Score table

"Score table" task description

This information system, created using the **IsFusion Platform**, should contain the functionality for keeping score at a hockey tournament.

A tournament is understood to mean a subset of games between teams (with two teams participating in each game), which result in points being awarded to the teams.

In each game, the result of each game can be a regular-time win of one of the teams (the winning team receives 3 points), an overtime win (the winning team receives 2 points, while the losing team receives 1), or a penalty shootout win (the winning team receives 2 points, while the losing team receives 1).

A team's score table ranking is determined by the total number of points. In case of a tie, additional parameters are considered: number of regular-time victories, number of overtime victories, number of penalty shootout victories, the difference between scores and misses, number of scores. Additional parameters for determining the final ranking are applied consecutively in a specified order until a status is achieved in which the results of the teams are uniquely ranked.

Defining the domain logic

Module declaration

We declare a module within which the required functionality will be implemented. We assign an arbitrary name to the module (for example, HockeyStats).

```
MODULE HockeyStats;
```

We define the use of functionality from other modules in the HockeyStats module. In particular, we need the system module **System**, in which some system elements used in the example are declared.

```
REQUIRE System, Utils;
```

Team definition

We introduce the concept of a team, for which we create a separate class using the corresponding instruction CLASS.

```
CLASS Team 'Team';
```

We assign a name to the class created (for example, "Team"), which will subsequently be used when building expressions, as well as a caption to display on custom forms (for example, "Team").

So that all teams can be easily identified when working with forms created later, we create a name for the team. In other words, we create a "Name" propert
that can be defined for objects of the "Team" class.

```
name 'Team name' = DATA STRING[30] (Team) IN base;
```

Thus, the team name is a data (user-entered) string-type property. Using the IN option, the created property is added to the predefined **base** property group . Object properties belonging to the **base** group will be automatically displayed on the dialog form for selecting an object of the "Team" class.

Game definition

We introduce the concept of the game and its attributes: date, participants (host team and guest team), and their names.

```
CLASS Game 'Game';

date 'Date' = DATA DATE (Game);
hostTeam = DATA Team (Game);
guestTeam = DATA Team (Game);
hostTeamName 'Hosts' (Game game) = name(hostTeam(game));
guestTeamName 'Guests' (Game game) = name(guestTeam(game));
```

The hostTeam and guestTeam properties are data object properties of a game, whose values are links to the host team and guest team, respectively (that is, to specific Team-class objects). Properties of the team names of the game hosts and guests (hostTeamName and guestTeamName) are created for subsequent use on forms. If the hostTeam and guestTeam properties are added to the form, the user will see the internal IDs of objects from the database.

We introduce the constraint that the game participants must be two different teams.

```
CONSTRAINT hostTeam(Game team) = guestTeam(team) CHECKED BY hostTeam, guestTeam MESSAGE 'Host and guest teams must be different';
```

The operating mechanism of this expression is as follows: when the host team or guest team of a game changes, the system checks the condition of equality of these teams (hostTeam(team) == guestTeam(team)), and if it is met the system blocks the application of changes to the database, and also gives the user the specified message ('Host and guest teams must be different'). In other words, the result of the expression specified after the CONSTRAINT operator must be NULL. In all other cases the restriction will be considered violated. In addition, thanks to the CHECKED BY block, the created constraint will filter teams when selecting a home team or a guest team for a game (that is, it will exclude the team already set as the opponent from the list of teams in the dialog that appears upon selecting a team).

We define the number of goals scored by each team during the game.

```
hostGoals 'H goals' = DATA INTEGER (Game);
guestGoals 'G goals' = DATA INTEGER (Game);
```

The defined properties use the INTEGER type, since the number of goals scored by each team is an integer.

We introduce the constraint that the game cannot end in a tie. The system should prohibit the user from setting an identical number of goals for both teams in the game, and issue a message with the specified text.

```
CONSTRAINT hostGoals(Game game) = guestGoals(game) MESSAGE 'The game cannot end in a draw';
```

Determining the winner of the game

We determine the winner of the game - the team that has scored more goals than its opponent.

```
winner(Game game) = IF hostGoals(game) > guestGoals(game)
                    THEN hostTeam(game)
                    ELSE guestTeam(game);
```

Here we use the operator IF... THEN... ELSE, which checks the condition that the host team in this game has scored more goals than the guest team. If it is met, the winner is the home team; if not, the guest team.

