|  |
| --- |
| **Player Auction system**  **Capabilities tested:**   * Use C# Programming to implement business layer of your application * Analyze complex business scenarios and create a data model * Use SQL Server to build SQL engine for an application * Automate the build using MSBuild Tool * Test, and Debug your application to make application ready for deployment |

**Problem Statement**

Design and implement an application program to automate the process of auctioning for cricket trophy. The application stores a collection of players and provides mechanism of displaying players details based on team name.

**The main menu contains following options:**

1. **Add a player**
2. **Display players**
3. **Exit**

It is redisplayed after each operation is completed, except for the **‘Exit’** option. The details for each option are specified in the rest of the document.

Your application program should follow the control flow as, start from class **PlayerAuctionSystemClient -> PlayerAuctionSystemManager->DAO class -> Database**

**Note:**

1. Create following three tables **‘Team’, ‘Player’** and **‘Team\_Player’** which will be used in the application. For Table **‘Team’**, manually insert the values as per values given in this document. The values for **‘Player’** and **‘Team\_Player’** must be inserted through your application.
2. Read the complete question before coding
3. **Upload your table script with inserted values along with project solution.**
4. All user defined exceptions must be created under Assessment.Exception project
5. All exceptions must be handled in Manager Class. DAO class should not handle any exception; it must throw it to Manager Class.
6. Loose coupling: Create a separate Dao interfaces and implementation classes. Database interaction code should be only in DAO classes. Dao classes should not have UI code.
7. **Write Nunit test cases for ‘PlayerAuctionSystemManager’ class**

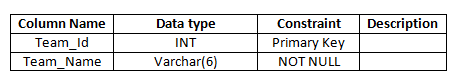
Write Nunit methods in NunitTest class under Assessment.Nunit project

* Create test case methods with exactly the same names given in the table below with appropriate annotations.
* The test case method naming convention is as follows:
  + The test case testing valid data is **testFunctionalityNameValidData**
  + The test case testing a particular invalid attribute is **testFunctionalityNameInvalidAttributeName or testFunctionalityNameNoRecords**

1. **Database design:**

**Create database “Player\_DB” and create the following three tables used for the application: “Team”, “Player” and a link table “Team\_Player”. Listed below is the table design with constraints.**

Table Name: **Team**

****

Below is the content of table **‘Team’,** which must be manually inserted into table**:**

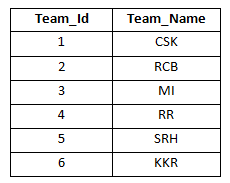
****

Table Name: **Player**

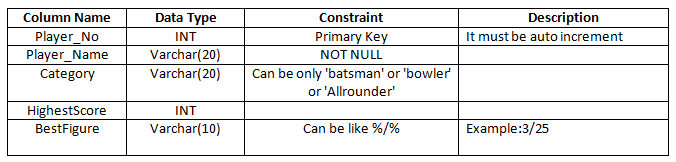
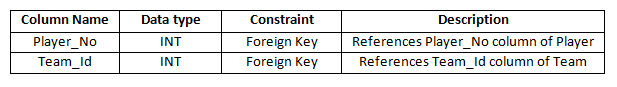
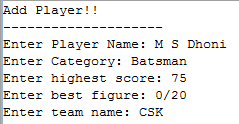
****

Table Name: **Team\_Player**

****

1. **Add a Player**

* When user selects option 1, the user will enter the **playerName, category, highestScore, bestFigure** and **teamName.**
* Sample output shown below.



Write code to display the output in below format by using the returned value if it is successfully saved in the database, otherwise display an appropriate exception/error message.



Your **PlayerAuctionSystemManager class** code should check for the business rules given below and throw appropriate exception, when exception thrown corresponding message should be displayed to user, which is as given in the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Rule No** | **Business constraint** | **User defined exceptions to be thrown** | **Message to user to be displayed** |
| 1 | **category** entered should be either 'Batsman','Bowler' or 'Allrounder' | If **category** is invalid, throw **InvalidCategoryException** | “Invalid category name please check your input” |
| 2 | **teamName** entered should be present in the database. | If **teamName** is invalid, throw **InvalidTeamNameException** | “Invalid team name, please check your input” |
| 3 | For a batsman**, highestScore** should be between 50 and 200 inclusive | If **category** is 'Batsman' and **highestScore** is invalid, throw **NotABatsmanException** | “Invalid Batsman, please check your input” |
| 4 | For a bowler, **bestFigure** should not be null and **highestScore** should not be negative | If category is 'Bowler' and **bestFigure** are invalid, throw **NotABowlerException** | “Invalid Bowler, please check your input” |
| 5 | Same player name for given category & team should not be repeated | If player name for given category & team is already exist in database, then it should throw **DuplicateEntryException**.  For example: If input player name is M S Dhoni, category is ‘Batsman’ and team name is ‘CSK’, if another player with same name, category, and team is already exist in database, then it must throw DuplicateEntryException | “Player details already exist in the database” |

Test the manager class by using sample data in the table given below corresponding for each of the test case.

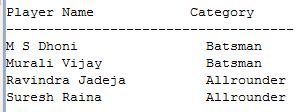


1. **Display Players:**

* When user selects Option 2, user will input **teamName**.
* Sample output shown below.



Write code to display the output in below sorted order (**sort by** **player’s name**) by using the returned values if the business rules are satisfied; otherwise display the appropriate exception message.



Your **PlayerAuctionSystemManager class** code should check for the business rules given below and throw appropriate exception, when exception thrown corresponding message is displayed to user, which is as given in the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Rule No** | **Business constraint** | **User defined exceptions to be thrown** | **Message to user to be displayed** |
| 1 | **teamName** entered should be present in the database. | If **teamName** is invalid, throw **InvalidTeamNameException** | “Invalid team name, please check your input” |

Test the manager class by using sample data in the table given below corresponding for each of the test case.



**4. Exit**

When user selects option 3, application should terminate

**Notes:**

1. **Provide required properties in entity classes**
2. **“Assessment.Dao” project should contain all the database interaction code. It should implement the interface IDataAccess**
3. **Assessment.Nunit.NunitTest class should provide methods to test business layer methods**
4. **Provide required methods in the AutionManager class**
5. **Provide custom exceptions classes in CustomExceptions project**
6. **“PlayerAuctionSystemClient” contains main () method, complete the code to interact with Manager Class.**