Challenge July 2022 - Extension

Evaluate Team & Player Performance

A second solution with DT5GL by Jack Jansonius – 19 September 2022

For a number of reasons, I find this a very nice and interesting challenge.

First, it is not a logical puzzle, but is about something that occurs in reality, namely sports teams that have players who participate in games and can be judged on their performance. The challenge involves 2 teams with 3 players each participating in 3 matches, but very easily you can think of hundreds of teams where thousands of players participate in tens of thousands of matches. In that respect, you also want a solution that can handle much larger numbers just as easily.

Second, this challenge asks a clear question, namely: Which team got the most points? and then you also expect a nicely formulated answer, for example: The winner is team Mustungs with 11 points!

Third, it is a challenge that can very easily be checked by hand. That team Mustungs is the winner with 11 points and team Eagles is a close second with 8 points can be figured out with little calculation in advance.

Fourth, this challenge is also very easy to expand with additional tasks, which immediately make the challenge much more complicated. That is exactly what I am going to do with this solution.

The extension to the challenge I propose states:
- which players have the highest average performance.
- Which players have the lowest average performance.

Again, this extension is very easy to calculate by hand. In which a nice complication immediately arises:

Brown: 3.33
Robinson: 1,00
Smith: -0.67
Black: 3.33
White: 1.00
Green: -1.67

The complication here is: there can be multiple players with a highest average and multiple players with a lowest average. Where, therefore, the answer must distinguish between 1 player (on 1 line) or an enumeration of 2 or more players.

The output of the program could now be:

```
-----------------------------------------------------------------------------------------
Winner is team Mustungs with 11 points!

Players with highest average (3.33):
- Brown (team: Mustungs)
- Black (team: Eagles)

Player with lowest average (-1.67): Green  (team: Eagles)
-----------------------------------------------------------------------------------------
```

Here, by the way, one can think of a very special (although very theoretical) test case, namely the situation where every player in the example on the website is given the same qualification, for example "good." Then all players have both the highest and lowest average, and in addition both teams have the most number of points:

```
---------------------------------------------------------------------------------------------
Teams with most points (18):
- Mustungs
- Eagles

Players with highest average (2.0):
- Brown (team: Mustungs)
- Robinson (team: Mustungs)
- Smith (team: Mustungs)
- Black (team: Eagles)
- White (team: Eagles)
- Green (team: Eagles)

Players with lowest average (2.0):
- Brown (team: Mustungs)
- Robinson (team: Mustungs)
- Smith (team: Mustungs)
- Black (team: Eagles)
- White (team: Eagles)
- Green (team: Eagles)
---------------------------------------------------------------------------------------------
```

Which thus provides another extension to the challenge: in some cases, multiple teams can have the maximum number of points.

Now the output given above is indeed what my program produces, but it still took some adjustments to the reasoning mechanism (and a new repeat instruction) to get this done. For example, I had already built in a repeat_until instruction for the March 2019 organ donation challenge[1], which allows me to perform exactly the nested loops as found in Jacob Feldman's solution, namely: Teams>Players>Game efficiencies[2]. And this repeat instruction is also found in my solution for Rebooking Passengers from Cancelled Flights (Oct-2016)[3].

But this repeat_until turns out to be inappropriate for the flow for displaying one or more players with a maximum average, and then one or more players with a minimum average; for this I had to come up with a repeat_while instruction.

1 See: https://dmcommunity.org/challenge/challenge-march-2019/
2 See: https://openrules.wordpress.com/2022/07/18/dealing-with-embedded-loops/ and appendix A.
3 See: https://dmcommunity.org/challenge/challenge-oct-2016/

So there is a difference between the repeat_until in my previous solution:

```
Table 0:
If:                                          | 0| 1|
'Next Game to process?'                      | Y| N|
Then:
Action is Process_game                       | X|  |
Action is Games_processed                    |  | X|
# .......
# Repeat until: Action is Games_processed
```

and the repeat_while that I'm going to use now:

```
Table 0:
If:                                          | 0| 1|
'Next Game to process?'                      | Y| N|
Then:
Eval_Game is Process_game                    | X|  |
Eval_Game is Finished                        |  | X|
# .......
# Repeat while: Eval_Game is Process_game
```

where the actual repeat instructions are found at the the respective goal attributes, namely:

```
GoalAttribute: Action
Repeat_until: Games_processed
```

and

```
GoalAttribute: Eval_Game
Repeat_while: Process_game
```

The difference in operation occurs when handling subsequent goal attributes (if there are any!): with repeat_while, subsequent goal attributes are not handled until the current one is completely finished; with repeat_until, for each proven goal attribute, a subsequent goal attribute is being handled.

