

Query Execution Analysis

Query 1. Finding the best goalkeeper







First, to find the goalkeeper, we have filtered down to skills like GKReflexes, GK Diving, Positioning, GK Handling, GK Kicking, Sprint Speed from the skills table, and the position is chosen from the player table. The GK was searched in the player table based on the position id and the average of the skills is calculated. If the players' skills are greater than the average value, the contract valid is less than 2021 and the market value should be around 10M. This query is executed and a series of tuples are found with the defined constraints.

```
WITH all_gk AS ( -- Create a temporary table for all GKs
    SELECT 'All Players', 'ALL GKs', 'Contract Valid Until',
        ROUND(AVG(s.gk_reflexes), 1) AS AVG_Reflexes, ROUND(AVG(s.gk_diving), 1) AS
    AVG_Diving,
        ROUND(AVG(s.Positioning), 1) AS AVG_Positioning, ROUND(AVG(s.gk_handling), 1) AS
    AVG_Handling,
        ROUND(AVG(s.gk_kicking), 1) AS AVG_Kicking, ROUND(AVG(s.sprint_speed), 1) AS
    AVG_SprintSpeed
    FROM skills AS s, positions as ps, player as p
    WHERE s.player_id = p.player_id AND ps.pos_id = p.position_id
    AND ps.position_name = 'Goal_Keeper'
)
```

```
SELECT p1.player_id, p1.name, ps1.position_name, p1.contract_valid_until, p1.market_value,
    s1.gk_reflexes, s1.gk_diving, s1.Positioning, s1.gk_handling, s1.gk_kicking, s1.sprint_speed
FROM player AS p1, positions as ps1, skills as s1
WHERE ps1.position_name = 'Goal_Keeper'
    AND s1.gk_reflexes > (SELECT AVG_Reflexes FROM all_gk)
    AND s1.gk_diving > (SELECT AVG_Diving FROM all_gk)
    AND s1.Positioning > (SELECT AVG_Positioning FROM all_gk)
    AND s1.gk_handling > (SELECT AVG_Handling FROM all_gk)
    AND s1.gk_kicking > (SELECT AVG_Kicking FROM all_gk)
    AND s1.sprint_speed > (SELECT AVG_SprintSpeed FROM all_gk)
    AND p1.contract_valid_until < 2021
    AND p1.market_value LIKE '1_M';
```

player_id integer	name character varying	position_name character varying	contract_valid_until integer	market_value character varying	gk_reflexes integer	gk_diving integer	positioning integer	gk_handling integer	gk_kicking integer	sprint_speed integer
75	Fernandinho	Goal_Keeper	2020	€18M	23	22	54	22	28	68
75	Fernandinho	Goal_Keeper	2020	€18M	37	27	92	25	31	75
75	Fernandinho	Goal_Keeper	2020	€18M	33	21	62	17	18	79
203	D. Subašić	Goal_Keeper	2020	€13M	23	22	54	22	28	68
203	D. Subašić	Goal_Keeper	2020	€13M	37	27	92	25	31	75
203	D. Subašić	Goal_Keeper	2020	€13M	33	21	62	17	18	79
221	S. Mandanda	Goal_Keeper	2020	€13M	23	22	54	22	28	68
221	S. Mandanda	Goal_Keeper	2020	€13M	37	27	92	25	31	75
221	S. Mandanda	Goal_Keeper	2020	€13M	33	21	62	17	18	79
303	J. Henderson	Goal_Keeper	2020	€18M	23	22	54	22	28	68

To analyze the statistics of the query, EXPLAIN keyword is executed along with the query. The cost estimated was 965.7 and the rows fetched are 33000.

