Project Phase 3

Screenshot of Connection to GCP - MySQL@GCP

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to statspot.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
akshath_sk@cloudshell:~ (statspot)$ gcloud sql connect stat-spot-db --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 2659
Server version: 8.0.26-google (Google)

Copyright (c) 2000, 2023, 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> show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| README_TO_RECOVER_A |
| information_schema  |
| mysql               |
| performance_schema  |
| stat-spot-db        |
| sys                 |
+--------------------+
6 rows in set (0.01 sec)

mysql> use stat-spot-db;
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_stat-spot-db |
+------------------------+
| Athlete                |
| Club                   |
| Coach                  |
| Compete                |
| Country                |
| Game                   |
| League                 |
| Medals                 |
| PartOf                 |
| Sport                  |
| Subscribe              |
| Tournament             |
| Train                  |
| UserDetail             |
+------------------------+
14 rows in set (0.00 sec)
```

Table DDL Commands

Total Tables Created = 13

Database Name - stat-spot-db

```
CREATE DATABASE IF NOT EXISTS `stat-spot-db`;
USE `stat-spot-db`;
```

Table Name - League

```
CREATE TABLE IF NOT EXISTS League (
leagueId INT,
type VARCHAR(128),
subType VARCHAR(128),
name VARCHAR(128),
PRIMARY KEY(leagueId)
);
```

Table Name - Country

```
CREATE TABLE IF NOT EXISTS Country (
countryId INT,
name VARCHAR(32),
continent VARCHAR(32),
PRIMARY KEY(countryId)
);
```

Table Name - Tournament

```
CREATE TABLE IF NOT EXISTS Tournament (
tournamentId INT,
name VARCHAR(32),
year INT,
description VARCHAR(256),
PRIMARY KEY(tournamentId)
);
```

Table Name - Club

```
CREATE TABLE IF NOT EXISTS Club (
clubId INT,
name VARCHAR(128),
owner VARCHAR(128),
established INT,
totalMarketValue REAL,
squadSize INT,
averageAge REAL,
totalNationalTeamPlayers INT,
PRIMARY KEY(clubId)
);
```

Table Name - Game

```
CREATE TABLE `Game` (
`gameId` int NOT NULL,
`season` varchar(128) DEFAULT NULL,
`round` varchar(128) DEFAULT NULL,
`gameDate` date DEFAULT NULL,
`homeClubId` int DEFAULT NULL,
`awayClubId` int DEFAULT NULL,
`homeClubGoals` int DEFAULT NULL,
`awayClubGoals` int DEFAULT NULL,
PRIMARY KEY (`gameId`),
CONSTRAINT `GameToClub-fk1` FOREIGN KEY (`homeClubId`) REFERENCES `Club`
(`clubId`),
CONSTRAINT `GameToClub-fk2` FOREIGN KEY (`awayClubId`) REFERENCES `Club`
(`clubId`)
);
```

Table Name - UserDetail

```
CREATE TABLE IF NOT EXISTS UserDetail (
userId INT,
userName VARCHAR(64),
password VARCHAR(32),
email VARCHAR(32),
firstName VARCHAR(32),
lastName VARCHAR(32),
PRIMARY KEY(userId)
);
```

Table Name - Sport

```
CREATE TABLE IF NOT EXISTS Sport (
sportId INT,
name VARCHAR(128),
description VARCHAR(256),
playerCount INT,
duration REAL,
PRIMARY KEY(sportId)
);
```

Table Name - Coach

```
CREATE TABLE IF NOT EXISTS Coach (
coachId INT,
name VARCHAR(128),
countryId INT,
dateOfBirth DATE,
clubId INT,
PRIMARY KEY(coachId),
CONSTRAINT `CocahToClub-fk1` FOREIGN KEY (`clubId`) REFERENCES `Club` (`clubId`),
CONSTRAINT `CocahToCountry-fk2` FOREIGN KEY (`countryId`) REFERENCES `Country`
(`countryId`)
);
```

Table Name - Athlete