Tables in the database

Again, the following tables are used:

Game:

	Teamid	Playerid	Date	Efficiency
1	1	1	04-01-2022	good
2	1	1	04-02-2022	better
3	1	1	04-03-2022	best
4	1	2	04-01-2022	worst
5	1	2	04-02-2022	better
6	1	2	04-03-2022	best
7	1	3	04-01-2022	bad
8	1	3	04-02-2022	good
9	1	3	04-03-2022	bad
10	2	4	04-01-2022	good
11	2	4	04-02-2022	better
12	2	4	04-03-2022	best
13	2	5	04-01-2022	worst
14	2	5	04-02-2022	better
15	2	5	04-03-2022	best
16	2	6	04-01-2022	bad
17	2	6	04-02-2022	good
18	2	6	04-03-2022	worst

Player:

	Id	Name
1	1	Brown
2	2	Robinson
3	3	Smith
4	4	Black
5	5	White
6	6	Green

Team:

	Id	Name
1	1	Mustungs
2	2	Eagles

and added to this:

Sum_per_team:

	teamid	totalpoints
1	1	11
2	2	8

Sum_per_player:

	playerid	totalgames	totalpoints
1	1	3	10
2	2	3	3
3	3	3	-2
4	4	3	10
5	5	3	3
6	6	3	-5

Note: content after running the process; in advance everything is set to 0.

Implementation of the decision model in DT5GL:

```
SQLite_database: "Database/TeamPlayerPerformance.sqlite3"

Table 0: Process all games and count team and player performances
If:                                          | 0| 1|
'Next Game to process?'                      | Y| N|
Then:
Eval_Game is Process_game                    | X|  |
Eval_Game is Finished                        |  | X|
# .......
# Repeat while: Eval_Game is Process_game


Proposition: 'Next Game to process?'
Obtain_instance_from_database_view: game


Table 1: display one or more teams with most points
If:                                          | 0| 1|
best_performing_teams.number > 1             | Y| N|
Then:
Eval_Team1 is Display_first_team             | X|  |
Eval_Team1 is Display_one_team               |  | X|
# .......


Table 2: display next teams with most points
If:                                          | 0| 1|
'Next Team with most points?'                | Y| N|
Then:
Eval_Team2 is Display_next_team              | X|  |
Eval_Team2 is Finished                       |  | X|
# .......
# Repeat while: Eval_Team2 is Display_next_team


Proposition: 'Next Team with most points?'
Obtain_instance_from_database_view: best_performing_teams


Table 3: display one or more players with highest average
If:                                          | 0| 1|
highest_rated_avg.number > 1                 | Y| N|
Then:
Eval_High1 is Display_first_player           | X|  |
Eval_High1 is Display_one_player             |  | X|
# .......


Table 4: display next players with highest average
If:                                          | 0| 1|
'Next Player with highest average?'          | Y| N|
Then:
Eval_High2 is Display_next_player            | X|  |
Eval_High2 is Finished                       |  | X|
# .......
# Repeat while: Eval_High2 is Display_next_player


Proposition: 'Next Player with highest average?'
Obtain_instance_from_database_view: highest_rated_players
```

```
Table 5: display one or more players with lowest average
If:                                              | 0| 1|
lowest_rated_avg.number > 1                      | Y| N|
Then:
Eval_Low1 is Display_first_player                | X|  |
Eval_Low1 is Display_one_player                  |  | X|
# .......


Table 6: display next players with lowest average
If:                                              | 0| 1|
'Next Player with lowest average?'               | Y| N|
Then:
Eval_Low2 is Display_next_player                 | X|  |
Eval_Low2 is Finished                            |  | X|
# .......
# Repeat while: Eval_Low2 is Display_next_player


Proposition: 'Next Player with lowest average?'
Obtain_instance_from_database_view: lowest_rated_players


Attribute: Efficiency  Type: Text
Obtain_value_from_database_view: game.efficiency

Attribute: points      Type: Integer
Equals:  5 if Efficiency == "best"    \
   else  3 if Efficiency == "better"  \
   else  2 if Efficiency == "good"    \
   else -2 if Efficiency == "bad"     \
   else -5 if Efficiency == "worst"   \
   else 99999


########################## Database views ############################

Database_view: game
With_attributes:
teamid, playerid, efficiency
Query:
SELECT teamid, playerid, efficiency
  FROM game
 LIMIT 1 OFFSET %s
With_arguments: game.auto_index


Database_view: best_performing_teams
With_attributes:
teamid, teamname, totalpoints, number
Query:
SELECT a.teamid, b.name, a.totalpoints, count(*) AS number
FROM sum_per_team AS a
JOIN team AS b on (a.teamid = b.id)
JOIN (SELECT count(distinct teamid)
           FROM sum_per_team
           WHERE totalpoints =
                 (SELECT max(totalpoints) FROM sum_per_team) GROUP BY teamid)
WHERE a.totalpoints = (SELECT max(totalpoints) FROM sum_per_team)
GROUP BY teamid
LIMIT 1 OFFSET %s
With_arguments: best_performing_teams.auto_index
```