#	Node	Plan	Rows
1.		Nested Loop Inner Join (cost=965.7..2863.73 rows=33000 width=78)	33000
2.		Aggregate (cost=965.56..965.58 rows=1 width=288)	1
3.		Hash Inner Join (cost=470.04..964.25 rows=86 width=24)  Hash Cond: (s.player_id = p.player_id)	86
4.		Seq Scan on skills as s (cost=0..425.07 rows=18207 width=28)	18207
5.		Hash (cost=468.97..468.97 rows=86 width=4)	86
6.		Hash Inner Join (cost=25.95..468.97 rows=86 width=4)  Hash Cond: (p.position_id = ps.pos_id)	86
7.		Seq Scan on player as p (cost=0..395.07 rows=18207 width=8)	18207
8.		Hash (cost=25.88..25.88 rows=6 width=4)	6
9.		Seq Scan on positions as ps (cost=0..25.88 rows=6 width=4)  Filter: ((position_name)::text = 'Goal Keeper'::text)	6
10.		CTE Scan (cost=0..0.02 rows=1 width=32)	1
11.		CTE Scan (cost=0..0.02 rows=1 width=32)	1
12.		CTE Scan (cost=0..0.02 rows=1 width=32)	1
13.		CTE Scan (cost=0..0.02 rows=1 width=32)	1
14.		CTE Scan (cost=0..0.02 rows=1 width=32)	1
15.		CTE Scan (cost=0..0.02 rows=1 width=32)	1
16.		Seq Scan on player as p1 (cost=0..486.11 rows=220 width=22)  Filter: ((contract_valid_until < 2021) AND ((market_value)::text ~ ' 1 M'::text))	220
17.		Materialize (cost=0..999.79 rows=150 width=56)	150
18.		Nested Loop Inner Join (cost=0..999.04 rows=150 width=56)	150
19.		Seq Scan on skills as s1 (cost=0..971.28 rows=25 width=24)  Filter: (((gk_reflexes)::numeric > \$1) AND ((gk_diving)::numeric > \$2) AND ((positioning)::numeric > \$3) AND ((gk_handling)::numeric > \$4) AND ((gk_kicking)::numeric > \$5) AND ((sprint_speed)::numeric > \$6))	25
20.		Materialize (cost=0..25.91 rows=6 width=32)	6
21.		Seq Scan on positions as ps1 (cost=0..25.88 rows=6 width=32)  Filter: ((position_name)::text = 'Goal Keeper'::text)	6

To improve the performance of the cost indexing was performed. Indexes were created on the players' table and the positions table. The indexing was done on the player id and the position id of the player table. It was also done on the position id of the position table. The cost was drastically decreased to 761.73 and the rows fetched were 5500.

#	Node	Rows
1.	Nested Loop Inner Join (cost=761.73..2289.45 rows=5500 width=78)	5500
2.	Aggregate (cost=761.59..761.61 rows=1 width=288)	1
3.	Hash Inner Join (cost=251.38..751.46 rows=674 width=24) ■ Hash Cond: (s.player_id = p.player_id)	674
4.	Seq Scan on skills as s (cost=0..425.07 rows=18207 width=28)	18207
5.	Hash (cost=242.95..242.95 rows=674 width=4)	674
6.	Nested Loop Inner Join (cost=13.51..242.95 rows=674 width=4)	674
7.	Seq Scan on positions as ps (cost=0..1.27 rows=1 width=4) ■ Filter: ((position_name)::text = 'Goal_Keeper'::text)	1
8.	Bitmap Heap Scan on player as p (cost=13.51..234.94 rows=674 width=8) ■ Recheck Cond: (position_id = ps.pos_id)	674
9.	Bitmap Index Scan using players_position_index (cost=0..13.34 rows=674 width=0) ■ Index Cond: (position_id = ps.pos_id)	674
10.	CTE Scan (cost=0..0.02 rows=1 width=32)	1
11.	CTE Scan (cost=0..0.02 rows=1 width=32)	1
12.	CTE Scan (cost=0..0.02 rows=1 width=32)	1
13.	CTE Scan (cost=0..0.02 rows=1 width=32)	1
14.	CTE Scan (cost=0..0.02 rows=1 width=32)	1
15.	CTE Scan (cost=0..0.02 rows=1 width=32)	1
16.	Seq Scan on player as p1 (cost=0..486.11 rows=220 width=22) ■ Filter: ((contract_valid_until < 2021) AND ((market_value)::text ~ '_1_M'::text))	220
17.	Materialize (cost=0..972.93 rows=25 width=56)	25
18.	Nested Loop Inner Join (cost=0..972.8 rows=25 width=56)	25
19.	Seq Scan on positions as ps1 (cost=0..1.27 rows=1 width=32) ■ Filter: ((position_name)::text = 'Goal_Keeper'::text)	1
20.	Seq Scan on skills as s1 (cost=0..971.28 rows=25 width=24) ■ Filter: (((gk_reflexes)::numeric > \$2) AND ((gk_diving)::numeric > \$3) AND ((positioning)::numeric > \$4) AND ((gk_handling)::numeric > \$5) AND ((gk_kicking)::numeric > \$6) AND ((sprint_speed)::numeric > \$7))	25

