tbl\_Genders create table tbl\_Genders (  GenderID integer Identity (1,1) Primary Key,  Gender nvarchar(10) unique not null )  insert into tbl\_Genders values('male') insert into tbl\_Genders values('female')   -- tbl\_GameConfiguration create table tbl\_GameConfiguration (  parameter nvarchar(20) not null,  [value] nvarchar(10) not null )  insert into tbl\_GameConfiguration values ('num\_of\_sets', '10') insert into tbl\_GameConfiguration values ('left\_win\_border', '1') insert into tbl\_GameConfiguration values ('right\_win\_border', '60')  go  -- tbl\_Users create table tbl\_Users (  UserId int Identity (1,1) Primary Key,  Username nvarchar(40) unique not null,  [Password] nvarchar(40) not null,  Firstname nvarchar(40) not null,  Lastname nvarchar(40) not null,  [Address] nvarchar(40),  CountryID integer foreign key references tbl\_Countries(CountryID),  Email nvarchar(40) unique not null,  GenderID integer foreign key references tbl\_Genders(GenderID),  BirthDate date,  [Status] bit default 0 )  go  -- tbl\_ScoreStatus create table tbl\_ScoreStatus (  UserID integer unique NOT NULL Foreign key references tbl\_Users(UserId),  Poinits integer default 1000,  GainsCount integer default 0,  LostsCount integer default 0 )  go  -- Custom Data Types create type ct\_FishLocation as table (  FishNumber int unique,  [Location] int default 0 ) go  -- Functions  -- fn\_Rand create view [dbo].[vv\_getRandValue] as select rand() as [value]  go  Create function [dbo].[fn\_Rand](@Lower int, @Upper int) returns int as Begin  DECLARE @Random INT;  if @Upper > @Lower  SELECT @Random = (1 + @Upper - @Lower) \* (SELECT Value FROM vv\_getRANDValue) + @Lower  Else  SELECT @Random = (1 + @Lower - @Upper) \* (SELECT Value FROM vv\_getRANDValue) + @Upper  return @Random end  go  -- fn\_drawFish create function fn\_drawFish() returns nvarchar(40) as begin  declare @pic\_fish nvarchar(40) = ' ><> '   return @pic\_fish end  go  -- fn\_updateFishermanRating create function fn\_updateFishermanRating(@fisherman\_id int) returns int as begin  declare @current\_rating int = (select rating from tbl\_Fishermans where FishermanID = @fisherman\_id)  declare @new\_rating int = @current\_rating + 1   return @new\_rating end  go  -- fn\_ValidEmail  Create Function fn\_ValidEmail(@Email varchar(40)) Returns bit as Begin    Declare @bitRetVal Bit   If (@Email <> '' and @Email not like '\_%@\_\_%.\_\_%')   Set @bitRetVal = 1 -- Invalid   Else  Set @bitRetVal = 0 -- Valid     Return @bitRetVal End  go  -- fn\_checkAge  Create Function fn\_checkAge (@birthdate date) Returns int Begin   declare @years int  declare @months int  declare @days int  declare @revstatus int   Set @years = DATEDIFF(yy, @birthdate, getdate())   Set @months = DATEDIFF(mm, @birthdate, getdate()) % 12     if MONTH(@birthdate) > MONTH(getdate())  Set @years = @years - 1   Else  Begin  if MONTH (@birthdate) = MONTH(getdate())  if DAY(@birthdate) > DAY(getdate())  Begin  Set @years = @years - 1  Set @months = @months - 1  End  End   if (@years < 13)  set @revstatus = 1  else  set @revstatus = 0   return @revstatus End  go  -- Procedures  -- sp\_SelectFishermanID create procedure sp\_SelectFishermanID as  select 'Please select FishermanID from following table:' as [Message], \* from  (select \* from tbl\_Fishermans) t  go  -- sp\_RunOneSet create procedure sp\_RunOneSet  @t\_fl [dbo].[ct\_FishLocation] readonly,  @set\_type int as -- Set type 0 - general, Set type 1 - finally  begin  declare @set\_result int = 0  declare @pic\_fish nvarchar(40)  declare @pic\_fish\_with\_location nvarchar(300)  declare @fish\_location int  declare @catched\_fish\_number int  declare @i int = 1    declare @fishermans\_count int = (select count(\*) from tbl\_Fishermans)    set @catched\_fish\_number = dbo.fn\_Rand(1, @fishermans\_count)    while (@i <= @fishermans\_count)  begin  set @pic\_fish\_with\_location = ''  set @fish\_location = (select [Location] from @t\_fl where FishNumber = @i)  set @pic\_fish = (select dbo.fn\_drawFish())  if (@set\_type = 0)  set @pic\_fish\_with\_location = space(@fish\_location) + @pic\_fish + CHAR(10)  else  begin  if (@catched\_fish\_number = @i)  begin  set @set\_result = @catched\_fish\_number  set @pic\_fish\_with\_location =   space(@fish\_location) + '-------' + CHAR(10) +  space(@fish\_location) + '|' + @pic\_fish + '|' + CHAR(10) +  space(@fish\_location) + '-------'  end  else  set @pic\_fish\_with\_location = space(@fish\_location) + @pic\_fish + CHAR(10)  end  print @pic\_fish\_with\_location  set @i = @i + 1  end  print '--------------------------------------------------------------------'   return @set\_result end  go  -- sp\_RunAllSets create procedure sp\_RunAllSets as begin  declare @i int = 1  declare @j int = 1  declare @new\_fish\_location int  declare @fisherman\_winner\_id int = 0  declare @fisherman\_winner\_name nvarchar(40)  declare @fisherman\_winner\_rating int  declare @sets\_count int =   convert(int,   (select [value] from tbl\_GameConfiguration where parameter = 'num\_of\_sets'))   declare @left\_win\_border int = convert(int,   (select [value] from tbl\_GameConfiguration where parameter = 'left\_win\_border')  )   declare @right\_win\_border int = convert(int,   (select [value] from tbl\_GameConfiguration where parameter = 'right\_win\_border')  )   declare @fishermans\_count int = (select count(\*) from tbl\_Fishermans)  declare @step\_size int = (@right\_win\_border - @left\_win\_border) / @sets\_count  declare @step\_jump int   declare @t\_fishlocation as [dbo].[ct\_FishLocation]   while (@j <= @fishermans\_count)  begin  insert into @t\_fishlocation values(@j, 0)  set @j = @j + 1  end    print '--------------------------------------------------------------------'  while (@i <= @sets\_count)  begin  print 'Round Number ' + convert(nvarchar(5), @i)  set @j = 1  while ( @j <= (select count(\*) from @t\_fishlocation) )  begin  set @step\_jump = dbo.fn\_Rand(1, @step\_size) + 2  set @new\_fish\_location = ((select [Location] from @t\_fishlocation where FishNumber = @j) + @step\_jump)  update @t\_fishlocation set [Location] = @new\_fish\_location where FishNumber = @j  set @j = @j + 1  end  if (@i = @sets\_count)  begin  print 'This is final round'   exec @fisherman\_winner\_id = sp\_RunOneSet @set\_type = 1, @t\_fl = @t\_fishlocation  set @fisherman\_winner\_name = (select [name] from tbl\_Fishermans where FishermanID = @fisherman\_winner\_id)  print 'The fisherman winner is: ' + @fisherman\_winner\_name  set @fisherman\_winner\_rating = (select dbo.fn\_updateFishermanRating(@fisherman\_winner\_id))  update tbl\_Fishermans set rating = @fisherman\_winner\_rating where FishermanID = @fisherman\_winner\_id  print 'The new rating of ' + @fisherman\_winner\_name + ' is ' + convert(varchar, @fisherman\_winner\_rating)  print ''  end  else  exec @fisherman\_winner\_id = sp\_RunOneSet @set\_type = 0, @t\_fl = @t\_fishlocation  set @i = @i + 1   end   return @fisherman\_winner\_id  end  go  -- sp\_RunFishermanGame create procedure sp\_RunFishermanGame  @user\_id int,  @fisherman\_id int as  begin  SET NOCOUNT ON    declare @fisherman\_winner\_id int  declare @current\_points int  declare @login\_status bit   set @current\_points = (select [Poinits] from tbl\_ScoreStatus where UserID = @user\_id)  if (@current\_points <= 0)  begin  throw 50001, 'You dont have enougth points for play this game, Please, contact with game administrator for fixing this issue', 1  end   set @login\_status = (select [Status] from tbl\_Users where UserID = @user\_id)  if (@login\_status = 0)  throw 50007, 'Please, make login before playing the game', 1   exec @fisherman\_winner\_id = sp\_RunAllSets  if (@fisherman\_winner\_id = @fisherman\_id)  begin  print 'Congratulation !!! You win 100 points!!!'  update tbl\_ScoreStatus set [GainsCount] = (  (select [GainsCount] from tbl\_ScoreStatus where UserID = @user\_id) + 1)  where UserID = @user\_id   update tbl\_ScoreStatus set [Poinits] = (  (select [Poinits] from tbl\_ScoreStatus where UserID = @user\_id) + 100)  where UserID = @user\_id  end  else  begin  print 'You lost. Unfortunately, you lost 100 points.'   