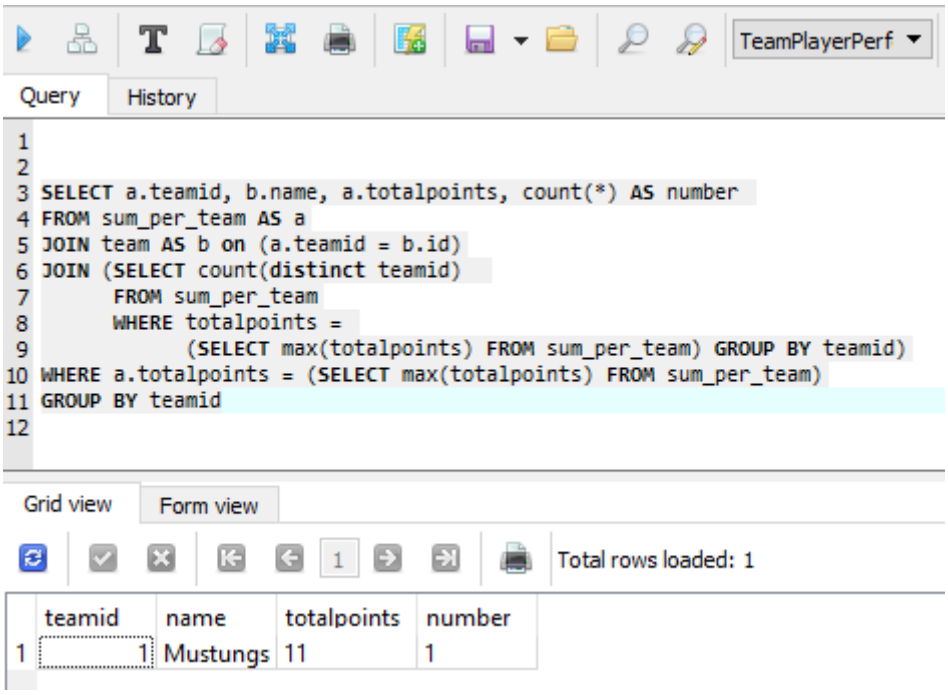
```
Database_view: highest_rated_avg
With_attributes:
number
Query:
SELECT count(distinct playerid) AS number
FROM sum_per_player
WHERE totalpoints*1.0/totalgames =
      (SELECT max(totalpoints*1.0/totalgames) FROM sum_per_player)
End_Query

Database_view: lowest_rated_avg
With_attributes:
number
Query:
SELECT count(distinct playerid) AS number
FROM sum_per_player
WHERE totalpoints*1.0/totalgames =
      (SELECT min(totalpoints*1.0/totalgames) FROM sum_per_player)
End_Query


Database_view: highest_rated_players
With_attributes:
playerid, playername, teamname, totalpoints, totalgames, avgpoints
Query:
SELECT a.playerid, b.name AS playername, c.name AS teamname,
       a.totalpoints,
       a.totalgames,
       round(totalpoints*1.0/totalgames, 2) AS avgpoints
FROM sum_per_player AS a
JOIN player AS b on (a.playerid = b.id)
JOIN (SELECT playerid, name
       FROM game
       JOIN team
       WHERE game.teamid = team.id) AS c
       on a.playerid = c.playerid
WHERE totalpoints*1.0/totalgames =
      (SELECT max(totalpoints*1.0/totalgames) FROM sum_per_player)
GROUP BY a.playerid
LIMIT 1 OFFSET %s
With_arguments: highest_rated_players.auto_index


Database_view: lowest_rated_players
With_attributes:
playerid, playername, teamname, totalpoints, totalgames, avgpoints
Query:
SELECT a.playerid, b.name AS playername, c.name AS teamname,
       a.totalpoints,
       a.totalgames,
       round(totalpoints*1.0/totalgames, 2) AS avgpoints
FROM sum_per_player AS a
JOIN player AS b on (a.playerid = b.id)
JOIN (SELECT playerid, name
       FROM game
       JOIN team
       WHERE game.teamid = team.id) AS c
       on a.playerid = c.playerid
WHERE totalpoints*1.0/totalgames =
      (SELECT min(totalpoints*1.0/totalgames) FROM sum_per_player)
GROUP BY a.playerid
LIMIT 1 OFFSET %s
With_arguments: lowest_rated_players.auto_index
```

```
########################## GoalAttributes ############################

GoalAttribute: Eval_Game
Repeat_while: Process_game

Case: Process_game
Print: "Efficiency for player %s is %s so add %s points for team with id: %s "
game.playerid  game.efficiency  points  game.teamid
>SQL:  "UPDATE sum_per_team "
-SQL:  "   SET totalpoints = totalpoints + %s "       points
<SQL:  " WHERE teamid = %s "                          game.teamid
>SQL:  "UPDATE sum_per_player "
-SQL:  "   SET totalgames = totalgames + 1,   "
-SQL:  "       totalpoints = totalpoints + %s "       points
