Soccer League Database Management System: A10
FINAL REPORT

CPS510 - Section 08
Dr. S.B.Tajali
Fall 2024

Aananth Kuganendra
Enithan Kamalachandran
Abdur-Rehman Rizvi
Aathushan Kugendran

TABLE OF CONTENTS

1 - APPLICATION DETAILS **3**

2 - ER MODEL **4**

3 - SCHEMA DESIGN **5**

4 - DESIGNING VIEWS/SIMPLE QUERIES **7**

5 - ADVANCED QUERIES BY UNIX SHELL IMPLEMENTATION **17**

6 - NORMALIZATION OF DATABASE / FUNCTIONAL DEPENDENCIES **28**

7 - NORMALIZATION OF DATABASE / 3NF (3rd Normal Form) **29**

8 - NORMALIZATION OF DATABASE / BCNF (Boyce-Codd Normal Form) **31**

9 - JAVA BASED UI **32**

10 - QUERIES IN RELATIONAL ALGEBRA **48**

11 - USAGE INSTRUCTIONS **50**

12 - CONCLUSION **50**

1 - APPLICATION DETAILS

This Soccer League Database Management System regulates a league-based system as well as team and player statistics for competitive soccer. This application is intended to be used by sports-event organizers who wish to start their own soccer league, and view player and team information. This application will allow event organizers to view the following:

- Team records: number of wins, draws, losses
- Team statistics: team with most goals, wins, etc.
- Match history of all teams: date, location, goals scored between both teams
- Goal history of matches: player who scored, player who assisted
- Match attendance data for each game
- Individual player statistics: goals, assists, and games played

This Soccer League DBMS establishes relationships among players, coaches, stadiums and the team they fall under. The league table is related to the matches played with match results directly impacting the standings on the table. These relationships also bring data constraints. A player cannot score more goals than his team. A team cannot win more games than they played. These logical constraints are reiterated in this Soccer League DBMS.

Figure 1.0: Relational Database Schema

Player

playerid	playername	teamid	games	goals	assists

Coach

teamid	coachname

Team

teamid	teamname	stadiumid

Stadium

stadiumid	stadiumname	area	seats	avgattendance

Game

gameid	hometeam	awayteam	homegoals	awaygoals	stadiumid	attendance	gamedate

Score

gameid	score	winner

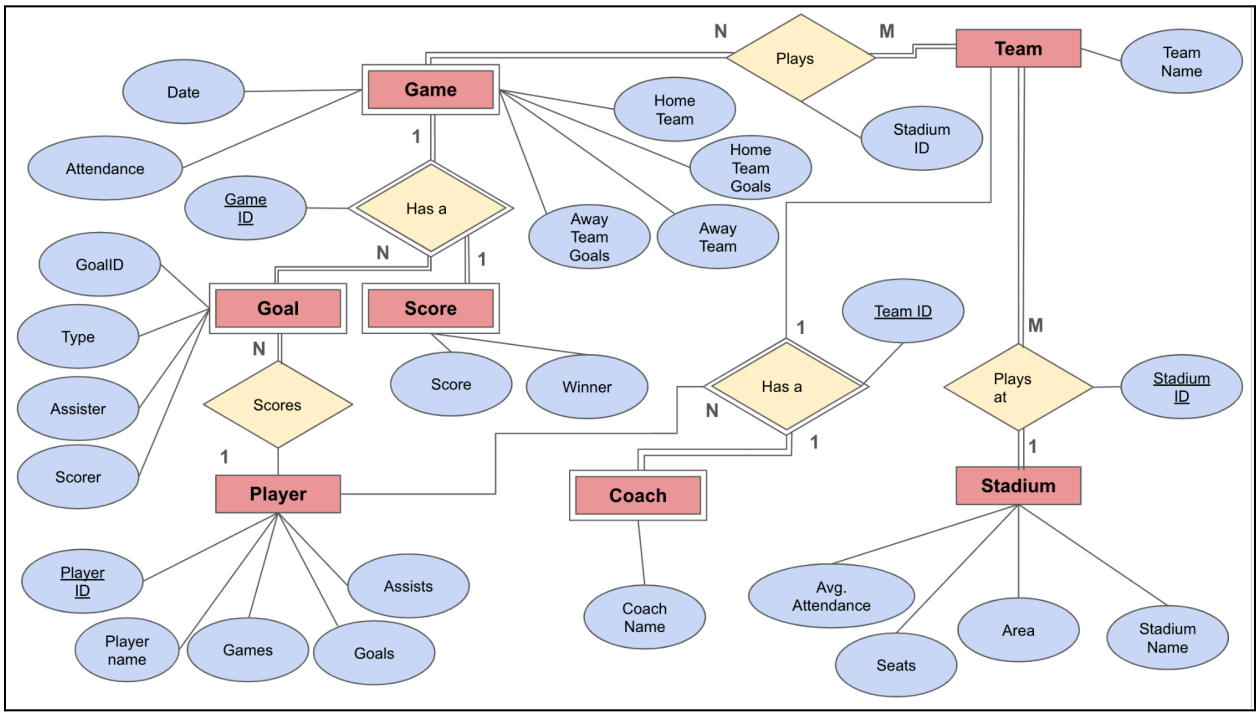
Goal

goalid	gameid	scorer	assister	goaltype

2 - ER MODEL

The following ER model was derived from the projected database schema. It consists of 7 entities with 5 relationships among them. Within the ER model, there are 7 cardinalities addressed, double lines used between entities and relationships to denote total participation, and bordered entities to denote weak entities.

Figure 2.0: ER Model for Database



3 - SCHEMA DESIGN

After visualizing the relationships between entities and their attributes, the proposed database scheme was designed in Oracle using SQL Developer. The SQL statements create tables while defining attributes with their data type (SMALLINT, VARCHAR, etc.) and any required constraints (PRIMARY KEY, NOT NULL, etc.).

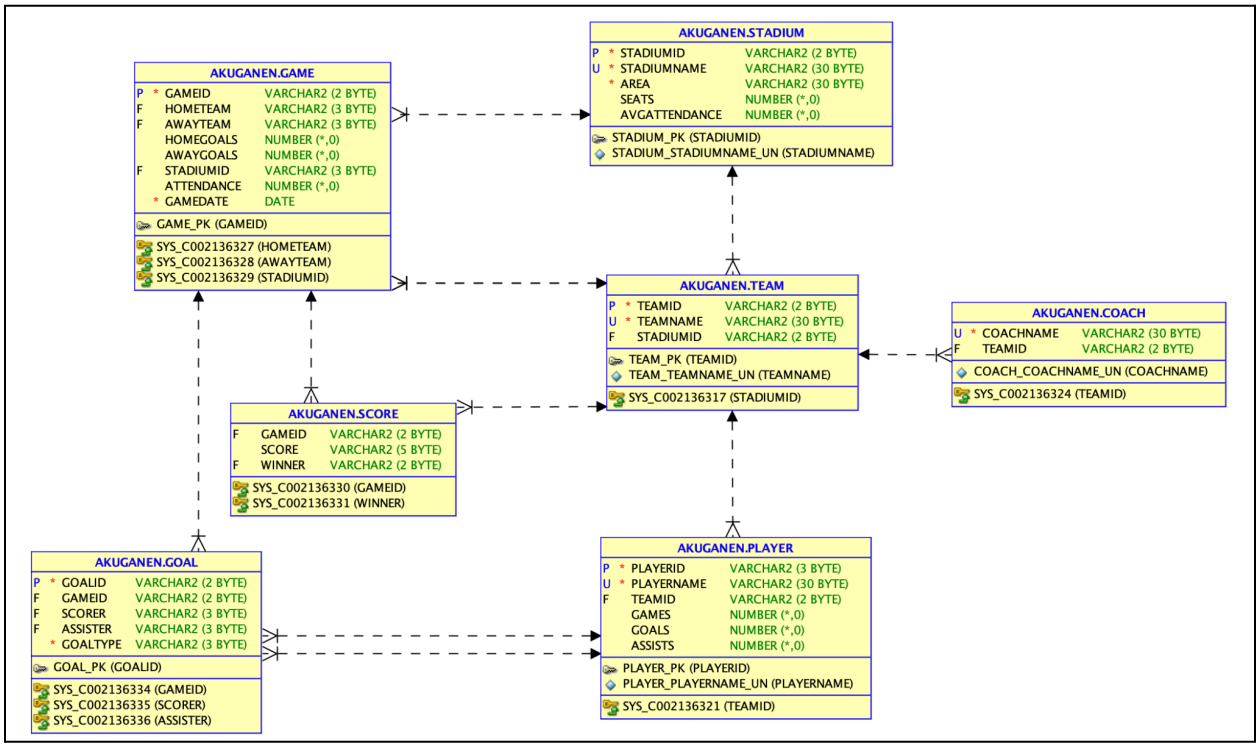
Figure 3.0: SQL Statements to Create Tables

```
CREATE TABLE stadium (
    stadiumid     VARCHAR(2) PRIMARY KEY,
    stadiumname   VARCHAR(30) NOT NULL UNIQUE,
    area          VARCHAR(30) NOT NULL,
    seats         INTEGER DEFAULT 0,
    avgattendance INTEGER DEFAULT 0
);

CREATE TABLE team (
    teamid    VARCHAR(2) PRIMARY KEY,
    teamname  VARCHAR(30) NOT NULL UNIQUE,
    stadiumid VARCHAR(2)
        REFERENCES stadium ( stadiumid )
);

CREATE TABLE player (
    playerid   VARCHAR(3) PRIMARY KEY,
    playername VARCHAR(30) NOT NULL UNIQUE,
    teamid     VARCHAR(2)
        REFERENCES team (teamid),
    games      SMALLINT DEFAULT 0,
    goals      SMALLINT DEFAULT 0,
    assists    SMALLINT DEFAULT 0
);

CREATE TABLE coach (
    coachname VARCHAR(30) NOT NULL UNIQUE,
    teamid    VARCHAR(2)
        REFERENCES team ( teamid )
);

CREATE TABLE game (
    gameid      VARCHAR(2) PRIMARY KEY,
    hometeam    VARCHAR(3)
        REFERENCES team ( teamid ),
    awayteam    VARCHAR(3)
        REFERENCES team ( teamid ),
    homegoals   SMALLINT,
    awaygoals   SMALLINT,
    stadiumid VARCHAR(3)
        REFERENCES stadium ( stadiumid ),
    attendance  INTEGER DEFAULT 0,
    gamedate    DATE NOT NULL
);
```

```
CREATE TABLE score (
    gameid  VARCHAR(2)
        REFERENCES game ( gameid ),
    score  VARCHAR(5),
    winner VARCHAR(4)
        REFERENCES team (teamid)
);

CREATE TABLE goal (
    goalid    VARCHAR(2)
    gameid    VARCHAR(2)
        REFERENCES game ( gameid ),
    scorer   VARCHAR(3)
        REFERENCES player (playerid),
    assister VARCHAR(3)
        REFERENCES player (playerid),
    goaltype     VARCHAR(3) NOT NULL
);
Exit;
```