```
CREATE TABLE IF NOT EXISTS Athlete (
athleteId INT,
name VARCHAR(128),
dateOfBirth DATE,
position VARCHAR(128),
marketValue REAL,
sex VARCHAR(12),
sportId INT,
countryId INT,
clubId INT,
PRIMARY KEY(athleteId),
CONSTRAINT `AthleteToSport-fk1` FOREIGN KEY (`sportId`) REFERENCES `Sport`
(`sportId`),
CONSTRAINT `AthleteToCountry-fk2` FOREIGN KEY (`countryId`) REFERENCES `Country`
(`countryId`),
CONSTRAINT `AthleteToClub-fk3` FOREIGN KEY (`clubId`) REFERENCES `Club` (`clubId`)
);
```

Table Name - Train

```
CREATE TABLE IF NOT EXISTS Train (
athleteId INT,
coachId INT,
PRIMARY KEY(athleteId, coachId),
CONSTRAINT `TrainToAthlete-fk1` FOREIGN KEY (`athleteId`) REFERENCES `Athlete`
(`athleteId`),
CONSTRAINT `TrainToCoach-fk2` FOREIGN KEY (`coachId`) REFERENCES `Coach`
(`coachId`)
);
```

Table Name - Compete

```
CREATE TABLE IF NOT EXISTS Compete (
tournamentId INT,
athleteId INT,
PRIMARY KEY(tournamentId, athleteId),
CONSTRAINT `CompeteToTournament-fk1` FOREIGN KEY (`tournamentId`) REFERENCES
`Tournament` (`tournamentId`),
CONSTRAINT `CompeteToAthlete-fk2` FOREIGN KEY (`athleteId`) REFERENCES `Athlete`
(`athleteId`)
);
```

Table Name - PartOf

```
CREATE TABLE IF NOT EXISTS PartOf (
clubId INT,
leagueId INT,
PRIMARY KEY(clubId, leagueId),
CONSTRAINT `PartOfToClub-fk1` FOREIGN KEY (`clubId`) REFERENCES `Club` (`clubId`),
CONSTRAINT `PartOfToLeague-fk2` FOREIGN KEY (`leagueId`) REFERENCES `League`
(`leagueId`)
);
```

Table Name - Participate

```
CREATE TABLE IF NOT EXISTS Participate (
clubId INT,
gameId INT,
PRIMARY KEY(clubId, gameId),
CONSTRAINT `ParticipateToClub-fk1` FOREIGN KEY (`clubId`) REFERENCES `Club`
(`clubId`),
CONSTRAINT `ParticipateToLeague-fk2` FOREIGN KEY (`gameId`) REFERENCES `Game`
(`gameId`)
);
```

Table Name - Subscribe

```
CREATE TABLE IF NOT EXISTS Subscribe (
gameId INT,
userId INT,
pay REAL,
PRIMARY KEY(gameId, userId),
CONSTRAINT `SubscribeToGame-fk1` FOREIGN KEY (`gameId`) REFERENCES `Game`
(`gameId`),
CONSTRAINT `SubscribeToUser-fk2` FOREIGN KEY (`userId`) REFERENCES `UserDetail`
(`userId`)
);
```

Table Name - Medals

```
CREATE TABLE IF NOT EXISTS Medals (
medalId INT,
tournamentId INT,
type VARCHAR(12),
athleteId INT,
PRIMARY KEY(medalId, tournamentId),
CONSTRAINT `MedalsToGame-fk1` FOREIGN KEY (`tournamentId`) REFERENCES
`Tournament` (`tournamentId`) ON DELETE CASCADE,
CONSTRAINT `MedalsToAthlete-fk2` FOREIGN KEY (`athleteId`) REFERENCES `Athlete`
(`athleteId`)
);
```

Counts of Records in Tables

Table	Row Count
Club	411
Country	587
Sport	46
Athlete	38050
League	43
UserDetail	1500
Coach	2305
Game	48098
Train	24
Compete	11085
PartOf	411
Subscribe	1500
Medals	11085

StatSpot data was loaded from 2 datasets -

1. 2021 Olympics in Tokyo - https://www.kaggle.com/datasets/arjunprasadsarkhel/2021-olympics-in-tokyo
2. Football Data from Transfermarkt - https://www.kaggle.com/datasets/davidcariboo/player-scores?select=games.csv

A java program (fatty jar) was created to generate the insert statements. This jar reads the data from these 2 kaggle datasets and creates insert statements taking into account the foreign key relationships.
This program can be found at - sp23-cs411-team047-selectStars/dml-sql-generator/src/main/java/org/uiuc/Main.java

A total of 13 tables were populated.
The main 3 tables are **Athlete (38050 rows), Coach (2305 rows),** and **Game (48098 rows)**.
The other tables with more than 1000 rows include - UserDetail (1500 rows), Compete (11085 rows), Subscribe (1500 rows), and Medals (11085 rows).