<SQL:  " WHERE playerid = %s "                        game.playerid

Case: Finished
Print: "    "


# DISPLAY ONE OR FIRST TEAM WITH MOST POINTS
GoalAttribute: Eval_Team1

Case: Display_first_team
Print: "Teams with most points (%s):"  best_performing_teams.totalpoints
Print: "- %s "                        best_performing_teams.teamname

Case: Display_one_team
Print: "Winner is team %s with %s points!"
best_performing_teams.teamname  best_performing_teams.totalpoints
Print: "        "

# DISPLAY NEXT TEAMS WITH MOST POINTS
GoalAttribute: Eval_Team2
Repeat_while: Display_next_team

Case: Display_next_team
Print: "- %s "                        best_performing_teams.teamname

Case: Finished
Print: "      "


# DISPLAY ONE OR FIRST PLAYER WITH HIGHEST AVERAGE
GoalAttribute: Eval_High1

Case: Display_first_player
Print: "Players with highest average (%s):"    highest_rated_players.avgpoints
Print: "- %s (team: %s)"
highest_rated_players.playername highest_rated_players.teamname

Case: Display_one_player
Print: "Player with highest average (%s): %s  (team: %s) "
highest_rated_players.avgpoints  highest_rated_players.playername
highest_rated_players.teamname
Print: "        "

# DISPLAY NEXT PLAYERS WITH HIGHEST AVERAGE
GoalAttribute: Eval_High2
Repeat_while: Display_next_player

Case: Display_next_player
Print: "- %s (team: %s)"
highest_rated_players.playername highest_rated_players.teamname

Case: Finished
Print: "      "
```

```
# DISPLAY ONE OR FIRST PLAYER WITH LOWEST AVERAGE
GoalAttribute: Eval_Low1

Case: Display_first_player
Print: "Players with lowest average (%s):"   lowest_rated_players.avgpoints
Print: "- %s (team: %s)"
lowest_rated_players.playername lowest_rated_players.teamname

Case: Display_one_player
Print: "Player with lowest average (%s): %s  (team: %s) "
lowest_rated_players.avgpoints  lowest_rated_players.playername
lowest_rated_players.teamname
Print: "        "


# DISPLAY NEXT PLAYERS WITH LOWEST AVERAGE
GoalAttribute: Eval_Low2
Repeat_while: Display_next_player

Case: Display_next_player
Print: "- %s (team: %s)"
lowest_rated_players.playername lowest_rated_players.teamname

Case: Finished
Print: "      "


Initial_database_setup: make_start_summation_per_team
Query:
    UPDATE sum_per_team SET totalpoints = 0
End_Query

Initial_database_setup: make_start_summation_per_player
Query:
    UPDATE sum_per_player SET totalgames = 0, totalpoints = 0
End_Query
```

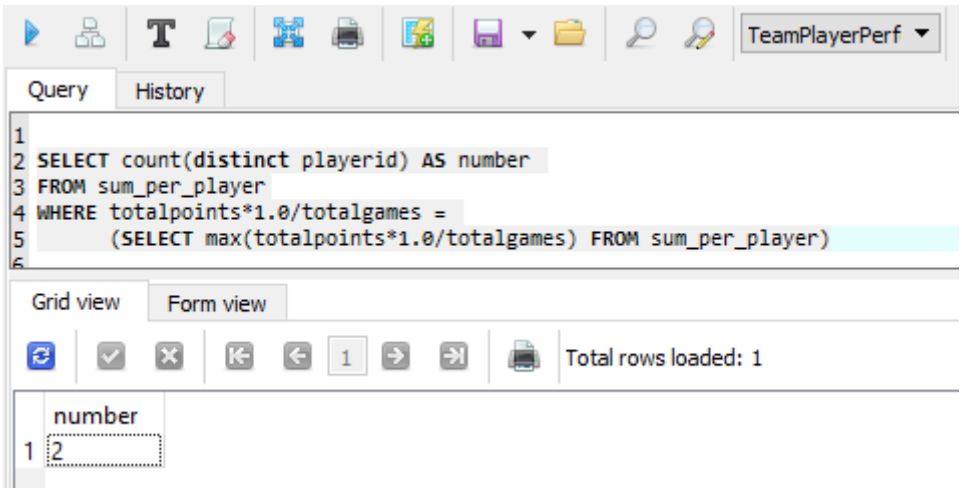
What do database views do? Check![4]

Database_view: best_performing_teams



Note: best_performing_teams.number is the number of teams with the most points.

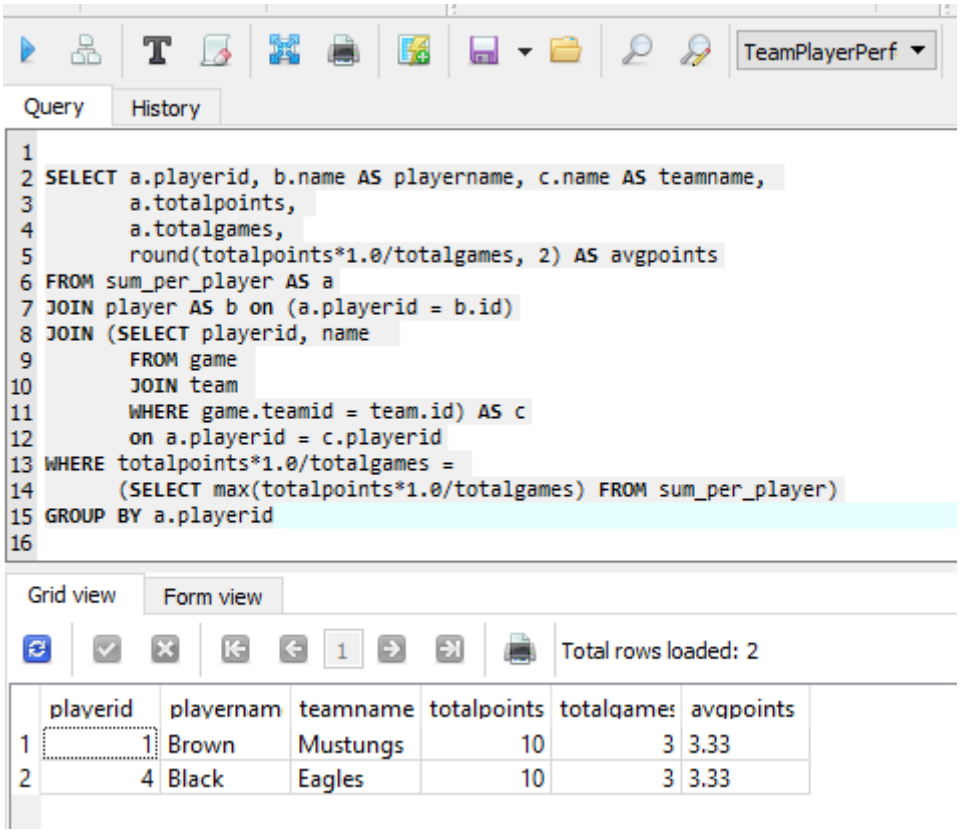
Database_view: highest_rated_avg



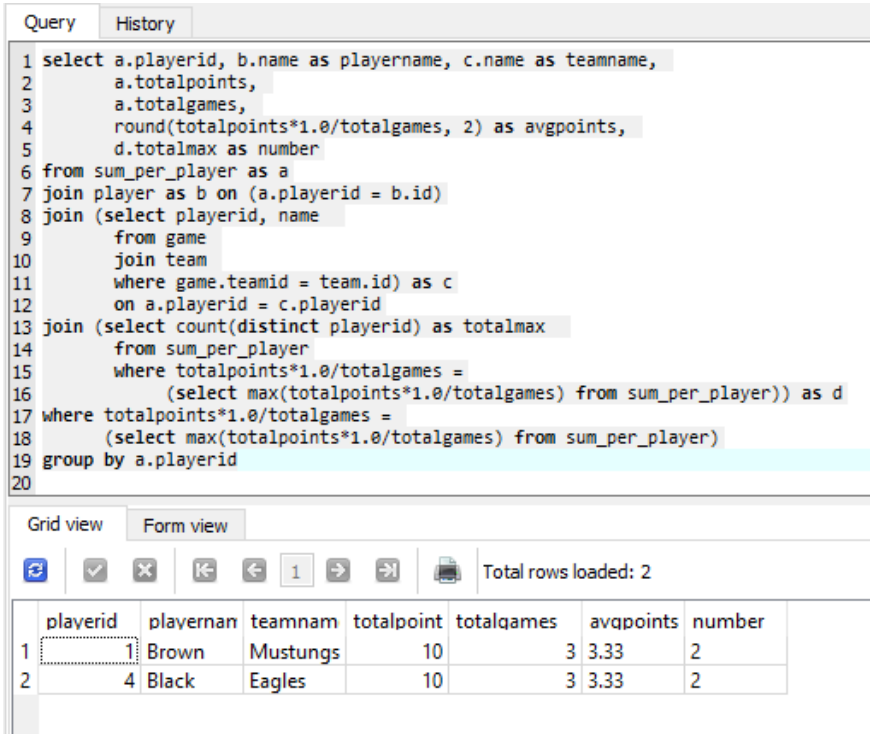
Note: highest_rated_avg.number is the number of people with highest average.

[4] Here the database tool SQLite was used, but gives the same results with PostgreSQL.
This requires the first instruction in the program: SQLite_database: "Database/TeamPlayerPerformance.sqlite3" be replaced with: PostgreSQL_database: "teamplayerperformance".
Connection to any other database tool is also possible, but not realized at this moment.

Database_view: highest_rated_players



Previous 2 database views can of course also be in 1 view (as done with best_performing_teams):



Appendix A: A comparison with Jacob Feldman's nested loops.[5]

```
Table 0: Process Teams
If:                                          | 0| 1|
'Next Team?'                                 | Y| N|
Then:
Action1 is Process_Players                   | X|  |
Action1 is Finished                          |  | X|
# .......
# Repeat until: Action1 is Finished

Table 1: Process players for this team
If:                                          | 0| 1|
'Next Player for this team?'                 | Y| N|
Then:
Action2 is Process_Performances              | X|  |
Action2 is Finished                          |  | X|
# .......
# Repeat until: Action2 is Finished

Table 2: Process performances for this player
If:                                          | 0| 1|
'Next Performance for this player?'          | Y| N|
Then:
Action3 is Process_game                      | X|  |
Action3 is Finished                          |  | X|
# .......
# Repeat until: Action3 is Finished


Proposition: 'Next Team?'
Obtain_instance_from_database_view: team

Proposition: 'Next Player for this team?'
Obtain_instance_from_database_view: player_for_team

Proposition: 'Next Performance for this player?'
Obtain_instance_from_database_view: performance_for_player


......
GoalAttribute: Action1
Repeat_until: Finished

Case: Finished
Print: "Winner is team %s with %s points!"  topscore.teamname topscore.totalpoints

Case: Process_Players
Print: "#REM# - TZT TEXT"


......
GoalAttribute: Action3
Repeat_until: Finished

Case: Finished
Print: "#REM# - TZT TEXT"

Case: Process_game
Print: "Efficiency for player %s is %s so add %s points for team with id: %s "
player_for_team.playerid  performance_for_player.efficiency  Points  team.teamid
......
```

[5] See: https://openrules.wordpress.com/2022/07/18/dealing-with-embedded-loops/

In my solution, I do not use nested loops to process the game records (which also do not need to be sorted); with this comparison I mainly want to show that the repeat_until instruction used here corresponds exactly to Jacob Feldman's for-loop.

```
Database_view: team
With_attributes:
teamid
Query:
SELECT distinct teamid
  FROM game
 LIMIT 1 OFFSET %s
With_arguments: team.auto_index


Database_view: player_for_team
With_attributes:
playerid
Query:
SELECT distinct playerid
  FROM game
 WHERE teamid = %s
 LIMIT 1 OFFSET %s
With_arguments: team.teamid, player_for_team.auto_index


Database_view: performance_for_player
With_attributes:
efficiency
Query:
SELECT efficiency
  FROM game
 WHERE teamid = %s
   AND playerid = %s
 LIMIT 1 OFFSET %s
With_arguments:
team.teamid, player_for_team.playerid, performance_for_player.auto_index

……




Efficiency for player 1 is good so add 2 points for team with id: 1
Efficiency for player 1 is better so add 3 points for team with id: 1
Efficiency for player 1 is best so add 5 points for team with id: 1
Efficiency for player 2 is worst so add -5 points for team with id: 1
Efficiency for player 2 is better so add 3 points for team with id: 1
Efficiency for player 2 is best so add 5 points for team with id: 1
Efficiency for player 3 is bad so add -2 points for team with id: 1
Efficiency for player 3 is good so add 2 points for team with id: 1
Efficiency for player 3 is bad so add -2 points for team with id: 1
Efficiency for player 4 is good so add 2 points for team with id: 2
Efficiency for player 4 is better so add 3 points for team with id: 2
Efficiency for player 4 is best so add 5 points for team with id: 2
Efficiency for player 5 is worst so add -5 points for team with id: 2
Efficiency for player 5 is better so add 3 points for team with id: 2
Efficiency for player 5 is best so add 5 points for team with id: 2
Efficiency for player 6 is bad so add -2 points for team with id: 2
Efficiency for player 6 is good so add 2 points for team with id: 2
Efficiency for player 6 is worst so add -5 points for team with id: 2
Winner is team Mustungs with 11 points!
```