After the SQL CREATE statements from **Figure 3.0** were executed, the following ER Model was generated through Oracle SQLDevloper's "Data Modeler" feature.

Figure 3.1: Oracle SQLDeveloper ER Model



4 - DESIGNING VIEWS/SIMPLE QUERIES

Once the tables were created on Oracle, they were populated using the SQL INSERT statements. The rows of PLAYER and STADIUM tables were then updated based on information from the GAME and GOAL table. This data served as “dummy” data for the rest of the project

Figure 4.0: SQL Statements to Populate Tables

```
INSERT INTO STADIUM (STADIUMID, STADIUMNAME, AREA, SEATS)
    VALUES ('00', 'Free-Agency', 'N/A', 0);
INSERT INTO STADIUM (STADIUMID, STADIUMNAME, AREA, SEATS)
    VALUES ('01', 'BMO Field', 'Toronto, ON', 40000);
INSERT INTO STADIUM (STADIUMID, STADIUMNAME, AREA, SEATS)
    VALUES ('02', 'Stade Olympique', 'Montreal, QC', 61004);
INSERT INTO STADIUM (STADIUMID, STADIUMNAME, AREA, SEATS)
    VALUES ('03', 'Commonwealth Stadium', 'Edmonton, AB', 56302);
INSERT INTO STADIUM (STADIUMID, STADIUMNAME, AREA, SEATS)
    VALUES ('04', 'BC Place', 'Vancouver, BC', 54320);
INSERT INTO STADIUM (STADIUMID, STADIUMNAME, AREA, SEATS)
    VALUES ('05', 'McMahon Stadium', 'Calgary, AB', 37317);

INSERT INTO TEAM (TEAMID, TEAMNAME, STADIUMID)
    VALUES ('FA', 'Free-Agents', '00');
INSERT INTO TEAM (TEAMID, TEAMNAME, STADIUMID)
    VALUES ('01', 'Toronto FC', '01');
INSERT INTO TEAM (TEAMID, TEAMNAME, STADIUMID)
    VALUES ('02', 'Strikers United', '02');
INSERT INTO TEAM (TEAMID, TEAMNAME, STADIUMID)
    VALUES ('03', 'Phoenix FC', '03');
INSERT INTO TEAM (TEAMID, TEAMNAME, STADIUMID)
    VALUES ('04', 'Red Raptors FC', '04');

INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('01', 'Lionel Messi', '01');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('02', 'Cristiano Ronaldo', '01');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('03', 'Kylian Mbappe', '01');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('04', 'Kevin De Bruyne', '01');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('05', 'Virgil van Dijk', '01');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('06', 'Mohamed Salah', '01');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('07', 'Sergio Ramos', '01');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('08', 'Robert Lewandowski', '01');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('09', 'Neymar Jr', '01');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('10', 'Luka Modric', '01');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('11', 'Erling Haaland', '01');
```

```
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('12', 'Karim Benzema', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('13', 'Harry Kane', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('14', 'Paul Pogba', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('15', 'Toni Kroos', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('16', 'Romelu Lukaku', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('17', 'Jadon Sancho', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('18', 'Gareth Bale', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('19', 'Eden Hazard', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('20', 'Alisson Becker', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('21', 'Gerard Pique', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('22', 'Thiago Silva', '02');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('23', 'Luis Suarez', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('24', 'Raheem Sterling', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('25', 'Phil Foden', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('26', 'Joshua Kimmich', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('27', 'Manuel Neuer', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('28', 'Leroy Sane', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('29', 'Riyad Mahrez', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('30', 'Pierre-Emerick Aubameyang', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('31', 'Aymeric Laporte', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('32', 'Andrew Robertson', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('33', 'Pele', '03');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('34', 'Bruno Fernandes', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('35', 'Joao Felix', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('36', 'Sadio Mane', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('37', 'Trent Alexander-Arnold', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('38', 'Marcus Rashford', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
```

```
    VALUES ('39', 'Mason Mount', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('40', 'Declan Rice', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('41', 'Diego Maradona', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('42', 'Kalidou Koulibaly', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('43', 'Paulo Dybala', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME, TEAMID)
    VALUES ('44', 'Zlatan Ibrahimovic', '04');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME)
    VALUES ('45', 'Kyrie Irving');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME)
    VALUES ('46', 'Kanye West');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME)
    VALUES ('47', 'Quandale Dingle');
INSERT INTO PLAYER (PLAYERID, PLAYERNAME)
    VALUES ('48', 'P Diddy');

INSERT INTO COACH (COACHNAME, TEAMID)
    VALUES ('J.Mourinho', '01');
INSERT INTO COACH (COACHNAME, TEAMID)
    VALUES ('E.Kamalachandran', '04');
INSERT INTO COACH (COACHNAME, TEAMID)
    VALUES ('H.BOB', '02');
INSERT INTO COACH (COACHNAME, TEAMID)
    VALUES ('J.JOE', '03');

INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('01', '01', '02', 3, 0, '01', 36232, TO_DATE('2024-10-12', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('02', '03', '04', 1, 2, '03', 46732, TO_DATE('2024-10-13', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('03', '02', '03', 2, 2, '02', 54337, TO_DATE('2024-10-19', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('04', '04', '01', 0, 1, '04', 37234, TO_DATE('2024-10-20', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('05', '01', '03', 0, 0, '01', 38765, TO_DATE('2024-10-26', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('06', '02', '04', 1, 0, '02', 57283, TO_DATE('2024-10-27', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('07', '03', '01', 2, 3, '03', 52352, TO_DATE('2024-11-02', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('08', '04', '02', 1, 0, '04', 35627, TO_DATE('2024-11-03', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
```

```
    VALUES ('09', '03', '02', 1, 2, '03', 41738, TO_DATE('2024-11-09', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('10', '01', '04', 0, 2, '01', 39875, TO_DATE('2024-11-10', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('11', '02', '01', 1, 2, '02', 60125, TO_DATE('2024-11-16', 'YYYY-MM-DD'));
INSERT INTO GAME (GAMEID, HOMETEAM, AWAYTEAM, HOMEGOALS, AWAYGOALS, STADIUMID,
ATTENDANCE, GAMEDATE)
    VALUES ('12', '04', '03', 1, 0, '04', 36783, TO_DATE('2024-11-17', 'YYYY-MM-DD'));


-- Populating GOAL table
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('01', '01',
'01', '04', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('02', '01',
'01', '03', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('03', '01',
'02', '01', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('04', '02',
'23', '24', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('05', '02',
'35', '34', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('06', '02',
'36', '37', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('07', '03',
'12', '14', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('08', '03',
'13', '15', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('09', '03',
'23', '25', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('10', '03',
'28', '26', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('11', '04',
'01', '04', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('12', '06',
'19', '14', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('13', '07',
'30', '29', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('14', '07',
'30', '25', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('15', '07',
'01', '06', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('16', '07',
'01', '09', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('17', '07',
'02', '01', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('18', '08',
'44', '40', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('19', '09',
'13', '17', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('20', '09',
'18', '15', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('21', '09',
'23', '32', 'G');
```

```
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('22', '10',
'43', '41', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('23', '10',
'35', '34', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('24', '11',
'12', '14', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('25', '11',
'11', '01', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('26', '11',
'02', '01', 'G');
INSERT INTO GOAL (GOALID, GAMEID, SCORER, ASSISTER, GOALTYPE) VALUES ('27', '12',
'34', '35', 'G');

INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('01', '3-0', '01');
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('02', '1-2', '04');
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('03', '2-2', NULL);
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('04', '0-1', '01');
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('05', '0-0', NULL);
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('06', '1-0', '02');
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('07', '2-3', '01');
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('08', '1-0', '04');
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('09', '1-2', '02');
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('10', '0-2', '04');
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('11', '1-2', '01');
INSERT INTO SCORE (GAMEID, SCORE, WINNER)
    VALUES ('12', '1-0', '04');

UPDATE PLAYER
SET GAMES = 6
WHERE TEAMID != 'FA';

UPDATE PLAYER
SET GOALS = (
    SELECT COUNT(*) -- all occurrences
    FROM GOAL
    WHERE playerid = scorer
);

UPDATE PLAYER
SET ASSISTS = (
    SELECT COUNT(*)
    FROM GOAL
    WHERE playerid = assister
);
```

```
UPDATE STADIUM s
SET AVGATTENDANCE = (
    SELECT ROUND(AVG(m.attendance), 2)
    FROM GAME m
    WHERE m.stadiumid = s.stadiumid
    GROUP BY m.stadiumid
);
```

With the tables populated, the following simple SQL queries – one for each entity, – were run in SQL Developer to observe the corresponding tables. Keywords such as ORDER BY and GROUP BY were used to further define the queries.

