Implementing Data Base Tables

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to finalproject-365721.
Use “gcloud config set project [PROJECT_ID]” to change to a different project.
vsikka02@cloudshell:~ (finalproject-365721)$ gcloud sql connect team-bis --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 93109
Server version: 8.0.26-google (Google)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use all_seasons
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------------+
| Tables_in_all_seasons |
+----------------------+
| Advanced_Statistics   |
| Players               |
| Season_Statistics     |
| Seasons               |
| mytable               |
+----------------------+
5 rows in set (0.00 sec)
```

DDL Commands

```
CREATE TABLE Players(SELECT player_name
 ,AVG(player_height) as player_height
 ,AVG(player_weight) as player_weight
 ,MAX(college) as college
 ,MAX(country) as country
 ,MAX(draft_year) as draft_year
 ,MAX(draft_round) as draft_round
 ,MAX(draft_number) as draft_number
FROM mytable
GROUP BY player_name
);

ALTER TABLE Players
ADD PRIMARY KEY(player_name);
```

```sql
CREATE TABLE Season_Statistics(SELECT player_name,
season,
pts,
reb,
ast,
age
FROM mytable
);

ALTER TABLE Season_Statistics
ADD FOREIGN KEY (player_name) REFERENCES Players(player_name);
```

```sql
CREATE TABLE Advanced_Statistics(SELECT player_name,
season,
net_rating,
oreb_pct,
dreb_pct,
usg_pct,
ts_pct,
ast_pct
FROM mytable
);

ALTER TABLE Advanced_Statistics
ADD FOREIGN KEY (player_name) REFERENCES Players(player_name);
```

```sql
CREATE TABLE Seasons(SELECT season,
 player_name,
 gp,
 age
 FROM mytable
);

ALTER TABLE Seasons
ADD FOREIGN KEY (player_name) REFERENCES Players(player_name);
```

Table Counts

```
mysql> SELECT COUNT(*)
    -> FROM Advanced_Statistics;
+----------+
| COUNT(*) |
+----------+
|    12305 |
+----------+
1 row in set (0.02 sec)
```

```
mysql> SELECT COUNT(*) FROM Players;
+----------+
| COUNT(*) |
+----------+
|     2463 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*)
    -> FROM Season_Statistics;
+----------+
| COUNT(*) |
+----------+
|    12305 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Seasons;
+----------+
| COUNT(*) |
+----------+
|    12305 |
+----------+
1 row in set (0.04 sec)
```

Advanced Queries

```
SELECT sub_query.player_name
FROM (SELECT ss.player_name as player_name, Round(SUM(ss.pts*s.gp),0) as
total_pts, Round(SUM(ss.reb*s.gp),0) as total_reb,Round(SUM(ss.ast*s.gp),0) as
total_ast
FROM Season_Statistics as ss JOIN Seasons as s ON (ss.season LIKE s.season and
ss.player_name LIKE s.player_name)
GROUP BY ss.player_name) sub_query
WHERE (sub_query.total_pts > 18000 AND sub_query.total_reb > 4000 AND
sub_query.total_ast > 4000) OR (sub_query.total_ast > 10000) OR (sub_query.total_reb
> 10000)
UNION
SELECT sub_query.player_name
FROM (SELECT ss.player_name as player_name, Round(AVG(ss.pts),0) as avg_pts,
Round(AVG(ss.reb),0) as avg_rebs ,Round(AVG(ss.ast),0) as avg_ast
FROM Season_Statistics as ss JOIN Seasons as s ON (ss.season LIKE s.season and
ss.player_name LIKE s.player_name)
GROUP BY ss.player_name) sub_query
WHERE (sub_query.avg_pts > 25 AND sub_query.avg_rebs > 4 AND
sub_query.avg_ast > 4) OR (sub_query.avg_rebs > 12) OR (sub_query.avg_pts > 24.5)
OR (sub_query.avg_ast > 10)
ORDER BY player_name
```

```
LIMIT 15;
```

```
mysql> SELECT sub_query.player_name FROM (SELECT ss.player_name as player_name, Round(SUM(ss.pts*s.gp),0) as total_pts, Round(SUM(ss.reb*s.gp),0) as total_reb,Round(SUM(ss.ast*s.gp
),0) as total_ast FROM Season_Statistics as ss JOIN Seasons as s ON (ss.season LIKE s.season and ss.player_name LIKE s.player_name) GROUP BY ss.player_name) sub_query WHERE (sub_qu
ery.total_pts > 18000 AND sub_query.total_reb > 4000 AND sub_query.total_ast > 4000) OR (sub_query.total_ast > 10000) OR (sub_query.total_reb > 10000) UNION  SELECT sub_query.playe
r_name FROM (SELECT ss.player_name as player_name, Round(AVG(ss.pts),0) as avg_pts, Round(AVG(ss.reb),0) as avg_rebs ,Round(AVG(ss.ast),0) as avg_ast FROM Season_Statistics as ss J
OIN Seasons as s ON (ss.season LIKE s.season and ss.player_name LIKE s.player_name) GROUP BY ss.player_name) sub_query WHERE (sub_query.avg_pts > 25 AND sub_query.avg_rebs > 4 AND
sub_query.avg_ast > 4) OR (sub_query.avg_rebs > 12) OR (sub_query.avg_pts > 24.5) OR (sub_query.avg_ast > 10) ORDER BY player_name LIMIT 15;
+-----------------+
| player_name     |
+-----------------+
| Allen Iverson   |
| Andre Drummond  |
| Ben Wallace     |
| Chris Paul      |
| Damian Lillard  |
| DeAndre Jordan  |
| Dennis Rodman   |
| Dirk Nowitzki   |
| Dwight Howard   |
| Dwyane Wade     |
| James Harden    |
| Jason Kidd      |
| Jayson Williams |
| Joe Johnson     |
| Joel Embiid     |
+-----------------+
15 rows in set (49.71 sec)

mysql>
```

<u>Subquery 2</u>

SELECT ss.player_name as player_name, Round(SUM(ss.pts*s.gp),0) as total_pts, Round(SUM(ss.reb*s.gp),0) as total_reb,Round(SUM(ss.ast*s.gp),0) as total_ast
FROM Season_Statistics as ss JOIN Seasons as s ON (ss.season LIKE s.season and ss.player_name LIKE s.player_name)
GROUP BY ss.player_name
ORDER BY ss.player_name
LIMIT 15;

```
mysql> SELECT ss.player_name as player_name, Round(SUM(ss.pts*s.gp),0) as total_pts, Round(SUM(ss.reb*s.gp),0) as total_reb,Round(SUM(ss.ast*s.gp),0) as total_ast
    -> FROM Season_Statistics as ss JOIN Seasons as s ON (ss.season LIKE s.season and ss.player_name LIKE s.player_name)
    -> GROUP BY ss.player_name
    -> ORDER BY ss.player_name
    -> LIMIT 15;
+---------------+-----------+-----------+-----------+
| player_name   | total_pts | total_reb | total_ast |
+---------------+-----------+-----------+-----------+
| A.C. Green    |      2220 |      2345 |       337 |
| A.J. Bramlett |         8 |        22 |         0 |
| A.J. Guyton   |       441 |        81 |       146 |
| Aaron Brooks  |      6264 |      1077 |      1923 |
| Aaron Gordon  |      6885 |      3325 |      1309 |
| Aaron Gray    |      1066 |      1192 |       212 |
| Aaron Harrison |        80 |        42 |        16 |
| Aaron Henry   |         2 |         1 |         0 |
| Aaron Holiday |      1794 |       429 |       586 |
| Aaron Jackson |         8 |         3 |         1 |
| Aaron McKie   |      4716 |      2163 |      1833 |
| Aaron Miles   |        15 |        13 |        25 |
| Aaron Nesmith |       414 |       217 |        44 |
| Aaron Wiggins |       415 |       180 |        70 |
| Aaron Williams |      4099 |      2763 |       487 |
+---------------+-----------+-----------+-----------+
15 rows in set (25.03 sec)

mysql>
```

<u>Indexing</u>

<u>ADVANCED QUERY #1</u>

```
    -> GROUP BY ss.player_name;
+----------------------------------------------------------------------------------------------------------------
+
| EXPLAIN

|
+----------------------------------------------------------------------------------------------------------------
+
| -> Table scan on <temporary>  (actual time=0.003..0.887 rows=2463 loops=1)
    -> Aggregate using temporary table  (actual time=55489.525..55490.557 rows=2463 loops=1)
        -> Filter: ((ss.season like s.season) and (ss.player_name like s.player_name))  (cost=14796414.93 rows=1826182) (actual time=23.589..55368.691 rows=12313 loops=1)
            -> Inner hash join (no condition)  (cost=14796414.93 rows=1826182) (actual time=19.899..24513.427 rows=151413024 loops=1)
                -> Table scan on s  (cost=0.01 rows=12183) (actual time=0.025..73.284 rows=12305 loops=1)
                -> Hash
                    -> Table scan on ss  (cost=1238.65 rows=12144) (actual time=0.024..11.933 rows=12305 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------
+
1 row in set (55.50 sec)
```

Original no index added.

```
mysql> CREATE INDEX season_index ON Seasons (season)
    -> ;
Query OK, 0 rows affected (0.13 sec)
Records
easonseasssgON,(ss.seasonULIKE.s.season)andess.player_nametLIKEts.player_name)SS
    -> GROUP BY ss.player_name;
+----------------------------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                                                 |
+----------------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.003..0.633 rows=2463 loops=1)
    -> Aggregate using temporary table  (actual time=32693.583..32694.352 rows=2463 loops=1)
        -> Filter: ((ss.season like s.season) and (ss.player_name like s.player_name))  (cost=14796414.93 rows=1826182) (actual time=13.652..32622.338 rows=12313 loops=1)
            -> Inner hash join (no condition)  (cost=14796414.93 rows=1826182) (actual time=11.822..14947.624 rows=151413024 loops=1)
                -> Table scan on s  (cost=0.01 rows=12183) (actual time=0.024..41.257 rows=12305 loops=1)
                -> Hash
                    -> Table scan on ss  (cost=1238.65 rows=12144) (actual time=0.028..7.005 rows=12305 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------+
1 row in set (32.71 sec)
```

Index on seasons

The goal of the first subquery was to compile all stats that players accumulate across their careers. For our first subquery, the first index that we tried was to index based on the season. We thought this could potentially make the process faster because it would have all seasons of a player relatively close to each other rather than in random order. Compared to the original runtime of 55 seconds, indexing on seasons was able to speed it up to 32.71 seconds.

```
easonseasssgON,(ss.seasonULIKE.s.season)andess.player_nametLIKEts.player_name)SS
    -> GROUP BY ss.player_name;
+----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN

                                                                                                                                 |
+----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.003..0.836 rows=2463 loops=1)
    -> Aggregate using temporary table  (actual time=60929.077..60930.112 rows=2463 loops=1)
        -> Filter: ((ss.season like s.season) and (ss.player_name like s.player_name))  (cost=14796414.93 rows=1826182) (actual time=13.071..60835.754 rows=12313 loops=1)
            -> Inner hash join (no condition)  (cost=14796414.93 rows=1826182) (actual time=11.301..26803.931 rows=151413024 loops=1)
                -> Table scan on s  (cost=0.01 rows=12183) (actual time=0.024..54.890 rows=12305 loops=1)
                -> Hash
                    -> Table scan on ss  (cost=1238.65 rows=12144) (actual time=0.029..7.023 rows=12305 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------+
1 row in set (1 min 0.94 sec)

mysql> SHOW INDEX FROM Seasons
    -> ;
+---------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+----------
--+
| Table   | Non_unique | Key_name     | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expressio
n |
+---------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+----------
--+
| Seasons |          1 | player_name  |            1 | player_name | A         |        2463 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| Seasons |          1 | season_index |            1 | season      | A         |          26 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| Seasons |          1 | gp_idx       |            1 | season      | A         |          26 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
```

Index on games played.

For our first subquery, the second index that we tried was to index based off of the games played. We thought this could potentially speed things up as it would have the database sorted by the amount of games they played in each season. However, this did not solve the problem as the time was originally at 55 seconds and this indexing bumped it up to 1 minute 9 seconds so we will not be using this.

```
mysql> CREATE INDEX age_idx ON Seasons (season)
    -> ;
Query OK, 0 rows affected, 1 warning (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 1

easonseasssgON,(ss.seasonULIKE.s.season)andess.player_nametLIKEts.player_name)SS
    -> GROUP BY ss.player_name;
+----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN

                                                                                                                                 |
+----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.004..0.613 rows=2463 loops=1)
    -> Aggregate using temporary table  (actual time=32705.544..32706.290 rows=2463 loops=1)
        -> Filter: ((ss.season like s.season) and (ss.player_name like s.player_name))  (cost=14796414.93 rows=1826182) (actual time=13.287..32636.169 rows=12313 loops=1)
            -> Inner hash join (no condition)  (cost=14796414.93 rows=1826182) (actual time=11.574..14915.025 rows=151413024 loops=1)
                -> Table scan on s  (cost=0.01 rows=12183) (actual time=0.028..36.895 rows=12305 loops=1)
                -> Hash
                    -> Table scan on ss  (cost=1238.65 rows=12144) (actual time=0.028..6.886 rows=12305 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------+
1 row in set (32.72 sec)
```

Index on age.

Our next index attempt was based on age. We thought that this would be smart because it would put consecutive seasons of a player relatively close to each other. This would be very similar to the indexing by season but we thought that this could potentially be better because it would be integer comparisons rather than a VARCHAR. The time ended up being 32.72 seconds, which is far better than using no index but the time for season based indexing was 32.71 seconds so we will be using that.

ADVANCED QUERY #2

```
-------------------------+
| -> Sort: player_name  (cost=2.50 rows=0) (actual time=0.032..0.034 rows=34 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.004 rows=34 loops=1)
        -> Union materialize with deduplication  (cost=7.50..7.50 rows=0) (actual time=66368.044..66368.048 rows=34 loops=1)
            -> Table scan on sub_query  (cost=2.50..2.50 rows=0) (actual time=0.001..0.003 rows=24 loops=1)
                -> Materialize  (cost=2.50..2.50 rows=0) (actual time=33633.342..33633.346 rows=24 loops=1)
                    -> Filter: (((round(sum((ss.pts * s.gp)),0) > 18000) and (round(sum((ss.reb * s.gp)),0) > 4000) and (round(sum((ss.ast * s.gp)),0) > 4000)) or (round(sum((ss.as
t * s.gp)),0) > 10000) or (round(sum((ss.reb * s.gp)),0) > 10000))  (actual time=33631.473..33633.290 rows=24 loops=1)
                        -> Table scan on <temporary>  (actual time=0.002..0.637 rows=2463 loops=1)
                            -> Aggregate using temporary table  (actual time=33631.401..33632.173 rows=2463 loops=1)
                                -> Filter: ((ss.season like s.season) and (ss.player_name like s.player_name))  (cost=14796414.93 rows=1826182) (actual time=13.115..33562.132 rows=
12313 loops=1)
                                    -> Inner hash join (no condition)  (cost=14796414.93 rows=1826182) (actual time=11.442..15085.553 rows=151413024 loops=1)
                                        -> Table scan on s  (cost=0.01 rows=12183) (actual time=0.022..40.056 rows=12305 loops=1)
                                        -> Hash
                                            -> Table scan on ss  (cost=1238.65 rows=12144) (actual time=0.025..6.556 rows=12305 loops=1)
            -> Table scan on sub_query  (cost=2.50..2.50 rows=0) (actual time=0.001..0.002 rows=13 loops=1)
                -> Materialize  (cost=2.50..2.50 rows=0) (actual time=32734.620..32734.622 rows=13 loops=1)
                    -> Filter: (((round(avg(ss.pts),0) > 25) and (round(avg(ss.reb),0) > 4) and (round(avg(ss.ast),0) > 4)) or (round(avg(ss.reb),0) > 12) or (round(avg(ss.pts),0)2
4.5) or (round(avg(ss.ast),0) > 10))  (actual time=32731.091..32734.567 rows=13 loops=1)
                        -> Table scan on <temporary>  (actual time=0.004..0.702 rows=2463 loops=1)
                            -> Aggregate using temporary table  (actual time=32731.072..32731.911 rows=2463 loops=1)
                                -> Filter: ((ss.season like s.season) and (ss.player_name like s.player_name))  (cost=14796414.93 rows=1826182) (actual time=12.625..32669.162 rows=
12313 loops=1)
                                    -> Inner hash join (no condition)  (cost=14796414.93 rows=1826182) (actual time=10.984..14867.713 rows=151413024 loops=1)
                                        -> Table scan on s  (cost=0.01 rows=12183) (actual time=0.024..35.004 rows=12305 loops=1)
                                        -> Hash
                                            -> Table scan on ss  (cost=1238.65 rows=12144) (actual time=0.035..6.539 rows=12305 loops=1)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------+
1 row in set (1 min 6.38 sec)
```

Original no index added.

```
-------------------------+
| -> Sort: player_name  (cost=2.50 rows=0) (actual time=0.046..0.048 rows=34 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.004 rows=34 loops=1)
        -> Union materialize with deduplication  (cost=7.50..7.50 rows=0) (actual time=67014.024..67014.027 rows=34 loops=1)
            -> Table scan on sub_query  (cost=2.50..2.50 rows=0) (actual time=0.001..0.004 rows=24 loops=1)
                -> Materialize  (cost=2.50..2.50 rows=0) (actual time=34061.247..34061.251 rows=24 loops=1)
                    -> Filter: (((round(sum((ss.pts * s.gp)),0) > 18000) and (round(sum((ss.reb * s.gp)),0) > 4000) and (round(sum((ss.ast * s.gp)),0) > 4000)) or (round(sum((ss.as
t * s.gp)),0) > 10000) or (round(sum((ss.reb * s.gp)),0) > 10000))  (actual time=34059.365..34061.206 rows=24 loops=1)
                        -> Table scan on <temporary>  (actual time=0.003..0.677 rows=2463 loops=1)
                            -> Aggregate using temporary table  (actual time=34059.282..34060.093 rows=2463 loops=1)
                                -> Filter: ((ss.season like s.season) and (ss.player_name like s.player_name))  (cost=14796414.93 rows=1826182) (actual time=13.369..33980.609 rows=
12313 loops=1)
                                    -> Inner hash join (no condition)  (cost=14796414.93 rows=1826182) (actual time=11.574..15407.832 rows=151413024 loops=1)
                                        -> Table scan on s  (cost=0.01 rows=12183) (actual time=0.020..43.830 rows=12305 loops=1)
                                        -> Hash
                                            -> Table scan on ss  (cost=1238.65 rows=12144) (actual time=0.025..6.876 rows=12305 loops=1)
            -> Table scan on sub_query  (cost=2.50..2.50 rows=0) (actual time=0.001..0.002 rows=13 loops=1)
                -> Materialize  (cost=2.50..2.50 rows=0) (actual time=32952.680..32952.682 rows=13 loops=1)
                    -> Filter: (((round(avg(ss.pts),0) > 25) and (round(avg(ss.reb),0) > 4) and (round(avg(ss.ast),0) > 4)) or (round(avg(ss.reb),0) > 12) or (round(avg(ss.pts),0)2
4.5) or (round(avg(ss.ast),0) > 10))  (actual time=32949.227..32952.634 rows=13 loops=1)
                        -> Table scan on <temporary>  (actual time=0.003..0.692 rows=2463 loops=1)
                            -> Aggregate using temporary table  (actual time=32949.209..32950.035 rows=2463 loops=1)
                                -> Filter: ((ss.season like s.season) and (ss.player_name like s.player_name))  (cost=14796414.93 rows=1826182) (actual time=12.524..32889.162 rows=
12313 loops=1)
                                    -> Inner hash join (no condition)  (cost=14796414.93 rows=1826182) (actual time=10.855..14840.489 rows=151413024 loops=1)
                                        -> Table scan on s  (cost=0.01 rows=12183) (actual time=0.025..34.637 rows=12305 loops=1)
                                        -> Hash
                                            -> Table scan on ss  (cost=1238.65 rows=12144) (actual time=0.048..6.676 rows=12305 loops=1)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

