Contents

CPS 510 - Final Assignment Report

Group Members:

- Jermaine Ganado, 500624506
- Mitchell Mohorovich, 500563037
- Frank Vumbaca, 500564107

Phase 1: Logical Database Design

1. Application Selection

The application chosen for this project was a houseleague/casual baseball/softball league management and statistics tracker. There are many companies and groups that run their own casual leagues and often the planning and statistics gathering is all done by hand or using other crude methods.

This application would allow users to be able to create teams and players to organize game seasons, as well as track the statistics of the games within those seasons.

2. ER Model

The initial ER model is displayed below.

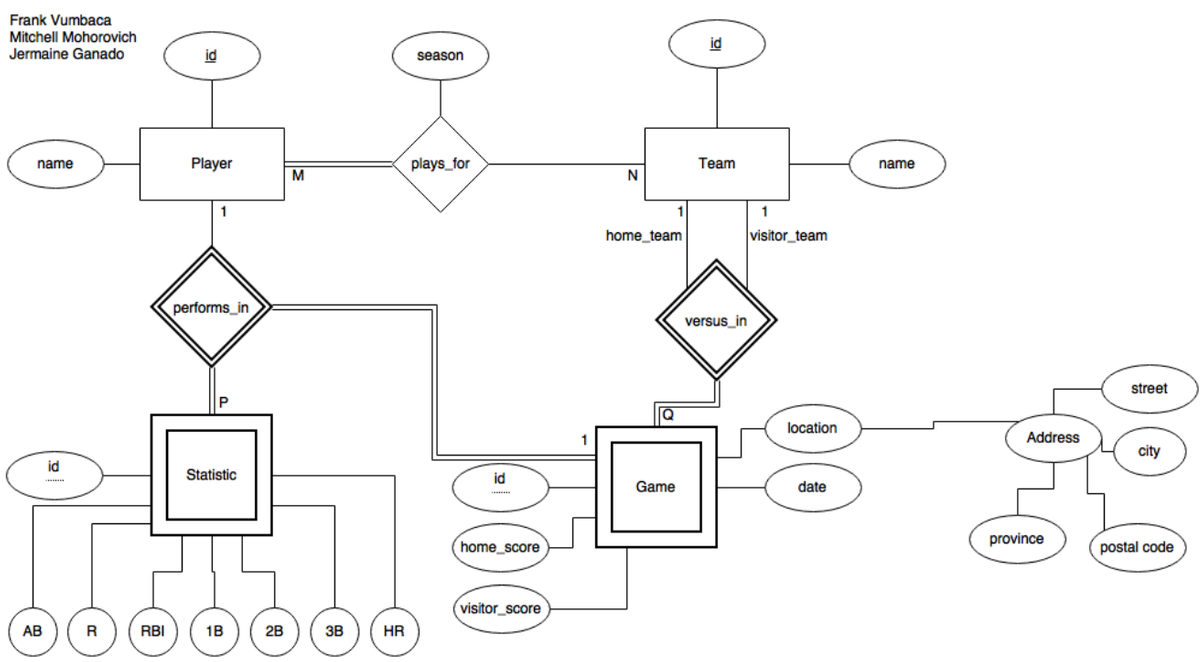

Figure 1: ER diagram

This ER diagram covered majority of the database design, but additional information was added in the next design phase in the EER diagram.

3. Schema Design

The EER model is displayed on the next page:

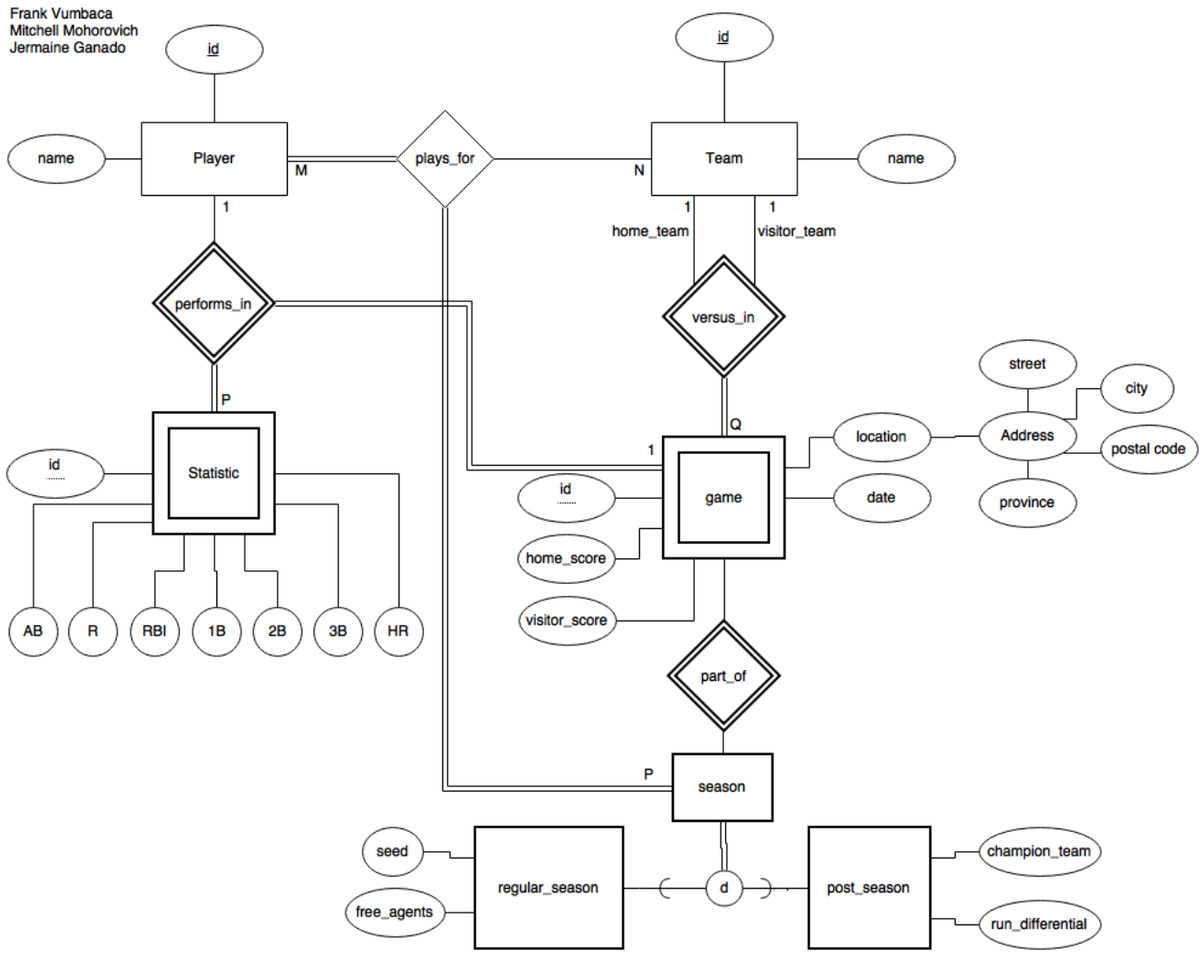

Figure 2: EER diagram

This shows the addition of the `Season` entity, and using aggregation, it shows there are different types of seasons: `regular` and `post` (playoffs) shown through disjoint specialization.

