PROJECT PHASE 4

Name: SOHAM NANDI
PSU ID: 930090452

New Questions

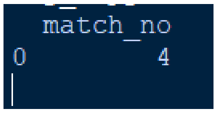
1. Write a query in SQL to find the total number of goalless draws have there in the entire tournament.

ANS:

```
SELECT COUNT(DISTINCT match_no )
FROM match_details
WHERE win_loss = 'D'
AND goal_score = '0';
```

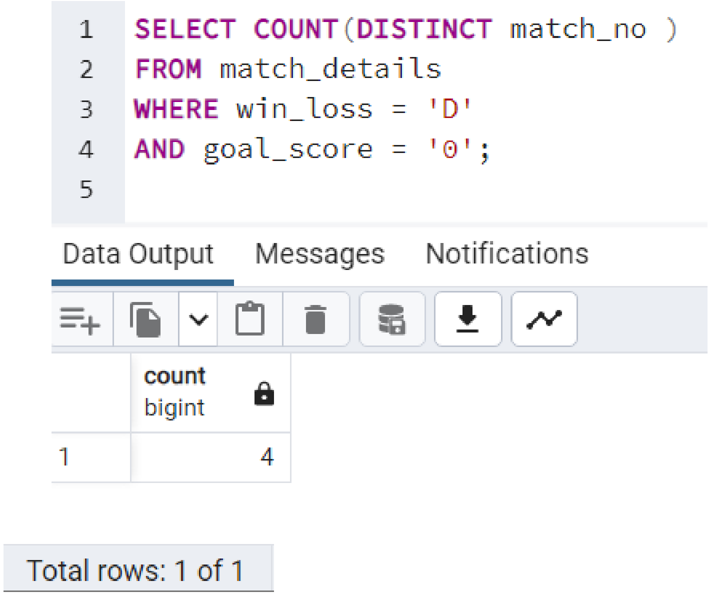
Output using Python:

No. of returned rows: 1



```
   match_no
0         4
```

Output of same query in pgAdmin:



```
SELECT COUNT(DISTINCT match_no )
FROM match_details
WHERE win_loss = 'D'
AND goal_score = '0';
```

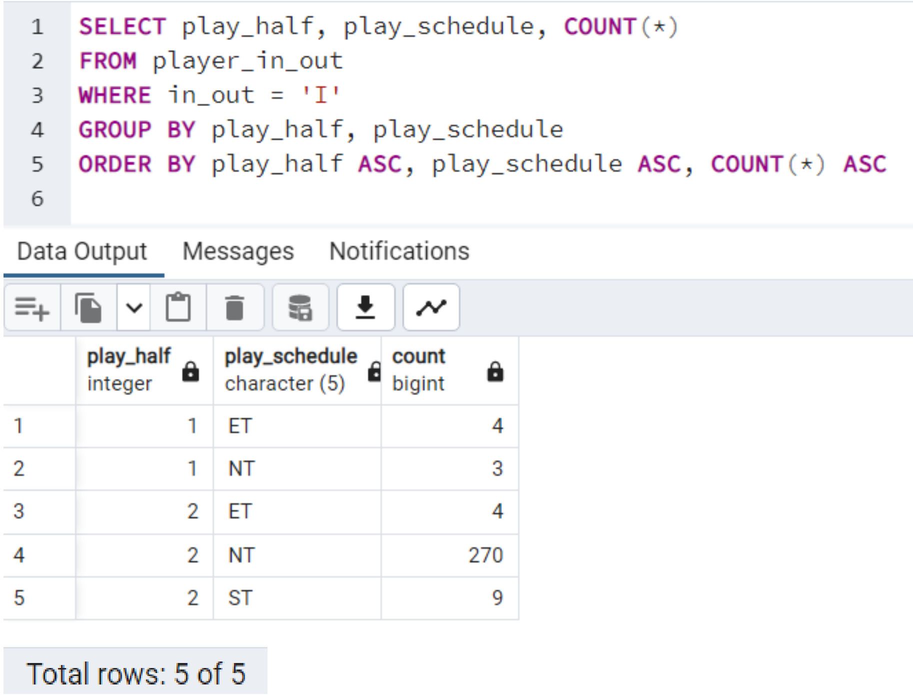
	count bigint
1	4

Total rows: 1 of 1

2. Write a query to get count the number of substitutes happened in various stage of play for the entire Tournament. Sort the result-set in ascending order by play-half, play-schedule and number of substitutes happened. Return play-half, play-schedule, number of substitutes happened

ANS: SELECT play_half, play_schedule, COUNT(*)
FROM player_in_out
WHERE in_out = ‘I’
GROUP BY play_half, play_schedule
ORDER BY play_half ASC, play_schedule ASC, COUNT(*) ASC

Output using pgAdmin:

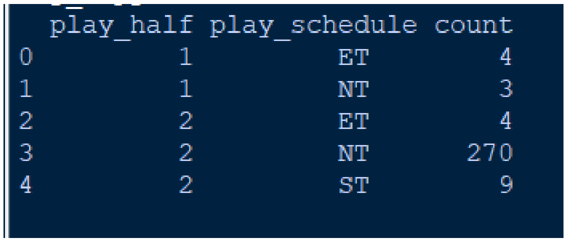


```sql
SELECT play_half, play_schedule, COUNT(*)
FROM player_in_out
WHERE in_out = 'I'
GROUP BY play_half, play_schedule
ORDER BY play_half ASC, play_schedule ASC, COUNT(*) ASC
```

	play_half integer	play_schedule character (5)	count bigint
1	1	ET	4
2	1	NT	3
3	2	ET	4
4	2	NT	270
5	2	ST	9

Total rows: 5 of 5

Output of same query using Python:

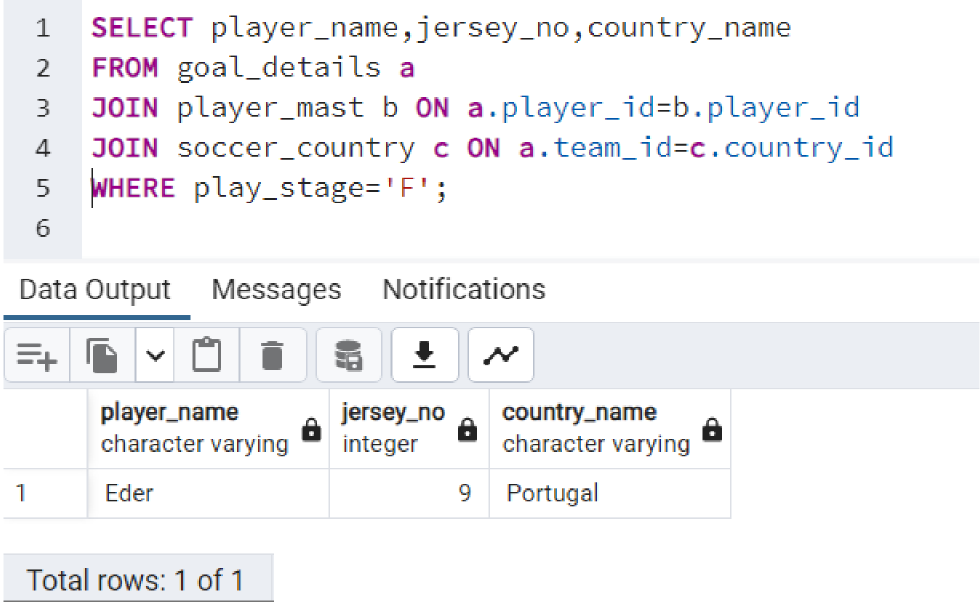


```
  play_half play_schedule count
0         1            ET     4
1         1            NT     3
2         2            ET     4
3         2            NT   270
4         2            ST     9
```

3. write a SQL query to find out who scored in the final. Return player name, jersey number and country name.

ANS: SELECT player_name,jersey_no,country_name
FROM goal_details a
JOIN player_mast b ON a.player_id=b.player_id
JOIN soccer_country c ON a.team_id=c.country_id
WHERE play_stage='F';

Output using pgAdmin:



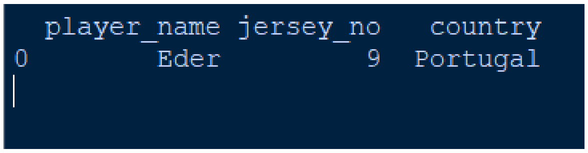
```
SELECT player_name,jersey_no,country_name
FROM goal_details a
JOIN player_mast b ON a.player_id=b.player_id
JOIN soccer_country c ON a.team_id=c.country_id
WHERE play_stage='F';

```

	player_name character varying	jersey_no integer	country_name character varying
1	Eder	9	Portugal

Total rows: 1 of 1

Output of same query using Python:



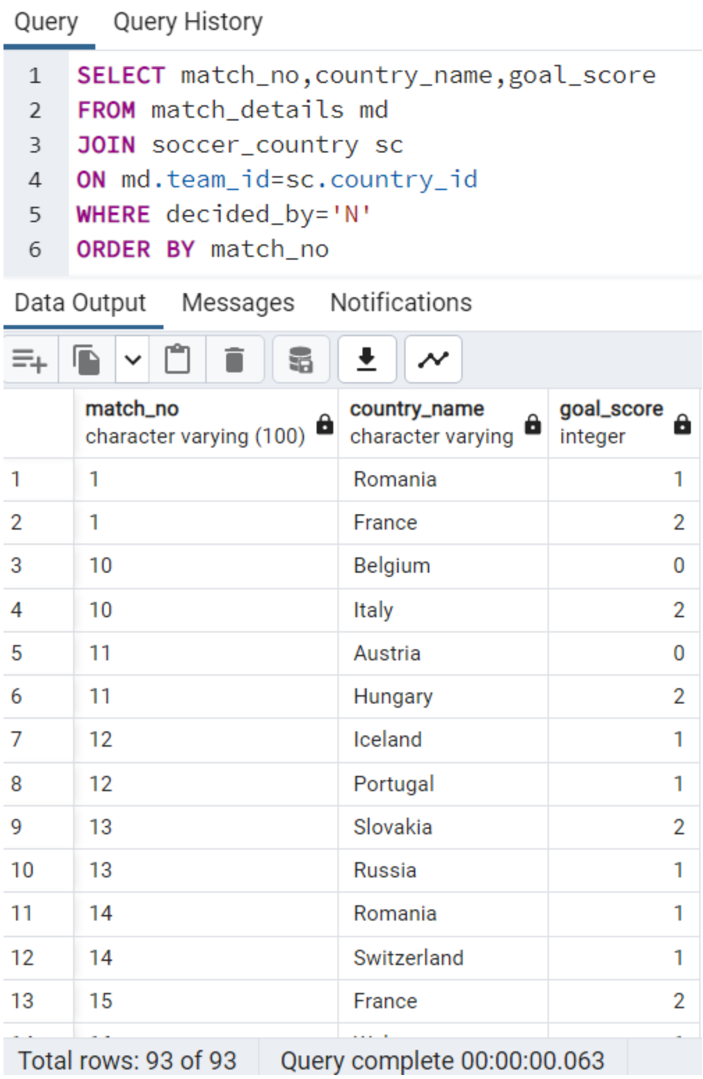
```
  player_name jersey_no   country
0        Eder         9  Portugal
```

4. write a SQL query to find the number of goals scored by each team in each match during normal play. Return match number, country name and goal score.

ANS:

```
SELECT match_no,country_name,goal_score
FROM match_details md
JOIN soccer_country sc
ON md.team_id=sc.country_id
WHERE decided_by='N'
ORDER BY match_no
```

Output using pgAdmin:

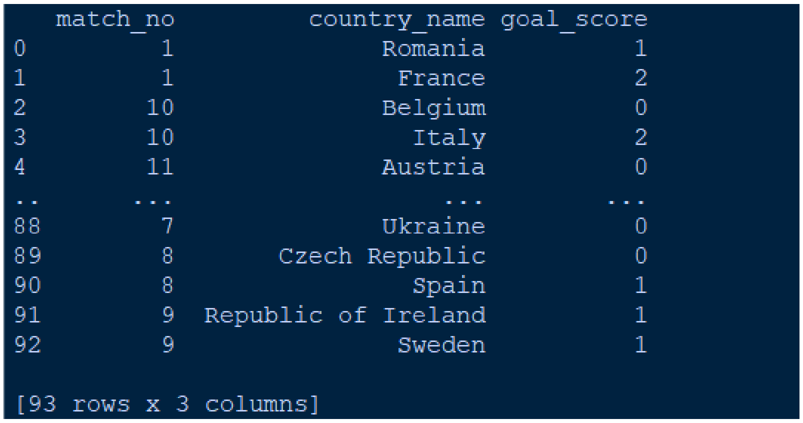


```sql
SELECT match_no,country_name,goal_score
FROM match_details md
JOIN soccer_country sc
ON md.team_id=sc.country_id
WHERE decided_by='N'
ORDER BY match_no
```

	match_no character varying (100)	country_name character varying	goal_score integer
1	1	Romania	1
2	1	France	2
3	10	Belgium	0
4	10	Italy	2
5	11	Austria	0
6	11	Hungary	2
7	12	Iceland	1
8	12	Portugal	1
9	13	Slovakia	2
10	13	Russia	1
11	14	Romania	1
12	14	Switzerland	1
13	15	France	2

Total rows: 93 of 93 Query complete 00:00:00.063

Output using Python:



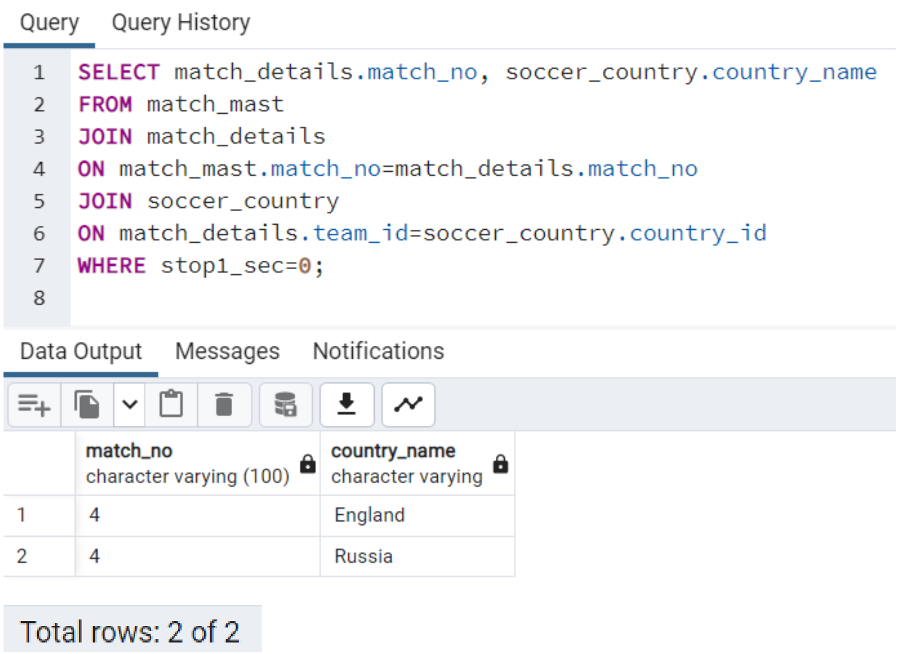
```
   match_no         country_name goal_score
0         1              Romania          1
1         1               France          2
2        10              Belgium          0
3        10                Italy          2
4        11              Austria          0
..      ...                  ...        ...
88        7              Ukraine          0
89        8       Czech Republic          0
90        8                Spain          1
91        9  Republic of Ireland          1
92        9               Sweden          1

[93 rows x 3 columns]
```

5. write a SQL query to find the match where there was no stoppage time in the first half. Return match number, country name.

ANS:
```
SELECT match_details.match_no, soccer_country.country_name
FROM match_mast
JOIN match_details
ON match_mast.match_no=match_details.match_no
JOIN soccer_country
ON match_details.team_id=soccer_country.country_id
WHERE stop1_sec=0;
```

Output using pgAdmin:

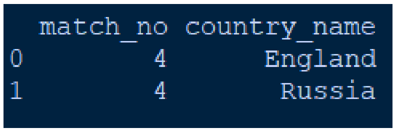


```
SELECT match_details.match_no, soccer_country.country_name
FROM match_mast
JOIN match_details
ON match_mast.match_no=match_details.match_no
JOIN soccer_country
ON match_details.team_id=soccer_country.country_id
WHERE stop1_sec=0;
```

	match_no character varying (100)	country_name character varying
1	4	England
2	4	Russia

Total rows: 2 of 2

Output using Python:

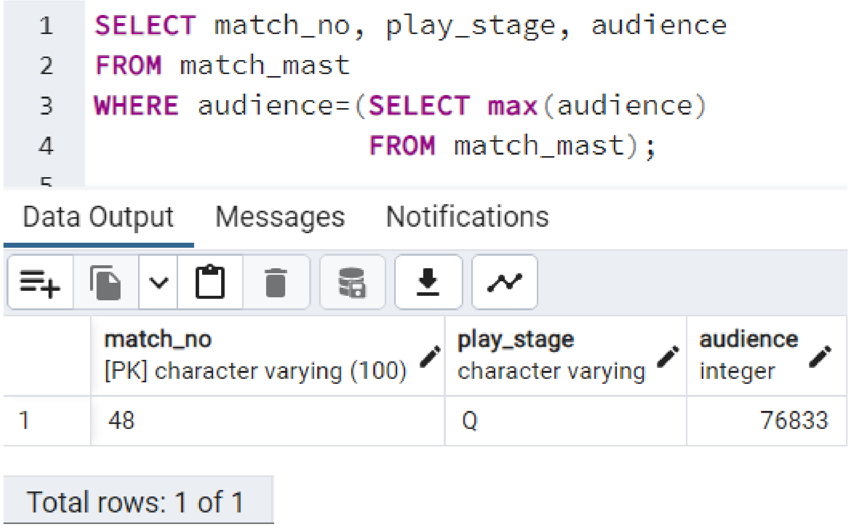


```
  match_no country_name
0        4      England
1        4       Russia
```

6. write a SQL query to find the highest audience match. Return match_no, play_stage, audience.