```
mysql> select count(*) from Club;
+----------+
| count(*) |
+----------+
|      411 |
+----------+
1 row in set (0.10 sec)

mysql> select count(*) from Country;
+----------+
| count(*) |
+----------+
|      587 |
+----------+
1 row in set (0.04 sec)

mysql> select count(*) from Sport;
+----------+
| count(*) |
+----------+
|       46 |
+----------+
1 row in set (0.06 sec)

mysql> select count(*) from Athlete;
+----------+
| count(*) |
+----------+
|    38050 |
+----------+
1 row in set (0.63 sec)

mysql> select count(*) from League;
+----------+
| count(*) |
+----------+
|       43 |
+----------+
1 row in set (0.07 sec)
```

```
mysql> select count(*) from UserDetail;
+----------+
| count(*) |
+----------+
|     1500 |
+----------+
1 row in set (0.08 sec)

mysql> select count(*) from Coach;
+----------+
| count(*) |
+----------+
|     2305 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from Game;
+----------+
| count(*) |
+----------+
|    48098 |
+----------+
1 row in set (1.00 sec)

mysql> select count(*) from Train;
+----------+
| count(*) |
+----------+
|       24 |
+----------+
1 row in set (0.05 sec)

mysql> select count(*) from Compete;
+----------+
| count(*) |
+----------+
|    11085 |
+----------+
1 row in set (0.28 sec)
```

```
mysql> select count(*) from PartOf;
+----------+
| count(*) |
+----------+
|      411 |
+----------+
1 row in set (0.05 sec)

mysql> select count(*) from Subscribe;
+----------+
| count(*) |
+----------+
|     1500 |
+----------+
1 row in set (0.13 sec)

mysql> select count(*) from Medals;
+----------+
| count(*) |
+----------+
|    11085 |
+----------+
1 row in set (0.23 sec)
```

Advanced Queries & Index Analysis

Query 1

Description

Get the total market value of all young players in the team who play in an attacking position. Young players are those who were born after 2000-01-01. Return top 15 highest market values per club.

Query

```
SELECT
	C.clubId,
	C.name,
	sum(A.marketValue) as TotalMarketValue
FROM
	`stat-spot-db`.Club C
		LEFT JOIN `stat-spot-db`.Athlete A on C.clubId = A.clubId
WHERE
	A.position = 'Attack'
	and
	A.dateOfBirth > '2000-01-01'
GROUP BY
	C.clubId,
	C.name
ORDER BY
	TotalMarketValue desc
LIMIT 15;
```

Output

ClubID	ClubName	TotalMarketValue
499	Vitesse Arnheim	1500
589	Antalyaspor	1400
3329	Fc Famalicao	1400
1083	Fk Rostov	1300
3205	Kayserispor	1300
79	Vfb Stuttgart	1300
1184	Krc Genk	1200
1519	Dundee United Fc	1200
370	Aberdeen Fc	1200
2414	Ac Horsens	1100
306	Sc Heerenveen	1100
449	Trabzonspor	1100
385	Fortuna Sittard	1100
173	Odense Boldklub	1100
24	Eintracht Frankfurt	1100

Returns 15 rows

Run Explain Analyze

