First Query:

Before Index:

```
EXPLAIN: -> Sort: `sum(t1.ydsnet)`  (actual time=815.889..815.891 rows=32 loops=1)
           -> Table scan on <temporary>  (actual time=0.002..0.008 rows=32 loops=1)
              -> Aggregate using temporary table  (actual time=815.849..815.857 rows=32 loops=1)
                 -> Table scan on t1  (cost=0.08..2.98 rows=38) (actual time=0.001..0.854 rows=7308 loops=1)
```

CREATE INDEX TEAMNAME_INDEX ON Rosters(TeamName);

After Index:

```
-> Sort: `sum(t1.ydsnet)`  (actual time=761.132..761.134 rows=32 loops=1)
   -> Table scan on <temporary>  (actual time=0.001..0.007 rows=32 loops=1)
      -> Aggregate using temporary table  (actual time=761.032..761.039 rows=32 loops=1)
         -> Table scan on t1  (cost=0.08..2.98 rows=38) (actual time=0.002..0.834 rows=7308 loops=1)
```

By providing pointers to the locations of data within a database, indexing makes it faster to query columns. Imagine you need to locate a certain piece of data that is stored in a sizable database. The computer will search through all of the rows in the database until it discovers this information. So in this case Indexing by TeamName will allow us to retrieve information in the TeamName column in this case it will be able to point and easily accessible that column which we use in in order to find out which teams made that play, especially because there are many plays attributed to just one TeamName inside our database so indexing by teams will allow us to query those columns in a more efficient manner and it is shown by the decrease in time.

CREATE INDEX PLAYID_INDEX ON PlayingIn(play_id);

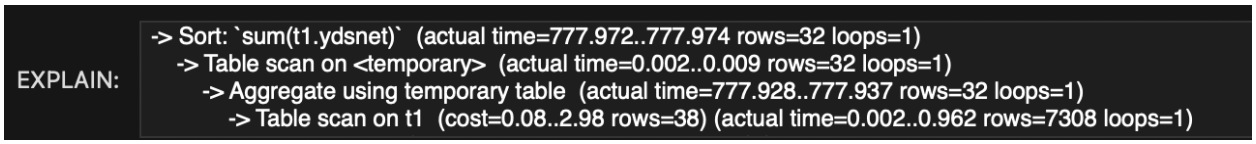
After Index:

```
EXPLAIN: -> Sort: `sum(t1.ydsnet)`  (actual time=753.364..753.367 rows=32 loops=1)
           -> Table scan on <temporary>  (actual time=0.002..0.006 rows=32 loops=1)
              -> Aggregate using temporary table  (actual time=753.319..753.326 rows=32 loops=1)
                 -> Table scan on t1  (cost=0.08..2.98 rows=38) (actual time=0.005..0.799 rows=7308 loops=1)
```

Again as I mentioned above with the definition of Indexing, it makes it faster to query columns and locate a piece of data that is stored in a database. In this case, indexing by play_id is that we use that information to find out what happened that play, who made that play, and also how that play went down(desc). Many players can make one play so there can a lot more players inside of each play_id which can be much slower with indexing. Indexing on the column play_id will make it faster to query that column to get to that respective data.

CREATE INDEX QTR_INDEX ON PlayByPlay(qtr);

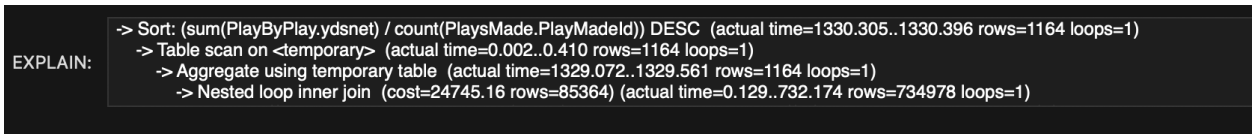
After Index:



We use qtr to measure the average number of passing yards per quarter per team. Run time ranges have decreased from 815.899 to 777.972. Qtr is much smaller because it can be divided into 1st, 2nd, 3rd, and 4th, and there can be multiple play_ids, net yards, and players assigned to just one quarter. So being able to access the column qtr will allow us to parse through the data a lot quicker shown by the decrease in time.

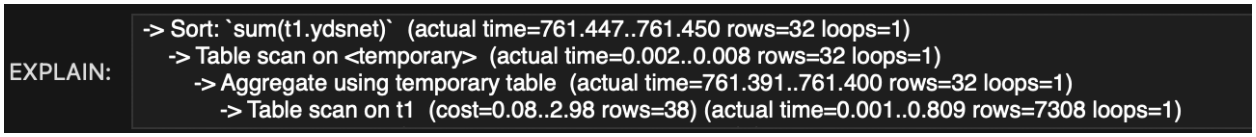
Second Query:

Before Index:



CREATE INDEX PLAYERNAME_INDEX ON Rosters(PlayerName);

After Index:



We use PlayerName in various different ways however the most common would be which player made the actual play during a qtr. Being able to index by PlayerName would allow us to query that column a lot faster because a single player(PlayerName can make multiple plays and have multiple PlayMadeId's attributed to him. So being able to access data in PlayerName much more efficiently when we index which we can see drastically decreased the amount of time as we can see above how it went down a significant amount from before the indexing.

CREATE INDEX YDSNET_INDEX ON PlaysMade(PlayMadeId);

After Index:

```
EXPLAIN: -> Sort: (sum(PlayByPlay.ydsnet) / count(PlaysMade.PlayMadeId)) DESC  (actual time=1303.358..1303.447 rows=1164 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.396 rows=1164 loops=1)
        -> Aggregate using temporary table  (actual time=1302.141..1302.599 rows=1164 loops=1)
            -> Nested loop inner join  (cost=24745.16 rows=85364) (actual time=0.078..706.077 rows=734978 loops=1)
```

We use PlayMadeId to attach who made that play, a defensive or offensive player to a specific play in the game. Being able to index by PlayMadeId would allow us to query that column faster. However, in this case, indexing does not actually query that column much faster because PlaysMadeId /is overwhelming the largest and has the most amount of columns. So being able to access it would not help much since it is already a huge chunk of what needs to be parsed before the indexing itself. That is why the indexing in this case does not bring a better effect on the query

CREATE INDEX YARDS_NET_INDEX ON PlayByPlay(ydsnet);

After Index:

```
EXPLAIN: -> Sort: `sum(t1.ydsnet)`  (actual time=757.425..757.427 rows=32 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.006 rows=32 loops=1)
        -> Aggregate using temporary table  (actual time=757.385..757.393 rows=32 loops=1)
            -> Table scan on t1  (cost=0.08..2.98 rows=38) (actual time=0.002..0.807 rows=7308 loops=1)
```

We use yds net to see how forward or backward the offense went during that play and also to measure passing yard totals, running yard totals, and receiving yard totals plus even defensive metrics such as interception yards, sack yards, etc. We can see that querying did bring a more effective result to our query because the yard totals column is much smaller than the other components such as PlaysMadeId so being able to access the column by indexing will allow us to locate the data in ydsnet much more effectively as we can see by the drastic decrease in time from before to after indexing.