Query 2. What were the most expensive players

*select * from player where market_value=(select Max(market_value) from player);*

Output:

id integer	player_id [PK] integer	name character varying	age integer	nationality character varying	wages character varying	market_value character varying	jersey_number integer	joined_date date	contract_valid_until integer	position_id integer
171919	102	Naldo	35	Brazil	€38K	€9M	29	2016-07-01	2020	1
120533	108	Pepe	35	Portugal	€57K	€9M	3	2017-07-04	2019	17
214584	111	F. Armani	31	Argentina	€22K	€9M	1	2018-01-12	2022	26
230312	240	Gabri Prestão	26	Brazil	€24K	€9M	12	2018-01-01	2021	26
183498	306	M. Parolo	33	Italy	€59K	€9M	16	2014-07-01	2020	16

Query Analysis:

#	Node	Rows	
		Plan	
	1.	Seq Scan on player as player (cost=440.6..881.18 rows=84 width=60)Filter: ((market_value)::text = \$0)	84
	2.	Aggregate (cost=440.59..440.6 rows=1 width=32)	1
	3.	Seq Scan on player as player_1 (cost=0..395.07 rows=18207 width=7)	18207

Query 3. What's the age distribution like? How is it related to the player's overall rating?

*select distinct p.age, r.overall_rating from player as p natural join ratings as r
group by p.age, r.overall_rating;*

Output:

	age integer	overall_rating integer
1	20	62
2	34	62
3	23	83
4	40	78
5	24	51
6	31	59
7	16	62
8	21	61
9	28	68
10	21	77

Query Analysis:

#	Node	Rows	
		Plan	
	1.	Aggregate (cost=1080.45..1094.37 rows=1392 width=8)	1392
	2.	Aggregate (cost=1059.57..1073.49 rows=1392 width=8)	1392
	3.	Hash Inner Join (cost=622.66..968.53 rows=18207 width=8)Hash Cond: (r.player_id = p.player_id)	18207
	4.	Seq Scan on ratings as r (cost=0..298.07 rows=18207 width=8)	18207
	5.	Hash (cost=395.07..395.07 rows=18207 width=8)	18207
	6.	Seq Scan on player as p (cost=0..395.07 rows=18207 width=8)	18207

Query 4. Which teams had the best average overall stats?