Figure 4.1: Simple SQL Queries with Resulting Tables

Query 1: STADIUM - Show all stadiums with 50000+ Capacity

```
SELECT stadiumname AS "Stadium", seats AS "Capacity"
    -- retrieves a table with these 3 columns
    -- AS gives a name to each column in the queried table
    FROM STADIUM
    WHERE seats > 50000 -- condtion: must have more than 50000 seats
    ORDER BY seats DESC; -- order: by descending seats number
```

Query 2: TEAM - Select all teams and show their stadiums

	Stadium Name	Capacity
1	Stade Olympique	61004
2	Commonwealth Stadium	56302
3	BC Place	54320

```
SELECT t.teamname AS "Team", s.stadiumname AS "Home Stadium"
    FROM TEAM t -- alias 't' used to clarify where attributes belong
    JOIN STADIUM s ON t.stadiumid = s.stadiumid;
       -- join two columns from different tables based on shared foreign key
```

	Team	Home Stadium
1	Red Raptors FC	BC Place
2	Toronto FC	BMO Field
3	Phoenix FC	Commonwealth Stadium
4	Free-Agents	Free-Agency
5	Strikers United	Stade Olympique

Query 3: PLAYER - Show the top goal-scorers

```
SELECT playername AS "Player", goals AS "Goals"
    FROM PLAYER
    WHERE goals > 0 -- players with at least 1 goal
    ORDER BY goals DESC;
```

	Player	Goals
1	Lionel Messi	5
2	Cristiano Ronaldo	3
3	Luis Suarez	3
4	Pierre-Emerick Aubameyang	2
5	Harry Kane	2
6	Joao Felix	2
7	Karim Benzema	2
8	Bruno Fernandes	1
9	Sadio Mane	1
10	Erling Haaland	1
11	Zlatan Ibrahimovic	1
12	Paulo Dybala	1
13	Eden Hazard	1
14	Gareth Bale	1
15	Leroy Sane	1

Query 4: COACH - Show coach and the team they manage

```
SELECT co.coachname AS "Coach", t.teamname AS "Team"
    FROM COACH co
    JOIN TEAM t ON co.teamid = t.teamid;
```

	Coach	Team
1	J.JOE	Phoenix FC
2	E.Kamalachandran	Red Raptors FC
3	H.BOB	Strikers United
4	J.Mourinho	Toronto FC

Query 5: GAME - Show the teams with the most goals at home

```
SELECT t.teamname AS "Team", SUM(homegoals) AS "Total Home Goals"
    FROM GAME g
    JOIN TEAM t ON g.hometeam = t.teamid
    GROUP BY t.teamname
    -- makes sure homegoals are grouped by teamname
    -- otherwise SUM(homegoals) returns the total homegoals for every team
    ORDER BY "Total Goals" DESC;
```

	Team	Total Home Goals
1	Phoenix FC	4
2	Strikers United	4
3	Toronto FC	3
4	Red Raptors FC	2

Query 6: GOAL - Show the total number of goals scored in the league

```
SELECT COUNT(*) AS "Totol League Goals" FROM GOAL;
```

	Totol League Goals
1	27

Query 7: SCORE - Show all the winning results of Toronto FC

```
SELECT s.gameid AS "Matchday", s.score AS "Result", t.teamname AS "Team"
    FROM SCORE s
    JOIN TEAM t ON s.winner = t.teamid
    WHERE t.teamname = 'Toronto FC';
```

	Matchday	Result	Team
1	1	3–0	Toronto FC
2	4	0–1	Toronto FC
3	7	2–3	Toronto FC
4	11	1–2	Toronto FC

Following the successful completion of simple queries, more advanced queries with the JOIN keyword were implemented as views.

Figure 4.2: Advanced SQL Queries (Views) with Resulting Tables

Query 8: TOP PERFORMERS - Show the best performing players based on Goals/Assists

```
CREATE VIEW top_performers AS
SELECT t.teamname AS "Team", p.playername AS "Player",
          p.goals AS "Goals", p.assists AS "Assists",
          (p.goals + p.assists) AS "Total Contributions"
    -- "Total Contributions" is a derived column from PLAYER table
    FROM PLAYER p
    JOIN TEAM t ON p.teamid = t.teamid
    WHERE p.goals > 0 OR p.assists > 0
    ORDER BY "Total Contributions" DESC, p.goals DESC, p.assists DESC;
    -- if multiple players has the same amount of contributions,
    -- they are then ordered by goals, however if their goals are the same,
    -- they are finally ordered by assists
```

	Team	Player	Goals	Assists	Total Contributions
1	Toronto FC	Lionel Messi	5	4	9
2	Toronto FC	Cristiano Ronaldo	3	0	3
3	Phoenix FC	Luis Suarez	3	0	3
4	Red Raptors FC	Joao Felix	2	1	3
5	Red Raptors FC	Bruno Fernandes	1	2	3
6	Strikers United	Paul Pogba	0	3	3
7	Phoenix FC	Pierre-Emerick Aubameyang	2	0	2
8	Strikers United	Karim Benzema	2	0	2
9	Strikers United	Harry Kane	2	0	2
10	Strikers United	Toni Kroos	0	2	2
11	Phoenix FC	Phil Foden	0	2	2
12	Toronto FC	Kevin De Bruyne	0	2	2
13	Toronto FC	Erling Haaland	1	0	1
14	Red Raptors FC	Paulo Dybala	1	0	1
15	Strikers United	Eden Hazard	1	0	1

Query 9: LEAGUE TABLE - Show the current league standings based on points

```
CREATE VIEW league_table AS
SELECT
    t.teamname AS "Team",
    SUM(CASE -- Calculating points (CASE: WHEN: THEN is like 'if statement')
            WHEN t.teamid = g.hometeam AND g.homegoals > g.awaygoals THEN 3
            -- Home win: More home goals than away goals -> Add 3 points
            WHEN t.teamid = g.awayteam AND g.awaygoals > g.homegoals THEN 3
            -- Away win: More away goals than home goals -> Add 3 points
            WHEN (t.teamid = g.hometeam OR t.teamid = g.awayteam)
                 AND g.homegoals = g.awaygoals THEN 1
                 -- Draw: Same amount of home and away goals -> Add 1 point
            ELSE 0  -- Loss -> Add 0 points
        END) AS "Points",
    COUNT(CASE
            WHEN t.teamid = g.hometeam AND g.homegoals > g.awaygoals THEN 1
            WHEN t.teamid = g.awayteam AND g.awaygoals > g.homegoals THEN 1
        END) AS "Wins",
    COUNT(CASE
            WHEN (t.teamid = g.hometeam OR t.teamid = g.awayteam)
                 AND g.homegoals = g.awaygoals THEN 1
        END) AS "Draws",
    COUNT(CASE
            WHEN t.teamid = g.hometeam AND g.homegoals < g.awaygoals THEN 1
            WHEN t.teamid = g.awayteam AND g.awaygoals < g.homegoals THEN 1
        END) AS "Losses",
    SUM(CASE
          WHEN t.teamid = g.hometeam THEN g.homegoals
          WHEN t.teamid = g.awayteam THEN g.awaygoals
    END) AS "Goals Scored",
    SUM(CASE
          WHEN t.teamid = g.hometeam THEN g.awaygoals
          WHEN t.teamid = g.awayteam THEN g.homegoals
    END) AS "Goals Conceded",
    (SUM(CASE
        WHEN t.teamid = g.hometeam THEN g.homegoals
        WHEN t.teamid = g.awayteam THEN g.awaygoals
     END)
    -
     SUM(CASE
        WHEN t.teamid = g.hometeam THEN g.awaygoals
        WHEN t.teamid = g.awayteam THEN g.homegoals
     END)) AS "Goal Difference"
FROM GAME g
JOIN TEAM t ON (t.teamid = g.hometeam OR t.teamid = g.awayteam)
GROUP BY t.teamname
ORDER BY "Points" DESC, "Wins" DESC, "Goal Difference" DESC;
```

	Team	Points	Wins	Draws	Losses	Goals Scored	Goals Conceded	Goal Difference
1	Toronto FC	13	4	1	1	9	5	4
2	Red Raptors FC	12	4	0	2	6	3	3
3	Strikers United	7	2	1	3	6	9	-3
4	Phoenix FC	2	0	2	4	6	10	-4

Query 10: MESSI_LOG - Show the games messi played in and the results

```
CREATE VIEW messi_log AS
SELECT m.gameid,
    CASE
        WHEN m.hometeam = p.teamid THEN t2.teamname
        WHEN m.awayteam = p.teamid THEN t1.teamname
        END AS "Opposition",
    s.score AS "Score"

    FROM GAME m
    JOIN PLAYER p ON (m.hometeam = p.teamid OR m.awayteam = p.teamid)
    JOIN SCORE s ON m.gameid = s.gameid
    JOIN TEAM t1 ON m.hometeam = t1.teamid
    JOIN TEAM t2 ON m.awayteam = t2.teamid
    WHERE p.playerid = 1; -- Select player to log based on ID
```