By a similar principle, the team that participated in the game and scored fewer goals than its opponent will be considered the loser.

```
looser(Game game) = IF hostGoals(game) > guestGoals(game)
                    THEN guestTeam(game)
                    ELSE hostTeam(game);
```

Determining game result

We introduce the concept of the possible game result with a predefined set of values: regular-time win, overtime win, penalty shootout win.

```
CLASS GameResult 'G/R' {
    win 'W',
    winOT 'LO',
    winSO 'LB'
}
```

For this purpose we create a GameResult class and add three static objects to it that are specified using expressions specified in braces { }. In this case, the values win, winOT, winSO and W, OW, SW will be stored in the system properties staticName and staticCaption, respectively.

We create the resultName property, which will return the caption of the game result (W, OW, or SW). To do this, we take the system property staticCaption, which is supported for all objects in the system, and constrain its signature using the IF operator, indicating that the object must be of the Game class. This property is added to the "base" property group so that it appears in the automatic dialog for selecting an object of the GameResult class.

```
resultName 'Name' (GameResult game) = staticCaption(game) IF game IS GameResult IN base;
```

We determine the result of a particular game. In case when one of the teams won by 2 or more goals, the game result is considered a regular-time win. If not, and only if not, the game result (the type of win for a given score) will be set by the user. However, the user cannot set a regular-time win as the game result.

```
userResult = DATA GameResult (Game);
result (Game game) = OVERRIDE userResult(game),
    (GameResult.win IF ((hostGoals(game) (-) guestGoals(game)) > 1 OR (guestGoals(game) (-) hostGoals(game)) > 1));
resultName 'G/R' (Game game) = resultName(result(game));

CONSTRAINT ((hostGoals(Game game) (-) guestGoals(game)) > 1 OR (hostGoals(game) (-) guestGoals(game)) < -1) AND userResult(game)
    MESSAGE 'The result of the game is determined automatically';
```

To determine the game result, the OVERRIDE operator is used, which returns the first non-**NULL** value in the order in which expressions are specified. In this case, calculating the "result" property will return either an object of the static class GameResult.win, if the goal difference in the game is greater than 1, or the value of the userResult data property.

In order to always determine a result for the game, we create a constraint that ensures that the user sets the value of the userResult property if the result is not calculated based on the game score.

```
CONSTRAINT ((hostGoals(Game game) (-) guestGoals(game)) < 2 AND (hostGoals(game) (-) guestGoals(game)) > -2) AND NOT userResult(game)
    MESSAGE 'Specify the result of the game';
```

The result of the NOT userResult(game) expression will be true only if userResult(game) is not defined (that is, if it is NULL). Thus, the constraint will be violated if the score difference is 1, and the type of win is not specified by the user.

Creating a score table

The score table is the ranking of the teams in a tournament - a list of teams sorted by ranking.

We define the indicators that determine the team's place on the scoreboard:

- number of games played by the team at home and away, and their total number

```
hostGamesPlayed = GROUP SUM 1 BY hostTeam(Game game);
guestGamesPlayed = GROUP SUM 1 BY guestTeam(Game game);
gamesPlayed 'G' (Team team) = hostGamesPlayed(team) (+)
guestGamesPlayed(team);
```

Here, the construction (+) is used instead of the arithmetic + to obtain the correct result if at least one of the terms has a value of NULL. Using (+) in this case is equivalent to replacing a possible NULL with 0. If one of the terms is NULL, then using the arithmetic + will also result in a value of NULL.

To determine the number of games played by the team at home and away, the GROUP SUM operator is used, which allows you to get the sum of the calculation results of a given expression for objects of a certain class, grouped by one or more of their attributes (similar to subtotals in Excel). Here the number 1 is specified for summation, and all games are grouped by guest team and host team (the BY instruction). As a result, for example, the hostGamesPlayed property determines for the team (since the result of the hostTeam property calculation is the Team-class object) the number (that is, the sum of the number 1 for all games where the host team is equal to the defined one) of games played as hosts (the hostTeam property is specified only for objects of the Game class). With this calculation the system analyzes all games entered into the system.