*select p.club, avg(r.overall_rating) from player as p natural join ratings as r
where p.club is not null
group by p.club;*

Output:

	club character varying	avg numeric
1	Guangzhou Evergrande Taobao FC	66.7142857142857143
2	SC Braga	74.8214285714285714
3	Henan Jianye FC	60.8928571428571429
4	FC Basel 1893	68.7777777777777778
5	Empoli	68.3333333333333333
6	Al Hazem	61.5666666666666667
7	Santos Laguna	68.7692307692307692
8	AD Alcorcón	67.4137931034482759
9	Gamba Osaka	61.1333333333333333
10	Chamois Niortais Football Club	63.0000000000000000

Query Analysis:

		Rows
#	Node	Plan
1.	Aggregate	(cost=1092.35..1100.49 rows=651 width=46)
2.	Hash Inner Join	(cost=656.64..1002.52 rows=17966 width=18)Hash Cond: (r.player_id = p.player_id)
3.	Seq Scan on ratings as r	(cost=0..298.07 rows=18207 width=8)
4.	Hash	(cost=432.07..432.07 rows=17966 width=18)
5.	Seq Scan on player as p	(cost=0..432.07 rows=17966 width=18)Filter: (club IS NOT NULL)

Query 5. Players with their average ratings

```
select distinct p.name, r.overall_rating
from player p, ratings r
where p.player_id = r.player_id
order by r.overall_rating desc;
```

Output:

	name character varying	overall_rating integer
1	Cristiano Ronaldo	94
2	L. Messi	94
3	Neymar Jr	92
4	De Gea	91
5	E. Hazard	91
6	K. De Bruyne	91
7	L. Modrić	91
8	L. Suárez	91
9	Sergio Ramos	91
10	D. Godín	90

Query Analysis:

#	Node	Rows	
		Plan	
	1.	Unique (cost=2256.88..2393.43 rows=18207 width=15)	18207
	2.	Sort (cost=2256.88..2302.4 rows=18207 width=15)	18207
	3.	Hash Inner Join (cost=622.66..968.53 rows=18207 width=15)Hash Cond: (r.player_id = p.player_id)	18207
	4.	Seq Scan on ratings as r (cost=0..298.07 rows=18207 width=8)	18207
	5.	Hash (cost=395.07..395.07 rows=18207 width=15)	18207
	6.	Seq Scan on player as p (cost=0..395.07 rows=18207 width=15)	18207

Query 6. Top 10 player's demographics

```
select p.name, pb.height, pb.weight, pb.stamina, pb.strength, pb.vision, ps.position_name
from player p, player_bio pb, positions ps
where p.player_id = pb.player_id
AND p.position_id = ps.pos_id
limit 10;
```

Output:

	name character varying	height character varying	weight character varying	stamina integer	strength integer	vision integer	position_name character varying
1	C. Deac	5'10	163lbs	84	70	69	Right_winger
2	B. Höwedes	6'2	170lbs	61	78	57	Left_winger
3	F. Soyalp	5'9	161lbs	65	61	66	Center_attacking_midfielder
4	K. Wimmer	6'2	187lbs	65	79	59	Right_midfielder
5	E. Bilen	6'1	176lbs	21	61	29	Right_attacking_midfielder
6	T. Guidara	5'8	150lbs	78	59	40	Left_midfielder
7	K. Schmeichel	6'2	196lbs	34	64	59	Right_back
8	L. Gamba	5'7	154lbs	68	54	78	Left Centre Midfielder
9	A. Turgeman	5'10	163lbs	80	76	46	Right_back
10	V. Slivka	6'3	185lbs	55	68	58	Striker

Query Analysis:

#	Node	Rows	
		Plan	
1.		Limit (cost=0.42..5.81 rows=10 width=66)	10
2.		Nested Loop Inner Join (cost=0.42..9798.33 rows=18207 width=66)	18207
3.		Nested Loop Inner Join (cost=0.29..6975.13 rows=18207 width=38)	18207
4.		Seq Scan on player_bio as pb (cost=0..362.07 rows=18207 width=27)	18207
5.		Index Scan using player_id on player as p (cost=0.29..0.36 rows=1 width=19) Index Cond: (player_id = pb.player_id)	1
6.		Index Scan using pos_id on positions as ps (cost=0.14..0.16 rows=1 width=36) Index Cond: (pos_id = p.position_id)	1