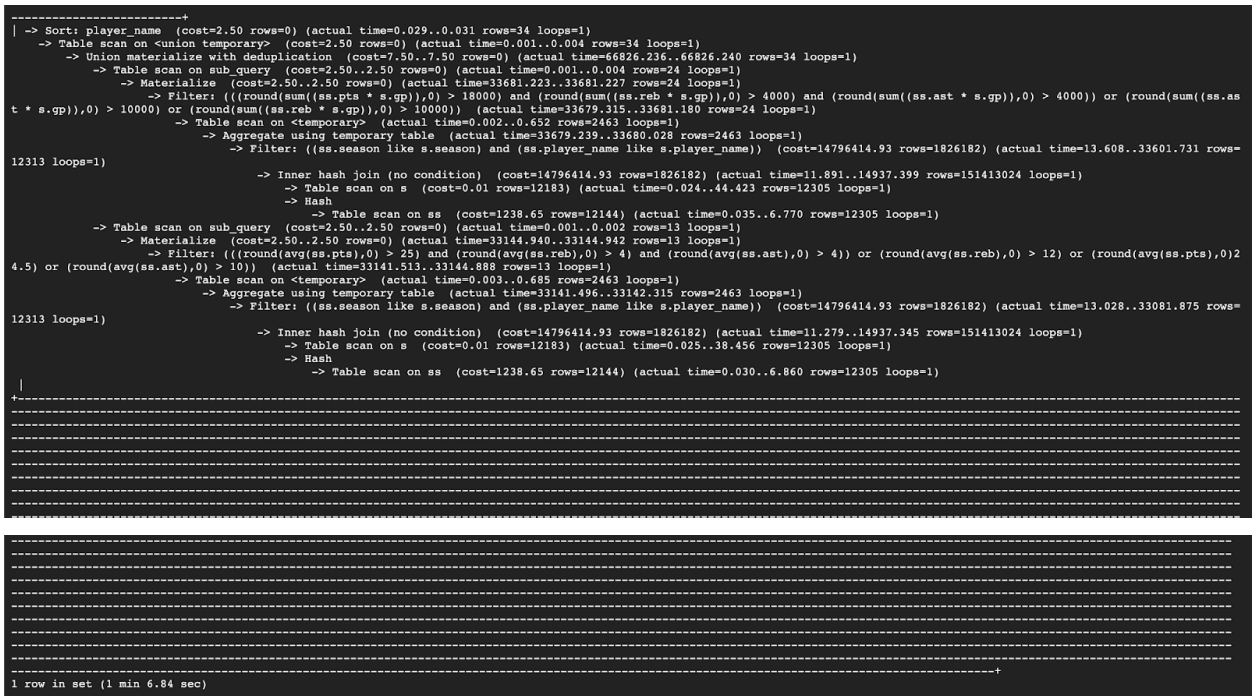
```
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------+
1 row in set (1 min 7.02 sec)

mysql>
```

Index on Rebounds

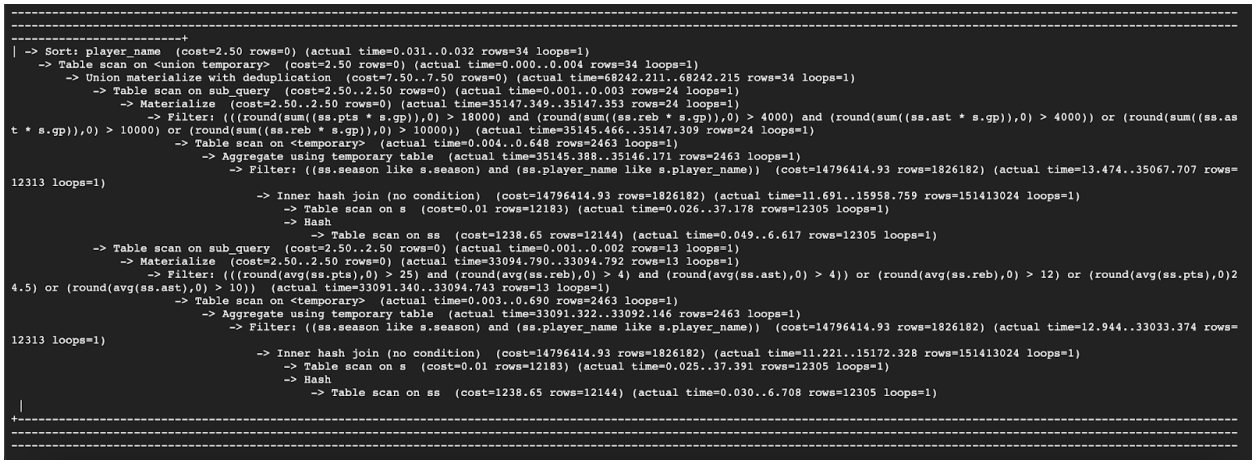
The goal of our second subquery was to create a hall of fame predictor metric using a union between two subqueries. Each subquery utilizes statistical metrics in order to calculate the hall of fame prediction. For the second advanced query, the first index that we tried was rebounding. A player's rebounding averages over successive seasons don't fluctuate too much which means all of one player's season should be pretty close together in the database. Compared to the original no-index run time of 1 minute 6.38

seconds, indexing using rebounds had a runtime of 1 minute 7.02 seconds. From this, we can see that the overall query slowed down in runtime.



Index on Points

The goal of our second subquery was to create a hall of fame predictor metric using a union between two subqueries. Each subquery utilizes statistical metrics in order to calculate the hall of fame prediction. For the second advanced query, the second index that we tried to index by was points. We tried to index by points because most player's average similar points per game over each season. Compared to the original no-index run time of 1 minute 6.38 seconds, indexing using points had a runtime of 1 minute 6.84 seconds. From this, we can see that the overall query slightly slowed down in runtime.



```
-> Sort: player_name  (cost=2.50 rows=0) (actual time=0.031..0.032 rows=34 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.000..0.004 rows=34 loops=1)
        -> Union materialize with deduplication  (cost=7.50..7.50 rows=0) (actual time=68242.211..68242.215 rows=34 loops=1)
            -> Table scan on sub_query  (cost=2.50..2.50 rows=0) (actual time=0.001..0.003 rows=24 loops=1)
                -> Materialize  (cost=2.50..2.50 rows=0) (actual time=35147.349..35147.353 rows=24 loops=1)
                    -> Filter: (((round(sum((ss.pts * s.gp)),0) > 18000) and (round(sum((ss.reb * s.gp)),0) > 4000) and (round(sum((ss.ast * s.gp)),0) > 4000)) or (round(sum((ss.as
t * s.gp)),0) > 10000) or (round(sum((ss.reb * s.gp)),0) > 10000))  (actual time=35145.466..35147.309 rows=24 loops=1)
                        -> Table scan on <temporary>  (actual time=0.004..0.648 rows=2463 loops=1)
                            -> Aggregate using temporary table  (actual time=35145.388..35146.171 rows=2463 loops=1)
                                -> Filter: ((ss.season like s.season) and (ss.player_name like s.player_name))  (cost=14796414.93 rows=1826182) (actual time=13.474..35067.707 rows=
12313 loops=1)
                                    -> Inner hash join (no condition)  (cost=14796414.93 rows=1826182) (actual time=11.691..15958.759 rows=151413024 loops=1)
                                        -> Table scan on s  (cost=0.01 rows=12183) (actual time=0.026..37.178 rows=12305 loops=1)
                                        -> Hash
                                            -> Table scan on ss  (cost=1238.65 rows=12144) (actual time=0.049..6.617 rows=12305 loops=1)
            -> Table scan on sub_query  (cost=2.50..2.50 rows=0) (actual time=0.001..0.002 rows=13 loops=1)
                -> Materialize  (cost=2.50..2.50 rows=0) (actual time=33094.790..33094.792 rows=13 loops=1)
                    -> Filter: (((round(avg(ss.pts),0) > 25) and (round(avg(ss.reb),0) > 4) and (round(avg(ss.ast),0) > 4)) or (round(avg(ss.reb),0) > 12) or (round(avg(ss.pts),0)2
4.5) or (round(avg(ss.ast),0) > 10))  (actual time=33091.340..33094.743 rows=13 loops=1)
                        -> Table scan on <temporary>  (actual time=0.003..0.690 rows=2463 loops=1)
                            -> Aggregate using temporary table  (actual time=33091.322..33092.146 rows=2463 loops=1)
                                -> Filter: ((ss.season like s.season) and (ss.player_name like s.player_name))  (cost=14796414.93 rows=1826182) (actual time=12.944..33033.374 rows=
12313 loops=1)
                                    -> Inner hash join (no condition)  (cost=14796414.93 rows=1826182) (actual time=11.221..15172.328 rows=151413024 loops=1)
                                        -> Table scan on s  (cost=0.01 rows=12183) (actual time=0.025..37.391 rows=12305 loops=1)
                                        -> Hash
                                            -> Table scan on ss  (cost=1238.65 rows=12144) (actual time=0.030..6.708 rows=12305 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (1 min 8.26 sec)
```

Index on Assists

The goal of our second subquery was to create a hall of fame predictor metric using a union between two subqueries. Each subquery utilizes statistical metrics in order to calculate the hall of fame prediction. For the second advanced query, the third index that we tried to index by was the assists category. Assists is a statistic that stays similar for most players on a year to year basis. This means that each year assist averages will be similar so all of one player's seasons will be close together in the data table. Compared to the original no-index run time of 1 minute 6.38 seconds, indexing using assists had a runtime of 1 minute 8.26 seconds. From this, we can see that the overall query significantly slowed down in runtime.