From this EER diagram, the schema of the database was created, and the first version of schema was developed. The table generation is shown below:

```
-- team table generation
CREATE TABLE MMOHOROV.TEAMS
(
  id INT PRIMARY KEY NOT NULL,
  name VARCHAR(25)
);


-- player table generation
CREATE TABLE MMOHOROV.PLAYERS
(
  id INT PRIMARY KEY NOT NULL,
  first_name VARCHAR(25),
  last_name VARCHAR(25)
);


-- season table generation
CREATE TABLE MMOHOROV.SEASONS
(
  id INT PRIMARY KEY,
  year INT NOT NULL,
  type VARCHAR(10) NOT NULL
);


-- plays_for table generation
CREATE TABLE MMOHOROV.PLAYS_FOR
(
  PLAYER_ID INT,
  SEASON_ID INT,
  TEAM_ID INT,
  CONSTRAINT PLAY PRIMARY KEY (PLAYER_ID, TEAM_ID, SEASON_ID),
```

```
  CONSTRAINT PLAYER_ID FOREIGN KEY (PLAYER_ID) REFERENCES PLAYERS (ID),
  CONSTRAINT SEASON_ID FOREIGN KEY (SEASON_ID) REFERENCES SEASONS (ID),
  CONSTRAINT TEAM_ID FOREIGN KEY (TEAM_ID) REFERENCES TEAMS (ID)
);


CREATE TABLE MMOHOROV.GAMES
(
  ID INT PRIMARY KEY NOT NULL,
  HOME_TEAM_ID INT NOT NULL,
  VISITOR_TEAM_ID INT NOT NULL,
  SEASON_ID INT NOT NULL,
  HOME_SCORE INT NOT NULL,
  VISITOR_SCORE INT NOT NULL,
  GAME_LOCATION VARCHAR2(50) NOT NULL,
  GAME_DATE TIMESTAMP NOT NULL,
  CONSTRAINT HOME_TEAM_ID FOREIGN KEY (HOME_TEAM_ID) REFERENCES TEAMS (ID),
  CONSTRAINT VISITOR_TEAM_ID FOREIGN KEY (VISITOR_TEAM_ID) REFERENCES TEAMS (ID),
  CONSTRAINT SEASON_ID FOREIGN KEY (SEASON_ID) REFERENCES SEASONS (ID)
);


-- creating STATISTICS table code
CREATE TABLE MMOHOROV.STATISTICS
(
  PLAYER_ID INT NOT NULL,
  GAME_ID INT NOT NULL,
  AB INT NOT NULL,
  R INT NOT NULL,
  RBI INT NOT NULL,
  "1B" INT NOT NULL,
  "2B" INT NOT NULL,
  "3B" INT NOT NULL,
  HR INT NOT NULL,
  CONSTRAINT STATISTICS_ID PRIMARY KEY (PLAYER_ID, GAME_ID),
  CONSTRAINT PLAYER_ID FOREIGN KEY (PLAYER_ID) REFERENCES PLAYERS (ID),
  CONSTRAINT GAME_ID FOREIGN KEY (GAME_ID) REFERENCES GAMES (ID)
);
```

Phase 2: Implementation

4. Designing Views/Simple Queries

After the schema was designed based on the EER diagram, simple queries were designed to basic functionality of the schema, and views were also designed.

The resulting SQL and results for thev views are shown below:

Simple Queries SQL

```
SELECT * FROM GAMES WHERE GAMES.VISITOR_SCORE = GAMES.HOME_SCORE;

SELECT * FROM GAMES WHERE GAMES.VISITOR_SCORE > GAMES.HOME_SCORE;

SELECT * FROM GAMES WHERE GAMES.VISITOR_SCORE < GAMES.HOME_SCORE;

SELECT * FROM PLAYERS WHERE PLAYERS.FIRST_NAME = 'Regina';

SELECT * FROM SEASONS WHERE SEASONS.YEAR < 2016;

SELECT * FROM SEASONS WHERE SEASONS.YEAR = 2016;

SELECT * FROM PLAYS_FOR WHERE PLAYS_FOR.TEAM_ID = 1;

SELECT * FROM STATISTICS WHERE STATISTICS.HR > 0;

SELECT * FROM TEAMS WHERE TEAMS.NAME = 'The Mathletes';
```

Relational Algebra

$\sigma_{visitor\_score=home\_score}(Games)$

$\sigma_{visitor\_score>home\_score}(Games)$

$\sigma_{visitor\_score<home\_score}(Games)$

$\sigma_{first\_name='Regina'}(Players)$

$\sigma_{year<2016}(Seasons)$

$\sigma_{year=2016}(Seasons)$

$\sigma_{team\_id=1}(Plays\_for)$

$\sigma_{statistics.hr>0}(Statistics)$

$\sigma_{name='TheMathletes'}(Teams)$

The following SQL queries below contain the view generation code for assignment 4.

View 1

View 1 creates a view that shows the statistics for every player for every season.

SQL View Creation Code

```
CREATE VIEW PLAYER_SEASON_STATS (
    PLAYER_ID,
    SEASON_ID,
    PLAYER_FNAME,
    PLAYER_LNAME,
    AB_AVG, R_AVG,
    RBI_AVG,
    "1B_AVG",
    "2B_AVG",
    "3B_AVG",
    "HR_AVG",
    H,
    BA)
AS
  SELECT
    s.PLAYER_ID,
    SEASONS.ID,
    PLAYERS.FIRST_NAME,
    PLAYERS.LAST_NAME,
    avg(s.AB),
    avg(s.R),
    avg(s.RBI),
    avg(s."1B"),
    avg(s."2B"),
    avg(s."3B"),
    avg(s.HR),
    sum(s."1B") + sum(s."2B") + sum(s."3B") + sum(s.HR),
    (sum(s."1B") + sum(s."2B") + sum(s."3B") + sum(s.HR) ) / sum(s.AB)
  FROM STATISTICS s
    INNER JOIN GAMES
      ON s.GAME_ID = GAMES.ID
    INNER JOIN SEASONS
      ON SEASONS.ID = GAMES.SEASON_ID
    RIGHT OUTER JOIN PLAYERS
      ON s.PLAYER_ID = PLAYERS.ID
  GROUP BY s.PLAYER_ID, SEASONS.ID, PLAYERS.FIRST_NAME, PLAYERS.LAST_NAME
  ORDER BY s.PLAYER_ID, SEASONS.ID
 WITH READ ONLY;
```