update tbl\_ScoreStatus set [LostsCount] = (  (select [LostsCount] from tbl\_ScoreStatus where UserID = @user\_id) + 1)  where UserID = @user\_id   update tbl\_ScoreStatus set [Poinits] = (  (select [Poinits] from tbl\_ScoreStatus where UserID = @user\_id) - 100)  where UserID = @user\_id  end   set @current\_points = (select [Poinits] from tbl\_ScoreStatus where UserID = @user\_id)  print 'You have ' + convert(varchar, @current\_points) + ' points' end  go  -- Check Username  create procedure sp\_If\_username\_exist  @user nvarchar(40) As Begin  declare @status bit   If ( (select count(\*) from tbl\_Users where Username = @user )> 0 )  set @status = 1   Else   set @status = 0    return @status End  go  -- sp\_checkpassword create procedure sp\_checkpassword   @pass varchar(40),  @user nvarchar(40) as  begin  declare @retval int     begin  if (len(@pass)< 7 or patindex('%[0-9]%', @pass ) <=0 or patindex('%[a-z]%' collate Latin1\_General\_Bin, @pass) <= 0 or PATINDEX('%[A-Z]%' collate Latin1\_General\_Bin, @pass ) <= 0)  set @retval = 1  else  set @retval = 0    end   if (@pass = @user)  begin  print 'The password can not be same as username'  set @retval = 1  end   if (@pass = 'password')  begin  print 'The password can not be word "password"'  set @retval = 1  end   return @retval end  go  -- Registration   Create procedure sp\_UserAdd  @Username nvarchar(40),  @Password nvarchar(40),  @Firstname nvarchar(40),  @Lastname nvarchar(40),  @Adress nvarchar(40),  @Country nvarchar(40),  @Email nvarchar(40),  @Gender nvarchar(10),  @Birthdate nvarchar(30) as  begin  declare @user\_prefix int  declare @new\_username nvarchar(40)  declare @if\_user\_exists bit  declare @password\_status int  declare @email\_status bit  declare @age\_status bit  declare @CountryID int  declare @GenderID int   set @Birthdate = CONVERT(date, @Birthdate)  set @CountryID = (select CountryID from tbl\_Countries where Country = @Country)  set @GenderID = (select GenderID from tbl\_Genders where Gender = @Gender)   -- Check username  exec @if\_user\_exists = sp\_If\_username\_exist @user = @Username  set @new\_username = @username  while (@if\_user\_exists = 1)  begin  print 'The user ' + @Username + ' already exists'  set @user\_prefix = dbo.fn\_Rand(1, 999)  set @new\_username = @Username + convert(varchar, @user\_prefix)  print 'The new username is ' + @new\_username  exec @if\_user\_exists = sp\_If\_username\_exist @user = @new\_username  end  set @username = @new\_username    -- Check Password   exec @password\_status = sp\_checkpassword @pass = @Password, @user = @Username  if (@password\_status = 1)  begin  throw 50002, 'The password is not correct !!! Please type new password', 1  end   --Check Email    set @email\_status = (select dbo.fn\_ValidEmail(@Email))  if (@email\_status = 1)  begin  throw 50003, 'The email is not valid !!! Please type new email', 1  end    --Check Minimum\_Age   set @age\_status = (select dbo.fn\_checkAge(@Birthdate))  if (@age\_status = 1)  begin  throw 50004, 'You are too young to play!! Try again in a few years', 1  end    Insert into tbl\_Users ( Username, Password, Firstname, Lastname, [Address], CountryID, Email, GenderID, BirthDate )  Values (@Username, @Password, @Firstname, @Lastname, @Adress, @CountryID, @Email, @GenderID, @Birthdate )   insert into tbl\_ScoreStatus (UserID) select UserId from tbl\_Users where Username = @Username   end  go  -- sp\_Login create procedure sp\_Login   @username nvarchar(40) as begin  declare @login\_status bit   set @login\_status = (select [Status] from tbl\_Users where Username = @username)   if (@login\_status = 1)  throw 50005, 'The user already login. Please, make logout begore login', 1  else  begin  update tbl\_Users set [Status] = 1 where Username = @username  print 'The user ' + @username + ' made login succefully'  end end  go  -- sp\_Logout create proceCREATE TABLE "topic" (  "id" serial NOT NULL PRIMARY KEY,  "forum\_id" integer NOT NULL,  "subject" varchar(255) NOT NULL ); ALTER TABLE "topic" ADD CONSTRAINT forum\_id FOREIGN KEY ("forum\_id") REFERENCES "forum" ("id");  -- Initials insert into "topic" ("forum\_id", "subject") values (2, 'D''artagnian'); dure sp\_Logout  @username nvarchar(40) as begin  declare @login\_status bit   set @login\_status = (select [Status] from tbl\_Users where Username = @username)   if (@login\_status = 1)  begin  update tbl\_Users set [Status] = 0 where Username = @username  print 'The user ' + @username + ' made logout succefully'  end  else  throw 50006, 'The user already logged out.', 1 end |