ANS:
```
SELECT match_no, play_stage, audience
FROM match_mast
WHERE audience=(SELECT max(audience)
                FROM match_mast);
```

Output using pgAdmin:

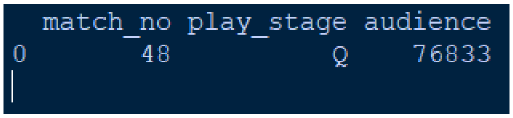


```
SELECT match_no, play_stage, audience
FROM match_mast
WHERE audience=(SELECT max(audience)
                FROM match_mast);
```

	match_no [PK] character varying (100)	play_stage character varying	audience integer
1	48	Q	76833

Total rows: 1 of 1

Output using Python:

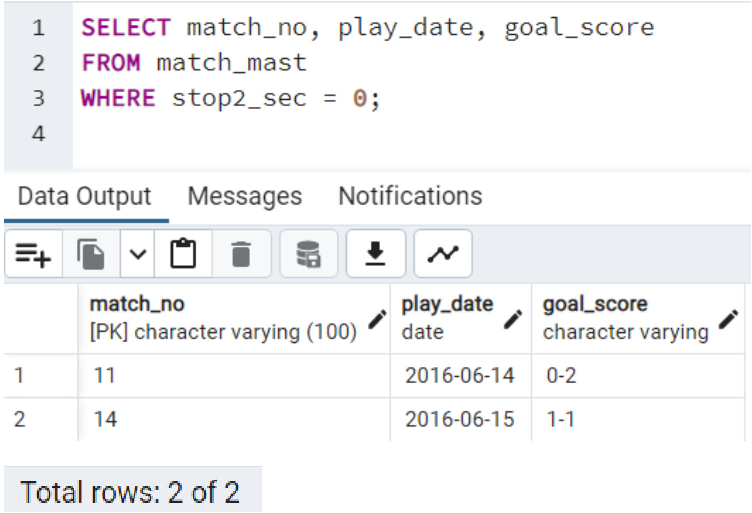


```
  match_no play_stage audience
0       48          Q    76833
```

7. Write a query in SQL to find the match no, date of play, and goal scored for that match in which no stoppage time have been added in 2nd half of play.

ANS: SELECT match_no, play_date, goal_score
FROM match_mast
WHERE stop2_sec = 0;

Output using pgAdmin:

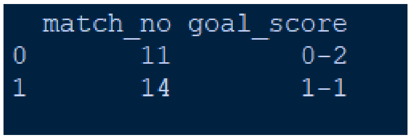


```
SELECT match_no, play_date, goal_score
FROM match_mast
WHERE stop2_sec = 0;
```

	match_no [PK] character varying (100)	play_date date	goal_score character varying
1	11	2016-06-14	0-2
2	14	2016-06-15	1-1

Total rows: 2 of 2

Output using Python:



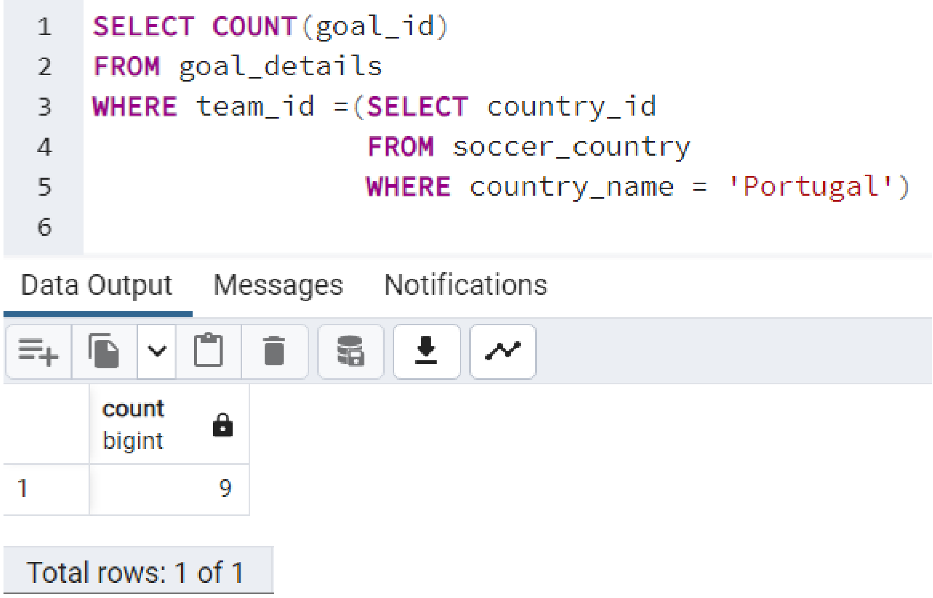
```
   match_no goal_score
0        11        0-2
1        14        1-1
```

8. From the goal_details and soccer_country tables, find the number of goals Germany scored at the tournament.

ANS:

```
SELECT COUNT(goal_id)
FROM goal_details
WHERE team_id =(SELECT country_id
                FROM soccer_country
                WHERE country_name = 'Portugal')
```

Output using pgAdmin:

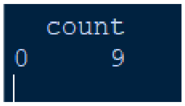


```
SELECT COUNT(goal_id)
FROM goal_details
WHERE team_id =(SELECT country_id
                FROM soccer_country
                WHERE country_name = 'Portugal')
```

	count bigint
1	9

Total rows: 1 of 1

Output using Python:



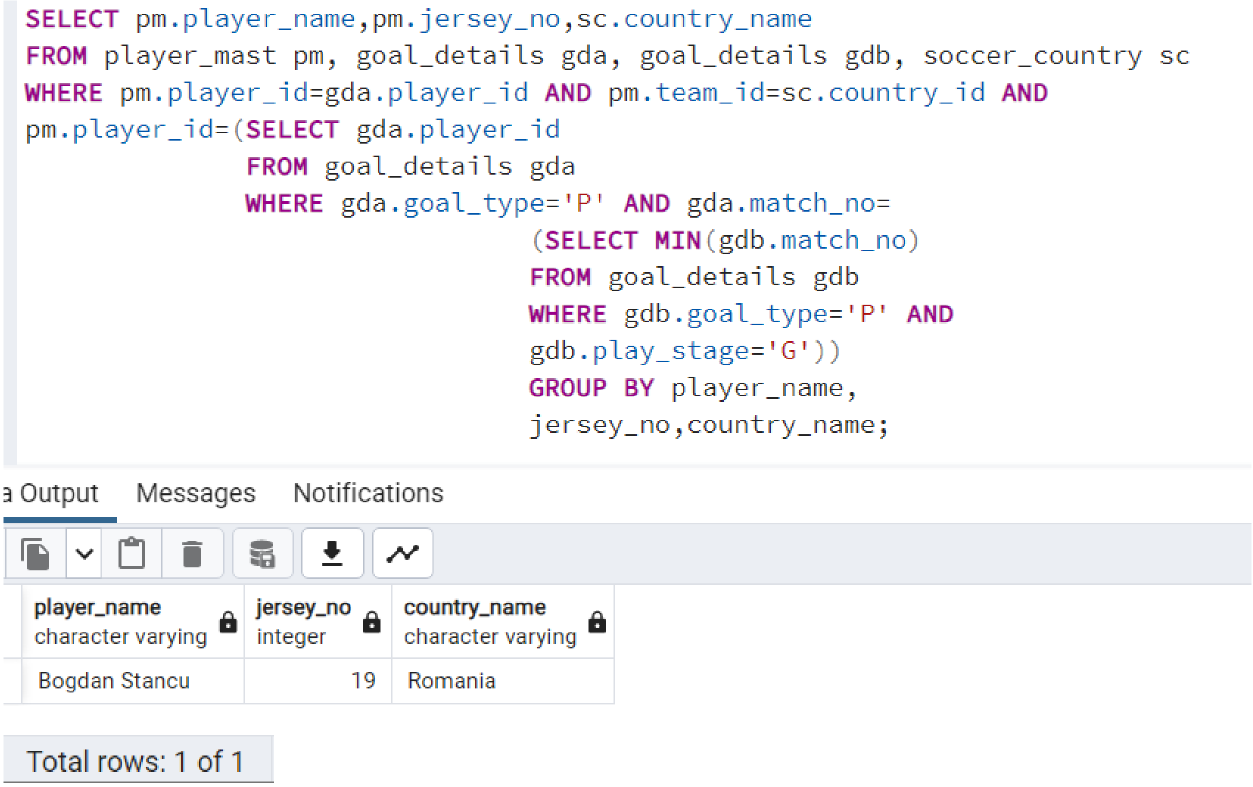
```
   count
0      9
```

9. Write a SQL query to find the player who scored the first penalty in the tournament. Return player name, Jersey number and country name.

