Challenge July 2022 - Extension

Evaluate Team & Player Performance

A third solution with DT5GL by Jack Jansonius – 15 October 2022

In my second solution to the July 2022 challenge, I proposed an extension to the original challenge:

- which teams have the most points (original challenge).
- which players have the highest average performance.
- which players have the lowest average performance.

Where for each of these challenges the possibility of one or more teams and players being able to meet them is taken into account.

Now what is remarkable about this second solution is that the decision tables are very simple, but the SQL queries used are quite complex.

In fact, some of the queries are so complex that I cannot get them to work in PostgreSQL; it gives me the following error message:

```
ERROR: subquery in FROM must have an alias
```

about which someone made the following irritated comment back in 2015:

“It is just unlucky syntax. As long as you are not referencing to that subquery, it does not matter what it's alias is. Personally, I'm using AS pg_sucks, meaning "well, here you have some redundant identifier, but you could generate some internally by yourself, damn postgres!" :)”[1]

But there is another reason to strongly simplify the SQL queries in the second solution. Indeed, with this solution I am not yet fulfilling a promise I made in an article on the DMC Web site:[2]:

“The good news is that even complicated queries can be made much more straightforward when combined with decision tables!”

So hence this third solution, where the decision tables are slightly extended (but still very simple), and the queries in SQL are strongly simplified.

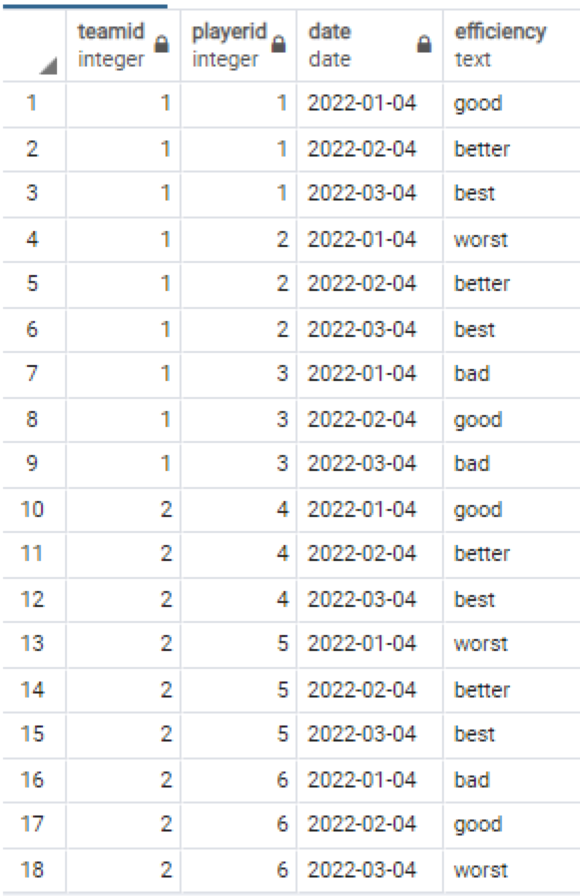
[1] https://stackoverflow.com/questions/14767209/subquery-in-from-must-have-an-alias
It seems that this redundant syntax is to be modified only in a future version of PostgreSQL.

[2] https://dmcommunity.org/2021/09/02/is-sql-for-business-or-it/

Tables in the database:

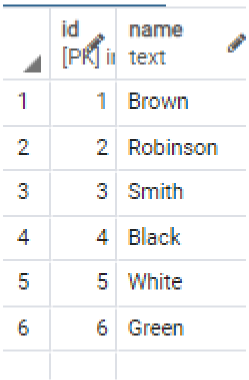
Again, the following tables are used (but now in PostgreSQL):

Game:



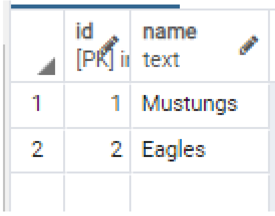
	teamid integer	playerid integer	date date	efficiency text
1	1	1	2022-01-04	good
2	1	1	2022-02-04	better
3	1	1	2022-03-04	best
4	1	2	2022-01-04	worst
5	1	2	2022-02-04	better
6	1	2	2022-03-04	best
7	1	3	2022-01-04	bad
8	1	3	2022-02-04	good
9	1	3	2022-03-04	bad
10	2	4	2022-01-04	good
11	2	4	2022-02-04	better
12	2	4	2022-03-04	best
13	2	5	2022-01-04	worst
14	2	5	2022-02-04	better
15	2	5	2022-03-04	best
16	2	6	2022-01-04	bad
17	2	6	2022-02-04	good
18	2	6	2022-03-04	worst