	GAMEID	Opposition	Score
1	1	Strikers United	3–0
2	4	Red Raptors FC	0–1
3	5	Phoenix FC	0–0
4	7	Phoenix FC	2–3
5	10	Red Raptors FC	0–2
6	11	Strikers United	1–2

Query 11: STADIUM_AVG - Show the attendance average for each stadium in each team

```
CREATE VIEW stadium_avgs AS
SELECT t.teamname AS "Team",
        s.stadiumname AS "Stadium",
        ROUND(AVG(m.attendance),2) AS "Attendance Avg."
        -- rounds avg attendance to 2 decimal places
FROM GAME m
JOIN STADIUM s ON m.stadiumid = s.stadiumid
JOIN TEAM t ON s.stadiumid = t.stadiumid
GROUP BY t.teamname, s.stadiumname;
```

	Team	Stadium	Attendance Avg.
1	Toronto FC	BMO Field	38290.67
2	Phoenix FC	Commonwealth Stadium	46940.67
3	Strikers United	Stade Olympique	57248.33
4	Red Raptors FC	BC Place	36548

5 - ADVANCED QUERIES BY UNIX SHELL IMPLEMENTATION

The creation of more advanced queries was required, but this time they would be implemented on a user interface (UI) developed in Bash script. More keywords such as UNION, EXISTS, and MINUS; clauses such as HAVING; and aggregate functions such as COUNT were used throughout these queries.

Figure 5.0: Advanced SQL Queries for Shell Script UI with Resulting Tables

Query 12: Show the players with the most goals and assists

```
SELECT p.playername AS "Player", p.goals AS "Goals" , p.assists AS "Assists"
FROM player p
WHERE EXISTS ( -- Checks if row exists: returns true
  SELECT 1 -- value 1 is returned if row exists (subquery)
  FROM goal g
  WHERE g.scorer = p.playerid
  GROUP BY p.playerid
  HAVING COUNT(*) > 0
)
OR EXISTS (
  SELECT 1
  FROM goal g
  WHERE g.assister = p.playerid
  GROUP BY p.playerid
  HAVING COUNT(*) > 0
)
ORDER BY p.goals DESC, p.assists DESC;
```

	Player	Goals	Assists
1	Lionel Messi	5	4
2	Cristiano Ronaldo	3	0
3	Luis Suarez	3	0
4	Joao Felix	2	1
5	Harry Kane	2	0
6	Pierre-Emerick Aubameyang	2	0
7	Karim Benzema	2	0
8	Bruno Fernandes	1	2
9	Sadio Mane	1	0
10	Leroy Sane	1	0

Query 13: Show all participants in a league: players and coaches

```
SELECT p.playername AS "Name",
       t1.teamname AS Team,
       'player' AS Role  -- Static string for players
FROM PLAYER p
JOIN TEAM t1 ON p.teamid = t1.teamid

UNION ALL  -- UNION ALL to includes all results without removing duplicates

SELECT  c.coachname,
        t2.teamname,
       'coach' AS Role  -- Static string for coaches
FROM COACH c
JOIN TEAM t2 ON c.teamid = t2.teamid
ORDER BY Team, Role;
```

	Name	TEAM	ROLE
1	J.JOE	Phoenix FC	coach
2	Raheem Sterling	Phoenix FC	player
3	Pele	Phoenix FC	player
4	Andrew Robertson	Phoenix FC	player
5	Aymeric Laporte	Phoenix FC	player
6	Pierre-Emerick Aubameyang	Phoenix FC	player
7	Riyad Mahrez	Phoenix FC	player
8	Leroy Sane	Phoenix FC	player
9	Manuel Neuer	Phoenix FC	player
10	Joshua Kimmich	Phoenix FC	player
11	Phil Foden	Phoenix FC	player
12	Luis Suarez	Phoenix FC	player
13	E.Kamalachandran	Red Raptors FC	coach
14	Joao Felix	Red Raptors FC	player

Query 14:: Show the games where attendance was above average

```
SELECT m.gameid AS game_id,
       t1.teamname AS "Home Team",
       t2.teamname AS "Away Team",
       s.stadiumname AS "Stadium",
       m.attendance
       FROM GAME m
       JOIN TEAM t1 ON m.hometeam = t1.teamid
       JOIN TEAM t2 ON m.awayteam = t2.teamid
       JOIN STADIUM s ON m.stadiumid = s.stadiumid

MINUS

SELECT m.gameid AS game_id,
       t1.teamname AS "Home Team",
       t2.teamname AS "Away Team",
       s.stadiumname AS "Stadium",
       m.attendance
       FROM GAME m
       JOIN TEAM t1 ON m.hometeam = t1.teamid
       JOIN TEAM t2 ON m.awayteam = t2.teamid
       JOIN STADIUM s ON m.stadiumid = s.stadiumid
       WHERE (m.attendance - s.avgattendance <= 0);
```

	GAME_ID	Home Team	Away Team	Stadium	ATTENDANCE
1	10	Toronto FC	Red Raptors FC	BMO Field	39875
2	11	Strikers United	Toronto FC	Stade Olympique	60125
3	12	Red Raptors FC	Phoenix FC	BC Place	36783
4	4	Red Raptors FC	Toronto FC	BC Place	37234
5	5	Toronto FC	Phoenix FC	BMO Field	38765
6	6	Strikers United	Red Raptors FC	Stade Olympique	57283
7	7	Phoenix FC	Toronto FC	Commonwealth Stadium	52352

*Query 15: Show most crucial players to their team (Goals + Assists/ Team Goals *100%)*

```
SELECT pc.playername AS "Player",
       t.teamname AS "Team",
       pc.goals + pc.assists AS "Goal contributions",
       tg.total_team_goals,
```

```
       ROUND((pc.goals + pc.assists) / tg.total_team_goals * 100, 2) AS
contribution_percentage
FROM (
    -- Calculating each player's contributions (goals + assists)
    SELECT p.playerid, p.playername, p.teamid,
           SUM(CASE WHEN g.scorer = p.playerid THEN 1 ELSE 0 END) AS goals,
           SUM(CASE WHEN g.assister = p.playerid THEN 1 ELSE 0 END) AS assists
    FROM player p
    LEFT JOIN goal g ON p.playerid = g.scorer OR p.playerid = g.assister
    GROUP BY p.playerid, p.playername, p.teamid
    HAVING p.teamid > '0' -- Do not include free-agents
) pc -- Subquery alias: "pc" = player contribution
-- Calculating total number of goals scored by each team
JOIN (
    SELECT p.teamid, SUM(CASE WHEN g.scorer IS NOT NULL THEN 1 ELSE 0 END) AS
total_team_goals
    FROM goal g
    JOIN player p ON g.scorer = p.playerid  -- Join to get the player's team for each
goal
    GROUP BY p.teamid
    HAVING p.teamid > '0' -- Do not include free-agents
) tg ON pc.teamid = tg.teamid -- Subquery alias: "tg" = team goals
-- Join with the team table to get team names
JOIN team t ON pc.teamid = t.teamid
WHERE tg.total_team_goals > 0  -- Exclude teams with no goals
ORDER BY contribution_percentage DESC;
```

	Player	Team	Goal contributions	TOTAL_TEAM_GOALS	CONTRIBUTION_PERCENTAGE
1	Lionel Messi	Toronto FC	9	9	100
2	Joao Felix	Red Raptors FC	3	6	50
3	Paul Pogba	Strikers United	3	6	50
4	Luis Suarez	Phoenix FC	3	6	50
5	Bruno Fernandes	Red Raptors FC	3	6	50
6	Toni Kroos	Strikers United	2	6	33.33
7	Karim Benzema	Strikers United	2	6	33.33
8	Cristiano Ronaldo	Toronto FC	3	9	33.33
9	Phil Foden	Phoenix FC	2	6	33.33
10	Harry Kane	Strikers United	2	6	33.33
11	Pierre-Emerick Aubameyang	Phoenix FC	2	6	33.33
12	Kevin De Bruyne	Toronto FC	2	9	22.22
13	Raheem Sterling	Phoenix FC	1	6	16.67
14	Paulo Dybala	Red Raptors FC	1	6	16.67

Query 16: Show all games with more than 3 goals

```
SELECT t1.teamname AS "Home Team",
       t2.teamname AS "Away Team",
       s.score
FROM score s
JOIN game g ON s.gameid = g.gameid
JOIN team t1 ON g.hometeam = t1.teamid
JOIN team t2 ON g.awayteam = t2.teamid
WHERE EXISTS (
    SELECT 1
    FROM game g2
    WHERE g2.gameid = g.gameid
    AND (g2.homegoals + g2.awaygoals) > 3
```

```
)
GROUP BY t1.teamname, t2.teamname, s.score
HAVING (TO_NUMBER(SUBSTR(s.score, 1, INSTR(s.score, '-') - 1)) +
         TO_NUMBER(SUBSTR(s.score, INSTR(s.score, '-') + 1))) > 3;
-- HAVING clause checks if score is more than 3 goals
-- INSTR returns the instance of a char in a sting: like charAT()
```

	Home Team	Away Team	SCORE
1	Strikers United	Phoenix FC	2–2
2	Phoenix FC	Toronto FC	2–3

Once the advanced queries from **Figure 5.0** were successfully executed, a shell script was implemented for a user interface (UI) that could be used in a computer's terminal. The shell script was able to connect to the database to create, delete, populate, view, and query tables.