ANS:

```
SELECT pm.player_name,pm.jersey_no,sc.country_name
FROM player_mast pm, goal_details gda, goal_details gdb, soccer_country sc
WHERE pm.player_id=gda.player_id AND pm.team_id=sc.country_id AND
pm.player_id=(SELECT gda.player_id
              FROM goal_details gda
              WHERE gda.goal_type='P' AND gda.match_no=
                                  (SELECT MIN(gdb.match_no)
                                  FROM goal_details gdb
                                  WHERE gdb.goal_type='P' AND
                                  gdb.play_stage='G'))
                                  GROUP BY player_name,
                                  jersey_no,country_name;
```

Output using pgAdmin:

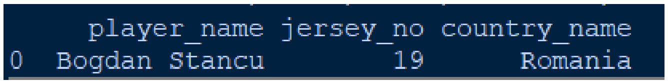


```
SELECT pm.player_name,pm.jersey_no,sc.country_name
FROM player_mast pm, goal_details gda, goal_details gdb, soccer_country sc
WHERE pm.player_id=gda.player_id AND pm.team_id=sc.country_id AND
pm.player_id=(SELECT gda.player_id
              FROM goal_details gda
              WHERE gda.goal_type='P' AND gda.match_no=
                                  (SELECT MIN(gdb.match_no)
                                  FROM goal_details gdb
                                  WHERE gdb.goal_type='P' AND
                                  gdb.play_stage='G'))
                                  GROUP BY player_name,
                                  jersey_no,country_name;
```

player_name character varying	jersey_no integer	country_name character varying
Bogdan Stancu	19	Romania

Total rows: 1 of 1

Output using Python:



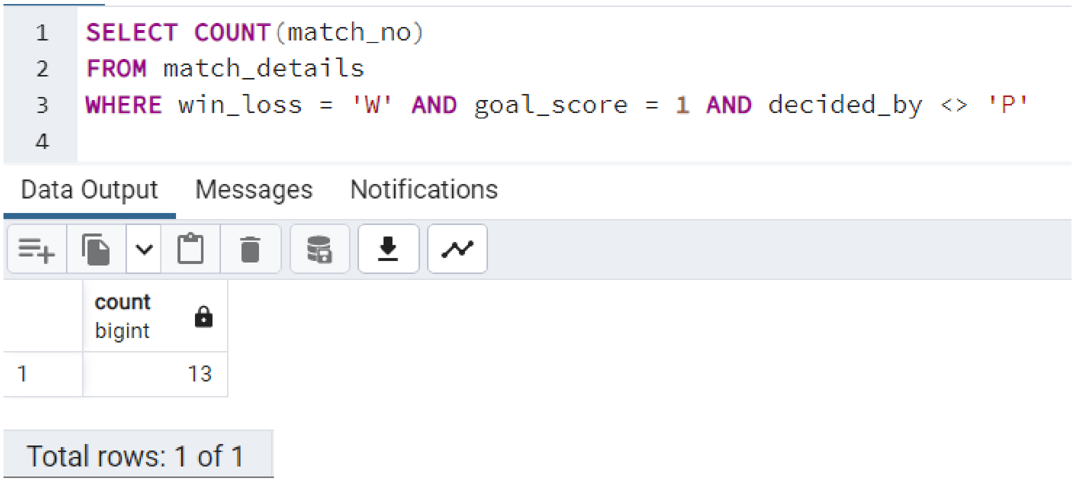
```
    player_name jersey_no country_name
0  Bogdan Stancu        19      Romania
```

10. Write a query to count the number of matches ending with only one goal win, except those matches, which was decided by penalty shoot-out.

ANS:

```
SELECT COUNT(match_no)
FROM match_details
WHERE win_loss = 'W' AND goal_score = 1 AND decided_by <> 'P'
```

Output using pgAdmin:

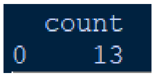


```
SELECT COUNT(match_no)
FROM match_details
WHERE win_loss = 'W' AND goal_score = 1 AND decided_by <> 'P'
```

	count bigint
1	13

Total rows: 1 of 1

Output using Python:



```
   count
0     13
```

Query Performance

1. write a SQL query to find the number of goals scored by each team in each match during normal play. Return match number, country name and goal score. (4th query)

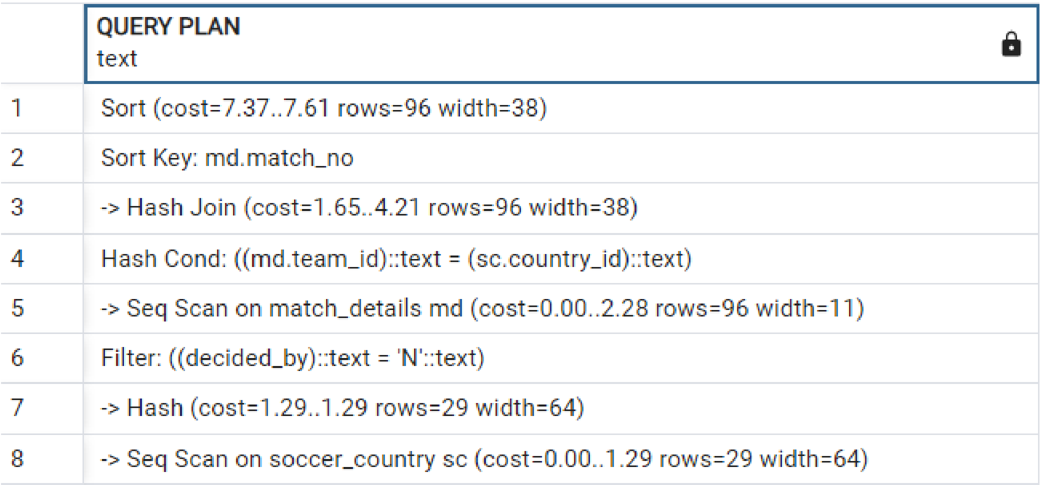
ANS:

```
SELECT match_no,country_name,goal_score
FROM match_details md
JOIN soccer_country sc
ON md.team_id=sc.country_id
WHERE decided_by='N'
ORDER BY match_no
```

Query Performance:

Query:

```
EXPLAIN
SELECT match_no,country_name,goal_score
FROM match_details md
JOIN soccer_country sc
ON md.team_id=sc.country_id
WHERE decided_by='N'
ORDER BY match_no
```



	QUERY PLAN text
1	Sort (cost=7.37..7.61 rows=96 width=38)
2	Sort Key: md.match_no
3	-> Hash Join (cost=1.65..4.21 rows=96 width=38)
4	Hash Cond: ((md.team_id)::text = (sc.country_id)::text)
5	-> Seq Scan on match_details md (cost=0.00..2.28 rows=96 width=11)
6	Filter: ((decided_by)::text = 'N'::text)
7	-> Hash (cost=1.29..1.29 rows=29 width=64)
8	-> Seq Scan on soccer_country sc (cost=0.00..1.29 rows=29 width=64)

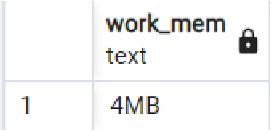
Join Algorithm Used: Hash Join

Reason:

The query that I had chosen used the Hash Join the reason behind this below**,**

Query:

show work_mem



	work_mem text
1	4MB

The maximum amount of memory that will be allocated to any query is 4MB

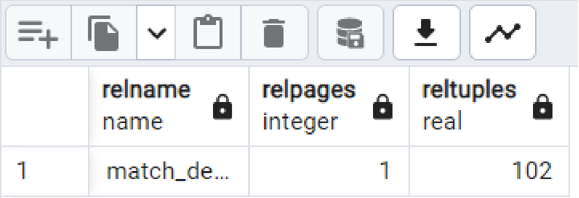
Size of each page is = 8KB

Number of Buffer Pages (BP) = 4MB/8KB = 512 buffer pages

- Another reason is there is an equi-join between outer and inner relation

Query:

select relname,relpages,reltuples
from pg_class
where relname='match_details';



	relname name	relpages integer	reltuples real
1	match_de...	1	102

Here, no. of pages in outer relation(M) = 1

As a result, the outer relation of the Join in this query can fit entirely in the buffer memory, the database will therefore employ the Hash Join in this instance. A hash table will be constructed over the outer relation match_details in this case, with tuples associated with each hash value depending on the hash function on the join property. We will later apply the same hashing function to the join attribute of the inner relation and search the outer relation's hash table for matches.

The estimated cost = 7.61
Actual time to run query = 0.262

Way to Improve Query Performance:

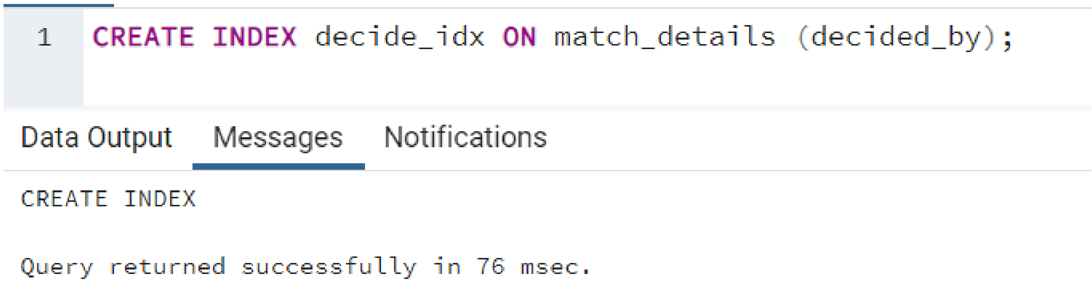
So, if we wanted to enhance the query's performance in this case, we could build a clustered index on ‘decided_by’ in ‘match_details’ table. The data will be in sorter order with indexes pointing to them after the clustered index on the ‘decided_by’ property is created. In order to match the predicate condition of decided_by='N', the database can use the index scan rather than the sequential scan.