Player:



	id [PK] i	name text
1	1	Brown
2	2	Robinson
3	3	Smith
4	4	Black
5	5	White
6	6	Green

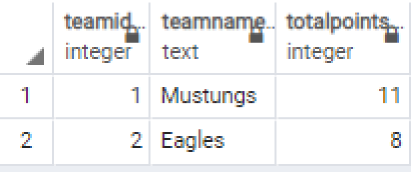
Team:



	id [PK] i	name text
1	1	Mustungs
2	2	Eagles

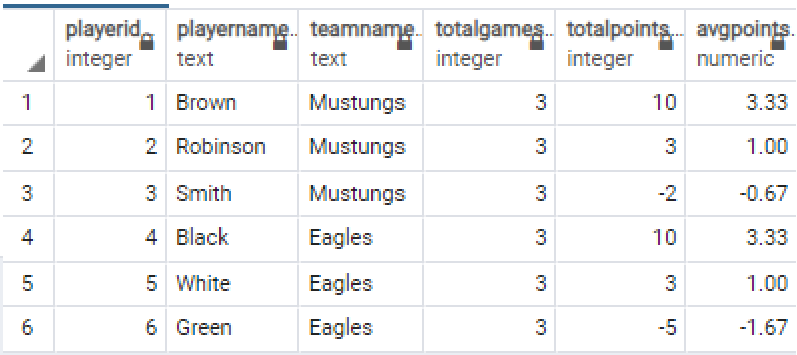
and added to this:

Sum_per_team:



	teamid.. integer	teamname.. text	totalpoints.. integer
1	1	Mustungs	11
2	2	Eagles	8

Sum_per_player:



	playerid.. integer	playername.. text	teamname.. text	totalgames.. integer	totalpoints.. integer	avgpoints.. numeric
1	1	Brown	Mustungs	3	10	3.33
2	2	Robinson	Mustungs	3	3	1.00
3	3	Smith	Mustungs	3	-2	-0.67
4	4	Black	Eagles	3	10	3.33
5	5	White	Eagles	3	3	1.00
6	6	Green	Eagles	3	-5	-1.67

Note: content after running the process; in advance everything is set to 0.
In addition, these tables are now created with initial SQL commands from the Player and Team tables, as can be found in the program source.