Figure 5.1: Bash/Shell Script for Terminal UI

menu.sh

```
#!/bin/bash

MainMenu()
{
    while [ "$CHOICE" != "START" ]
    do
    clear
        echo "================================================================="
        echo "|                      SOCCER LEAGUE DATABASE                   |"
        echo "|             Main Menu - Select Desired Operation(s):          |"
        echo "|         <CTRL-Z Anytime to Enter Interactive CMD Prompt>       |"
        echo "-----------------------------------------------------------------"
        echo " $IS_SELECTEDM M) View Manual"
        echo " "
        echo " $IS_SELECTED1 1) Drop Tables"
        echo " $IS_SELECTED2 2) Create Tables"
        echo " $IS_SELECTED3 3) Populate Tables"
        echo " $IS_SELECTED4 4) Query Tables"
       echo " $IS_SELECTED5 5) View Tables"
       echo " $IS_SELECTED6 6) Custom SQL"
        echo " "
        echo " $IS_SELECTEDX X) Force/Stop/Kill Oracle DB"
        echo " "
        echo " $IS_SELECTEDE E) End/Exit"
        echo "Choose: "
        read CHOICE
        if [ "$CHOICE" == "0" ]
        then
            echo "Nothing Here"
           sleep 5
        elif [ "$CHOICE" == "1" ]
        then
            bash drop_tables.sh
          echo "Tables dropped. Press any key to continue"
            while true; do
              read -rsn1 key  # Read a single character silently
```

```
      if [[ -n "$key" ]]; then
             echo "Key pressed. Continuing..."
             break  # Exit the loop if a key is pressed
      fi
    done
 elif [ "$CHOICE" == "2" ]
 then
  bash create_tables.sh
    echo "Tables created. Press any key to continue"
     while true; do
         read -rsn1 key  # Read a single character silently
         if [[ -n "$key" ]]; then
                 echo "Key pressed. Continuing..."
                 break  # Exit the loop if a key is pressed
         fi
     done
 elif [ "$CHOICE" == "3" ]
 then
     bash populate_tables.sh
    echo "Tables populated. Press any key to continue"
     while true; do
         read -rsn1 key  # Read a single character silently
         if [[ -n "$key" ]]; then
                 echo "Key pressed. Continuing..."
                 break  # Exit the loop if a key is pressed
         fi
     done
 elif [ "$CHOICE" == "4" ]
 then
    bash queries.sh
    echo "Press any key to continue"
     while true; do
         read -rsn1 key  # Read a single character silently
         if [[ -n "$key" ]]; then
                 echo "Key pressed. Continuing..."
                 break  # Exit the loop if a key is pressed
         fi
     done
elif [ "$CHOICE" == "5" ]
 then
     bash views.sh
    echo "Viewing all tables. Press any key to continue"
     while true; do
         read -rsn1 key  # Read a single character silently
         if [[ -n "$key" ]]; then
                 echo "Key pressed. Continuing..."
                 break  # Exit the loop if a key is pressed
         fi
     done
elif [ "$CHOICE" == "6" ]
 then
    echo "Enter your own SQL statement. Press ENTER to run it."
   echo "To exit, enter  MM, then press ENTER"
   while true; do
       # Prompt the user for a SQL statement
```

```
            read -p "SQL> " sql_query

                 # Check if the user entered MM to exit
                   if [ "$sql_query" == "MM" ]; then
                          echo "Returning to main menu..."
                          break
                   fi
            # Execute the SQL query
            echo "$sql_query" | sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.torontomu.ca)(P
ort=1521))(CONNECT_DATA=(SID=orcl)))"
            SET LINESIZE 200;
          done
        elif [ "$CHOICE" == "E" ]
        then
            exit
        fi
    done
}
#--COMMENTS BLOCK--
# Main Program
#--COMMENTS BLOCK--
ProgramStart()
{
    StartMessage
    while [ 1 ]
    do
        MainMenu
    done
}
ProgramStart
```

Figure 5.1.a: Bash Shell UI Main Page (menu.sh)

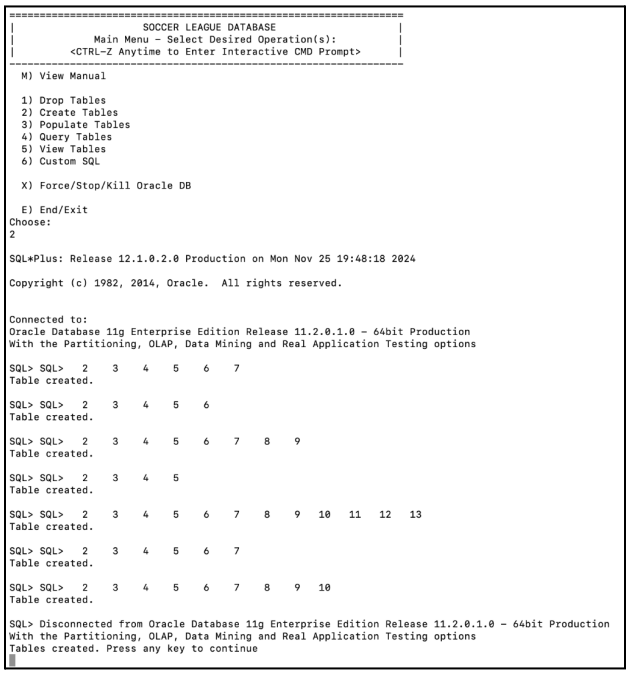
```
================================================================
|                    SOCCER LEAGUE DATABASE                    |
|           Main Menu - Select Desired Operation(s):           |
|       <CTRL-Z Anytime to Enter Interactive CMD Prompt>       |
----------------------------------------------------------------
  M) View Manual

  1) Drop Tables
  2) Create Tables
  3) Populate Tables
  4) Query Tables
  5) View Tables
  6) Custom SQL

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
```

create_tables.sh

```
#!/bin/sh
# Optional: Uncomment the following line if you need to set the library path
# export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib

# Ensure the connection string is correctly formatted
```

```
sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.torontomu.ca)(P
ort=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF
        ** Insert SQL to create tables from Figure 3.0 **
exit;
EOF
```

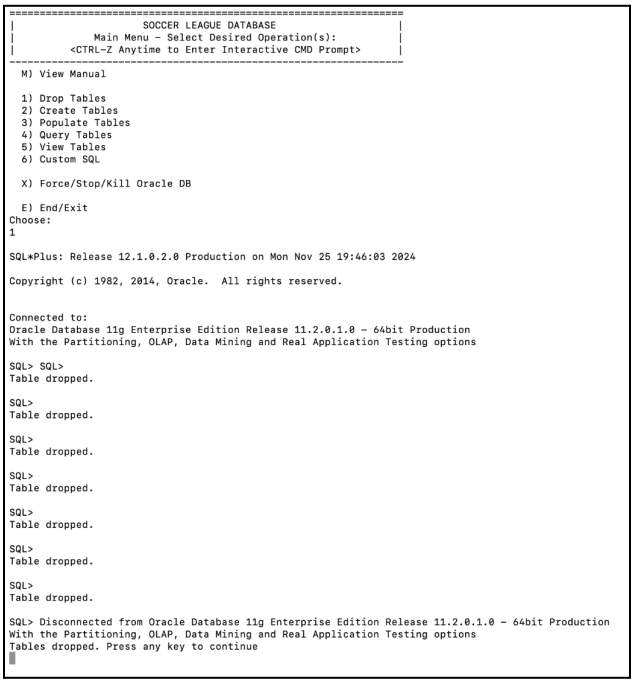
Figure 5.1.b: Bash Shell UI After Creating Tables (create.sh)



drop_tables.sh

```
#!/bin/sh
# export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.torontomu.ca)(P
ort=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF

DROP TABLE SCORE CASCADE CONSTRAINTS;
DROP TABLE GOAL CASCADE CONSTRAINTS;
DROP TABLE GAME CASCADE CONSTRAINTS;
DROP TABLE COACH CASCADE CONSTRAINTS;
DROP TABLE PLAYER CASCADE CONSTRAINTS;
DROP TABLE TEAM CASCADE CONSTRAINTS;
DROP TABLE STADIUM CASCADE CONSTRAINTS;
exit;
EOF
```

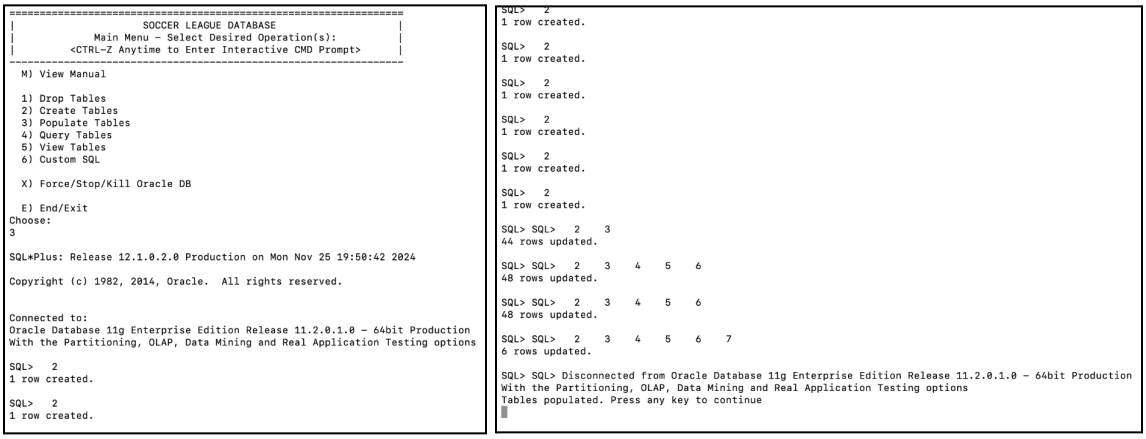
Figure 5.1.c: Bash Shell UI After Dropping Tables (drop_tables.sh)



populate_tables.sh

```
#!/bin/sh
# Optional: Uncomment the following line if you need to set the library path
# export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib

# Ensure the connection string is correctly formatted
sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.torontomu.ca)(P
ort=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF
        ** Insert SQL to create tables from Figure 4.0 **
exit;
EOF
```