The efficiency of the query will not be improved by creating indexes on other attributes since they will not speed up the data-scanning process.

TO create the Index:

Query:

```
CREATE INDEX decide_idx ON match_details (decided_by);
```



```
CREATE INDEX decide_idx ON match_details (decided_by);
```

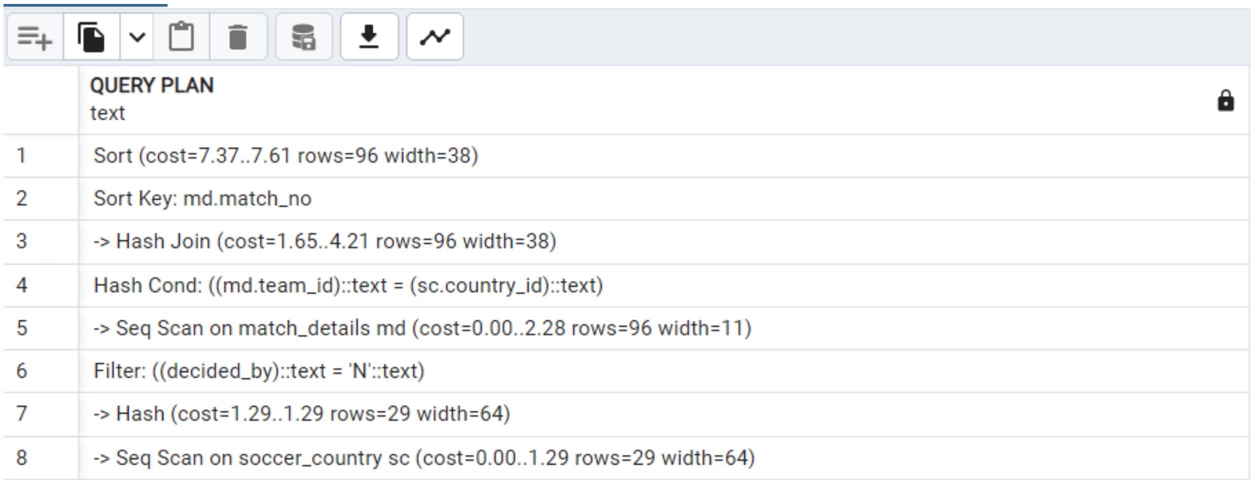
```
CREATE INDEX

Query returned successfully in 76 msec.
```

After Creating Index:

Query:

```
EXPLAIN
SELECT match_no,country_name,goal_score
FROM match_details md
JOIN soccer_country sc
ON md.team_id=sc.country_id
WHERE decided_by='N'
ORDER BY match_no
```



	QUERY PLAN text
1	Sort (cost=7.37..7.61 rows=96 width=38)
2	Sort Key: md.match_no
3	-> Hash Join (cost=1.65..4.21 rows=96 width=38)
4	Hash Cond: ((md.team_id)::text = (sc.country_id)::text)
5	-> Seq Scan on match_details md (cost=0.00..2.28 rows=96 width=11)
6	Filter: ((decided_by)::text = 'N'::text)
7	-> Hash (cost=1.29..1.29 rows=29 width=64)
8	-> Seq Scan on soccer_country sc (cost=0.00..1.29 rows=29 width=64)

Result:

Even after creating index on the ‘decided_by’ attribute of ‘match_details’ table the query performance is not improving in postgres. As per my understanding if there were more amount of data, it might use indexing to improve the query performance.

2. write a SQL query to find the match where there was no stoppage time in the first half. Return match number, country name.

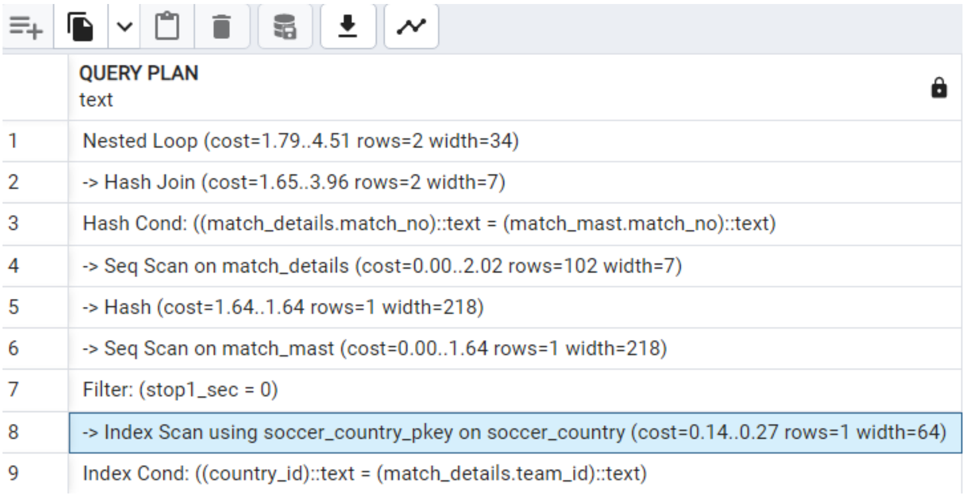
ANS:

```
SELECT match_details.match_no, soccer_country.country_name
FROM match_mast
JOIN match_details
ON match_mast.match_no=match_details.match_no
JOIN soccer_country
ON match_details.team_id=soccer_country.country_id
WHERE stop1_sec=0;
```

Query Performance:

Query:

```
EXPLAIN
SELECT match_details.match_no, soccer_country.country_name
 FROM match_mast
 JOIN match_details
 ON match_mast.match_no=match_details.match_no
 JOIN soccer_country
 ON match_details.team_id=soccer_country.country_id
 WHERE stop1_sec=0;
```



	QUERY PLAN text
1	Nested Loop (cost=1.79..4.51 rows=2 width=34)
2	-> Hash Join (cost=1.65..3.96 rows=2 width=7)
3	Hash Cond: ((match_details.match_no)::text = (match_mast.match_no)::text)
4	-> Seq Scan on match_details (cost=0.00..2.02 rows=102 width=7)
5	-> Hash (cost=1.64..1.64 rows=1 width=218)
6	-> Seq Scan on match_mast (cost=0.00..1.64 rows=1 width=218)
7	Filter: (stop1_sec = 0)
8	-> Index Scan using soccer_country_pkey on soccer_country (cost=0.14..0.27 rows=1 width=64)
9	Index Cond: ((country_id)::text = (match_details.team_id)::text)

Join Algorithm Used: Nested Loop Join, Hash Join

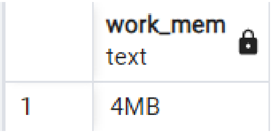
Reason:

Nested loop is using is ‘soccer_country’ and ‘match_details’ table
Hash join is using in ‘match_details’ and ‘match_mast’ tables.
The query that I had chosen used the Nested Loop Join the reason behind this below

The query that I had chosen used the Hash Join the reason behind this below**,**

Query:

show work_mem



	work_mem text
1	4MB

The maximum amount of memory that will be allocated to any query is 4MB

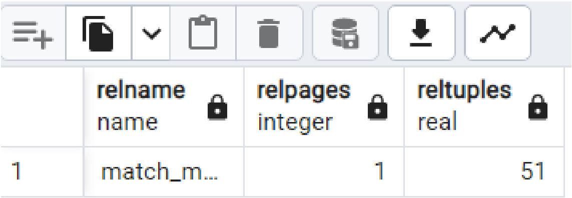
Size of each page is = 8KB

Number of Buffer Pages (BP) = 4MB/8KB = 512 buffer pages

- Another reason is there is an equi-join between outer and inner relation

Query:

select relname,relpages,reltuples
from pg_class
where relname='soccer_country';



	relname name	relpages integer	reltuples real
1	match_m...	1	51

Here, no. of pages in outer relation(M) = 1

'soccer_country' is the outer relation in this instance, and the result of the hash join is the inner relation. Because "country_id" is a primary key and will therefore by default be used to build an index, it will be searched over in order to perform nested loop join. This is also the reason pgadmin uses nested loops so that the 'country_id' can be found without having to search through all of the tuples in the outer relation. There is no need to build a hash table for a hash join or sort the 'soccer_country' tuples over the 'country_id' attribute for a sort merge join because the index has already been built. It therefore does not use other joins, such as sort or hash join.

Way to Improve Query Performance:

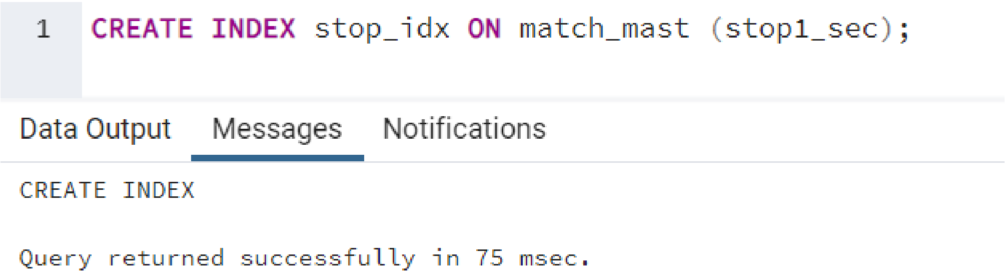
In this case, we can therefore create a clustered index on the'stop1 sec' attribute of the'match mast' to enhance the performance of the query. The data will be in sorter order over the'stop1 sec' attribute with indexes pointing to them after the clustered index on the'stop1 sec' attribute is created. Therefore, the database can obtain the maximum date by using an index scan rather than a sequential scan.