Implementation of the decision model in DT5GL:

```
PostgreSQL_database: "teamplayerperformance"

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


# DISPLAY TEAM(S) WITH MOST POINTS ###################################
rTable 1: display one or more teams with highest average
If:                                                         | 0| 1| 2|
'First team with most points'                               | Y| Y| Y|
'Next team with most points'                                | Y| Y| N|
next_team_most.totalpoints = first_team_most.totalpoints    | Y| N| -|
Then:
Eval_Team1 is Display_first_teams                           | X|  |  |
Eval_Team1 is Display_one_team                              |  | X| X|
# .......


Proposition: 'First team with most points'
Obtain_instance_from_database_view: first_team_most

Proposition: 'Next team with most points'
Obtain_instance_from_database_view: next_team_most


Table 2: display next teams with most points
If:                                                         | 0| 1| 2|
'Next team with most points'                                | Y| Y| N|
next_team_most.totalpoints = first_team_most.totalpoints    | Y| N| -|
Then:
Eval_Team2 is Display_next_team                             | X|  |  |
Eval_Team2 is Finished                                      |  | X| X|
# .......
# Repeat while: Eval_Team2 is Display_next_team
# DISPLAY PLAYER(S) WITH HIGHEST AVERAGE #############################
rTable 3: display one or more players with highest average
If:                                                         | 0| 1| 2|
'First player with highest average'                         | Y| Y| Y|
'Next player with highest average'                          | Y| Y| N|
next_player_high.avgpoints = first_player_high.avgpoints    | Y| N| -|
Then:
Eval_High1 is Display_first_players                         | X|  |  |
Eval_High1 is Display_one_player                            |  | X| X|
# .......


Proposition: 'First player with highest average'
Obtain_instance_from_database_view: first_player_high

Proposition: 'Next player with highest average'
Obtain_instance_from_database_view: next_player_high


Table 4: display next players with highest average
If:                                                         | 0| 1| 2|
'Next player with highest average'                          | Y| Y| N|
next_player_high.avgpoints = first_player_high.avgpoints    | Y| N| -|
Then:
Eval_High2 is Display_next_player                           | X|  |  |
Eval_High2 is Finished                                      |  | X| X|
# .......
# Repeat while: Eval_High2 is Display_next_player



# DISPLAY PLAYER(S) WITH LOWEST AVERAGE ##############################
rTable 5: display one or more players with lowest average
If:                                                         | 0| 1| 2|
'First player with lowest average'                          | Y| Y| Y|
'Next player with lowest average'                           | Y| Y| N|
next_player_low.avgpoints = first_player_low.avgpoints      | Y| N| -|
Then:
Eval_Low1 is Display_first_players                          | X|  |  |
Eval_Low1 is Display_one_player                             |  | X| X|
# .......


Proposition: 'First player with lowest average'
Obtain_instance_from_database_view: first_player_low

Proposition: 'Next player with lowest average'
Obtain_instance_from_database_view: next_player_low


Table 6: display next players with lowest average
If:                                                         | 0| 1| 2|
'Next player with lowest average'                           | Y| Y| N|
next_player_low.avgpoints = first_player_low.avgpoints      | Y| N| -|
Then:
Eval_Low2 is Display_next_player                            | X|  |  |
Eval_Low2 is Finished                                       |  | X| X|
# .......
# Repeat while: Eval_Low2 is Display_next_player
```

```
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


Attribute: first_team_most.totalpoints     Type: Integer
Attribute: next_team_most.totalpoints      Type: Integer

Attribute: first_player_high.avgpoints     Type: Real
Attribute: next_player_high.avgpoints      Type: Real

Attribute: first_player_low.avgpoints      Type: Real
Attribute: next_player_low.avgpoints       Type: Real


Attribute: next_most_index_starting_at_1   Type: Integer
Equals:    next_team_most.auto_index + 1

Attribute: next_high_index_starting_at_1   Type: Integer
Equals:    next_player_high.auto_index + 1

Attribute: next_low_index_starting_at_1    Type: Integer
Equals:    next_player_low.auto_index + 1


########################### 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


# DISPLAY TEAM(S) WITH MOST POINTS
Database_view: first_team_most
With_attributes: teamid, teamname, totalpoints
Query:
SELECT teamid, teamname, totalpoints
  FROM sum_per_team
 ORDER BY totalpoints DESC, teamid ASC
 LIMIT 1 OFFSET 0
End_Query

Repeatable_database_view=>[3]: next_team_most
With_attributes: teamid, teamname, totalpoints
Query:
SELECT teamid, teamname, totalpoints
  FROM sum_per_team
 ORDER BY totalpoints DESC, teamid ASC
 LIMIT 1 OFFSET %s
With_arguments: next_most_index_starting_at_1
```

[3] In regard to avoiding an infinite loop in decision table 2, 'refreshing' the database view is necessary here! On the other hand, the database view for the first team found should not be refreshed.

```
# DISPLAY PLAYER(S) WITH HIGHEST AVERAGE
Database_view: first_player_high
With_attributes:
playerid, playername, teamname, avgpoints
Query:
SELECT playerid, playername, teamname, avgpoints
  FROM sum_per_player
 ORDER BY avgpoints DESC, playerid ASC
 LIMIT 1 OFFSET 0
End_Query

Repeatable_database_view=>: next_player_high
With_attributes:
playerid, playername, teamname, avgpoints
Query:
SELECT playerid, playername, teamname, avgpoints
  FROM sum_per_player
 ORDER BY avgpoints DESC, playerid ASC
 LIMIT 1 OFFSET %s
With_arguments: next_high_index_starting_at_1


# DISPLAY PLAYER(S) WITH LOWEST AVERAGE
Database_view: first_player_low
With_attributes:
playerid, playername, teamname, avgpoints
Query:
SELECT playerid, playername, teamname, avgpoints
  FROM sum_per_player
 ORDER BY avgpoints ASC, playerid ASC
 LIMIT 1 OFFSET 0
End_Query

Repeatable_database_view=>: next_player_low
With_attributes:
playerid, playername, teamname, avgpoints
Query:
SELECT playerid, playername, teamname, avgpoints
  FROM sum_per_player
 ORDER BY avgpoints ASC, playerid ASC
 LIMIT 1 OFFSET %s
With_arguments: next_low_index_starting_at_1


########################### 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
>SQL:  "UPDATE sum_per_player "
-SQL:  "   SET avgpoints = round(totalpoints*1.0/totalgames, 2) "
<SQL:  " WHERE playerid = %s "                        game.playerid


Case: Finished
Print: "    "
```