SQL Query Code

```
SELECT
  PLAYER_ID AS "Player ID",
  PLAYER_FNAME AS "First Name",
  PLAYER_LNAME AS "Last Name",
  SEASON_ID AS "Season ID",
  TRUNC(AB_AVG, 1) AS "S AB",
  TRUNC(R_AVG, 1) AS "S R",
  TRUNC(RBI_AVG, 1) AS "S RBI",
  TRUNC("1B_AVG", 1) AS "S 1B",
  TRUNC("2B_AVG", 1) AS "S 2B",
  TRUNC("3B_AVG", 1) AS "S 3B",
  TRUNC(HR_AVG, 1) AS "S HR",
  TRUNC(H, 1) AS "S Hits",
```

```
  TRUNC(BA, 1) AS "SBA"
FROM PLAYER_SEASON_STATS;
```

Result

Player ID	First Name	Last Name	Season ID	S AB	S R	S RBI	S 1B	S 2B	S 3B	S HR	S Hits	S BA
0	Cady	Heron	1	7	0.7	0.7	0	0	0.2	0.5	3	0.1
0	Cady	Heron	2	5	0.2	0.7	0	0	0	0.2	1	0
1	Regina	George	1	7	0.5	0.7	0	0	0.2	0.2	2	0
1	Regina	George	2	5	0	0	0	0	0	0	0	0
2	Gretchen	Wieners	1	7	0.2	0	0.2	0.2	0	0	2	0
2	Gretchen	Wieners	2	5	0.2	0.2	0	0.2	0	0.2	2	0.1
3	Janis	Ian	1	7	0.5	0.2	0	0	0.2	0	1	0
3	Janis	Ian	2	5	0.2	0.2	0.2	0	0	0	1	0
4	Karen	Smith	1	6	0.5	0	0.5	0.5	0	0.2	5	0.2
4	Karen	Smith	2	5	0.5	0.5	0.2	0.2	0	0	2	0.1
5	Kevin	Gnapoor	1	6	0.2	0.5	0.5	0	0	0.2	3	0.1
5	Kevin	Gnapoor	2	5	0.5	0.5	0.2	0.5	0	0	3	0.1
6	Trang	Pak	1	6	0	0.7	0.2	0	0	0	1	0
6	Trang	Pak	2	5	0.5	0.2	0.5	0	0.2	0	3	0.1
7	Aaron	Samuels	1	6	0.2	0.5	0.2	0	0	0	1	0
7	Aaron	Samuels	2	5	0.2	0	0	0	0	0.2	1	0
8	Glenn	Cocoo	1	6	0	0	0	0.2	0	0	1	0
8	Glenn	Cocoo	2	5	0.2	0.5	0	0.2	0.2	0	2	0.1
9	Shane	Oman	1	6	0	0	0.2	0	0	0	1	0
9	Shane	Oman	2	5	0.5	0.2	0.2	0	0	0.2	2	0.1
10	Ms.	Norbury	1	7	0.2	0.2	0	0.2	0	0	1	0
10	Ms.	Norbury	2	5	0.5	0.2	0.5	0.2	0	0	3	0.1
11	Mrs.	George	1	7	0.2	0.7	0	0.2	0	0.2	2	0
11	Mrs.	George	2	5	0.2	0.5	0.2	0	0.5	0	3	0.1
12	Caroline	Krafft	1	6	0.2	0	0	0.2	0	0	1	0
12	Caroline	Krafft	2	5	0.5	0.2	0	0.2	0.2	0	2	0.1
13	Coach	Carr	1	6	0	0	0.2	0	0	0	1	0
13	Coach	Carr	2	5	0.2	0.7	0	0	0.2	0.2	2	0.1
14	Principal	Duvall	1	6	0.5	0.2	0	0	0.2	0.2	2	0
14	Principal	Duvall	2	5	0	0	0	0	0	0	0	0
15	Emma	Gerber	1	6	0	0	0	0	0	0	0	0
15	Emma	Gerber	2	5	0.5	0	0	0.2	0	0	1	0
16	Taylor	Wedell	1	6	0.2	0	0.2	0	0	0	1	0
17	Kristen	Hadley	1	6	0.5	0.5	0.5	0	0	0	2	0
17	Kristen	Hadley	2	5	0.5	0.2	0.5	0	0	0	2	0.1
18	Dawn	Schweitzer	1	6	0	0	0.2	0	0	0	1	0
18	Dawn	Schweitzer	2	5	0.5	0.2	0.2	0	0.2	0	2	0.1
19	Tim	Pak	1	6	0.2	0.2	0	0	0	0	0	0
19	Tim	Pak	2	5	0.2	0.5	0	0.2	0	0	1	0
20	Damian	Leigh	2	5	0.5	0.2	0.2	0	0	0	1	0

View 2

This view shows the career statistics of every player in the database.

SQL Creation Code

```
CREATE VIEW PLAYER_CAREER_STATS (
    PLAYER_ID,
    PLAYER_FNAME,
    PLAYER_LNAME,
    AB_CAREER,
    R_CAREER,
    RBI_CAREER,
    "1B_CAREER",
    "2B_CAREER",
    "3B_CAREER",
    "HR_CAREER",
    "H_CAREER",
    "BA_CAREER"
) AS
  SELECT
    PLAYER_ID,
    PLAYER_FNAME,
    PLAYER_LNAME,
    avg(AB_AVG),
```

```
    avg(R_AVG),
    avg(RBI_AVG),
    avg("1B_AVG"),
    avg("2B_AVG"),
    avg("3B_AVG"),
    avg(HR_AVG),
    avg(H),
    avg(BA)
  FROM PLAYER_SEASON_STATS
  GROUP BY PLAYER_ID, PLAYER_FNAME, PLAYER_LNAME
  ORDER BY PLAYER_LNAME, PLAYER_FNAME, PLAYER_ID;
```

SQL Query Code

```
SELECT
  PLAYER_ID AS "Player ID",
  PLAYER_FNAME AS "First Name",
  PLAYER_LNAME AS "Last Name",
  TRUNC(AB_CAREER, 1) AS "Career AB",
  TRUNC(R_CAREER, 1) AS "Career R",
  TRUNC(RBI_CAREER, 1) AS "Career RBI",
  TRUNC("1B_CAREER", 1) AS "Career 1B",
  TRUNC("2B_CAREER", 1) AS "Career 2B",
  TRUNC("3B_CAREER", 1) AS "Career 3B",
  TRUNC(HR_CAREER, 1) AS "Career HR",
  TRUNC(H_CAREER, 1) AS "Career Hits",
  TRUNC(BA_CAREER, 1) AS "Career Batting Average"
FROM PLAYER_CAREER_STATS;
```

Result

Player ID	First Name	Last Name	Career AB	Career R	Career RBI	Career 1B	Career 2B	Career 3B	Career HR	Career Hits	Career Batting Average
13	Coach	Carr	5.5	0.1	0.3	0.1	0	0.1	0.1	1.5	0
8	Glenn	Cocoo	5.5	0.1	0.2	0	0.2	0.1	0	1.5	0
14	Principal	Duvall	5.5	0.2	0.1	0	0	0.1	0.1	1	0
11	Mrs.	George	6	0.2	0.6	0.1	0.1	0.2	0.1	2.5	0.1
1	Regina	George	6	0.2	0.3	0	0	0.1	0.1	1	0
15	Emma	Gerber	5.5	0.2	0	0	0.1	0	0	0.5	0
5	Kevin	Gnapoor	5.5	0.3	0.5	0.3	0.2	0	0.1	3	0.1
17	Kristen	Hadley	5.5	0.5	0.3	0.5	0	0	0	2	0
0	Cady	Heron	6	0.5	0.7	0	0	0.1	0.3	2	0
3	Janis	Ian	6	0.3	0.2	0.1	0	0.1	0	1	0
12	Caroline	Krafft	5.5	0.3	0.1	0	0.2	0.1	0	1.5	0
20	Damian	Leigh	5	0.5	0.2	0.2	0	0	0	1	0
10	Ms.	Norbury	6	0.3	0.2	0.2	0.2	0	0	2	0
9	Shane	Oman	5.5	0.2	0.1	0.2	0	0	0.1	1.5	0
19	Tim	Pak	5.5	0.2	0.3	0	0.1	0	0	0.5	0
6	Trang	Pak	5.5	0.2	0.5	0.3	0	0.1	0	2	0
7	Aaron	Samuels	5.5	0.2	0.2	0.1	0	0	0.1	1	0
18	Dawn	Schweitzer	5.5	0.2	0.1	0.2	0	0.1	0	1.5	0
4	Karen	Smith	5.5	0.5	0.2	0.3	0.3	0	0.1	3.5	0.1
16	Taylor	Wedell	6	0.2	0	0.2	0	0	0	1	0
2	Gretchen	Wieners	6	0.2	0.1	0.1	0.2	0	0.1	2	0

5. Advanced Queries and Unix Shell Implementation

For this phase of the project, a shell script was written to allow for easy schema creation population and querying. A screenshot of the interface is attached below:

Those queries are shown below with their respective results.

Query 1

This query uses two sub queries unioned to compute the total runs each team had for each season. Two subqueries were used to extract the runs when the team is away and when the team is home. The two values are then summed to find the total for each team for each seasons.

SQL Query

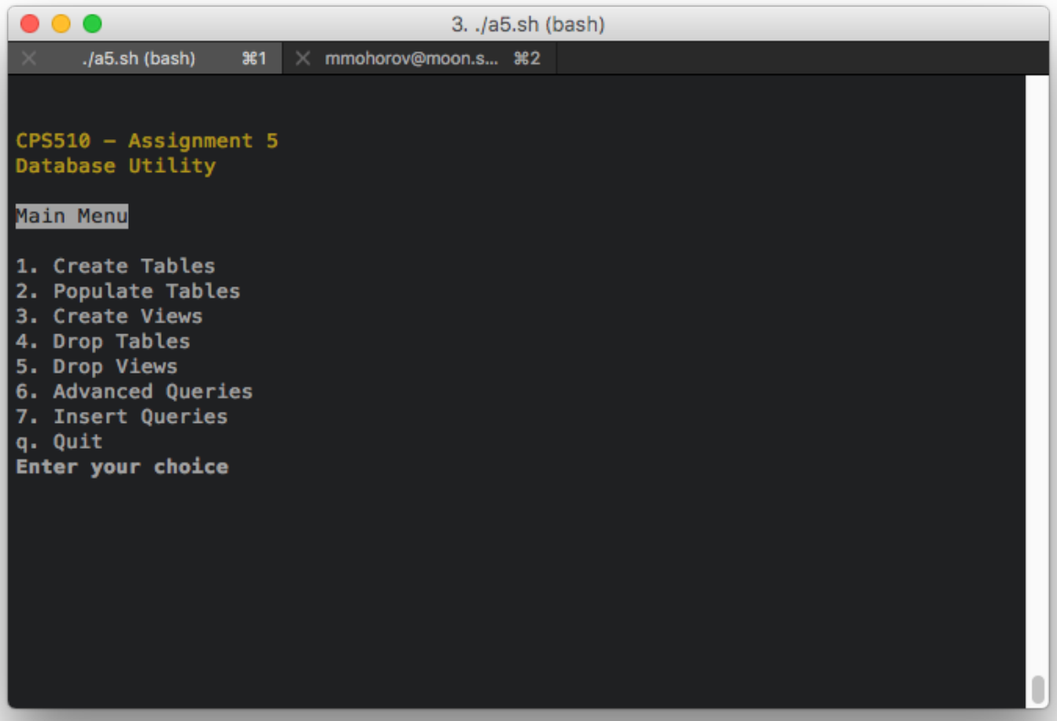

Figure 3: bash script ui screenshot

```
SELECT
  SEASON_ID,
  TEAM_ID,
  TEAMS.NAME AS "Team Name",
  SEASONS.YEAR AS "Season Year",
  SEASONS.TYPE AS "Season Type",
  sum(SCORE_SUM) AS "Team Runs Per Season"
FROM (
  SELECT
    SEASON_ID,
    HOME_TEAM_ID AS TEAM_ID,
    sum(HOME_SCORE) AS SCORE_SUM
  FROM GAMES
  GROUP BY SEASON_ID, HOME_TEAM_ID
  UNION
  SELECT
    SEASON_ID,
    VISITOR_TEAM_ID,
    sum(VISITOR_SCORE) AS VISITOR_SCORE_SUM
  FROM GAMES
  GROUP BY SEASON_ID, VISITOR_TEAM_ID
  )
  JOIN TEAMS ON TEAM_ID = TEAMS.ID
  JOIN SEASONS ON SEASON_ID = SEASONS.ID
GROUP BY SEASON_ID, TEAM_ID, TEAMS.NAME, SEASONS.YEAR, Seasons.TYPE
ORDER BY SEASON_ID ASC, TEAM_ID ASC;
```

Relational Algebra

$home \leftarrow \pi_{season\_id,home\_team\_id,SUMhome\_score}(Games)$

$visitor \leftarrow \pi_{season\_id,visitor\_team\_id,SUMvisitor\_score}(Games)$

$union \leftarrow home \cup visitor$

$team\_join \leftarrow union \bowtie_{home\_team\_id=id} Teams$

$seasons\_join \leftarrow \bowtie_{season\_id=id} seasons$

$\pi_{season\_id,team\_id,teams.name,seasons.year,seasons.type,SUMscore\_sum}(seasons\_join)$

Result

SEASON_ID	TEAM_ID	Team Name	Season Year	Season Type	Team Runs Per Season
1	1	The Plastics	2015	regular	12
1	2	The Mathletes	2015	regular	13
2	1	The Plastics	2016	regular	13
2	2	The Mathletes	2016	regular	14

Query 2

This query uses one subquery pulling every player's season's performance data. The performance data is then partitioned by season, and ordered by their batting average. The players with the top 5 batting averages are selected with the where clause and finally ordered by season then ranking. The result is a table showing the top five players for every season.

SQL Query

```
SELECT
  RN AS "Season Ranking",
  PLAYER_FNAME AS "Player First Name",
  PLAYER_LNAME AS "Player Last Name",
  SEASON_ID AS "Season ID",
  SEASONS.YEAR AS "Season Year",
  SEASONS.TYPE AS "Season Type",
  TRUNC(BA, 3) AS "Batting Average"
FROM
  (
    SELECT PLAYER_ID, PLAYER_FNAME, PLAYER_LNAME, SEASON_ID, BA,
      ROW_NUMBER() OVER (PARTITION BY SEASON_ID ORDER BY BA DESC) RN
    FROM   PLAYER_SEASON_STATS
  ) a
  JOIN SEASONS ON SEASON_ID = SEASONS.ID
WHERE RN <= 5
ORDER BY SEASON_ID, BA DESC;
```

Result

Season Ranking	Player First Name	Player Last Name	Season ID	Season Year	Season Type	Batting Average
1	Karen	Smith	1	2015	regular	0.208
2	Kevin	Gnapoor	1	2015	regular	0.125
3	Cady	Heron	1	2015	regular	0.107
4	Principal	Duvall	1	2015	regular	0.083
5	Kristen	Hadley	1	2015	regular	0.083
1	Kevin	Gnapoor	2	2016	regular	0.15
2	Mrs.	George	2	2016	regular	0.15
3	Ms.	Norbury	2	2016	regular	0.15
4	Trang	Pak	2	2016	regular	0.15
5	Gretchen	Wieners	2	2016	regular	0.1

6. Normalization of Database and Functional Dependencies

This section of the report contains advanced queries from Assignment 5. The first two queries were previously submitted in the last report.

Query 1

This query uses two sub queries unioned to compute the total runs each team had for each season. Two subqueries were used to extract the runs when the team is away and when the team is home. The two values are then summed to find the total for each team for each seasons.

SQL Query

```
SELECT
  SEASON_ID,
  TEAM_ID,
  TEAMS.NAME AS "Team Name",
  SEASONS.YEAR AS "Season Year",
  SEASONS.TYPE AS "Season Type",
  sum(SCORE_SUM) AS "Team Runs Per Season"
FROM (
  SELECT
```

```
    SEASON_ID,
    HOME_TEAM_ID AS TEAM_ID,
    sum(HOME_SCORE) AS SCORE_SUM
  FROM GAMES
  GROUP BY SEASON_ID, HOME_TEAM_ID
  UNION
  SELECT
    SEASON_ID,
    VISITOR_TEAM_ID,
    sum(VISITOR_SCORE) AS VISITOR_SCORE_SUM
  FROM GAMES
  GROUP BY SEASON_ID, VISITOR_TEAM_ID
  )
  JOIN TEAMS ON TEAM_ID = TEAMS.ID
  JOIN SEASONS ON SEASON_ID = SEASONS.ID
GROUP BY SEASON_ID, TEAM_ID, TEAMS.NAME, SEASONS.YEAR, Seasons.TYPE
ORDER BY SEASON_ID ASC, TEAM_ID ASC;
```

Relational Algebra

$home \leftarrow_{season\_id,home\_team\_id,SUMhome\_score} (Games)$

$visitor \leftarrow_{season\_id,visitor\_team\_id,SUMvisitor\_score} (Games)$

$union \leftarrow home \cup visitor$

$team\_join \leftarrow union \bowtie_{home\_team\_id=id} Teams$

$seasons\_join \leftarrow team\_join \bowtie_{season\_id=id} seasons$

$\pi_{season\_id,team\_id,teams.name,seasons.year,seasons.type,SUMscore\_sum}(seasons\_join)$

Result

SEASON_ID	TEAM_ID	Team Name	Season Year	Season Type	Team Runs Per Season
1	1	The Plastics	2015	regular	12
1	2	The Mathletes	2015	regular	13
2	1	The Plastics	2016	regular	13
2	2	The Mathletes	2016	regular	14

Query 2

This query uses one subquery pulling every player's season's performance data. The performance data is then partitioned by season, and ordered by their batting average. The players with the top 5 batting averages are selected with the where clause and finally ordered by season then ranking. The result is a table showing the top five players for every season.

SQL Query

```
SELECT
  RN AS "Season Ranking",
  PLAYER_FNAME AS "Player First Name",
  PLAYER_LNAME AS "Player Last Name",
  SEASON_ID AS "Season ID",
  SEASONS.YEAR AS "Season Year",
  SEASONS.TYPE AS "Season Type",
  TRUNC(BA, 3) AS "Batting Average"
FROM
  (
    SELECT PLAYER_ID, PLAYER_FNAME, PLAYER_LNAME, SEASON_ID, BA,
      ROW_NUMBER() OVER (PARTITION BY SEASON_ID ORDER BY BA DESC) RN
    FROM   PLAYER_SEASON_STATS
  ) a
  JOIN SEASONS ON SEASON_ID = SEASONS.ID
WHERE RN <= 5
ORDER BY SEASON_ID, BA DESC;
```

Relational Algebra

$subquery \leftarrow \pi_{player\_id,player\_fname,player\_lname,season\_id,ba,ROW\_NUMBER()OVER(PARTITIONBYseason\_id}(player\_season\_stats)$

$condition \leftarrow \sigma_{ROW\_NUMBER()OVER(PARTITIONBYseason\_id)\leq 5}(subquery)$

$\pi_{ROW\_NUMBER()OVER(PARTITIONBY season\_id),player\_fname,player\_lname,season\_id,seasons.year,seasons.type,TRUNC(ba,3)}$ $(condition)$

Result

Season Ranking	Player First Name	Player Last Name	Season ID	Season Year	Season Type	Batting Average
1	Karen	Smith	1	2015	regular	0.208
2	Kevin	Gnapoor	1	2015	regular	0.125
3	Cady	Heron	1	2015	regular	0.107
4	Principal	Duvall	1	2015	regular	0.083
5	Kristen	Hadley	1	2015	regular	0.083
1	Kevin	Gnapoor	2	2016	regular	0.15
2	Mrs.	George	2	2016	regular	0.15
3	Ms.	Norbury	2	2016	regular	0.15
4	Trang	Pak	2	2016	regular	0.15
5	Gretchen	Wieners	2	2016	regular	0.1

Query 3

This query, given a date range, calculatings a standing of the top players based off of the games played during that range. The players are ranked based on their batting average.