Making an index for other attributes won't help the query perform better because they won't speed up the process of searching the data for the maximum date. In this case, the index is therefore useless.

TO create the Index:

Query:

```
CREATE INDEX stop_idx ON match_mast (stop1_sec);
```



```
CREATE INDEX stop_idx ON match_mast (stop1_sec);
```

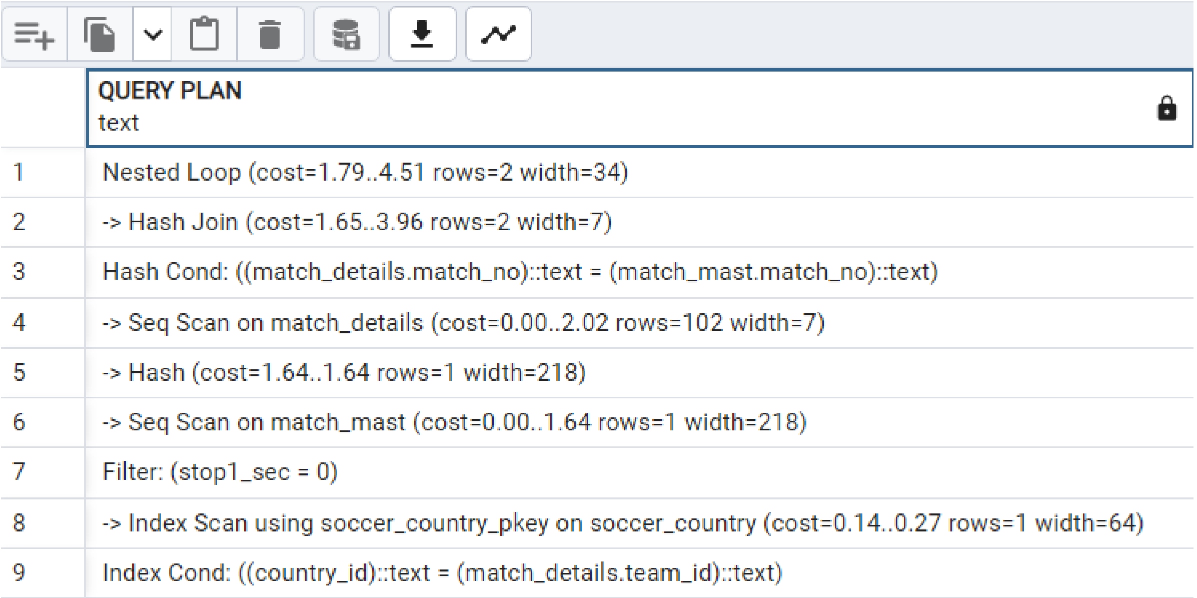
```
CREATE INDEX

Query returned successfully in 75 msec.
```

After Creating Index:

Query:

```
EXPLAIN
SELECT match_details.match_no, soccer_country.country_name
FROM match_mast
JOIN match_details
ON match_mast.match_no=match_details.match_no
JOIN soccer_country
ON match_details.team_id=soccer_country.country_id
WHERE stop1_sec=0;
```



	QUERY PLAN text
1	Nested Loop (cost=1.79..4.51 rows=2 width=34)
2	-> Hash Join (cost=1.65..3.96 rows=2 width=7)
3	Hash Cond: ((match_details.match_no)::text = (match_mast.match_no)::text)
4	-> Seq Scan on match_details (cost=0.00..2.02 rows=102 width=7)
5	-> Hash (cost=1.64..1.64 rows=1 width=218)
6	-> Seq Scan on match_mast (cost=0.00..1.64 rows=1 width=218)
7	Filter: (stop1_sec = 0)
8	-> Index Scan using soccer_country_pkey on soccer_country (cost=0.14..0.27 rows=1 width=64)
9	Index Cond: ((country_id)::text = (match_details.team_id)::text)

Result:

Even after creating index on the ‘decided_by’ attribute of ‘match_details’ table the query performance is not improving in postgres. As per my understanding if there were more amount of data, it might use indexing to improve the query performance.

Query Plan

1. write a SQL query to find out who scored in the final. Return player name, jersey number and country name.

ANS:

Query:

```
SELECT player_name,jersey_no,country_name
FROM goal_details a
JOIN player_mast b ON a.player_id=b.player_id
JOIN soccer_country c ON a.team_id=c.country_id
WHERE play_stage='F';
```

Query Plan:

```
EXPLAIN
SELECT player_name,jersey_no,country_name
FROM goal_details a
JOIN player_mast b ON a.player_id=b.player_id
JOIN soccer_country c ON a.team_id=c.country_id
WHERE play_stage='F';
```

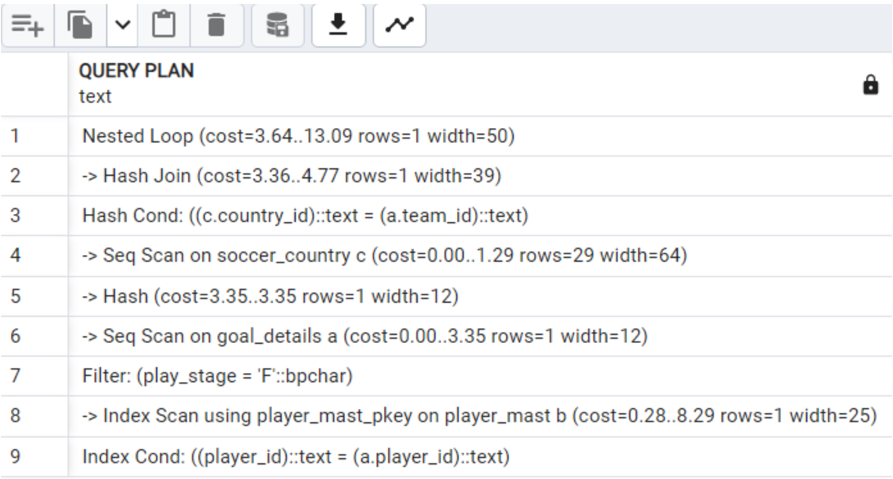
OUTPUT:

Reason:

The query that I had chosen used Hash Join.

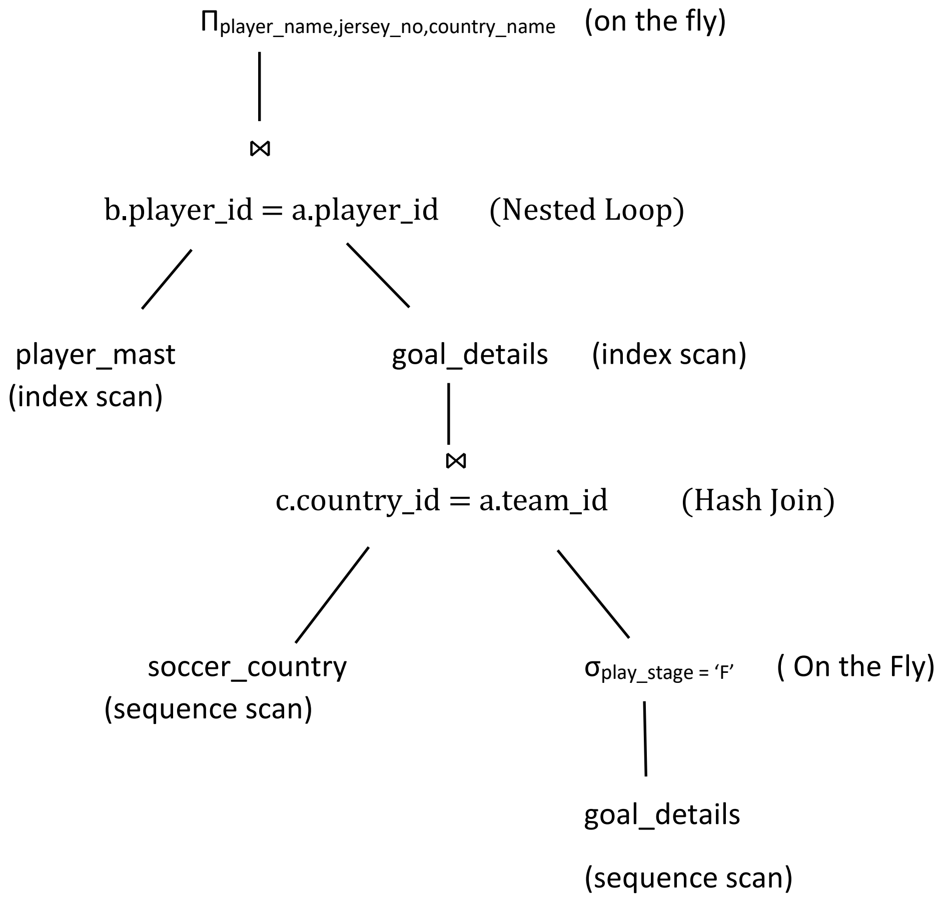
The Hash join is on the relations Rent Data and Tenant Data

Here the outer relation is Rent_data and the Inner Relation is Tenant_Data.