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

Case: Display_first_teams
Print: "Teams with most points (%s):"
first_team_most.totalpoints
Print: "- %s "                                             first_team_most.teamname
Print: "- %s "                                             next_team_most.teamname


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

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

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

Case: Finished
Print: "      "


# DISPLAY ONE PLAYER OR FIRST 2 PLAYERS WITH HIGHEST AVERAGE
GoalAttribute: Eval_High1

Case: Display_first_players
Print: "Players with highest average (%s):"            first_player_high.avgpoints
Print: "- %s (team: %s)"                               first_player_high.playername
first_player_high.teamname
Print: "- %s (team: %s)"   next_player_high.playername next_player_high.teamname


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

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

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

Case: Finished
Print: "      "
```

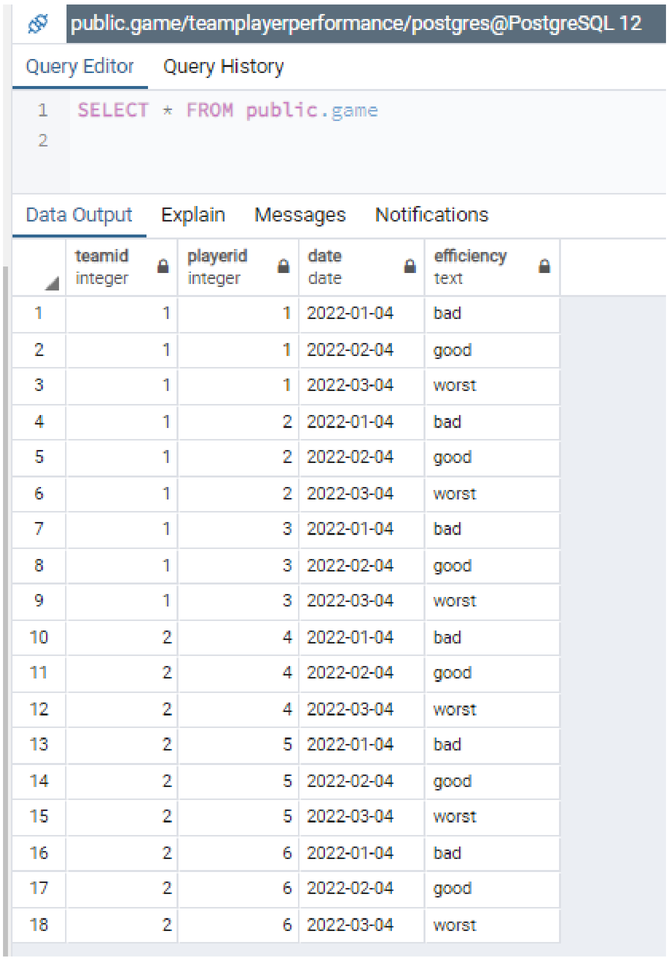
```
# DISPLAY ONE PLAYER OR FIRST 2 PLAYERS WITH LOWEST AVERAGE
GoalAttribute: Eval_Low1

Case: Display_first_players
Print: "Players with lowest average (%s):"             first_player_low.avgpoints
Print: "- %s (team: %s)"   first_player_low.playername first_player_low.teamname
Print: "- %s (team: %s)"   next_player_low.playername  next_player_low.teamname

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


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

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

Case: Finished
Print: "      "


Initial_database_setup: delete_table_sum_per_team
Query:
    DROP TABLE sum_per_team
End_Query

Initial_database_setup: initialize_sum_per_team_from_games
Query:
    CREATE TABLE sum_per_team AS
    SELECT distinct a.teamid, b.name AS teamname, 0 as totalpoints
      FROM game AS a
      JOIN team AS b on (a.teamid = b.id)
     ORDER BY a.teamid
End_Query


Initial_database_setup: delete_table_sum_per_player
Query:
    DROP TABLE sum_per_player
End_Query


Initial_database_setup: initialize_sum_per_player_from_games
Query:
    CREATE TABLE sum_per_player AS
    SELECT distinct a.playerid, b.name AS playername, c.name AS teamname,
           0 AS totalgames, 0 AS totalpoints, 0.0 AS avgpoints
      FROM game AS a
      JOIN player AS b on (a.playerid = b.id)
      JOIN team AS c on (a.teamid = c.id)
     ORDER BY a.playerid
End_Query
```

Testrun 1 (using data from the website):