```
mysql> EXPLAIN ANALYZE SELECT C.clubId as ClubID, C.name as ClubName, sum(A.marketValue) as TotalMarketValue
    -> FROM `stat-spot-db`.Club C LEFT JOIN `stat-spot-db`.Athlete A on C.clubId = A.clubId
    -> WHERE A.position = 'Attack' and A.dateOfBirth > '2000-01-01'
    -> GROUP BY C.clubId, C.name
    -> ORDER BY TotalMarketValue desc
    -> LIMIT 15;
```

Explain Analyze Result

-> Limit: 15 row(s) (actual time=21.942..21.944 rows=15 loops=1)
    -> Sort: TotalMarketValue DESC, limit input to 15 row(s) per chunk (actual time=21.941..21.942 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=0.002..0.054 rows=311 loops=1)
            -> Aggregate using temporary table (actual time=21.797..21.868 rows=311 loops=1)
                -> Nested loop inner join (cost=4096.56 rows=1201) (actual time=2.374..20.303 rows=1747 loops=1)
                    -> Filter: ((A.position = 'Attack') and (A.dateOfBirth > DATE'2000-01-01') and (A.clubId is not null)) (cost=3676.15 rows=1201) (actual time=2.354..18.571 rows=1747 loops=1)
                        -> Table scan on A (cost=3676.15 rows=36039) (actual time=0.073..13.481 rows=38050 loops=1)
                    -> Single-row index lookup on C using PRIMARY (clubId=A.clubId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1747)

Add Index 1 on “marketValue”

CREATE INDEX athlete_marketValue
on `stat-spot-db`.Athlete(marketValue);

Field	Type	Null	Key	Default	Extra
athleteId	int	NO	PRI	NULL	
name	varchar(128)	YES		NULL	
dateOfBirth	date	YES		NULL	
position	varchar(128)	YES		NULL	
marketValue	double	YES	MUL	NULL	
sex	varchar(12)	YES		NULL	
sportId	int	YES	MUL	NULL	
countryId	int	YES	MUL	NULL	
clubId	int	YES	MUL	NULL	

Run Explain Analyze after Index 1

-> Limit: 15 row(s) (actual time=21.425..21.427 rows=15 loops=1)
    -> Sort: TotalMarketValue DESC, limit input to 15 row(s) per chunk (actual time=21.424..21.425 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=0.002..0.048 rows=311 loops=1)
            -> Aggregate using temporary table (actual time=21.300..21.365 rows=311 loops=1)
                -> Nested loop inner join (cost=4096.56 rows=1201) (actual time=2.094..19.837 rows=1747 loops=1)
                    -> Filter: ((A.position = 'Attack') and (A.dateOfBirth > DATE'2000-01-01') and (A.clubId is not null)) (cost=3676.15 rows=1201) (actual time=2.079..18.202 rows=1747 loops=1)
                        -> Table scan on A (cost=3676.15 rows=36039) (actual time=0.080..13.256 rows=38050 loops=1)
                    -> Single-row index lookup on C using PRIMARY (clubId=A.clubId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1747)

Analysis -
Adding an index on Athlete.marketValue improves performance only marginally. This is mainly because the marketValue is used during the aggregation on the temporary table as seen in the query above. This column is not used during the table scan or during the filtering process. **Removing this index as it is not providing significant performance improvement.**

Add Index 2 on “position”

CREATE INDEX athlete_position
on `stat-spot-db`.Athlete(position);

Field	Type	Null	Key	Default	Extra
athleteId	int	NO	PRI	NULL	
name	varchar(128)	YES		NULL	
dateOfBirth	date	YES		NULL	
position	varchar(128)	YES	MUL	NULL	
marketValue	double	YES		NULL	
sex	varchar(12)	YES		NULL	
sportId	int	YES	MUL	NULL	
countryId	int	YES	MUL	NULL	
clubId	int	YES	MUL	NULL	

Run Explain Analyze after Index 2

-> Limit: 15 row(s) (actual time=16.486..16.488 rows=15 loops=1)
    -> Sort: TotalMarketValue DESC, limit input to 15 row(s) per chunk (actual time=16.485..16.486 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=0.002..0.030 rows=311 loops=1)
            -> Aggregate using temporary table (actual time=16.384..16.430 rows=311 loops=1)
                -> Nested loop inner join (cost=2689.70 rows=5495) (actual time=2.070..14.881 rows=1747 loops=1)
                    -> Filter: ((A.dateOfBirth > DATE'2000-01-01') and (A.clubId is not null)) (cost=766.30 rows=5495) (actual time=2.057..13.171 rows=1747 loops=1)
                        -> Index lookup on A using athlete_position (position='Attack') (cost=766.30 rows=16488) (actual time=0.200..12.394 rows=9162 loops=1)
                    -> Single-row index lookup on C using PRIMARY (clubId=A.clubId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1747)

Analysis-
Adding an index on Athlete.position provides a very significant performance improvement as well as reduction of cost. This is mainly because the Table scan on Athlete is now moved to an index lookup using the unclustered index on position. This gave a cost improvement from 3676.15 to 766.30. **Keeping this index as it gave a significant performance improvement.**

Add Index 3 on “dateOfBirth”

CREATE INDEX athlete_dob
on `stat-spot-db`.Athlete(dateOfBirth);

Field	Type	Null	Key	Default	Extra
athleteId	int	NO	PRI	NULL	
name	varchar(128)	YES		NULL	
dateOfBirth	date	YES	MUL	NULL	
position	varchar(128)	YES	MUL	NULL	
marketValue	double	YES		NULL	
sex	varchar(12)	YES		NULL	
sportId	int	YES	MUL	NULL	
countryId	int	YES	MUL	NULL	
clubId	int	YES	MUL	NULL	

Run Explain Analyze after Index 3

-> Limit: 15 row(s) (actual time=15.551..15.553 rows=15 loops=1)
    -> Sort: TotalMarketValue DESC, limit input to 15 row(s) per chunk (actual time=15.550..15.551 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=0.002..0.028 rows=311 loops=1)
            -> Aggregate using temporary table (actual time=15.451..15.495 rows=311 loops=1)
                -> Nested loop inner join (cost=2396.37 rows=4844) (actual time=1.942..14.071 rows=1747 loops=1)
                    -> Filter: ((A.dateOfBirth > DATE'2000-01-01') and (A.clubId is not null)) (cost=701.11 rows=4844) (actual time=1.930..12.495 rows=1747 loops=1)
                        -> Index lookup on A using athlete_position (position='Attack') (cost=701.11 rows=16488) (actual time=0.183..11.736 rows=9162 loops=1)
                    -> Single-row index lookup on C using PRIMARY (clubId=A.clubId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1747)

Analysis -
Adding an index on Athlete.dateOfBirth provides relatively lower performance improvement as compared to Athlete.position as the index lookup still uses Athlete.position. This filter on the Table gave a cost improvement from 766.30 (with Athlete.position) to 701.11. **Keeping this index as it gave performance improvements.**

Query 2

Description

For the 2021 season find the leagues with the lowest home goals. Don’t consider the goals scored in the Fifa Club World Cup.

Query

```
SELECT
	L.leagueId,
	L.type,
	sum(G.homeClubGoals) as HomeGoalsTotal
FROM
	`stat-spot-db`.League L
		INNER JOIN PartOf P on P.leagueId = L.leagueId
		INNER JOIN Club C on C.clubId = P.clubId
		INNER JOIN Game G on G.homeClubId = C.clubId
WHERE
	G.season = '2021'
	and
	L.name not in ('fifa_club_world_cup')
GROUP BY
	L.leagueId,
	L.type
ORDER BY
	HomeGoalsTotal
LIMIT 15;
```

Output

leagueId	type	HomeGoalsTotal
25	Ukrainian Cup	219
36	Sydbank Pokalen	235
33	Kypello Elladas	282
28	Sfa Cup	310
22	Russian Cup	440
8	Belgian Supercup	499
13	Allianz Cup	505
5	Toto Knvb Beker	553
10	Copa Del Rey	620
35	Superligaen	625
16	Trophee Des Ch…	628
2	Dfb Pokal	653
30	Italy Cup	676
18	Efl Cup	710

Returns 14 rows

Run Explain Analyze

```
mysql> EXPLAIN ANALYZE
    -> SELECT  L.leagueId, L.type, sum(G.homeClubGoals) as HomeGoalsTotal
    -> FROM  `stat-spot-db`.League L INNER JOIN PartOf P on P.leagueId = L.leagueId
    -> INNER JOIN Club C on C.clubId = P.clubId
    -> INNER JOIN Game G on G.homeClubId = C.clubId
    -> WHERE G.season = '2021' and L.name not in ('fifa_club_world_cup')
    -> GROUP BY L.leagueId, L.type
    -> ORDER BY HomeGoalsTotal
    -> LIMIT 15;
```

Explain Analyze Result

-> Limit: 15 row(s) (actual time=40.809..40.811 rows=14 loops=1)
   -> Sort: HomeGoalsTotal, limit input to 15 row(s) per chunk (actual time=40.808..40.809 rows=14 loops=1)
      -> Table scan on <temporary> (actual time=0.001..0.004 rows=14 loops=1)
         -> Aggregate using temporary table (actual time=40.783..40.786 rows=14 loops=1)
            -> Nested loop inner join (cost=9537.07 rows=4213) (actual time=17.558..36.481 rows=4484 loops=1)
               -> Nested loop inner join (cost=8062.46 rows=4213) (actual time=17.548..32.259 rows=4484 loops=1)
                  -> Nested loop inner join (cost=6424.01 rows=4681) (actual time=17.536..29.278 rows=4484 loops=1)
                     -> Filter: ((G.season = '2021') and (G.homeClubId is not null)) (cost=4785.55 rows=4681) (actual time=17.510..21.753 rows=4484 loops=1)
                        -> Table scan on G (cost=4785.55 rows=46813) (actual time=0.090..15.083 rows=48098 loops=1)
                     -> Index lookup on P using PRIMARY (clubId=G.homeClubId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=4484)
                  -> Filter: (L.`name` <> 'fifa_club_world_cup') (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=4484)
                     -> Single-row index lookup on L using PRIMARY (leagueId=P.leagueId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=4484)
               -> Single-row index lookup on C using PRIMARY (clubId=G.homeClubId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=4484)

Add Index 1 on “homeClubGoals”

CREATE INDEX game_homeClubGoals
on `stat-spot-db`.Game(homeClubGoals);

Field	Type	Null	Key	Default	Extra
gameId	int	NO	PRI	NULL	
season	varchar(128)	YES		NULL	
round	varchar(128)	YES		NULL	
gameDate	date	YES		NULL	
homeClubId	int	YES	MUL	NULL	
awayClubId	int	YES	MUL	NULL	
homeClubGoals	int	YES	MUL	NULL	
awayClubGoals	int	YES		NULL	

Run Explain Analyze after Index 1

-> Limit: 15 row(s) (actual time=40.760..40.762 rows=14 loops=1)
    -> Sort: HomeGoalsTotal, limit input to 15 row(s) per chunk (actual time=40.759..40.760 rows=14 loops=1)
        -> Table scan on <temporary> (actual time=0.001..0.004 rows=14 loops=1)
            -> Aggregate using temporary table (actual time=40.735..40.738 rows=14 loops=1)
                -> Nested loop inner join (cost=9537.07 rows=4213) (actual time=17.166..36.391 rows=4484 loops=1)
                    -> Nested loop inner join (cost=8062.46 rows=4213) (actual time=17.160..32.116 rows=4484 loops=1)
                        -> Nested loop inner join (cost=6424.01 rows=4681) (actual time=17.151..29.034 rows=4484 loops=1)
                            -> Filter: ((G.season = '2021') and (G.homeClubId is not null)) (cost=4785.55 rows=4681) (actual time=17.126..21.392 rows=4484 loops=1)
                                -> Table scan on G (cost=4785.55 rows=46813) (actual time=0.067..14.890 rows=48098 loops=1)
                            -> Index lookup on P using PRIMARY (clubId=G.homeClubId) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=4484)
                        -> Filter: (L.`name` <> 'fifa_club_world_cup') (cost=0.25 rows=1) (actual time=0.000..0.001 rows=1 loops=4484)
                            -> Single-row index lookup on L using PRIMARY (leagueId=P.leagueId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=4484)
                    -> Single-row index lookup on C using PRIMARY (clubId=G.homeClubId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=4484)

Analysis -

Adding an index on Game.homeClubGoals improves performance only marginally. This is mainly because the homeClubGoals is used during the aggregation on the temporary table as seen in the query above. This column is not used during the table scan or during the filtering process. **Removing this index as it is not providing significant performance improvement.**

Add Index 2 on “season”

```
CREATE INDEX game_season
on `stat-spot-db`.Game(season);
```

Field	Type	Null	Key	Default	Extra
gameId	int	NO	PRI	NULL	
season	varchar(128)	YES	MUL	NULL	
round	varchar(128)	YES		NULL	
gameDate	date	YES		NULL	
homeClubId	int	YES	MUL	NULL	
awayClubId	int	YES	MUL	NULL	
homeClubGoals	int	YES		NULL	
awayClubGoals	int	YES		NULL	

Run Explain Analyze after Index 2

```
-> Limit: 15 row(s)  (actual time=26.359..26.361 rows=14 loops=1)
    -> Sort: HomeGoalsTotal, limit input to 15 row(s) per chunk  (actual time=26.358..26.359 rows=14 loops=1)
        -> Table scan on <temporary>  (actual time=0.002..0.004 rows=14 loops=1)
            -> Aggregate using temporary table  (actual time=26.327..26.330 rows=14 loops=1)
                -> Nested loop inner join  (cost=5312.41 rows=4036) (actual time=0.259..21.698 rows=4484 loops=1)
                    -> Nested loop inner join  (cost=3899.95 rows=4036) (actual time=0.226..17.281 rows=4484 loops=1)
                        -> Nested loop inner join  (cost=2330.55 rows=4484) (actual time=0.215..14.242 rows=4484 loops=1)
                            -> Filter: (G.homeClubId is not null)  (cost=761.15 rows=4484) (actual time=0.205..6.524 rows=4484 loops=1)
                                -> Index lookup on G using game_season (season='2021')  (cost=761.15 rows=4484) (actual time=0.204..6.122 rows=4484 loops=1)
                            -> Index lookup on P using PRIMARY (clubId=G.homeClubId)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=4484)
                        -> Filter: (L.`name` <> 'fifa_club_world_cup')  (cost=0.25 rows=1) (actual time=0.000..0.001 rows=1 loops=4484)
                    -> Single-row index lookup on L using PRIMARY (leagueId=P.leagueId)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=4484)
```

-> Single-row index lookup on C using PRIMARY (clubId=G.homeClubId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=4484)

Analysis-
Adding an index on Game.season provides a very significant performance improvement as well as reduction of cost. This is mainly because the Table scan on Game is now moved to an index lookup using the unclustered index on season. This gave a cost improvement from 4785.55 to 761.15. **Keeping this index as it gave a significant performance improvement.**

Add Index 3 on League “name”

CREATE INDEX league_name
on `stat-spot-db`.League(name);

Field	Type	Null	Key	Default	Extra
leagueId	int	NO	PRI	NULL	
type	varchar(128)	YES		NULL	
subType	varchar(128)	YES		NULL	
name	varchar(128)	YES	MUL	NULL	

Run Explain Analyze after Index 3

-> Limit: 15 row(s) (actual time=24.571..24.573 rows=14 loops=1)
-> Sort: HomeGoalsTotal, limit input to 15 row(s) per chunk (actual time=24.570..24.571 rows=14 loops=1)
-> Table scan on <temporary> (actual time=0.002..0.004 rows=14 loops=1)
-> Aggregate using temporary table (actual time=24.545..24.548 rows=14 loops=1)
-> Nested loop inner join (cost=5432.85 rows=4380) (actual time=0.172..20.144 rows=4484 loops=1)
-> Nested loop inner join (cost=3899.95 rows=4380) (actual time=0.167..16.027 rows=4484 loops=1)
-> Nested loop inner join (cost=2330.55 rows=4484) (actual time=0.160..13.043 rows=4484 loops=1)
-> Filter: (G.homeClubId is not null) (cost=761.15 rows=4484) (actual time=0.154..5.809 rows=4484 loops=1)
-> Index lookup on G using game_season (season='2021') (cost=761.15 rows=4484) (actual time=0.152..5.435 rows=4484 loops=1)
-> Index lookup on P using PRIMARY (clubId=G.homeClubId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=4484)
-> Filter: (L.`name` <> 'fifa_club_world_cup') (cost=0.25 rows=1) (actual time=0.000..0.001 rows=1 loops=4484)
-> Single-row index lookup on L using PRIMARY (leagueId=P.leagueId) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=4484)

-> Single-row index lookup on C using PRIMARY (clubId=G.homeClubId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=4484)

Analysis -

Adding an index on League.name provides relatively lower performance improvement as compared to Game.season as the index lookup still uses Game.season. **Keeping this index as it gave an overall performance improvement. This can be useful when the table size increases in the future and for other queries as we expect many queries based on the League.name column in our application.** The performance is good and is not impacted by this index as the amount of data in the league table is small in the current snapshot of the database.