	QUERY PLAN text
1	Nested Loop (cost=3.64..13.09 rows=1 width=50)
2	-> Hash Join (cost=3.36..4.77 rows=1 width=39)
3	Hash Cond: ((c.country_id)::text = (a.team_id)::text)
4	-> Seq Scan on soccer_country c (cost=0.00..1.29 rows=29 width=64)
5	-> Hash (cost=3.35..3.35 rows=1 width=12)
6	-> Seq Scan on goal_details a (cost=0.00..3.35 rows=1 width=12)
7	Filter: (play_stage = 'F'::bpchar)
8	-> Index Scan using player_mast_pkey on player_mast b (cost=0.28..8.29 rows=1 width=25)
9	Index Cond: ((player_id)::text = (a.player_id)::text)

Physical Query Plan:


$\Pi_{player\_name,jersey\_no,country\_name}$ (on the fly)
⋈
b.player_id = a.player_id (Nested Loop)
player_mast
(index scan)
goal_details (index scan)
⋈
c.country_id = a.team_id (Hash Join)
soccer_country
(sequence scan)
$\sigma_{play\_stage = 'F'}$ (On the Fly)
goal_details
(sequence scan)

- Postgres is using nested loop and hash join both

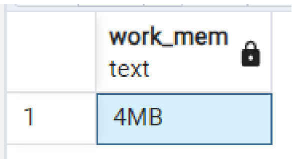
Reason:

The query that I had chosen used Nested Loop and Hash Join.
The Hash join is on the relations soccer_country and goal_details. 'soccer_country' is outer relation here and 'goal_details' is inner relation.
The Nested Loop Join is on the relations 'player_mast' and 'goal_details'. Here the outer relation is 'player_mast' and the Inner Relation is 'goal_details'.

- Another reason is there is an equi-join between outer and inner relation

Query:

```
show work_mem
```



	work_mem text
1	4MB

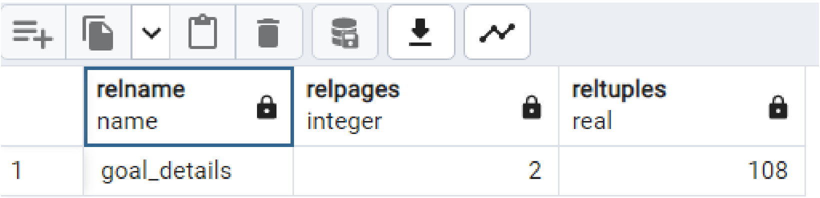
The maximum amount of memory that will be allocated to any query is 4MB

Size of each page is = 8KB

Number of Buffer Pages (BP) = 4MB/8KB = 512 buffer pages

Query:

```
select relname,relpages,reltuples
from pg_class
where relname=' soccer_country';
```



	relname name	relpages integer	reltuples real
1	goal_details	2	108

Here no of pages in outer relation (M) = 2

'soccer_country' is the outer relation in this instance, and the result of the hash join is the inner relation. Because "country_id" is a primary key and will therefore by default be used to build an index, it will be searched over in order to perform nested loop join. This is also the reason pgadmin uses nested loops so that the 'country_id' can be found without having to search through all of the tuples in the outer relation. There is no need to build a hash table for a hash join or sort the 'soccer_country' tuples over the 'country_id' attribute for a sort merge join because the index has already been built. It therefore does not use other joins, such as sort or hash join.

2. write a SQL query to find the number of goals scored by each team in each match during normal play. Return match number, country name and goal score.

   ANS:

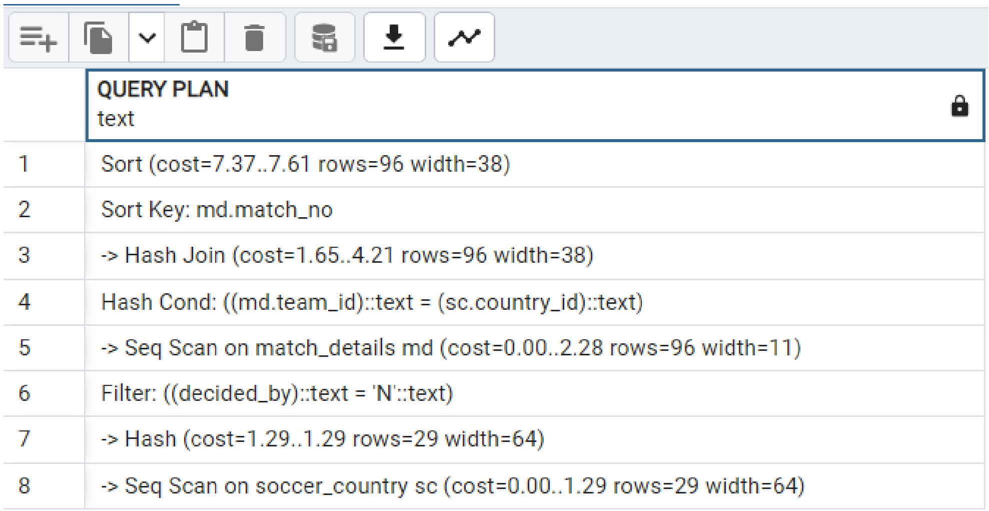
```
SELECT match_no,country_name,goal_score
FROM match_details md
JOIN soccer_country sc
ON md.team_id=sc.country_id
```

```
WHERE decided_by='N'
ORDER BY match_no
```

Query Plan:

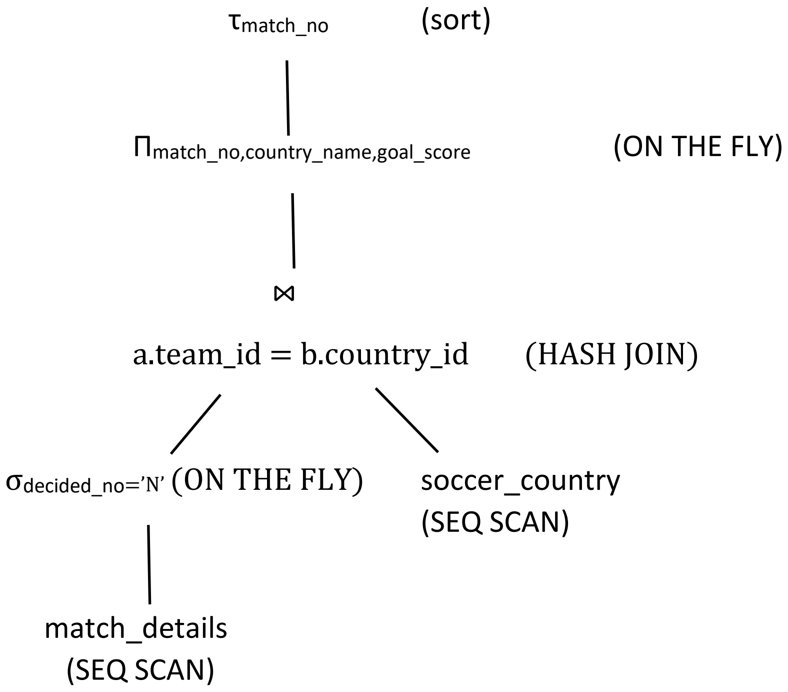
```
EXPLAIN
SELECT match_no,country_name,goal_score
FROM match_details md
JOIN soccer_country sc
ON md.team_id=sc.country_id
WHERE decided_by='N'
ORDER BY match_no
```

Output:



	QUERY PLAN text
1	Sort (cost=7.37..7.61 rows=96 width=38)
2	Sort Key: md.match_no
3	-> Hash Join (cost=1.65..4.21 rows=96 width=38)
4	Hash Cond: ((md.team_id)::text = (sc.country_id)::text)
5	-> Seq Scan on match_details md (cost=0.00..2.28 rows=96 width=11)
6	Filter: ((decided_by)::text = 'N'::text)
7	-> Hash (cost=1.29..1.29 rows=29 width=64)
8	-> Seq Scan on soccer_country sc (cost=0.00..1.29 rows=29 width=64)

Physical Query Plan:


τ_{match_no} (sort)
$\Pi_{match\_no,country\_name,goal\_score}$ (ON THE FLY)
$\bowtie$
a.team_id = b.country_id (HASH JOIN)
$\sigma_{decided\_no='N'}$ (ON THE FLY)
soccer_country (SEQ SCAN)
match_details (SEQ SCAN)

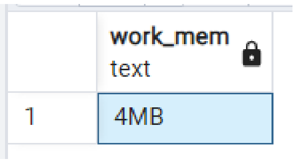
- **The join algorithm chosen by the Postgres is Hash Join**

Reason:

The query I had chosen used Hash Join. It is on match_details and soccer_country. Here the outer relation is match_details and the Inner Relation is soccer_country.