SQL Query

```
SELECT
  p.FIRST_NAME AS "FIRST NAME",
  p.LAST_NAME AS "LAST NAME",
  BA,
  GP AS "GAMES PLAYED",
  BA / GP AS "AVG POINTS PER GAME"
FROM PLAYERS p
  RIGHT JOIN (
               SELECT i.PLAYER_ID, i.BA, i.GP
               FROM
                 (
                   SELECT
                     COUNT(*) AS "GP",
                     PLAYER_ID,
                     (SUM("1B") + SUM("2B") + SUM("3B") + SUM(HR)) / SUM(AB) AS BA
                   FROM GAMES
                     LEFT JOIN STATISTICS s2 ON GAMES.ID = s2.GAME_ID
                   WHERE
                     GAME_DATE > TO_DATE(:sd) AND GAME_DATE < TO_DATE(:ed)
                   GROUP BY s2.PLAYER_ID
                 ) i
             ) s ON p.ID = s.PLAYER_ID ORDER BY BA DESC;
```

Relational Algebra (5)

$condition \leftarrow \sigma_{game\_date>TO\_DATE(:sd)ANDgame\_date<TO\_DATE(:ed)}(Games)$

$subquery \leftarrow_{COUNT(*),player\_id,ba}$ $(condition)$

$join \leftarrow subquery \bowtie_{games.id=statistics.game\_id}$ $(statistics)$

$subselect \leftarrow \pi_{player\_id,ba,gp}(join)$

$join2 \leftarrow players \bowtie_{players.id=player\_id}$ $(subselect)$

$result \leftarrow \pi_{first\_name,last\_name,ba,gp,ba/gp}(join2)$

Result

Given the values `16-10-03` and `16-10-13` for the bind variables `:sd` and `:ed`, the following result is produced:

FIRST NAME	LAST NAME	BA	GAMES PLAYED	AVG POINTS PER GAME
Kevin	Gnapoor	0.2	2	0.1
Trang	Pak	0.2	2	0.1
Ms.	Norbury	0.2	2	0.1
Shane	Oman	0.2	2	0.1
Janis	Ian	0.1	2	0.05
Coach	Carr	0.1	2	0.05

FIRST NAME	LAST NAME	BA	GAMES PLAYED	AVG POINTS PER GAME
Gretchen	Wieners	0.1	2	0.05
Cady	Heron	0.1	2	0.05
Karen	Smith	0.1	2	0.05
Mrs.	George	0.1	2	0.05
Emma	Gerber	0.1	2	0.05
Kristen	Hadley	0.1	2	0.05
Glenn	Cocoo	0.1	2	0.05
Tim	Pak	0	2	0
Dawn	Schweitzer	0	2	0
Principal	Duvall	0	2	0
Caroline	Krafft	0	2	0
Aaron	Samuels	0	2	0
Damian	Leigh	0	2	0
Regina	George	0	2	0

Query 4

The follwing query will produce the winning team id and name for every game, expanding on the information present in the `GAMES` table.

SQL Query

```
SELECT
  GAME_ID,
  SEASON_ID,
  HOME_TEAM_ID,
  VISITOR_TEAM_ID,
  HOME_SCORE,
  VISITOR_SCORE,
  WINNING_TEAM_ID,
  TEAMS.NAME AS WINNING_TEAM_NAME
FROM (
  SELECT
    GAME_ID,
    SEASON_ID,
    HOME_TEAM_ID,
    VISITOR_TEAM_ID,
    HOME_SCORE,
    VISITOR_SCORE,
    (CASE
     WHEN RUN_DIFFERENTIAL < 0
       THEN VISITOR_TEAM_ID
     WHEN RUN_DIFFERENTIAL > 0
       THEN HOME_TEAM_ID
     ELSE NULL
     END) AS WINNING_TEAM_ID
  FROM (
    SELECT
      GAMES.ID                              AS GAME_ID,
      GAMES.SEASON_ID                       AS SEASON_ID,
      GAMES.HOME_TEAM_ID                    AS HOME_TEAM_ID,
      GAMES.VISITOR_TEAM_ID                 AS VISITOR_TEAM_ID,
      GAMES.HOME_SCORE                      AS HOME_SCORE,
      GAMES.VISITOR_SCORE                   AS VISITOR_SCORE,
      GAMES.HOME_SCORE - GAMES.VISITOR_SCORE AS RUN_DIFFERENTIAL
    FROM GAMES
  )
  ) JOIN TEAMS ON WINNING_TEAM_ID = TEAMS.ID
ORDER BY SEASON_ID, GAME_ID;
```

Relational Algebra (6)

$inner \leftarrow \pi_{id,season\_id,home\_team\_id,visitor\_team\_id,home\_score,visitor\_score,home\_score-visitor\_score}(Games)$

$inner2 \leftarrow \pi_{game\_id,season\_id,home\_team\_id,visitor\_team\_id,home\_score,visitor\_score,winning\_team\_id}(inner)$

$result \leftarrow \pi_{game\_id,season\_id,home\_team\_id,visitor\_team\_id,home\_score,visitor\_score,winning\_team\_id,teams.name}(inner2) \bowtie_{winning\_team\_id=teams.id} teams$

Result

GAME ID	SEASON ID	HOME TEAM ID	VISITOR TEAM ID	HOME SCORE	VISITOR SCORE	WINNING TEAM ID	WINNING TEAM NAME
1	1	1	2	5	2	1	The Plastics
2	1	1	2	3	6	2	The Mathletes
4	1	2	1	3	2	2	The Mathletes
5	2	1	2	8	2	1	The Plastics
6	2	1	2	2	3	2	The Mathletes
7	2	1	2	2	4	2	The Mathletes
8	2	1	2	1	5	2	The Mathletes

Query 5

This query will provide the total number of games won per season per team, alongside the team name for readability.

SQL Query

```
SELECT
  TEAMS.ID,
  TEAMS.NAME,
  SEASON_ID,
  COUNT(WINNING_TEAM_ID) AS WINS
FROM TEAMS
  JOIN (
    SELECT
      GAME_ID,
      SEASON_ID,
      HOME_TEAM_ID,
      VISITOR_TEAM_ID,
      CASE WHEN RUN_DIFFERENTIAL > 0
        THEN HOME_TEAM_ID
      WHEN RUN_DIFFERENTIAL < 0
        THEN VISITOR_TEAM_ID
      ELSE NULL END WINNING_TEAM_ID
    FROM (
      SELECT
        GAMES.ID                                AS GAME_ID,
        GAMES.SEASON_ID                         AS SEASON_ID,
        GAMES.HOME_TEAM_ID                      AS HOME_TEAM_ID,
        GAMES.VISITOR_TEAM_ID                   AS VISITOR_TEAM_ID,
        GAMES.HOME_SCORE                        AS HOME_SCORE,
        GAMES.VISITOR_SCORE                     AS VISITOR_SCORE,
        GAMES.HOME_SCORE - GAMES.VISITOR_SCORE AS RUN_DIFFERENTIAL
      FROM GAMES
    )
    )
    ON TEAMS.ID = WINNING_TEAM_ID
GROUP BY TEAMS.ID, TEAMS.NAME, SEASON_ID
ORDER BY SEASON_ID, TEAMS.ID;
```