Figure 5.1.d: Bash Shell UI After Populating Tables (populate_tables.sh)

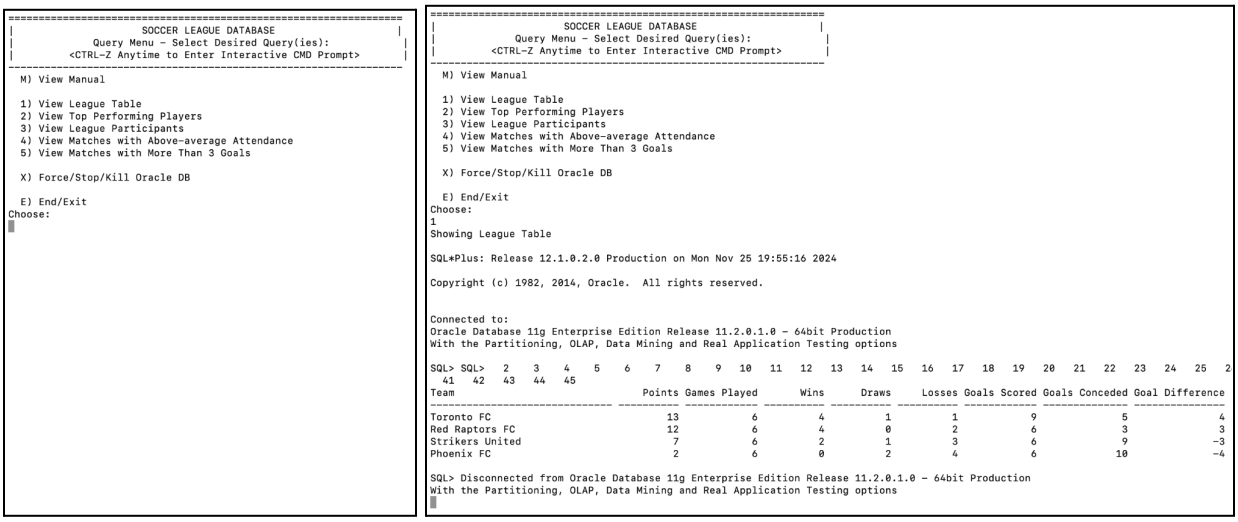


query_tables.sh

```
MainMenu()
{
    while [ "$CHOICE" != "START" ]
    do
        clear
        echo "=============================================================="
        echo "|                     SOCCER LEAGUE DATABASE                 |"
        echo "|             Query Menu - Select Desired Query(ies):         |"
        echo "|         <CTRL-Z Anytime to Enter Interactive CMD Prompt>    |"
        echo "--------------------------------------------------------------"
        echo " $IS_SELECTEDM M) View Manual"
        echo " "
        echo " $IS_SELECTED1 1) View League Table"
        echo " $IS_SELECTED2 2) View Top Performing Players"
        echo " $IS_SELECTED3 3) View League Participants"
        echo " $IS_SELECTED4 4) View Matches with Above-average Attendance"
        echo " $IS_SELECTED5 5) View Matches with More Than 3 Goals"
        echo " "
        echo " $IS_SELECTEDX X) Force/Stop/Kill Oracle DB"
        echo " "
        echo " $IS_SELECTEDE E) End/Exit"
        echo "Choose: "
        read CHOICE
        if [ "$CHOICE" == "0" ]; then
            echo "Nothing Here"
            sleep 5
        elif [ "$CHOICE" == "1" ]; then
            echo "Showing League Table"
sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.torontomu.ca)(P
ort=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF
SET LINESIZE 200;
        ** Insert SQL to view league table: Query 9 from Figure 4.2 **
EOF
            while true; do
                read -rsn1 key  # Read a single character silently
                if [[ -n "$key" ]]; then
                    echo "Key pressed. Continuing..."
                    break  # Exit the loop if a key is pressed
                fi
            done
        elif [ "$CHOICE" == "2" ]; then
            echo "Showing Top Performers"
sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.torontomu.ca)(P
ort=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF
        ** Insert SQL to view top performers: Query 12 from Figure 5.0 **
EOF
            while true; do
                read -rsn1 key  # Read a single character silently
                if [[ -n "$key" ]]; then
                    echo "Key pressed. Continuing..."
                    break  # Exit the loop if a key is pressed
```

```
                fi
            done
        elif [ "$CHOICE" == "3" ]; then
            echo "Showing League Participants"
sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.torontomu.ca)(P
ort=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF
        ** Insert SQL to view league participants: Query 13 from Figure 5.0 **
EOF
            while true; do
                read -rsn1 key  # Read a single character silently
                if [[ -n "$key" ]]; then
                    echo "Key pressed. Continuing..."
                    break  # Exit the loop if a key is pressed
                fi
            done
        elif [ "$CHOICE" == "4" ]; then
            echo "Showing Matches with Above-average Attendance"
sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.torontomu.ca)(P
ort=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF
        ** Insert SQL to view games with above average attendance: Query 14 from Figure 5.0 **
EOF
            while true; do
                read -rsn1 key  # Read a single character silently
                if [[ -n "$key" ]]; then
                    echo "Key pressed. Continuing..."
                    break  # Exit the loop if a key is pressed
                fi
            done
        elif [ "$CHOICE" == "5" ]; then
            echo "Showing all games with more than 3 goals"
sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.torontomu.ca)(P
ort=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF
        ** Insert SQL to view games with more than 3 goals: Query 16 from Figure 5.0 **
EOF
            while true; do
                read -rsn1 key  # Read a single character silently
                if [[ -n "$key" ]]; then
                    echo "Key pressed. Continuing..."
                    break  # Exit the loop if a key is pressed
                fi
            done
        elif [ "$CHOICE" == "X" ]; then
            echo "Stopping Oracle..."
            # Your logic for force/stopping/killing Oracle DB goes here
        elif [ "$CHOICE" == "E" ]; then
            exit
        fi
    done
}
MainMenu
```

Figure 5.1.e: Bash Shell UI Query Page After Querying League Table (query_tables.sh)

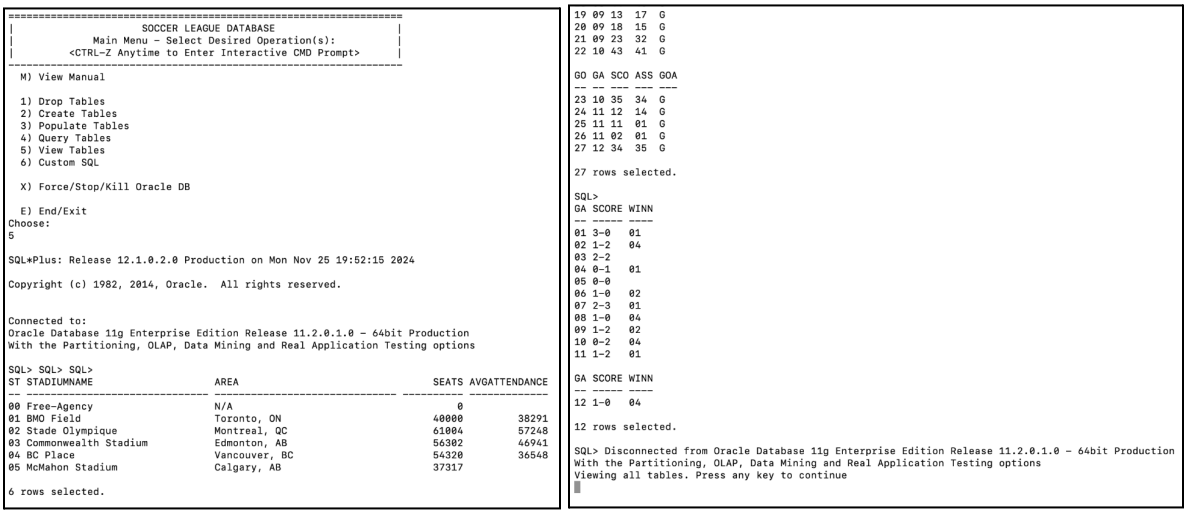


```
============================================================
|                    SOCCER LEAGUE DATABASE                |
|            Query Menu - Select Desired Query(ies):       |
|        <CTRL-Z Anytime to Enter Interactive CMD Prompt>   |
------------------------------------------------------------
  M) View Manual

  1) View League Table
  2) View Top Performing Players
  3) View League Participants
  4) View Matches with Above-average Attendance
  5) View Matches with More Than 3 Goals

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
1
Showing League Table

SQL*Plus: Release 12.1.0.2.0 Production on Mon Nov 25 19:55:16 2024

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL>   2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21   22   23   24   25   2
  41   42   43   44   45
Team                           Points Games Played       Wins      Draws     Losses Goals Scored Goals Conceded Goal Difference
------------------------------ ------ ------------ ---------- ---------- ---------- ------------- --------------- ---------------
Toronto FC                         13            6          4          1          1             9               5               4
Red Raptors FC                     12            6          4          0          2             6               3               3
Strikers United                     7            6          2          1          3             6               9              -3
Phoenix FC                          2            6          0          2          4             6              10              -4

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
```

view_tables.sh

```
#!/bin/sh
# export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
# Ensure the connection string is correctly formatted
sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.ryerson.ca)(Por
t=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF
SET LINESIZE 200;
SELECT * FROM stadium;
SELECT * FROM team;
SELECT * FROM player;
SELECT * FROM coach;
SELECT * FROM game;
SELECT * FROM goal;
SELECT * FROM score;
exit;
EOF
```