- number of games won in regular time, in overtime, and in extra time

```
gamesWonBy(Team team, GameResult type) = OVERRIDE
[GROUP SUM 1 BY winner(Game game), result(game)]
(team, type), 0;

gamesWon 'W' (Team team) = gamesWonBy(team,
GameResult.win);
gamesWonOT 'WO' (Team team) = gamesWonBy(team,
GameResult.winOT);
gamesWonSO 'WB' (Team team) = gamesWonBy(team,
GameResult.winSO);
```

Since the logic for determining the number of wins of each type for a team is almost identical, we create and use the intermediate property gamesWonByResult, which is defined for a pair of objects (arguments). This property calculates for the team (first argument) the number of wins of a given type (second argument). The value of the gamesWonBy property is calculated with the OVERRIDE operator, which takes as input an expression specified in brackets ([=...]) and 0. If the expression value is **NULL**, the result of the whole property will be the value 0. A nested expression is specified in square brackets using the GROUP SUM construct. Using a given expression in brackets is identical to using a previously defined property with a similar expression. Thus, the construction [=...] allows you to simply reduce the number of lines of code. Here, GROUP SUM returns the total sum on number 1 for all games grouped by game winner and game result.

The total result of the gamesWonByResult property will be the number of wins of a given type for a given team, or zero if the team did not have any wins of this type (that is, if [= GROUP SUM 1 BY winner(Game game), result(game)] for a given team and type of win is NULL).

- number of games played in regular time, in overtime and in extra time (determined by analogy with the above-specified properties of the number of wins)

```
gamesLostBy(Team team, GameResult type) = OVERRIDE
[GROUP SUM 1 BY looser(Game game), result(game)]
(team, type), 0;

gamesLost 'L' (Team team) = gamesLostBy(team,
GameResult.win);
gamesLostOT 'LO' (Team team) = gamesLostBy(team,
GameResult.winOT);
gamesLostSO 'LB' (Team team) = gamesLostBy(team,
GameResult.winSO);
```

We calculate the number of points scored by the team in the tournament. The calculation is the sum of the number of wins of a particular type for each team, multiplied by the number of points awarded.

```
points 'Points' (Team team) = gamesWon
(team) * 3 + (gamesWonSO(team) + gamesWonOT
(team)) * 2 + gamesLostOT(team) +
gamesLostSO(team);
```

To be used as additional indicators for ranking teams, we calculate the total number of goals scored and missed by the team.

```
hostGoalsScored = GROUP SUM hostGoals(Game game) BY
hostTeam(game);
guestGoalsScored = GROUP SUM guestGoals(Game game) BY
guestTeam(game);
goalsScored 'Scored goals' (Team team) = OVERRIDE
hostGoalsScored(team) (+) guestGoalsScored(team), 0 IF
team IS Team;

hostGoalsConceded = GROUP SUM guestGoals(Game game) BY
hostTeam(game);
guestGoalsConceded = GROUP SUM hostGoals(Game game) BY
guestTeam(game);
goalsConceded 'Conceded goals' (Team team) = OVERRIDE
hostGoalsConceded(team) (+) guestGoalsConceded(team), 0
IF team IS Team;
```

We determine the place of the team on the scoreboard.

```
place 'Rank' (Team team) = PARTITION SUM 1 ORDER DESC points
(team), gamesWon(team), gamesWonOT(team), gamesWonSO(team),
                                               (OVERRIDE
goalsScored(team) (-) goalsConceded(team), 0), goalsScored(team);
```

The "place" property "Team place on the score table" is determined using the construction PARTITION SUM, which for all objects of a certain class in a cumulative total, the sequence of which is specified by the ORDER operator, calculates the sum of the results of the calculation of a specified expression. It is important to remember that the values of all properties involved in determining the order must not be NULL. For this purpose, the penultimate expression uses the OVERRIDE operator so that the number 0 is used instead of NULL.

Thus, the logic for determining the "place" property for each command is as follows:

- all teams are arranged in a sequence (ranked) in descending order of the values of certain parameters (number of points scored, games won in regular time, and other properties specified after the ORDER DESC operator)
- The sum of the values of the specified SUM expression (in this case, number 1) is calculated for each team. The sum is calculated for all teams preceding that team in the ranked list (including that team). That is, 1 for the first team, 1+1 for the second, 1+1+1 for the third, etc.

Defining view logic

We add an interface that allows you to work with the developed system, entering data into the system and obtaining the necessary information from it. The form being created will consist of two vertically arranged blocks, in the upper of which the user will be able to add, modify, and delete games with all their attributes, while in the lower one there will be a score table displaying the results of the games and allowing to add or delete teams and change their names.

We declare the form with a name and caption. We add to the form a block of objects of the Game class with all the properties defined in the system. We also place a button on the form for adding a new game and deleting it.

```
FORM MainForm 'Score table'
    OBJECTS game = Game
    PROPERTIES(game) date, hostTeamName, hostGoals, guestGoals,
guestTeamName, resultName, NEW, DELETE
;
```

The FORM instruction creates an empty form with a certain default functionality. Using the OBJECTS game=Game expression, a "game" object is added to the form: a table view block containing all instances of the Game class entered in the system. Using the expression PROPERTIES(game) with a the subsequent listing of a subset of properties, the specified properties are added to the form, and objects of the "game" block are passed to them as arguments. In addition to previously created properties, the actions NEW and DELETE are also placed on the form, which will visually appear as buttons and allow you to add and remove objects of the Game class.

Data properties displayed on a form that are of a primitive type (date, hostGoals, guestGoals) will visually appear as cells that can be filled and changed by the user. Calculated properties that return an attribute of another object (hostTeamName, guestTeamName, resultName) will appear as cells. When these are clicked, a dialog box with the list of their objects and base group properties will be shown (for example, when clicking on the cell hostTeamName "Guests" a dialog box appears with the list of teams). In the dialog box you can select one of the objects, thus changing the property value for the object of the original form (for example, changing the game host team).

We extend the form by adding a score table block to it. The score table will be shown as a list of teams (objects of the Team class) with their statistical indicators, sorted by rating using the ORDER BY operator.

```
EXTEND FORM MainForm
    OBJECTS team = Team
    PROPERTIES(team) place, name, gamesPlayed, gamesWon, gamesWonOT, gamesWonSO,
                     gamesLostSO, gamesLostOT, gamesLost, goalsScored, goalsConceded, points,
NEW, DELETE
    ORDERS place(team)
;
```

The above form can be defined with a single block of code without using the EXTEND instruction.

```
FORM MainFormSingle 'Score table'
    OBJECTS game = Game
    PROPERTIES(game) date, hostTeamName, hostGoals, guestGoals, guestTeamName, resultName, NEW
, DELETE

    OBJECTS team = Team
    PROPERTIES(team) place, name, gamesPlayed, gamesWon, gamesWonOT, gamesWonSO,
                     gamesLostSO, gamesLostOT, gamesLost, goalsScored, goalsConceded, points,
NEW, DELETE
    ORDERS place(team)
;
```

We place the created form on the main menu of the program - the predefined navigator **root** folder - and indicate that it should be positioned by the very first element in front of the system menu item **Administration**.

```
NAVIGATOR
{
    NEW
MainForm F
IRST;
}
```

The process of creating an information system is completed.