Relational Algebra (7)

$inner \leftarrow \pi_{id,season\_id,home\_team\_id,visitor\_team\_id,home\_score,visitor\_score,home\_score-visitor\_score}(Games)$

$inner2 \leftarrow \pi_{game\_id,season\_id,home\_team\_id,visitor\_team\_id,home\_score,visitor\_score,winning\_team\_id}(inner)$

$result \leftarrow \pi_{teams.id,teams.name,season\_id,COUNTwinning\_team\_id}(inner2) \bowtie_{winning\_team\_id=teams.id} (teams)$

Result

ID	NAME	SEASON_ID	WINS
1	The Plastics	1	1
2	The Mathletes	1	2
1	The Plastics	2	1
2	The Mathletes	2	3

Functional Dependencies

TEAMS

```
ID -> Name
```

PLAYERS

```
ID -> First Name
ID -> Last Name
```

SEASONS

```
ID -> Year
ID -> Type
```

PLAYS_FOR

```
PLAYER_ID, SEASON_ID -> TEAM_ID
```

GAMES

```
ID -> HOME_TEAM_ID
ID -> VISITOR_TEAM_ID
ID -> SEASON_ID
ID -> HOME_SCORE
ID -> VISITOR_SCORE
ID -> GAME_LOCATION
ID -> GAME_DATE
```

STATISTICS

```
PLAYER_ID, GAME_ID -> AB
PLAYER_ID, GAME_ID -> H
PLAYER_ID, GAME_ID -> R
PLAYER_ID, GAME_ID -> RBI
PLAYER_ID, GAME_ID -> 1B
PLAYER_ID, GAME_ID -> 2B
PLAYER_ID, GAME_ID -> 3B
PLAYER_ID, GAME_ID -> HR
```

7. Normalization: 3rd NF

No changes had to be made to convert the schema into 3rd NF since all tables were defined with their own primary keys or candidate foreign keys.

8. Normalization: BCNF

For this phase of the project, most of the tables were already in BCNF form, only one table had to be modified which resulted in a new table being created.

`CREATE TABLE` Modifications

For this assignment, the `STATISTICS` table was normalized and changed to BCNF form.

The original table generation code was:

```
CREATE TABLE MMOHOROV.STATISTICS
(
 PLAYER_ID INT NOT NULL,
 GAME_ID INT NOT NULL,
 AB INT NOT NULL,
 H INT NOT NULL,
 R INT NOT NULL,
 RBI INT NOT NULL,
 "1B" INT NOT NULL,
 "2B" INT NOT NULL,
 "3B" INT NOT NULL,
 HR INT NOT NULL,
 CONSTRAINT STATISTICS_ID PRIMARY KEY (PLAYER_ID, GAME_ID),
 CONSTRAINT PLAYER_ID_S FOREIGN KEY (PLAYER_ID) REFERENCES PLAYERS (ID),
 CONSTRAINT GAME_ID FOREIGN KEY (GAME_ID) REFERENCES GAMES (ID)
 )
```

This was changed to:

```
CREATE TABLE MMOHOROV.STATISTICS
(
 ID INT PRIMARY KEY NOT NULL,
 AB INT NOT NULL,
 H INT NOT NULL,
 R INT NOT NULL,
 RBI INT NOT NULL,
 "1B" INT NOT NULL,
 "2B" INT NOT NULL,
 "3B" INT NOT NULL,
 HR INT NOT NULL
 );
```

Which removed the references to the primary keys of other entities.

To maintain the same relationship, a new table `PERFORMED_IN` was created which maps the `ID` of players with the `ID` of games with the `ID` of a statistic entity. The SQL table generation code for this new table is below.

```
CREATE TABLE MMOHOROV.PERFORMED_IN
(
 PLAYER_ID INT NOT NULL,
 GAME_ID INT NOT NULL,
 STATISTIC_ID INT NOT NULL,
 CONSTRAINT PERFORMED_IN_PERFORMANCE_ID PRIMARY KEY
 (PLAYER_ID, GAME_ID, STATISTIC_ID),
 CONSTRAINT PERFORMED_IN_PLAYER_ID_S FOREIGN KEY
 (PLAYER_ID) REFERENCES PLAYERS (ID),
 CONSTRAINT PERFORMED_IN_GAME_ID FOREIGN KEY
 (GAME_ID) REFERENCES GAMES (ID),
 CONSTRAINT PERFORMED_IN_STATISTIC_ID FOREIGN KEY
 (STATISTIC_ID) REFERENCES STATISTICS(ID)
 );
```

`INSERT` Modification

Since the `STATISTICS` table was modified, the insertions had to be modified as well. All the foreign keys that were present in every `STATISTICS INSERT` was moved to a new `INSERT` for the `PERFORMS_IN` table.

`VIEW` Modification

One view which used the previous `STATISTICS` table had to be modified in order to work with the new normalized `STATISTICS` table.

```
CREATE VIEW PLAYER_SEASON_STATS (
    PLAYER_ID,
    SEASON_ID,
    PLAYER_FNAME,
    PLAYER_LNAME,
    AB_AVG, R_AVG,
    RBI_AVG,
    "1B_AVG",
    "2B_AVG",
    "3B_AVG",
    "HR_AVG",
    H,
    BA)
AS
  SELECT
    s.PLAYER_ID,
    SEASONS.ID,
    PLAYERS.FIRST_NAME,
    PLAYERS.LAST_NAME,
    avg(s.AB),
    avg(s.R),
    avg(s.RBI),
    avg(s."1B"),
    avg(s."2B"),
    avg(s."3B"),
    avg(s.HR),
    sum(s."1B") + sum(s."2B") + sum(s."3B") + sum(s.HR),
    (sum(s."1B") + sum(s."2B") + sum(s."3B") + sum(s.HR) ) / sum(s.AB)
  FROM STATISTICS s
    INNER JOIN GAMES
      ON s.GAME_ID = GAMES.ID
    INNER JOIN SEASONS
```

```
      ON SEASONS.ID = GAMES.SEASON_ID
    RIGHT OUTER JOIN PLAYERS
      ON s.PLAYER_ID = PLAYERS.ID
  GROUP BY s.PLAYER_ID, SEASONS.ID, PLAYERS.FIRST_NAME, PLAYERS.LAST_NAME
  ORDER BY s.PLAYER_ID, SEASONS.ID
  WITH READ ONLY;
```