```
PostgreSQL_database: "teamplayerperformance"
Proposition: 'Next Game to process?'
Proposition: 'First team with most points'
Proposition: 'Next team with most points'
Proposition: 'First player with highest average'
Proposition: 'Next player with highest average'
Proposition: 'First player with lowest average'
Proposition: 'Next player with lowest average'
Attribute: Efficiency  Type: Text
Attribute: points      Type: Integer
Attribute: first_team_most.totalpoints    Type: Integer
Attribute: next_team_most.totalpoints     Type: Integer
Attribute: first_player_high.avgpoints    Type: Real
Attribute: next_player_high.avgpoints     Type: Real
Attribute: first_player_low.avgpoints     Type: Real
Attribute: next_player_low.avgpoints      Type: Real
Attribute: next_most_index_starting_at_1  Type: Integer
Attribute: next_high_index_starting_at_1  Type: Integer
Attribute: next_low_index_starting_at_1   Type: Integer
Database_view: game
Database_view: first_team_most
Repeatable_database_view=>: next_team_most
Database_view: first_player_high
Repeatable_database_view=>: next_player_high
Database_view: first_player_low
Repeatable_database_view=>: next_player_low
GoalAttribute: Eval_Game
GoalAttribute: Eval_Team1
GoalAttribute: Eval_Team2
GoalAttribute: Eval_High1
GoalAttribute: Eval_High2
GoalAttribute: Eval_Low1
GoalAttribute: Eval_Low2
Initial_database_setup: delete_table_sum_per_team
Initial_database_setup: initialize_sum_per_team_from_games
Initial_database_setup: delete_table_sum_per_player
Initial_database_setup: initialize_sum_per_player_from_games

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!

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

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

Time elapsed: 0:00:01.473003
```

Testrun 2 (the special test case of solution 2):

Now all players get player Green's performance score:



```
SELECT * FROM public.game
```

	teamid integer	playerid integer	date date	efficiency text
1	1	1	2022-01-04	bad
2	1	1	2022-02-04	good
3	1	1	2022-03-04	worst
4	1	2	2022-01-04	bad
5	1	2	2022-02-04	good
6	1	2	2022-03-04	worst
7	1	3	2022-01-04	bad
8	1	3	2022-02-04	good
9	1	3	2022-03-04	worst
10	2	4	2022-01-04	bad
11	2	4	2022-02-04	good
12	2	4	2022-03-04	worst
13	2	5	2022-01-04	bad
14	2	5	2022-02-04	good
15	2	5	2022-03-04	worst
16	2	6	2022-01-04	bad
17	2	6	2022-02-04	good
18	2	6	2022-03-04	worst

```
Teams with most points (-15):
- Mustungs
- Eagles

Players with highest average (-1.67):
- Brown (team: Mustungs)
- Robinson (team: Mustungs)
- Smith (team: Mustungs)
- Black (team: Eagles)
- White (team: Eagles)
- Green (team: Eagles)

Players with lowest average (-1.67):
- Brown (team: Mustungs)
- Robinson (team: Mustungs)
- Smith (team: Mustungs)
- Black (team: Eagles)
- White (team: Eagles)
- Green (team: Eagles)

Time elapsed: 0:00:01.562221
```

Concluding remark

I implemented the simplification of the SQL queries in this solution first in SQLite and then in PostgreSQL under the assumption that these simplified queries would produce the same results in both tools.

That turned out to be a misconception for a while when I ran the second test run in PostgreSQL:

```
Players with highest average (-1.67):
- Brown (team: Mustungs)
- Brown (team: Mustungs)
- Smith (team: Mustungs)
- Black (team: Eagles)
- White (team: Eagles)
- Green (team: Eagles)

Players with lowest average (-1.67):
- Brown (team: Mustungs)
- Brown (team: Mustungs)
- Smith (team: Mustungs)
- Black (team: Eagles)
- White (team: Eagles)
- Green (team: Eagles)
```

Some research learned that PostgreSQL requires just a little more information in a query than SQLite:

```
SELECT playerid, playername, teamname, avgpoints
  FROM sum_per_player
 ORDER BY avgpoints DESC, playerid ASC
 LIMIT 1 OFFSET %s
```

and that had to be added to 6 queries.

Not entirely consistently, without this addition, PostgreSQL provides the same value for offset 0 as offset 1 (namely Brown).

But even with this addition, this remains a query that can be easily understood by anyone after a simple beginner's course in SQL.