The complete source code (on GitHub)

```
MODULE HockeyStats;
// logic description: https://documentation.lsfusion.org/pages/viewpage.action?pageId=2228240
REQUIRE System, Utils;

CLASS Team 'Team';

name 'Team name' = DATA STRING[30] (Team) IN base;

CLASS Game 'Game';

date 'Date' = DATA DATE (Game);
hostTeam = DATA Team (Game);
guestTeam = DATA Team (Game);
hostTeamName 'Hosts' (Game game) = name(hostTeam(game));
guestTeamName 'Guests' (Game game) = name(guestTeam(game));

CONSTRAINT hostTeam(Game team) = guestTeam(team) CHECKED BY hostTeam, guestTeam MESSAGE 'Host
and guest teams must be different';

hostGoals 'H goals' = DATA INTEGER (Game);
guestGoals 'G goals' = DATA INTEGER (Game);

CONSTRAINT hostGoals(Game game) = guestGoals(game) MESSAGE 'The game cannot end in a draw';

winner(Game game) = IF hostGoals(game) > guestGoals(game)
                    THEN hostTeam(game)
                    ELSE guestTeam(game);

looser(Game game) = IF hostGoals(game) > guestGoals(game)
                    THEN guestTeam(game)
                    ELSE hostTeam(game);

CLASS GameResult 'G/R' {
    win 'W',
    winOT 'LO',
    winSO 'LB'
}

resultName 'Name' (GameResult game) = staticCaption(game) IF game IS GameResult IN base;

userResult = DATA GameResult (Game);
result (Game game) = OVERRIDE userResult(game),
    (GameResult.win IF ((hostGoals(game) (-) guestGoals(game)) > 1 OR (guestGoals(game) (-)
hostGoals(game)) > 1));
resultName 'G/R' (Game game) = resultName(result(game));

CONSTRAINT ((hostGoals(Game game) (-) guestGoals(game)) > 1 OR (hostGoals(game) (-) guestGoals
(game)) < -1) AND userResult(game)
    MESSAGE 'The result of the game is determined automatically';

CONSTRAINT ((hostGoals(Game game) (-) guestGoals(game)) < 2 AND (hostGoals(game) (-)
guestGoals(game)) > -2) AND NOT userResult(game)
    MESSAGE 'Specify the result of the game';

hostGamesPlayed = GROUP SUM 1 BY hostTeam(Game game);
guestGamesPlayed = GROUP SUM 1 BY guestTeam(Game game);
gamesPlayed 'G' (Team team) = hostGamesPlayed(team) (+) guestGamesPlayed(team);

gamesWonBy(Team team, GameResult type) = OVERRIDE [GROUP SUM 1 BY winner(Game game), result
(game)](team, type), 0;

gamesWon 'W' (Team team) = gamesWonBy(team, GameResult.win);
gamesWonOT 'WO' (Team team) = gamesWonBy(team, GameResult.winOT);
gamesWonSO 'WB' (Team team) = gamesWonBy(team, GameResult.winSO);

gamesLostBy(Team team, GameResult type) = OVERRIDE [GROUP SUM 1 BY looser(Game game), result
(game)](team, type), 0;

gamesLost 'L' (Team team) = gamesLostBy(team, GameResult.win);
gamesLostOT 'LO' (Team team) = gamesLostBy(team, GameResult.winOT);
gamesLostSO 'LB' (Team team) = gamesLostBy(team, GameResult.winSO);

points 'Points' (Team team) = gamesWon(team) * 3 + (gamesWonSO(team) + gamesWonOT(team)) * 2
+ gamesLostOT(team) + gamesLostSO(team);

hostGoalsScored = GROUP SUM hostGoals(Game game) BY hostTeam(game);
guestGoalsScored = GROUP SUM guestGoals(Game game) BY guestTeam(game);
goalsScored 'Scored goals' (Team team) = OVERRIDE hostGoalsScored(team) (+) guestGoalsScored
(team), 0 IF team IS Team;

hostGoalsConceded = GROUP SUM guestGoals(Game game) BY hostTeam(game);
guestGoalsConceded = GROUP SUM hostGoals(Game game) BY guestTeam(game);
goalsConceded 'Conceded goals' (Team team) = OVERRIDE hostGoalsConceded(team) (+)
guestGoalsConceded(team), 0 IF team IS Team;

place 'Rank' (Team team) = PARTITION SUM 1 ORDER DESC points(team), gamesWon(team), gamesWonOT
(team), gamesWonSO(team),
                                               (OVERRIDE goalsScored(team) (-) goalsConceded
(team), 0), goalsScored(team);

FORM MainForm 'Score table'
    OBJECTS game = Game
    PROPERTIES(game) date, hostTeamName, hostGoals, guestGoals, guestTeamName, resultName, NEW
, DELETE
;

EXTEND FORM MainForm
    OBJECTS team = Team
    PROPERTIES(team) place, name, gamesPlayed, gamesWon, gamesWonOT, gamesWonSO,
                     gamesLostSO, gamesLostOT, gamesLost, goalsScored, goalsConceded, points,
NEW, DELETE
    ORDERS place(team)
;

FORM MainFormSingle 'Score table'
    OBJECTS game = Game
    PROPERTIES(game) date, hostTeamName, hostGoals, guestGoals, guestTeamName, resultName, NEW
, DELETE
```

```

    OBJECTS team = Team
    PROPERTIES(team) place, name, gamesPlayed, gamesWon, gamesWonOT, gamesWonSO,
                     gamesLostSO, gamesLostOT, gamesLost, goalsScored, goalsConceded, points,
NEW, DELETE
    ORDERS place(team)
;

NAVIGATOR {
    NEW MainForm FIRST;
}

CLASS Event;
date = DATA DATE (Event);
date(Event e) <- currentDate() WHEN SET(e IS Event);
title = DATA STRING (Event);
title(Event e) <- 'Event' + e WHEN SET(e IS Event);

FORM calendar
    OBJECTS e=Event CUSTOM 'calendar'
    PROPERTIES (e) date, title, NEW, EDIT, DELETE
;

NAVIGATOR {
    NEW calendar;
}
```