Q1. Write a drill SQL query to list the team and player data. Specifically display team name, team wins, team losses player name, player shots and player goals.

SQL query correctly joins the team and player statistics based on their respective team IDs, successfully retrieving the desired fields from both **teams** and **players** tables as specified.

```sql
SELECT
    teamData.name AS team_name,
    teamData.wins AS team_wins,
    teamData.losses AS team_losses,
    playerStats.name AS player_name,
    playerStats.shots AS player_shots,
    playerStats.goals AS player_goals
FROM
    mssql.`teams` AS teamData
INNER JOIN
    mssql.`players` AS playerStats ON teamData.id = playerStats.teamid;
```

NetId: snamani

Show 10 entries

NetId: snamani

Search:

Show / hide columns

team_name	team_wins	team_losses	player_name	player_shots	player_goals
syracuse	11	2	sam	56	23
syracuse	11	2	sarah	85	34
syracuse	11	2	steve	60	20
syracuse	11	2	stone	33	10
syracuse	11	2	sean	26	9
syracuse	11	2	sly	78	15
syracuse	11	2	sol	52	20
syracuse	11	2	shree	20	4
syracuse	11	2	shelly	10	2
syracuse	11	2	swede	90	50

Q2.Write a drill SQL query to display the gamestream. Label each of the columns in the gamestream with their appropriate columns names from the data dictionary.

The gamestream data has been displayed correctly with the appropriate names from minio.

```
SELECT
    columns[0] AS event,
    columns[1] AS time_stamp,
    columns[2] AS team_ID,
    columns[3] AS jersey_number,
    columns[4] AS team_goals
FROM
    minio.`gamestream.txt`
```

NetId: snamani

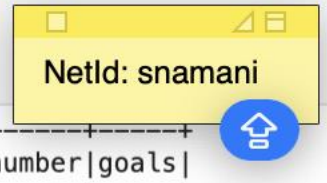
event	time_stamp	team_ID	jersey_number	team_goals
0	59:51	101	2	0
1	57:06	101	6	0
2	56:13	205	8	1
3	55:25	101	4	0
4	55:03	101	1	1
5	54:50	101	17	0
6	54:14	205	8	0
7	53:59	101	9	0
8	53:23	101	2	0
9	51:21	101	13	0

NetId: snamani

Q3. Write pyspark code (in SQL or DataFrame API) to display the gamestream. Label each of the columns in the gamestream with their appropriate columns names from the data dictionary.

Similarly the game stream data has been displayed correctly in spark with the appropriate column names

```
# Q3
from pyspark.sql.window import Window
from pyspark.sql import functions as func

split_cols = func.split(df_min['value'],' ')

df_label = df_min \
    .withColumn('event_id', split_cols.getItem(0).cast('int')) \
    .withColumn('timestamp', split_cols.getItem(1)) \
    .withColumn('team_ID', split_cols.getItem(2).cast('int')) \
    .withColumn('jersey_number', split_cols.getItem(3).cast('int')) \
    .withColumn('goals', split_cols.getItem(4).cast('int'))

df_2 = df_label.drop('value')

df_2.write.format("mongo") \
    .mode("overwrite") \
    .option("database", "sidearm") \
    .option("collection", "boxscores") \
    .save()

df_2.show(80)
```




```
+--------+---------+-------+-------------+-----+
|event_id|timestamp|team_ID|jersey_number|goals|
+--------+---------+-------+-------------+-----+
|       0|    59:51|    101|            2|    0|
|       1|    57:06|    101|            6|    0|
|       2|    56:13|    205|            8|    1|
|       3|    55:25|    101|            4|    0|
|       4|    55:03|    101|            1|    1|
|       5|    54:50|    101|           17|    0|
|       6|    54:14|    205|            8|    0|
|       7|    53:59|    101|            9|    0|
|       8|    53:23|    101|            2|    0|
```

Q4) Write pyspark code (in SQL or DataFrame API) to group the gamestream by team/player jersey number adding up the shots and goals. Specifically:

- One row per team / jersey number in the gamestream.
- Values dependent on team and player: total shots and goals for each player.
- Value dependent on only team: total goals (this should repeat for every row with the same team id)

The game stream data has been correctly grouped according to the team. And the total team goals column was correctly calculated as a sum of all the goals in each team.

```
# Q4

df_2.createOrReplaceTempView("gamestream")

sql_query = """

SELECT gs.team_ID, gs.jersey_number, COUNT(*) AS shots, SUM(gs.goals) AS goals, tg.team_goals
FROM gamestream gs
JOIN
    (SELECT team_ID, SUM(goals) AS team_goals
         FROM gamestream
         WHERE team_ID > 0
     GROUP BY team_ID)
tg ON gs.team_ID = tg.team_ID

WHERE gs.team_ID > 0
GROUP BY gs.team_ID, gs.jersey_number, tg.team_goals
ORDER BY gs.team_ID, gs.jersey_number
"""

result = spark.sql(sql_query)

result.show()
```

NetId: snamani

```
[Stage 366:=====================================>                  (134 + 1) / 200]
+-------+-------------+-----+-----+----------+
|team_ID|jersey_number|shots|goals|team_goals|
+-------+-------------+-----+-----+----------+
|    101|            1|    7|    6|        13|
|    101|            2|    6|    1|        13|
|    101|            4|    5|    1|        13|
|    101|            6|    4|    2|        13|
|    101|            8|    3|    0|        13|
|    101|            9|    4|    0|        13|
|    101|           10|    3|    1|        13|
|    101|           13|    5|    1|        13|
|    101|           15|    3|    1|        13|
|    101|           17|    1|    0|        13|
|    205|            1|    3|    3|         9|
|    205|            2|    3|    1|         9|
|    205|            3|    1|    0|         9|
|    205|            5|    1|    1|         9|
|    205|            8|    2|    1|         9|
|    205|            9|    4|    0|         9|
|    205|           15|    2|    2|         9|
|    205|           16|    1|    0|         9|
|    205|           17|    3|    1|         9|
|    205|           22|    1|    0|         9|
+-------+-------------+-----+-----+----------+
```

Q5. Use your output from 3. to include the most most current event id and timestamp for that point in time in the game. Same row level as 3. but now you include the latest event_id and timestamp.

The latest time stamp and the latest event id has been added to the data frame.

```
#Q5

from pyspark.sql import functions as func

df_2.createOrReplaceTempView("gamestream")

latest_event_and_timestamp = spark.sql("""
    SELECT
        MAX(event_id) AS latest_eventid,
        MAX(timestamp) AS latest_timestamp
    FROM gamestream
    WHERE team_ID != 0
""").collect()[0]

latest_eventid = latest_event_and_timestamp['latest_eventid']
latest_timestamp = latest_event_and_timestamp['latest_timestamp']

result_2 = result.withColumn("latest_eventid", func.lit(latest_eventid))\
                 .withColumn("latest_timestamp", func.lit(latest_timestamp))

result_2.show()
```

NetId: snamani

```
[Stage 372:=============================================>          (160 + 2) / 200]
+-------+-------------+-----+-----+----------+--------------+----------------+
|team_ID|jersey_number|shots|goals|team_goals|latest_eventid|latest_timestamp|
+-------+-------------+-----+-----+----------+--------------+----------------+
|    101|            1|    7|    6|        13|            61|           59:51|
|    101|            2|    6|    1|        13|            61|           59:51|
|    101|            4|    5|    1|        13|            61|           59:51|
|    101|            6|    4|    2|        13|            61|           59:51|
|    101|            8|    3|    0|        13|            61|           59:51|
|    101|            9|    4|    0|        13|            61|           59:51|
|    101|           10|    3|    1|        13|            61|           59:51|
|    101|           13|    5|    1|        13|            61|           59:51|
```

Q6.Write pyspark code (in SQL or DataFrame API) to join the output from question 4 with the player and team reference data mssql so that you have the data necessary for the box score.

The game stream data has successfully joined with the player data taken from mssql.

```
# Q6
players_df = spark.read.format("com.microsoft.sqlserver.jdbc.spark") \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .option("url", mssql_url) \
    .option("dbtable", "players") \
    .option("user", mssql_user) \
    .option("password", mssql_pw) \
    .load()

teams_df = spark.read.format("com.microsoft.sqlserver.jdbc.spark") \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .option("url", mssql_url) \
    .option("dbtable", "teams") \
    .option("user", mssql_user) \
    .option("password", mssql_pw) \
    .load()

result.createOrReplaceTempView("result")
players_df.createOrReplaceTempView("players")
teams_df.createOrReplaceTempView("teams")

df_3 = spark.sql("""
SELECT r.*, p.name as player_name, t.name as team_name, t.conference, t.wins, t.losses
FROM result r
LEFT JOIN players p ON p.teamid = r.team_ID AND r.jersey_number = p.number
LEFT JOIN teams t ON r.team_ID = t.id
""")

df_3.show()
```

NetId: snamani

```
+-------+-------------+-----+-----+----------+-----------+-------------+----------+----+------+
|team_ID|jersey_number|shots|goals|team_goals|player_name|    team_name|conference|wins|losses|
+-------+-------------+-----+-----+----------+-----------+-------------+----------+----+------+
|    101|            9|    4|    0|        13|        sol|     syracuse|       acc|  11|     2|
|    101|           10|    3|    1|        13|      swede|     syracuse|       acc|  11|     2|
|    101|            2|    6|    1|        13|      steve|     syracuse|       acc|  11|     2|
|    101|            8|    3|    0|        13|        sly|     syracuse|       acc|  11|     2|
|    101|           13|    5|    1|        13|      stone|     syracuse|       acc|  11|     2|
|    101|            6|    4|    2|        13|        sam|     syracuse|       acc|  11|     2|
|    101|           15|    3|    1|        13|     shelly|     syracuse|       acc|  11|     2|
|    101|            1|    7|    6|        13|      sarah|     syracuse|       acc|  11|     2|
|    101|            4|    5|    1|        13|      shree|     syracuse|       acc|  11|     2|
|    101|           17|    1|    0|        13|       sean|     syracuse|       acc|  11|     2|
|    205|            9|    4|    0|         9|      julie|johns hopkins|     big10|   9|     4|
|    205|            2|    3|    1|         9|      james|johns hopkins|     big10|   9|     4|
|    205|           16|    1|    0|         9|      jimmy|johns hopkins|     big10|   9|     4|
```

Q7. Write pyspark code (in SQL or DataFrame API) to transform the output from question 5 into the box score document structure shown in part 3.1.

The output has been converted into a document structure.

```python
# Q7
from pyspark.sql.window import Window
import pyspark.sql.functions as F

# Aggregation remains the same
team_stats_agg = df_3.groupBy("team_ID").agg(
    F.first("conference").alias("conference"),
    F.first("wins").alias("wins"),
    F.first("losses").alias("losses"),
    F.sum("goals").alias("score"),
    F.collect_list(
        F.struct(
            "jersey_number",
            "player_name",
            "shots",
            "goals",
            F.when(F.col("shots") > 0, F.col("goals") / F.col("shots")).otherwise(F.lit(0)).alias("pct")
        )
    ).alias("players")
)

# Calculate max score over each partition
windowSpec = Window.partitionBy()
team_stats_agg = team_stats_agg.withColumn("max_score", F.max("score").over(windowSpec))

# Determine the status based on max score
team_stats_agg = team_stats_agg.withColumn(
    "status",
    F.when(F.col("score") == F.col("max_score"), "tied").otherwise(
        F.when(F.col("score") < F.col("max_score"), "losing").otherwise("winning")
    )
).drop("max_score")

# Filter for specific teams
home_team_stats = team_stats_agg.filter(F.col("team_ID") == 101)
away_team_stats = team_stats_agg.filter(F.col("team_ID") == 205)

# Convert to JSON - alternative method
home_json = home_team_stats.toJSON().first()
away_json = away_team_stats.toJSON().first()

import json

# Convert JSON strings back to dictionaries
home_dict = json.loads(home_json)
away_dict = json.loads(away_json)

# Construct the box score document with dictionaries
box_score_document = {
    "_id": latest_eventid,
    "timestamp": latest_timestamp,
    "home": home_dict,
    "away": away_dict
}

print(box_score_document)
```

```
    "timestamp" : latest_timestamp,
    "home": home_dict,
    "away": away_dict
}

print(box_score_document)
```

NetId: snamani

```
[Stage 505:================================================>    (183 + 1) / 200]
{'_id': 61, 'timestamp': '59:51', 'home': {'team_ID': 101, 'conference': 'acc', 'wins': 11, 'losses': 2, 's
core': 14, 'players': [{'jersey_number': 9, 'player_name': 'sol', 'shots': 4, 'goals': 0, 'pct': 0.0}, {'je
rsey_number': 10, 'player_name': 'swede', 'shots': 3, 'goals': 1, 'pct': 0.3333333333333333}, {'jersey_numb
er': 2, 'player_name': 'steve', 'shots': 7, 'goals': 2, 'pct': 0.2857142857142857}, {'jersey_number': 8, 'p
layer_name': 'sly', 'shots': 3, 'goals': 0, 'pct': 0.0}, {'jersey_number': 13, 'player_name': 'stone', 'sho
ts': 5, 'goals': 1, 'pct': 0.2}, {'jersey_number': 6, 'player_name': 'sam', 'shots': 4, 'goals': 2, 'pct':
0.5}, {'jersey_number': 15, 'player_name': 'shelly', 'shots': 3, 'goals': 1, 'pct': 0.3333333333333333}, {'
jersey_number': 1, 'player_name': 'sarah', 'shots': 7, 'goals': 6, 'pct': 0.8571428571428571}, {'jersey_num
ber': 4, 'player_name': 'shree', 'shots': 5, 'goals': 1, 'pct': 0.2}, {'jersey_number': 17, 'player_name':
'sean', 'shots': 1, 'goals': 0, 'pct': 0.0}], 'status': 'tied'}, 'away': {'team_ID': 205, 'conference': 'bi
g10', 'wins': 9, 'losses': 4, 'score': 9, 'players': [{'jersey_number': 9, 'player_name': 'julie', 'shots':
4, 'goals': 0, 'pct': 0.0}, {'jersey_number': 2, 'player_name': 'james', 'shots': 3, 'goals': 1, 'pct': 0.3
333333333333333}, {'jersey_number': 16, 'player_name': 'jimmy', 'shots': 1, 'goals': 0, 'pct': 0.0}, {'jers
ey_number': 3, 'player_name': 'jane', 'shots': 1, 'goals': 0, 'pct': 0.0}, {'jersey_number': 15, 'player_na
me': 'jane', 'shots': 2, 'goals': 2, 'pct': 1.0}, {'jersey_number': 5, 'player_name': 'jimmy', 'shots': 1,
'goals': 1, 'pct': 1.0}, {'jersey_number': 1, 'player_name': 'jimmy', 'shots': 3, 'goals': 3, 'pct': 1.0},
{'jersey_number': 22, 'player_name': 'julie', 'shots': 1, 'goals': 0, 'pct': 0.0}, {'jersey_number': 17, 'p
layer_name': 'james', 'shots': 3, 'goals': 1, 'pct': 0.3333333333333333}, {'jersey_number': 8, 'player_nam
e': 'julie', 'shots': 2, 'goals': 1, 'pct': 0.5}], 'status': 'losing'}}
```

Q8) Write pyspark code (in SQL or DataFrame API) to write the box score completed in question 6 to the mongo.sidearm.boxscores collection. The document should be keyed by event_id.

The box scores are written onto the mongo db in sidearms.

I did not provide a proper screen shot proof that the box scores are written into the mongo db.

```
# Q8
import json

json_boxscore = json.dumps(box_score_document)
df_boxscores = spark.read.json(spark.sparkContext.parallelize([box_score_json]))

df_boxscores.write.format("mongo") \
    .option("database", "sidearm") \
    .option("collection", "boxscores") \
    .mode("append") \
    .save()
```

NetId: snamani

Q9. Combine parts 4-7 into a single pyspark script that will run the entire process of creating the box score document. Make sure to run this a couple of times while the game stream is going on so there are at least 5 box score events.

After combining all the codes from 4 – 7 I had it run for more than 5 times so that there are a few box score events.

```
# Q9

df_2.createOrReplaceTempView("gamestream")

sql_query = """

SELECT gs.team_ID, gs.jersey_number, COUNT(*) AS shots, SUM(gs.goals) AS goals, tg.team_goals
FROM gamestream gs
JOIN
    (SELECT team_ID, SUM(goals) AS team_goals
         FROM gamestream
         WHERE team_ID > 0
     GROUP BY team_ID)
tg ON gs.team_ID = tg.team_ID

WHERE gs.team_ID > 0
GROUP BY gs.team_ID, gs.jersey_number, tg.team_goals
ORDER BY gs.team_ID, gs.jersey_number
"""

result = spark.sql(sql_query)


from pyspark.sql import functions as func

df_2.createOrReplaceTempView("gamestream")

latest_event_and_timestamp = spark.sql("""
    SELECT
        MAX(event_id) AS latest_eventid,
        MAX(timestamp) AS latest_timestamp
    FROM gamestream
    WHERE team_ID != 0
""").collect()[0]

latest_eventid = latest_event_and_timestamp['latest_eventid']
latest_timestamp = latest_event_and_timestamp['latest_timestamp']

result_2 = result.withColumn("latest_eventid", func.lit(latest_eventid))\
                    .withColumn("latest_timestamp", func.lit(latest_timestamp))


players_df = spark.read.format("com.microsoft.sqlserver.jdbc.spark") \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .option("url", mssql_url) \
    .option("dbtable", "players") \
    .option("user", mssql_user) \
    .option("password", mssql_pw) \
    .load()

teams_df = spark.read.format("com.microsoft.sqlserver.jdbc.spark") \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .option("url", mssql_url) \
    .option("dbtable", "teams") \
    .option("user", mssql_user) \
    .option("password", mssql_pw) \
    .load()

result.createOrReplaceTempView("result")
```

```
result.createOrReplaceTempView("result")
players_df.createOrReplaceTempView("players")
teams_df.createOrReplaceTempView("teams")

df_3 = spark.sql("""
SELECT r.*, p.name as player_name, t.name as team_name, t.conference, t.wins, t.losses
FROM result r
LEFT JOIN players p ON p.teamid = r.team_ID AND r.jersey_number = p.number
LEFT JOIN teams t ON r.team_ID = t.id
""")

from pyspark.sql.window import Window
import pyspark.sql.functions as F

team_stats_agg = df_3.groupBy("team_ID").agg(
    F.first("conference").alias("conference"),
    F.first("wins").alias("wins"),
    F.first("losses").alias("losses"),
    F.sum("goals").alias("score"),
    F.collect_list(
        F.struct(
            "jersey_number",
            "player_name",
            "shots",
            "goals",
            F.when(F.col("shots") > 0, F.col("goals") / F.col("shots")).otherwise(F.lit(0)).alias("pct")
        )
    ).alias("players")
)

# Calculate max score over each partition
windowSpec = Window.partitionBy()
team_stats_agg = team_stats_agg.withColumn("max_score", F.max("score").over(windowSpec))

team_stats_agg = team_stats_agg.withColumn(
    "status",
    F.when(F.col("score") == F.col("max_score"), "tied").otherwise(
        F.when(F.col("score") < F.col("max_score"), "losing").otherwise("winning")
    )
).drop("max_score")

# Filter for specific teams
home_team_stats = team_stats_agg.filter(F.col("team_ID") == 101)
away_team_stats = team_stats_agg.filter(F.col("team_ID") == 205)

import json

home_dict = json.loads(home_json)
away_dict = json.loads(away_json)


box_score_document = {
    "_id": latest_eventid,
    "timestamp": latest_timestamp,
    "home": home_dict,
    "away": away_dict
}

print(box_score_document)
```

NetId: snamani

```
{'_id': 62, 'timestamp': '59:51', 'home': {'team_ID': 101, 'conference': 'acc', 'wins': 11, 'losses': 2, 's
core': 14, 'players': [{'jersey_number': 9, 'player_name': 'sol', 'shots': 4, 'goals': 0, 'pct': 0.0}, {'je
rsey_number': 10, 'player_name': 'swede', 'shots': 3, 'goals': 1, 'pct': 0.3333333333333333}, {'jersey_numb
er': 2, 'player_name': 'steve', 'shots': 7, 'goals': 2, 'pct': 0.2857142857142857}, {'jersey_number': 8, 'p
layer_name': 'sly', 'shots': 3, 'goals': 0, 'pct': 0.0}, {'jersey_number': 13, 'player_name': 'stone', 'sho
ts': 5, 'goals': 1, 'pct': 0.2}, {'jersey_number': 6, 'player_name': 'sam', 'shots': 4, 'goals': 2, 'pct':
0.5}, {'jersey_number': 15, 'player_name': 'shelly', 'shots': 3, 'goals': 1, 'pct': 0.3333333333333333}, {'
jersey_number': 1, 'pl                                    'shots': 7, 'goals': 6, 'pct': 0.8571428571428571}, {'jersey_num
ber': 4, 'player_name'                                    5, 'goals': 1, 'pct': 0.2}, {'jersey_number': 17, 'player_name':
'sean', 'shots': 1, 'g                                    0}], 'status': 'tied'}, 'away': {'team_ID': 205, 'conference': 'bi
g10', 'wins': 9, 'losses': 4,  'score': 9, 'players': [{'jersey_number': 9, 'player_name': 'julie', 'shots':
4, 'goals': 0, 'pct': 0.0}, {'jersey_number': 2, 'player_name': 'james', 'shots': 3, 'goals': 1, 'pct': 0.3
333333333333333}, {'jersey_number': 16, 'player_name': 'jimmy', 'shots': 1, 'goals': 0, 'pct': 0.0}, {'jers
ey_number': 3, 'player_name': 'jane', 'shots': 1, 'goals': 0, 'pct': 0.0}, {'jersey_number': 15, 'player_na
me': 'jane', 'shots': 2, 'goals': 2, 'pct': 1.0}, {'jersey_number': 5, 'player_name': 'jimmy', 'shots': 1,
'goals': 1, 'pct': 1.0}, {'jersey_number': 1, 'player_name': 'jimmy', 'shots': 3, 'goals': 3, 'pct': 1.0},
{'jersey_number': 22, 'player_name': 'julie', 'shots': 1, 'goals': 0, 'pct': 0.0}, {'jersey_number': 17, 'p
layer_name': 'james', 'shots': 3, 'goals': 1, 'pct': 0.3333333333333333}, {'jersey_number': 8, 'player_nam
e': 'julie', 'shots': 2, 'goals': 1, 'pct': 0.5}], 'status': 'losing'}}
```


Q10. Write a drill SQL query to display the latest box score. The latest value should be derived from the data. not hard-coded eg. 56

Here is a drill query to pull the latest box scores.

```
SELECT * FROM mongo.sidearm.boxscores where _id = (SELECT MAX(_id) FROM mongo.sidearm.boxscores)
```

NetId: snamani

Show 10 entries Search: Show / hide columns

_id	event_id	timestamp	team_ID	jersey_number	goals
e\xED1\xEB\x87\xA0\x06*\x11\xA6\x90o	61	11:33	101	2	0

Showing 1 to 1 of 1 entries Previous 1 Next

NetId: snamani

Q11. When the game is complete, write pyspark code (in SQL or DataFrame API) update the wins and losses for the teams in the teams table. Specifically, load the teams table and update it, then display the updated data frame.

By the end of the game the wins and losses has been updated.
But yhe problem here is the scores of team Syracuse has not being updated and the jonhopkins has been updated.

```
#Q11
# Extracting the status of both teams from the box score JSON document
home_status = box_score_document["home"]["status"]
away_status = box_score_document["away"]["status"]

def modify_team_records(teams_df, team_id_val, team_status):
    updated_teams_df = teams_df.withColumn(
        'updated_wins',
        F.when((F.col('id') == team_id_val) & (F.lit(team_status) == 'winning'), F.col('wins') + 1)
        .otherwise(F.col('wins'))
    ).withColumn(
        'updated_losses',
        F.when((F.col('id') == team_id_val) & (F.lit(team_status) == 'losing'), F.col('losses') + 1)
        .otherwise(F.col('losses'))
    )
    return updated_teams_df

# Updating team statistics based on the status
df_teams_updated = modify_team_records(teams_df, 101, home_status)
df_teams_updated = modify_team_records(teams_df, 205, away_status)

df_teams_updated.show()
```

NetId: snamani

```
+---+-------------+----------+----+------+-----------+--------------+
| id|         name|conference|wins|losses|updated_wins|updated_losses|
+---+-------------+----------+----+------+-----------+--------------+
|101|     syracuse|       acc|  11|     2|          11|             2|
|205|johns hopkins|     big10|   9|     4|           9|             5|
+---+-------------+----------+----+------+-----------+--------------+
```

Q12. Write pyspark code (in SQL or DataFrame API) to write the updated in question 11 to a new mssql.sidearmdb.teams2 table.

The updated stats have been pushed to the mssql.

There is no screenshot providing proof that the new table has been updated in mssql

```
# Q12
df_teams_updated.write.format("com.microsoft.sqlserver.jdbc.spark") \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .option("url", mssql_url) \
    .option("dbtable", "teams2") \
    .option("user", mssql_user) \
    .option("password", mssql_pw) \
    .mode("overwrite") \
    .save()
```

NetId: snamani

Q13. When the game is complete, write pyspark code (in SQL or DataFrame API) update the shots and goals for the players in the players table. Specifically, load the players table and update it, then display the updated data frame.

After the completion of the game, the code will execute an update on the **players** DataFrame using PySpark, refreshing the 'shots' and 'goals' columns with new data and then displaying the revised DataFrame with the latest statistics

```
# Q13
new_stats = players_df.alias('players').join(
    df_3.alias('new_stats'),
    (players_df['name'] == df_3['player_name']) &
    (players_df['number'] == df_3['jersey_number']),
    'left'
)

new_stats.show()
```

NetId: snamani

id	name	number	shots	goals	teamid	team_ID	jersey_number	shots	goals	team_goals	player_name	team_name	conference	wins	losses
10	swede	10	90	50	101	101	10	3	1	14	swede	syracuse	acc	11	2
7	sol	9	52	20	101	101	9	4	0	14	sol	syracuse	acc	11	2
20	julie	22	83	19	205	205	22	1	0	9	julie	johns hopkins	big10	9	4
3	steve	2	60	20	101	101	2	7	2	14	steve	syracuse	acc	11	2
14	jane	15	82	46	205	205	15	2	2	9	jane	johns hopkins	big10	9	4
2	sarah	1	85	34	101	101	1	7	6	14	sarah	syracuse	acc	11	2
9	shelly	15	10	2	101	101	15	3	1	14	shelly	syracuse	acc	11	2
18	jane	3	91	40	205	205	3	1	0	9	jane	johns hopkins	big10	9	4
12	julie	9	10	0	205	205	9	4	0	9	julie	johns hopkins	big10	9	4

Q14. Write pyspark code (in SQL or DataFrame API) to write the updated in question 11 to a new mssql.sidearmdb.players2 table.

The new statistics are being written to the mssql's sidearmdb database.

I did not provide a proof that the new statistics have been updated to a new table in sidearmdb

```
# Q14
new_stats.write.format("com.microsoft.sqlserver.jdbc.spark") \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .option("url", mssql_url) \
    .option("dbtable", "players2") \
    .option("user", mssql_user) \
    .option("password", mssql_pw) \
    .mode("overwrite") \
    .save()
```

NetId: snamani

Q15. Re-write drill SQL query from question 1 to use the updated players2 and teams2 tables.

Here is the drill query used the updated players2 and teams2 tables

```
1  SELECT t.name, t.wins, t.losses, p.name, p.shots, p.goals
2  FROM players2 p
3  join teams2 t on p.teamid = t.id
```

NetId: snamani

name	wins	losses	name0	shots	goals
syracuse	11	2	swede	93	51
syracuse	11	2	sol	56	20
johns hopkins	9	4	julie	84	19
syracuse	11	2	steve	67	22
johns hopkins	9	4	jane	84	48
syracuse	11	2	sarah	92	40
syracuse	11	2	shelly	13	3
johns hopkins	9	4	jane	92	40
johns hopkins	9	4	julie	14	0
johns hopkins	9	4	jimmy	43	30

NetId: snamani