Query:

show work_mem



	work_mem text
1	4MB

The maximum amount of memory that will be allocated to any query is 4MB

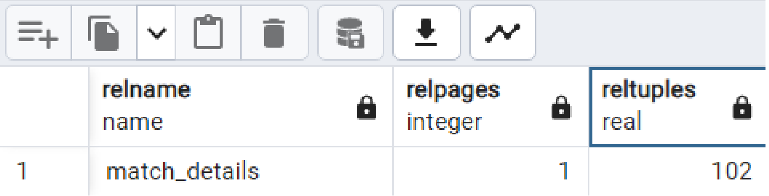
Size of each page is = 8KB

Number of Buffer Pages (BP) = 4MB/8KB = 512 buffer pages

Query:

```
select relname,relpages,reltuples
from pg_class
where relname='match_details';
```

Output:



	relname name	relpages integer	reltuples real
1	match_details	1	102

Here no of pages in outer relation (M) = 1

In this scenario, the outer relation of the Join in this query can fit entirely in the buffer memory. The database will therefore employ the Hash Join in this instance. (equi join pending)

In this case, a hash table will be built over the outer relation i.e., ‘match_details’, and each hash value will have its corresponding tuples associated with it depending on the hash function on the join attribute (team_id). The join property of the inner relation (soccer_country) will later be hashed using the same function, and we will search the outer relation's hash table for any matches. Additionally, sequential search through the hash characteristics of both relations is used when constructing the hash join.

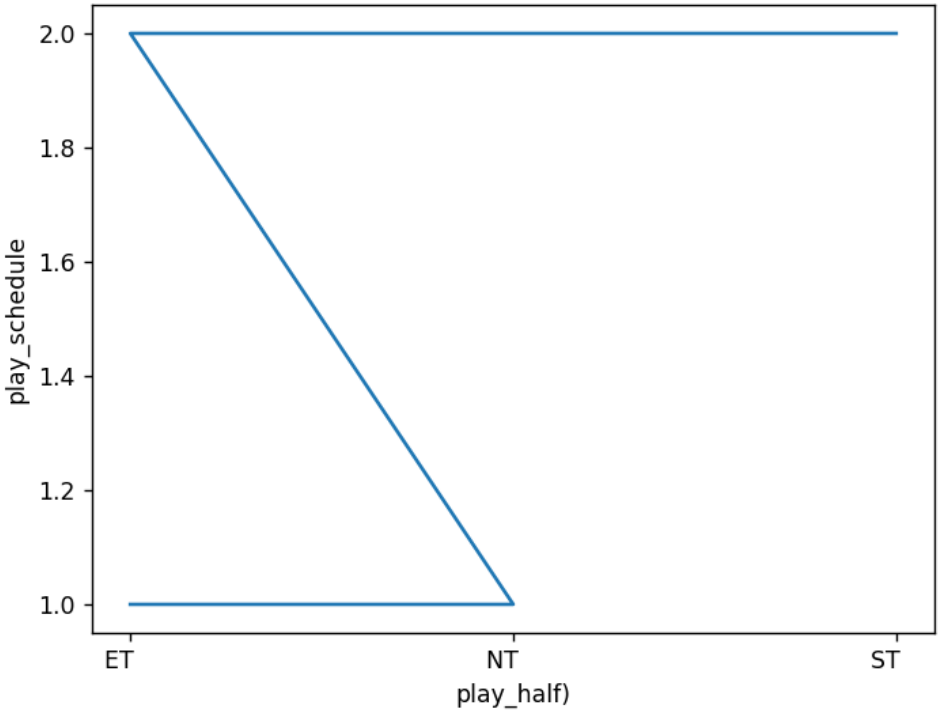
Visualization

1. Write a query to get count the number of substitutes happened in various stage of play for the entire Tournament. Sort the result-set in ascending order by play-half, play-schedule and number of substitutes happened. Return play-half, play-schedule, number of substitutes happened

ANS:

```
SELECT play_half, play_schedule, COUNT(*)
FROM player_in_out
WHERE in_out = 'I'
GROUP BY play_half, play_schedule
ORDER BY play_half ASC, play_schedule ASC, COUNT(*) ASC
```

Visualization Output:



To plot the above vizualization first I connected connected the postgresql database to the python environment using the psycopg2 adapter. Then I had written the query in the python code that returns the output as a datafram.

Later using matplotlib library of the pyhton I plotted the output using the code above.

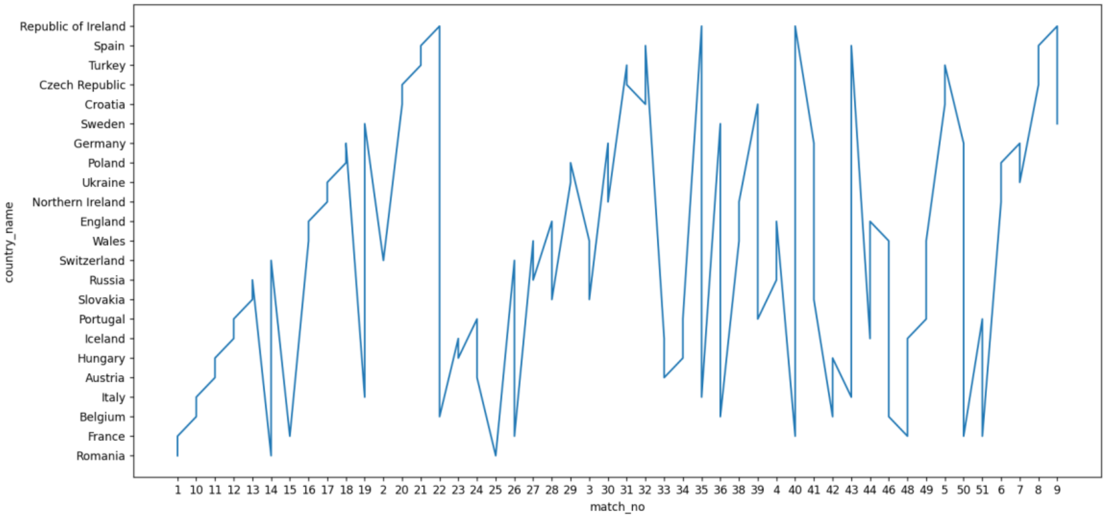
I've taken count in x axis and play_half in y axis and plotted the data.

2. write a SQL query to find the number of goals scored by each team in each match during normal play. Return match number, country name and goal score.

ANS:

```
SELECT match_no,country_name,goal_score
FROM match_details md
JOIN soccer_country sc
ON md.team_id=sc.country_id
WHERE decided_by='N'
ORDER BY match_no
```

Visualization Output:



To plot the above vizualization first I connected connected the postgresql database to the python environment using the psycopg2 adapter. Then I had written the query in the python code that returns the output as a datafram.

Later using matplotlib library of the pyhton I plotted the output using the code above.

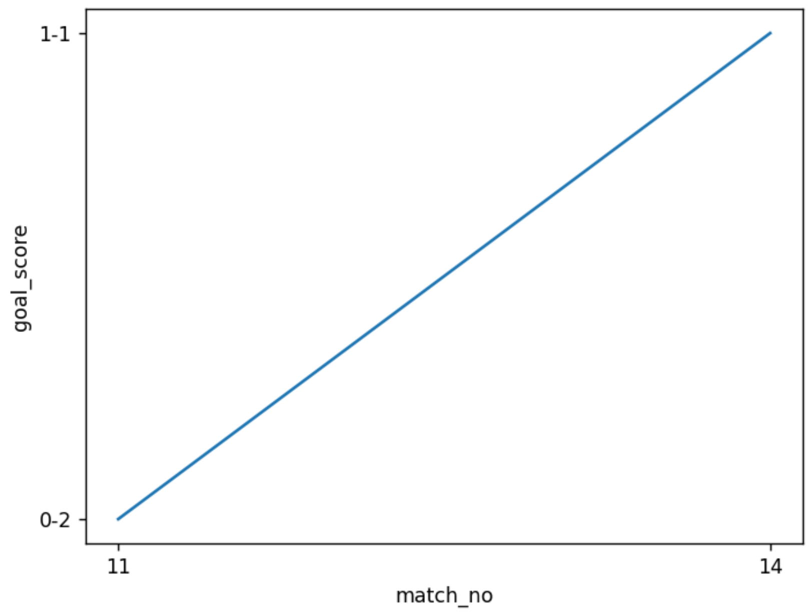
I've plotted match_number in x- axis and country_name in y axis. Tha's how I've visualized the whole data.

3. Write a query in SQL to find the match no, date of play, and goal scored for that match in which no stoppage time have been added in 2nd half of play.

ANS:

```
SELECT match_no, play_date, goal_score
FROM match_mast
WHERE stop2_sec = 0;
```

Visulization Output:



To plot the above vizualization first I connected connected the postgresql database to the python environment using the psycopg2 adapter. Then I had written the query in the python code that returns the output as a datafram.

Later using matplotlib library of the pyhton I plotted the output using the code above.

In this case I've plotted match_no and goal_score. match_no has been plotted in x-axis and goal_score in y-axis and then I've visualized the data I got.

PRESENTATION

Zoom Record Link:
https://pdx.zoom.us/rec/share/9hEkW2rQWm697Yu263CWdhEMUiBbxNGjv_VCHGTaR8w78HZSIXencN43Xg6_kzJm.tWVCZ8ROTYvkPKci?startTime=1670481971000

Drive Link:
https://drive.google.com/file/d/1o2RkxrZ9wkhu3KELZqJF8n9Ms82hDs55/view?usp=sharing