An additional join was used to map the player's ID with the newly added statistics ID.

```
CREATE VIEW PLAYER_SEASON_STATS (
    PLAYER_ID,
    SEASON_ID,
    PLAYER_FNAME,
    PLAYER_LNAME,
    AB_AVG, R_AVG,
    RBI_AVG,
    "1B_AVG",
    "2B_AVG",
    "3B_AVG",
    "HR_AVG",
    H,
    BA)
AS
  SELECT
    pi.PLAYER_ID,
    SEASONS.ID,
    PLAYERS.FIRST_NAME,
    PLAYERS.LAST_NAME,
    avg(s.AB),
    avg(s.R),
    avg(s.RBI),
    avg(s."1B"),
    avg(s."2B"),
    avg(s."3B"),
    avg(s.HR),
    sum(s."1B") + sum(s."2B") + sum(s."3B") + sum(s.HR),
    (sum(s."1B") + sum(s."2B") + sum(s."3B") + sum(s.HR) ) / sum(s.AB)
  FROM PERFORMED_IN pi
  LEFT JOIN STATISTICS s ON pi.STATISTIC_ID=s.ID
    INNER JOIN GAMES
      ON pi.GAME_ID = GAMES.ID
    INNER JOIN SEASONS
      ON SEASONS.ID = GAMES.SEASON_ID
    RIGHT OUTER JOIN PLAYERS
      ON pi.PLAYER_ID = PLAYERS.ID
  GROUP BY pi.PLAYER_ID, SEASONS.ID, PLAYERS.FIRST_NAME, PLAYERS.LAST_NAME
  ORDER BY pi.PLAYER_ID, SEASONS.ID
  WITH READ ONLY;
```

`DROP` Modification

The only necessary change to dropping tables is the addition of the following statement:

```
DROP TABLE MMOHOROV.PERFORMED_IN;
```

Which drops the new `PERFORMED_IN` table.

9. Java Application UI

For the Java application UI, the Java swing library was used to create a simple, easy to use UI. The functionality of this UI closely matched the shell script demo for assignment 5.

The Java application is runnable locally, and can connect to to the ryerson moons if the command below is executed with a SCS moon account name specified. This creates an SSH tunnel to the oracle.scs.ryerson.ca server through moon.scs.ryerson.ca

```
ssh -N -L1521:oracle.scs.ryerson.ca:1521 [scs_username]@moon.scs.ryerson.ca
```

As shown in the screenshot above, the Java UI contains similar functionality to the shell script. It allows for users to easily create tables, populate tables, create views and also drop views and tables in one step. A field is provided to allow users to execute any arbitrary SQL queries.

At the top of the window there are two dropdowns that are populated based on the existence of tables or views. Once the tables and views are created, users can select them and have their contents outputted to the console for inspection.

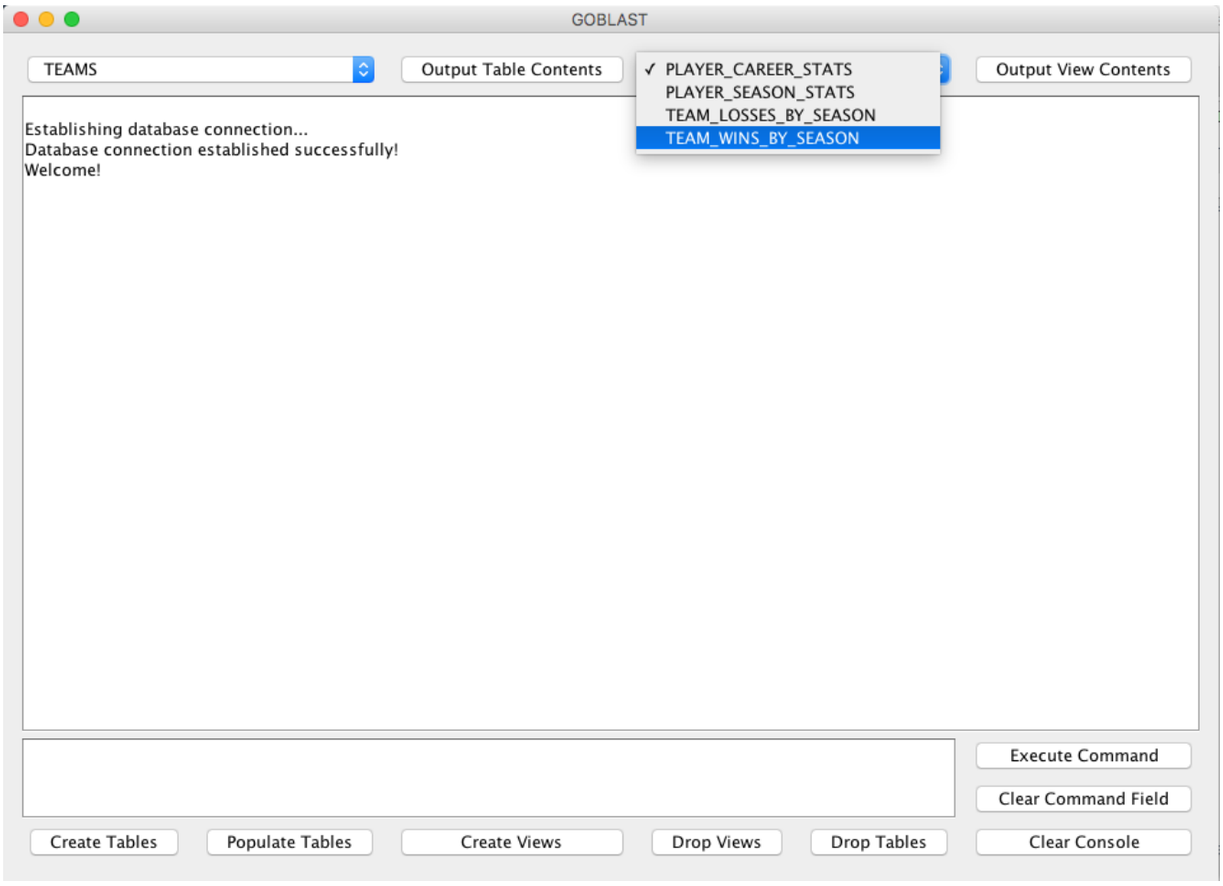

Figure 4: Java based database UI

Phase 3: Documentation

10. Final Documentation and Project

Relational Algebra

For the final documentation phase, the basic and advanced queries had relational algebra added to their respective sections of the report.

Normalization

During assignment 8, all tables were converted into BCNF form. This was done by making every single entity have its own primary key. This is the case for every table, except for the `PLAYS_FOR` and `PERFORMS_IN` tables, which are relation tables. In those two tables, they contain foreign keys, *but contain no other non-key attributes.*

Design Experience

Through progressing through this project, my group quickly realized that designing a database schema is no trivial task. We spent many hours designing and redesigning almost every part of the schema, and for the implementation as well.