Figure 5.1.f: Bash Shell UI After Viewing Tables (view_tables.sh)



```
============================================================
|                    SOCCER LEAGUE DATABASE                |
|             Main Menu - Select Desired Operation(s):     |
|        <CTRL-Z Anytime to Enter Interactive CMD Prompt>   |
------------------------------------------------------------
  M) View Manual

  1) Drop Tables
  2) Create Tables
  3) Populate Tables
  4) Query Tables
  5) View Tables
  6) Custom SQL

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
5

SQL*Plus: Release 12.1.0.2.0 Production on Mon Nov 25 19:52:15 2024

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL> SQL>
ST STADIUMNAME                 AREA                           SEATS AVGATTENDANCE
-- --------------------------- ------------------------------ ----------- --------------
00 Free-Agency                 N/A                                      0
01 BMO Field                   Toronto, ON                          40000          38291
02 Stade Olympique             Montreal, QC                         61004          57248
03 Commonwealth Stadium        Edmonton, AB                         56302          46941
04 BC Place                    Vancouver, BC                        54320          36548
05 McMahon Stadium             Calgary, AB                          37317

6 rows selected.
```

```
19 09 13  17  G
20 09 18  15  G
21 09 23  32  G
22 10 43  41  G

GO GA SCO ASS GOA
-- -- --- --- ---
23 10 35  34  G
24 11 12  14  G
25 11 11  01  G
26 11 02  01  G
27 12 34  35  G

27 rows selected.

SQL>
GA SCORE WINN
-- ----- ----
01 3-0   01
02 1-2   04
03 2-2
04 0-1   01
05 0-0
06 1-0   02
07 2-3   01
08 1-0   04
09 1-2   02
10 0-2   04
11 1-2   01

GA SCORE WINN
-- ----- ----
12 1-0   04

12 rows selected.

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Viewing all tables. Press any key to continue
```

6 - NORMALIZATION OF DATABASE / FUNCTIONAL DEPENDENCIES

Following the creation of the Bash shell UI was the normalization process of the database. This process began with examining the functional dependencies within each entity. The functional dependencies for each table are denoted by "FD" with the primary keys being underlined. Due to the simplicity of the database design, with one primary key per table, the functional dependencies were straightforward.

Figure 6.0: Functional Dependencies of Each Database Table

Player

playerid	playername	teamid	games	goals	assists

FD: **playerid → playername, teamid, games, goals, assists**

Coach

teamid	coachname

FD: **teamid → coachname**

Team

teamid	teamname	stadiumid

FD: **teamid → teamname, stadiumid**

Stadium

stadiumid	stadiumname	area	seats	avgattendance

FD: **stadiumid → stadiumname, area, seats, avgattendance**

Game

gameid	hometeam	awayteam	homegoals	awaygoals	stadiumid	attendance	gamedate

FD: **gameid → hometeam, awayteam, homegoals, awaygoals, stadiumid, attendance, gamedate**

Score

gameid	score	winner

FD: **gameid → score, winner**

Goal

goalid	gameid	scorer	assister	goaltype

FD: **goalid → gameid, scorer, assister, goaltype**

7 - NORMALIZATION OF DATABASE / 3NF (3rd Normal Form)

With the functional dependencies from **Figure 6.0**, the next step was to verify that all database tables were 3NF. If not, changes had to be made. In the case of this particular database, all tables were 3NF upon creation.

All the tables are in 3NF as they satisfy the following criteria.

- 1NF (atomic values: one per table cell), 2NF(no partial dependency)
- No transitive dependency

The following algorithm was used to verify the tables being 3NF. The algorithm input consists of a schema (table) *R*, a set of functional dependencies *F* defined on *R*, and a candidate key *K* of the table R. The following steps are taken:

1. Find the prime attributes of the table *R*
2. Check each functional dependency $X \rightarrow Y$ for the following conditions
    - Condition 1: *X* is a superkey
    - Condition 2: Every attribute in *Y* is a prime attribute
3. If any of the two conditions are violated by a functional dependency, the table is not 3NF. However if 1 of the conditions is satisfied, the table is 3NF

For *Player* table:
R = (playerid, playername, teamid, games, goals, assists)
F = {playerid → playername, teamid, games, goals, assists}
K = playerid

1. Prime Attribute is playerid
2. For functional dependency set *F*, the left side is always a superkey fulfilling Condition 1.
3. *Player* is in 3NF

For *Coach* table:
R = (teamid, coachname)
F = {teamid → coachname}
K = teamid

1. Prime Attribute is teamid
2. For functional dependency set *F*, the left side is always a superkey fulfilling Condition 1.
3. *Coach* is in 3NF

For *Team* table:
R = (teamid, teamname, stadiumid)
F = {teamid → teamname, stadiumid}
K = teamid

1. Prime Attribute is teamid
2. For functional dependency set *F*, the left side is always a superkey fulfilling Condition 1.
3. *Team* is in 3NF

For *Stadium* table:

R = (stadiumid, stadiumname, area, seats, avgattendance)

F = {stadiumid → stadiumname, area, seats, avgattendance}

K = stadiumid

1. Prime Attribute is stadiumid
2. For functional dependency set *F*, the left side is always a superkey fulfilling Condition 1.
3. *Stadium* is in 3NF

For *Game* table:

R = (gameid, hometeam, awayteam, homegoals, awaygoals, stadiumid, attendance, gamedate)

F = {gameid → hometeam, awayteam, homegoals, awaygoals, stadiumid, attendance, gamedate}

K = gameid

1. Prime Attribute is gameid
2. For functional dependency set *F*, the left side is always a superkey fulfilling Condition 1.
3. *Game* is in 3NF

For *Score* table:

R = (gameid, score, winner)

F = {gameid → score, winner}

K = gameid

1. Prime Attribute is gameid
2. For functional dependency set *F*, the left side is always a superkey fulfilling Condition 1.
3. *Score* is in 3NF

For *Goal* table:

R = (goalid, gameid, scorer, assister, goaltype)

F = {goalid → gameid, scorer, assister, goaltype}

K = goalid

1. Prime Attribute is goalid
2. For functional dependency set *F*, the left side is always a superkey fulfilling Condition 1.
3. *Goal* is in 3NF

8 - NORMALIZATION OF DATABASE / BCNF (Boyce-Codd Normal Form)

After verifying that the tables were 3NF, the tables were to then be normalized to BCNF if required. An algorithm very similar to the one used in 3NF verification was applied to verify the normal form of each table. BCNF has stricter requirements than 3NF.

All tables are BCNF if they satisfy the following conditions

- The tables are 3NF
- For any functional dependency where A → B, A is always a superkey.

Due to this slight difference, a small tweak in the original algorithm is required.

The algorithm input consists of a schema (table) *R*, a set of functional dependencies *F* defined on *R*, and a candidate key *K* of the table R. The following steps are taken:

1. Find the prime attributes of the table *R*
2. Check each functional dependency $X \rightarrow Y$. Only condition is that *X* must be a superkey.
3. If this condition is met, the table is in BCNF.

The only change in the algorithm is the removal of Condition 2, which would make a table 3NF but may violate BCNF, as *X* (left-hand side of FD: determinant) must strictly be a superkey in BCNF.

Since all the frequency dependencies were found to contain superkeys on the left-hand side of each functional dependency for 3NF verification, the tables are also confirmed to be BCNF. This algorithm did not have to be implemented for verification. To view how the algorithm would be implemented, refer to the algorithm of the previous section and exclude the 2nd condition: 7 - NORMALIZATION / 3NF.

9 - JAVA BASED UI

The project concluded with creating a Java-based application. Java Swing was used to develop a graphical user interface. Java's SQL connectivity features were used to connect to Oracle 11g Database. As a result, this application was able to run DDL and DML commands to modify and query the database.

Figure 9.0: Java Application Code for Soccer League Application

DatabaseConnection.java

```
import java.sql.*; // Imports Java SQL library

// This class manages the connection to the database and executes queries
public class DatabaseConnection {

    // Database credentials
    private final String url = "jdbc:oracle:thin:@oracle.cs.torontomu.ca:1521:orcl";
// Database URL
    private final String username = "username";  // Oracle username
    private final String password = "password";  // Oracle password

    private static DatabaseConnection instance; // Singleton instance
    private Connection connection;  // Persistent connection object

    // Private constructor to prevent direct instantiation
    private DatabaseConnection() {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            this.connection = DriverManager.getConnection(url, username, password);
            this.connection.setAutoCommit(false); // Disable auto-commit
            System.out.println("Connected!"); // Only printed once
        } catch (ClassNotFoundException | SQLException e) {
            System.err.println("Failed to connect to the database.");
            e.printStackTrace();
        }
    }

    // Public method to get the singleton instance
    public static DatabaseConnection getInstance() {
        if (instance == null) {
            instance = new DatabaseConnection();
        }
        return instance;
    }

    public Connection getConnection() {
        return connection;
    }

    // Verify Connection
    public boolean isConnected(){
        if (connection != null){ // Connection exisits if not null
            return true;
        }
        else{
```

```
            return false;
        }
    }

    // Execute a DDL/DML statement (e.g., DROP TABLE, INSERT, UPDATE)
    public void executeUpdate(String query) throws SQLException {
        try (Statement statement = connection.createStatement()) {
            statement.executeUpdate(query); // Use executeUpdate for DDL/DML
        }
    }

    // Execute a SELECT query and return the ResultSet
    public ResultSet executeQuery(String query) throws SQLException {
        Statement statement = connection.createStatement();
        return statement.executeQuery(query); // Caller is responsible for closing the
ResultSet
    }

    // Commit the transaction
    public void commit() throws SQLException {
        if (connection != null) {
            connection.commit();
        }
    }

    // Rollback the transaction
    public void rollback() throws SQLException {
        if (connection != null) {
            connection.rollback();
        }
    }

    // Close the connection and clean up resources
    public void close() {
        try {
            if (connection != null && !connection.isClosed()) {
                connection.close();
                System.out.println("Connection closed.");
            }
        } catch (SQLException e) {
            System.err.println("Failed to close the connection.");
            e.printStackTrace();
        }
    }
}
```

MainGUI.java

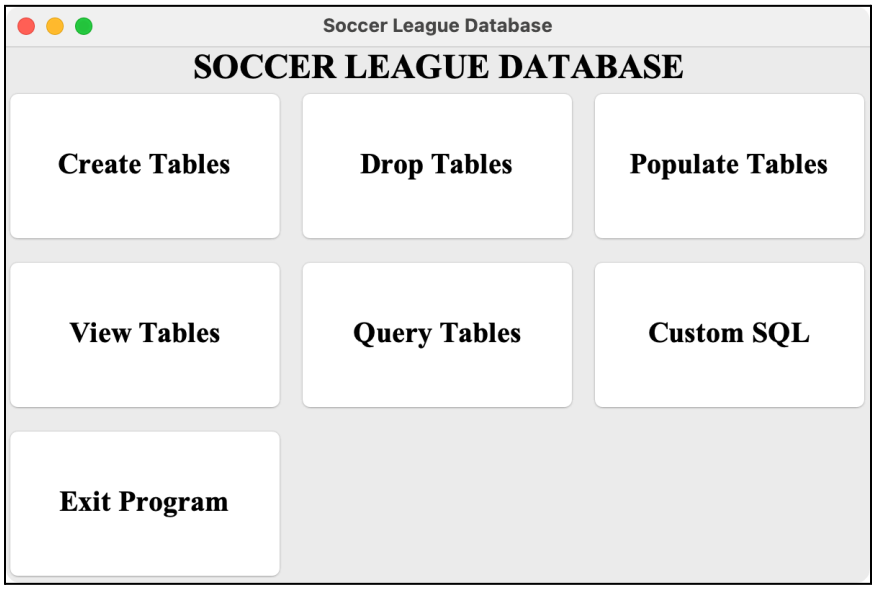
```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
```

```
import java.sql.SQLException;
import java.util.Vector;

public class MainGUI extends JFrame {
    private CardLayout cardLayout;
    private JPanel mainPanel;

    public MainGUI() {
        setTitle("Soccer League Database");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600, 400);
        setLocationRelativeTo(null); // Center the window on the screen

        // Initialize CardLayout and mainPanel
        cardLayout = new CardLayout();
        mainPanel = new JPanel(cardLayout);

        // Add the main menu panel
        JPanel mainMenuPanel = createMainMenuPanel();
        mainPanel.add(mainMenuPanel, "Main Menu");
        mainPanel.setBackground(Color.BLUE);

        // Add other panels
        mainPanel.add(new CreateTables(this), "Create Tables");
        mainPanel.add(new DropTables(this), "Drop Tables");
        mainPanel.add(new PopulateTables(this), "Populate Tables");
        mainPanel.add(new ViewTables(this), "View Tables");
        mainPanel.add(new QueryTables(this), "Query Tables");
        mainPanel.add(new CustomSQL(this), "Custom SQL");

        add(mainPanel);
    }

    private JPanel createMainMenuPanel() {
        JPanel menuPanel = new JPanel(new BorderLayout());
        JLabel titleLabel = new JLabel("SOCCER LEAGUE DATABASE",
SwingConstants.CENTER);
        titleLabel.setFont(new Font("Times New Roman", Font.BOLD, 24));
        menuPanel.add(titleLabel, BorderLayout.NORTH);

        JPanel buttonPanel = new JPanel(new GridLayout(3, 2, 10, 10));
        Color LightBlue = new Color(173, 216, 230);

        JButton createButton = new JButton("Create Tables");
        JButton dropButton = new JButton("Drop Tables");
        JButton populateButton = new JButton("Populate Tables");
        JButton viewButton = new JButton("View Tables");
        JButton queryButton = new JButton("Query Tables");
        JButton SQLButton = new JButton("Custom SQL");
        JButton exitButton = new JButton("Exit Program");

        JButton[] buttons = {createButton, dropButton, populateButton, viewButton,
queryButton, SQLButton, exitButton};

        for (JButton button : buttons) {
```

```
            button.setFont(new Font("Times New Roman", Font.BOLD, 20));
            button.setBackground(LightBlue);
            buttonPanel.add(button);
        }

        // Add action listeners for navigation
        createButton.addActionListener(e -> cardLayout.show(mainPanel, "Create
Tables"));
        dropButton.addActionListener(e -> cardLayout.show(mainPanel, "Drop Tables"));
        populateButton.addActionListener(e -> cardLayout.show(mainPanel, "Populate
Tables"));
        viewButton.addActionListener(e -> cardLayout.show(mainPanel, "View Tables"));
        queryButton.addActionListener(e -> cardLayout.show(mainPanel, "Query
Tables"));
        SQLButton.addActionListener(e -> cardLayout.show(mainPanel, "Custom SQL"));
        exitButton.addActionListener(e -> System.exit(0));

        menuPanel.add(buttonPanel, BorderLayout.CENTER);
        return menuPanel;
    }

    public static void executeButtonActionEvent(JButton tableButton,
DatabaseConnection databaseConnection, String query) {
        tableButton.addActionListener(actionEvent -> {
            try {
                ResultSet queryResult = databaseConnection.executeQuery(query);
                JTable queryResultTable = new JTable(buildTableModel(queryResult));
                JOptionPane.showMessageDialog(null, new
JScrollPane(queryResultTable));
            }
            catch (SQLException e) {
                //JOptionPane.showMessageDialog(null, e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
                e.printStackTrace();
            }
        });
    }

    public static DefaultTableModel buildTableModel(ResultSet queryResult) throws
SQLException {

        ResultSetMetaData queryMetaData = queryResult.getMetaData();
        int columnCount = queryMetaData.getColumnCount();
        Vector<String> columnNames = new Vector<>();
        Vector<Vector<Object>> queryDataVector = new Vector<>();

        for (int columnNumber = 1; columnNumber <= columnCount; columnNumber++) {
            columnNames.add(queryMetaData.getColumnName(columnNumber));
        }

        while (queryResult.next()) {
            Vector<Object> tempDataVector = new Vector<>();
            for (int columnIndex = 1; columnIndex <= columnCount; columnIndex++) {
                tempDataVector.add(queryResult.getObject(columnIndex));
            }
```

```
            queryDataVector.add(tempDataVector);
        }
        return new DefaultTableModel(queryDataVector, columnNames);
    }


    // Method to switch back to the main menu
    public void showMainMenu() {
        cardLayout.show(mainPanel, "Main Menu");
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            MainGUI mainGUI = new MainGUI();
            mainGUI.setVisible(true);
        });
    }
}
```

Figure 9.0.a: Main Page of Soccer League Application (MainGUI.java)

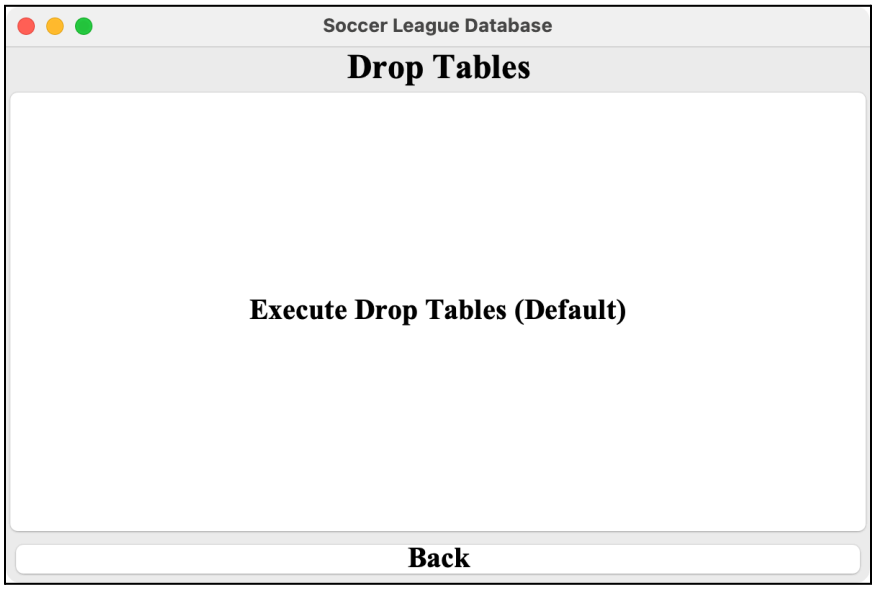


DropTables.java

```
import javax.swing.*;
import java.awt.*;
import java.sql.*;

public class DropTables extends JPanel {
    private final DatabaseConnection dbConnection;

    public DropTables(MainGUI mainGUI) {
        this.dbConnection = DatabaseConnection.getInstance(); // