

1. Screenshot of connecting to GCP

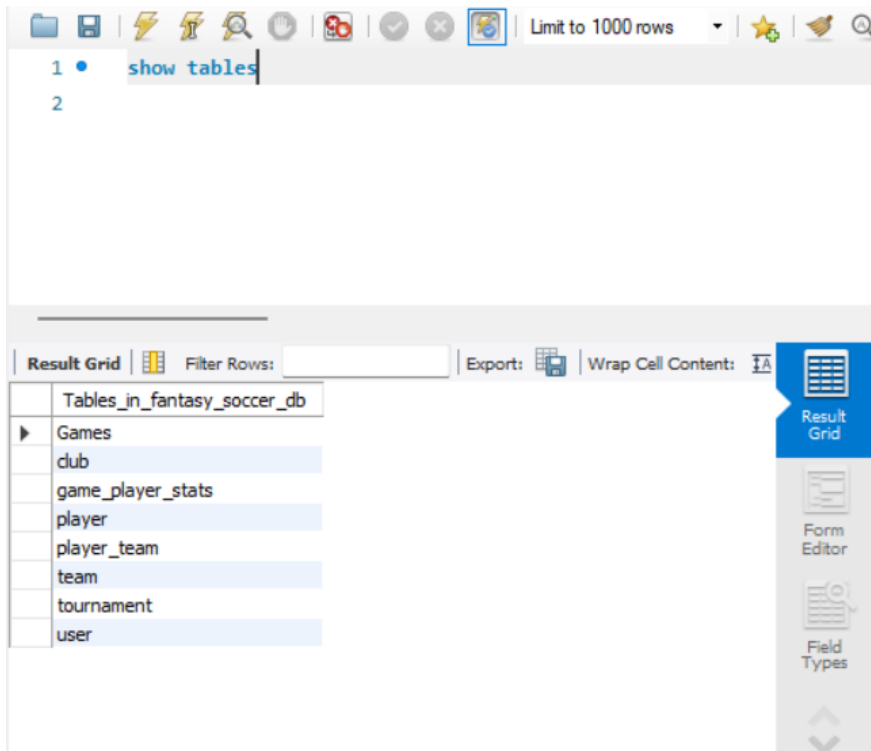
The screenshot shows the Google Cloud SQL console for an instance named 'PlyRank - Fantasy Soccer'. The instance is in the 'PRIMARY INSTANCE' state. The 'Overview' tab is selected, showing a 'Chart' for 'CPU utilisation' and a 'Connect to this instance' section. The 'Connect to this instance' section displays the 'Public IP address' as '34.121.2.8' and the 'Connection name' as 'apt-weft-378020:us-central1:plyrank'. A 'Need help connecting?' link is provided. Below the console, a terminal window shows the MySQL command-line interface. The terminal output includes the MySQL version (8.0.26-google), the MySQL connection ID (97), and the MySQL server version (8.0.26-google). The terminal also shows the creation of the 'fantasy_soccer_db' database, the use of the 'fantasy_soccer_db' database, the creation of the 'club' table, and the execution of a query to select the count of rows in the 'club' table, which returns 411. The terminal also shows the creation of the 'user' table and the execution of a query to select the count of rows in the 'user' table, which returns 5.

Google Cloud SQL console for instance 'PlyRank - Fantasy Soccer'. The instance is in the 'PRIMARY INSTANCE' state. The 'Overview' tab is selected, showing a 'Chart' for 'CPU utilisation' and a 'Connect to this instance' section. The 'Connect to this instance' section displays the 'Public IP address' as '34.121.2.8' and the 'Connection name' as 'apt-weft-378020:us-central1:plyrank'. A 'Need help connecting?' link is provided. Below the console, a terminal window shows the MySQL command-line interface. The terminal output includes the MySQL version (8.0.26-google), the MySQL connection ID (97), and the MySQL server version (8.0.26-google). The terminal also shows the creation of the 'fantasy_soccer_db' database, the use of the 'fantasy_soccer_db' database, the creation of the 'club' table, and the execution of a query to select the count of rows in the 'club' table, which returns 411. The terminal also shows the creation of the 'user' table and the execution of a query to select the count of rows in the 'user' table, which returns 5.

The screenshot shows the MySQL Workbench interface. The 'Query Editor' is open, displaying a 'CREATE TABLE' statement for a table named 'tournament'. The statement includes columns for 'id', 'name', 'start_date', 'venue', 'type', 'url', and 'primary key'. The 'Result Grid' is visible, showing the results of a query executed in the 'club' table. The query is 'select count(*) from club;', and the result is a single row with the value '411'. The 'Output' pane at the bottom shows the execution log, including the time taken to execute the query (0.031 sec) and the number of rows returned (1 row(s)).

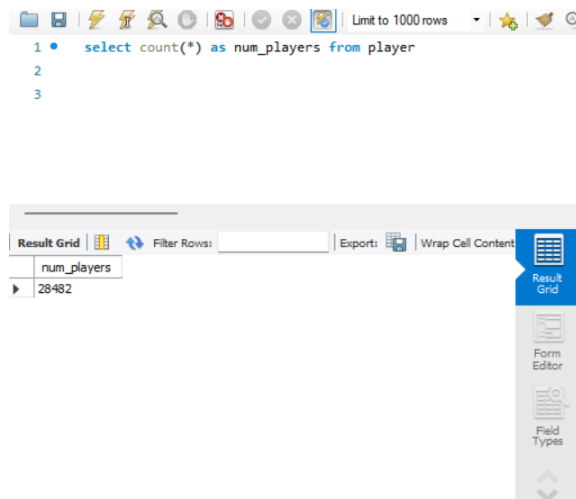
MySQL Workbench interface showing a query editor with a 'CREATE TABLE' statement and a result grid showing the count of rows in the 'club' table. The query is 'select count(*) from club;', and the result is a single row with the value '411'. The 'Output' pane at the bottom shows the execution log, including the time taken to execute the query (0.031 sec) and the number of rows returned (1 row(s)).

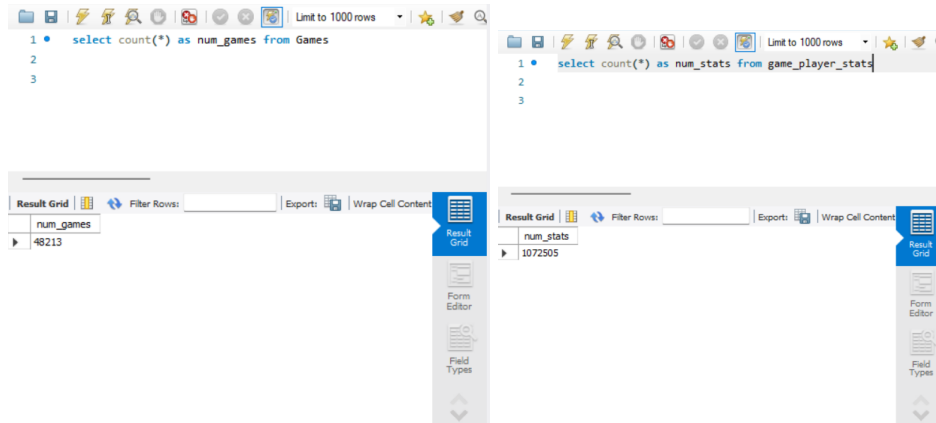
2. Screenshot of implemented tables



3. Screenshot for data insertion in GCP SQL

a.





4. Advanced Queries and Results

- a. Given a tournament name , find all the stats of players in the tournament

```
SELECT player_id,
       name,
       sum(goals) AS goals,
       sum(assists) AS assists,
       sum(yellow_cards) AS yellow_cards,
       sum(red_cards) AS red_cards,
       sum(goals) + sum(assists) AS total_points
FROM game_player_stats
INNER JOIN player ON player.id = game_player_stats.player_id
WHERE game_id in
      (SELECT id
       FROM Games
       WHERE tournament_id = (SELECT id FROM tournament WHERE name =
'LaLiga'))
GROUP BY player_id
ORDER BY total_points DESC
LIMIT 15;
```

MySQL Workbench

CS411 x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

fantasy_soccer_db

Tables

club

Columns

id

name

url

Indexes

Foreign Keys

Triggers

game_player_stats

Games

player

player_team

team

tournament

user

Views

Stored Procedures

Functions

sys

Administration Schemas

Information

Schema: fantasy_soccer_db

Query 1

Limit to 1000 rows

```

1 • SELECT player_id,
2     name,
3     sum(goals) AS goals,
4     sum(assists) AS assists,
5     sum(yellow_cards) AS yellow_cards,
6     sum(red_cards) AS red_cards,
7     sum(goals) + sum(assists) AS total_points
8 FROM game_player_stats
9 INNER JOIN player ON player.id = game_player_stats.player_id
10 WHERE game_id in
11 (SELECT id
12  FROM Games
13   WHERE tournament_id = (SELECT id FROM tournament WHERE name = 'LaLiga'))
14 GROUP BY player_id
15 ORDER BY total_points DESC
16 LIMIT 15;
17

```

Result Grid

Filter Rows:

Exports

Wrap Cell Contents

Fetch rows:

player_id	name	goals	assists	yellow_cards	red_cards	total_points
28003	Lionel Messi	226	107	27	0	333
44352	Luis Suárez	173	76	48	0	249
18922	Karim Benzema	154	73	5	0	227
125781	Antoine Griezmann	125	56	35	1	181
8198	Cristiano Ronaldo	134	39	11	1	173
72047	Iago Aspas	127	45	59	0	172
177467	Gerard Moreno	105	30	33	0	135
59561	Dani Parejo	53	57	71	2	110
351478	Mikel Oyarzabal	65	40	12	1	105
34601	Raúl García	69	28	75	1	97
255508	Iñaki Williams	59	37	22	0	96
252677	José Luis Morales	61	32	34	0	93
68290	Neymar	59	34	19	1	93
39381	Gareth Bale	63	28	22	1	91
122155	Willian José	69	20	24	1	89

Result 5 x

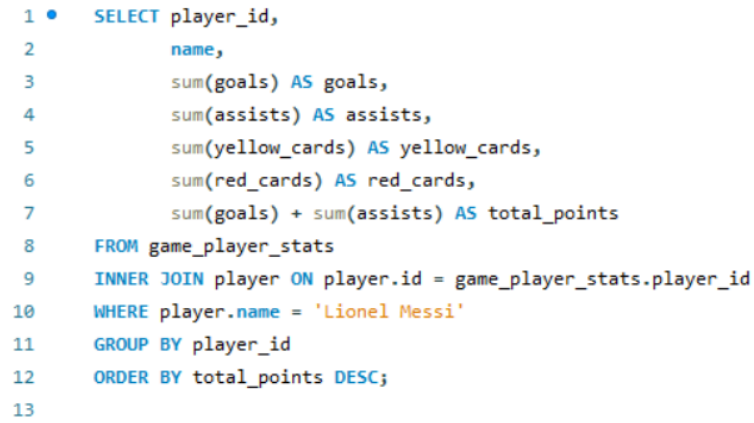
Read Only

b. Fetch lifetime stats of a player

```

SELECT player_id,
       name,
       sum(goals) AS goals,
       sum(assists) AS assists,
       sum(yellow_cards) AS yellow_cards,
       sum(red_cards) AS red_cards,
       sum(goals) + sum(assists) AS total_points
FROM game_player_stats
INNER JOIN player ON player.id = game_player_stats.player_id
WHERE player.name = 'Lionel Messi'
GROUP BY player_id
ORDER BY total_points DESC;

```



Result Grid							
			Filter Rows:		Export:		Wrap Cell Content:
	player_id	name	goals	assists	yellow_cards	red_cards	total_points
▶	28003	Lionel Messi	329	178	44	1	507

- c. Fetch top 15 players based on goals and assists for a club

```
Select player_id, player.name, club.name, sum(goals) as goals, sum(assists) as
assists
from game_player_stats
    inner join player on player.id = game_player_stats.player_id
    inner join club on club.id = player.club_id
where club.name = 'Fc Barcelona'
group by player_id
order by goals desc, assists desc
limit 15;
```

	player_id	name	name	goals	assists
1	38253	Robert Lewandowski	Fc Barcelona	336	71
2	26399	Sergio Agüero	Fc Barcelona	170	39
3	411295	Raphinha	Fc Barcelona	56	35
4	288230	Ousmane Dembélé	Fc Barcelona	51	55
5	294808	Franck Kessié	Fc Barcelona	43	16
6	112515	Marcos Alonso	Fc Barcelona	34	29
7	398184	Ferran Torres	Fc Barcelona	30	19
8	18944	Gerard Piqué	Fc Barcelona	30	7
9	69751	Jordi Alba	Fc Barcelona	20	78
10	326330	Frenkie de Jong	Fc Barcelona	19	27
11	683840	Pedri	Fc Barcelona	15	5
12	466810	Ansu Fati	Fc Barcelona	14	7
13	85370	Sergi Roberto	Fc Barcelona	12	34
14	15951	Dani Alves	Fc Barcelona	11	42
15	411975	Jules Kounde	Fc Barcelona	10	8

5. Indexing analysis

- a. Fetch lifetime stats of a player

No Index:

```
1 • explain analyze
2 SELECT player_id,
3         name,
4         sum(goals) AS goals,
5         sum(assists) AS assists,
6         sum(yellow_cards) AS yellow_cards,
7         sum(red_cards) AS red_cards,
8         sum(goals) + sum(assists) AS total_points
9 FROM game_player_stats
10 INNER JOIN player ON player.id = game_player_stats.player_id
11 WHERE player.name = 'Lionel Messi'
12 GROUP BY player_id
13 ORDER BY total_points DESC;
14
```

Form Editor | Navigate: ⏮ ⏪ 1 / 1 ⏩ ⏭

EXPLAIN:

```
-> Sort: (sum(game_player_stats.goals) + sum(game_player_stats.assists)) DESC (actual
time=11.951..11.951 rows=1 loops=1)
-> Table scan on <temporary> (actual time=0.003..0.003 rows=1 loops=1)
-> Aggregate using temporary table (actual time=11.922..11.922 rows=1 loops=1)
```

Index on player(name):

```
1 • create index player_name on player(name);
2 • explain analyze
3 SELECT player_id,
4         name,
5         sum(goals) AS goals,
6         sum(assists) AS assists,
7         sum(yellow_cards) AS yellow_cards,
8         sum(red_cards) AS red_cards,
9         sum(goals) + sum(assists) AS total_points
10 FROM game_player_stats
11 INNER JOIN player ON player.id = game_player_stats.player_id
12 WHERE player.name = 'Lionel Messi'
13 GROUP BY player_id
14 ORDER BY total_points DESC;
15
```

Form Editor | Navigate: ⏮ ⏪ ⏩ ⏭

EXPLAIN: -> Sort: (sum(game_player_stats.goals) + sum(game_player_stats.assists)) DESC (actual time=0.890..0.890 rows=1 loops=1)
-> Table scan on <temporary> (actual time=0.001..0.001 rows=1 loops=1)
-> Aggregate using temporary table (actual time=0.845..0.845 rows=1 loops=1)

Index on game_player_stats(red_cards):

```
1 • create index goals_idx on game_player_stats(goals);
2 • explain analyze
3 SELECT player_id,
4         name,
5         sum(goals) AS goals,
6         sum(assists) AS assists,
7         sum(yellow_cards) AS yellow_cards,
8         sum(red_cards) AS red_cards,
9         sum(goals) + sum(assists) AS total_points
10 FROM game_player_stats
11 INNER JOIN player ON player.id = game_player_stats.player_id
12 WHERE player.name = 'Lionel Messi'
13 GROUP BY player_id
14 ORDER BY total_points DESC;
15
```

Form Editor | Navigate: ⏮ ⏪ 1 / 1 ⏩ ⏭

EXPLAIN: -> Sort: (sum(game_player_stats.goals) + sum(game_player_stats.assists)) DESC (actual time=12.801..12.801 rows=1 loops=1)
-> Table scan on <temporary> (actual time=0.004..0.004 rows=1 loops=1)
-> Aggregate using temporary table (actual time=12.737..12.737 rows=1 loops=1)
-> Nested loop inner join (cost=19779.21 rows=158487) (actual time=1.074..12.206 rows=395 loops=1)

Index on game_player_stats(goals):

```
1 • create index goals_idx on game_player_stats(goals);
2 • explain analyze
3 SELECT player_id,
4         name,
5         sum(goals) AS goals,
6         sum(assists) AS assists,
7         sum(yellow_cards) AS yellow_cards,
8         sum(red_cards) AS red_cards,
9         sum(goals) + sum(assists) AS total_points
10 FROM game_player_stats
11 INNER JOIN player ON player.id = game_player_stats.player_id
12 WHERE player.name = 'Lionel Messi'
13 GROUP BY player_id
14 ORDER BY total_points DESC;
15
```

Form Editor | Navigate: ⏮ ⏪ 1 / 1 ⏩ ⏭

EXPLAIN: -> Sort: (sum(game_player_stats.goals) + sum(game_player_stats.assists)) DESC (actual time=12.801..12.801 rows=1 loops=1)
-> Table scan on <temporary> (actual time=0.004..0.004 rows=1 loops=1)
-> Aggregate using temporary table (actual time=12.737..12.737 rows=1 loops=1)
-> Nested loop inner join (cost=19779.21 rows=158487) (actual time=1.074..12.206 rows=395 loops=1)

Analysis: For the lifetime stats of a player query, we chose the index on player(name) because the time spent is significantly shorter than without using any indices. The reduction in time is likely due to the WHERE command for the player 'Lionel Messi,' since we do not search the entire table for instances with the player name 'Lionel Messi' and instead use the index. The game_player_stats(red_cards) and game_player_stats(goals) are not a part of the WHERE command and have little effect on the time spent, and performance of the query .

Hence we would select adding index on name field in player table - **player(name)** for this query.

- b. Fetch players based on goals and assists for a given club
No Index

The screenshot shows a database IDE with a query editor and an execution plan. The query is as follows:

```

7  explain analyze select player_id, player.name, club.name, sum(goals) as goals, sum(assists) as assists
8  from game_player_stats
9      inner join player on player.id = game_player_stats.player_id
10     inner join club on club.id = player.club_id
11  where club.name = 'Fc Barcelona'
12  group by player_id
13  order by goals desc, assists desc
14  limit 15;

```

The execution plan shows the following steps:

- 1 -> Limit: 15 row(s) (actual time=104.324..104.326 rows=15 loops=1)
- > Sort: goals DESC, assists DESC, limit input to 15 row(s) per chunk (actual time=104.323..104.325 rows=15 loops=1)
- > Table scan on <temporary> (actual time=0.003..0.014 rows=36 loops=1)
- > Aggregate using temporary table (actual time=104.266..104.280 rows=36 loops=1)
- > Nested loop inner join (cost=22392.71 rows=158873) (actual time=36.245..98.743 rows=5467 loops=1)
- > Nested loop inner join (cost=3290.96 rows=2952) (actual time=33.204..34.521 rows=50 loops=1)
- > Filter: (club.name = 'Fc Barcelona') (cost=44.23 rows=41) (actual time=0.923..2.188 rows=1 loops=1)
- > Table scan on club (cost=44.23 rows=411) (actual time=0.906..2.127 rows=411 loops=1)
- > Index lookup on player using player_club_idx (club_id=club.id) (cost=71.99 rows=72) (actual time=32.279..32.325 rows=50 loops=1)
- > Index lookup on game_player_stats using PRIMARY (player_id=player.id) (cost=1.09 rows=54) (actual time=0.899..1.276 rows=109 loops=50)

Adding Index on goals field in game_palyer_stats : This helps reduce the sort time from 114ms to 7.781 ms

The screenshot shows a database IDE with a query window and an EXPLAIN output window. The query is as follows:

```
create index goals on game_player_stats(goals);
explain analyze select player_id, player.name, club.name, sum(goals) as goals, sum(assists) as assists
from game_player_stats
inner join player on player.id = game_player_stats.player_id
inner join club on club.id = player.club_id
where club.name = 'Fc Barcelona'
group by player_id
order by goals desc, assists desc
limit 15;
```

The EXPLAIN output shows the following execution plan:

```
1 -> Limit: 15 row(s) (actual time=7.782..7.785 rows=15 loops=1)
    -> Sort: goals DESC, assists DESC, limit input to 15 row(s) per chunk (actual time=7.781..7.783 rows=15 loops=1)
    -> Table scan on <temporary> (actual time=0.002..0.010 rows=36 loops=1)
    -> Aggregate using temporary table (actual time=7.734..7.745 rows=36 loops=1)
    -> Nested loop inner join (cost=19846.41 rows=158873) (actual time=0.276..2.833 rows=5467 loops=1)
    -> Nested loop inner join (cost=3155.24 rows=2952) (actual time=0.233..0.400 rows=50 loops=1)
    -> Filter: (club.name = 'Fc Barcelona') (cost=42.35 rows=41) (actual time=0.093..0.240 rows=1 loops=1)
    -> Table scan on club (cost=42.35 rows=411) (actual time=0.078..0.191 rows=411 loops=1)
    -> Index lookup on player using player_club_idx (club_id=club.id) (cost=68.73 rows=72) (actual time=0.138..0.155 rows=50 loops=1)
    -> Index lookup on game_player_stats using PRIMARY (player_id=player.id) (cost=0.27 rows=54) (actual time=0.011..0.042 rows=109 loops=50)
```

Adding Index on club name

It helps in reducing the cost on club table lookup . Now the lookup is for only one row compared to 411 rows without index.

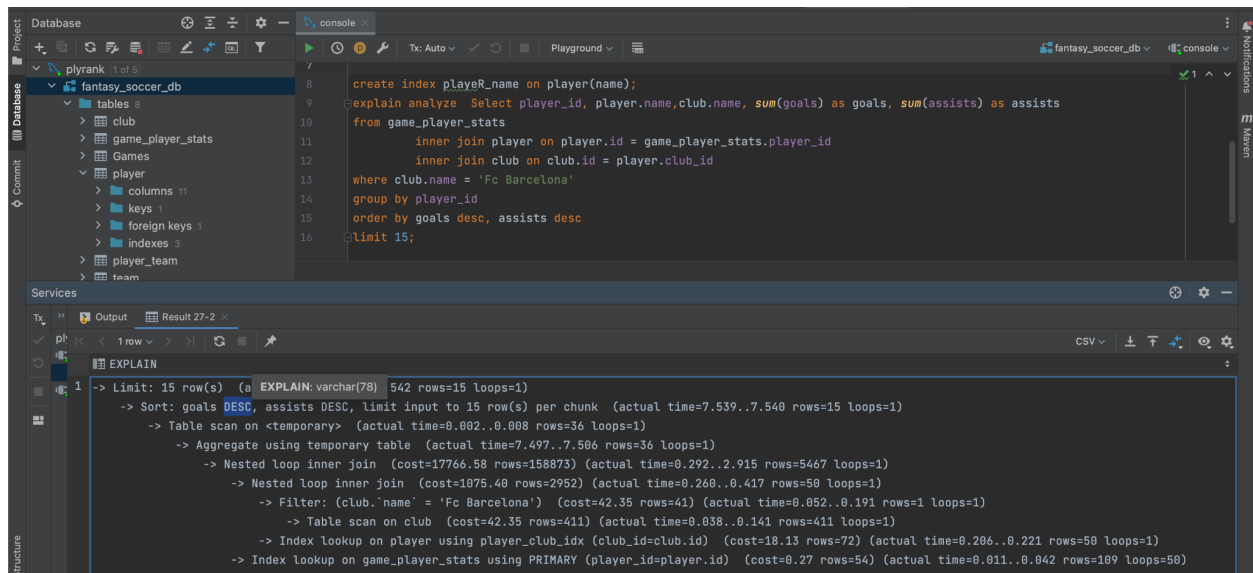
The screenshot shows the same database IDE with the same query, but now with an index on the club name. The query is as follows:

```
create index club_name on club(name);
explain analyze select player_id, player.name, club.name, sum(goals) as goals, sum(assists) as assists
from game_player_stats
inner join player on player.id = game_player_stats.player_id
inner join club on club.id = player.club_id
where club.name = 'Fc Barcelona'
group by player_id
order by goals desc, assists desc
limit 15;
```

The EXPLAIN output shows the following execution plan:

```
1 -> Limit: 15 row(s) (actual time=8.397..8.400 rows=15 loops=1)
    -> Sort: goals DESC, assists DESC, limit input to 15 row(s) per chunk (actual time=8.396..8.398 rows=15 loops=1)
    -> Table scan on <temporary> (actual time=0.003..0.015 rows=36 loops=1)
    -> Aggregate using temporary table (actual time=8.326..8.341 rows=36 loops=1)
    -> Nested loop inner join (cost=482.20 rows=3866) (actual time=0.571..3.266 rows=5467 loops=1)
    -> Nested loop inner join (cost=76.89 rows=72) (actual time=0.533..0.563 rows=50 loops=1)
    -> Index lookup on club using club_name (name='Fc Barcelona') (cost=0.35 rows=1) (actual time=0.009..0.017 rows=1 loops=1)
    -> Index lookup on player using player_club_idx (club_id=club.id) (cost=75.74 rows=72) (actual time=0.522..0.541 rows=50 loops=1)
    -> Index lookup on game_player_stats using PRIMARY (player_id=player.id) (cost=0.35 rows=54) (actual time=0.012..0.046 rows=109 loops=50)
```

Adding index on name in player table - player.name - This doesn't help much



The screenshot shows a database IDE with a project named 'fantasy_soccer_db'. The left sidebar shows a tree view of the database schema, including tables like 'club', 'game_player_stats', 'player', 'player_team', and 'team'. The main editor displays a SQL query in the 'console' tab:

```
create index playerR_name on player(name);
explain analyze select player_id, player.name, club.name, sum(goals) as goals, sum(assists) as assists
from game_player_stats
inner join player on player.id = game_player_stats.player_id
inner join club on club.id = player.club_id
where club.name = 'Fc Barcelona'
group by player_id
order by goals desc, assists desc
limit 15;
```

The 'Services' panel at the bottom shows the 'EXPLAIN' output for the query. The output indicates that the query is limited to 15 rows and uses a nested loop inner join. The most significant performance improvement is seen in the 'Index lookup on player using player_club_idx (club_id=club.id)' step, which reduces the row search from 411 rows to 1 row.

```
1 -> Limit: 15 row(s) (a) EXPLAIN: varchar(78) 542 rows=15 loops=1
-> Sort: goals DESC, assists DESC, limit input to 15 row(s) per chunk (actual time=7.539..7.540 rows=15 loops=1)
-> Table scan on <temporary> (actual time=0.002..0.008 rows=36 loops=1)
-> Aggregate using temporary table (actual time=7.497..7.506 rows=36 loops=1)
-> Nested loop inner join (cost=17766.58 rows=158873) (actual time=0.292..2.915 rows=5467 loops=1)
-> Nested loop inner join (cost=1075.40 rows=2952) (actual time=0.260..0.417 rows=50 loops=1)
-> Filter: (club.name = 'Fc Barcelona') (cost=42.35 rows=41) (actual time=0.052..0.191 rows=1 loops=1)
-> Table scan on club (cost=42.35 rows=411) (actual time=0.038..0.141 rows=411 loops=1)
-> Index lookup on player using player_club_idx (club_id=club.id) (cost=18.13 rows=72) (actual time=0.206..0.221 rows=50 loops=1)
-> Index lookup on game_player_stats using PRIMARY (player_id=player.id) (cost=0.27 rows=54) (actual time=0.011..0.042 rows=109 loops=50)
```

Analysis : We see that when we add index on club.name , the row search is reduced to only 1 row from 411 rows.

Adding an index on club.name helps because it enables the database to quickly locate the rows in the club table that match the WHERE clause club.name = 'Fc Barcelona'. Without this index, the database would need to scan the entire club table to find the matching rows, which could be very slow if the table is large.

Note that the ORDER BY clause also affects the performance of the query. We can see that adding indexes on the goals column improves the sorting performance by a huge amount. However, adding indexes on other attributes doesn't make a significant difference for this particular query, since the JOIN conditions and the WHERE clause involve only the club and player tables, and the player_id attribute is already the primary key of the game_player_stats table

We would select a combination of index - First on **club.name** field and **game_player_stats.goals** field .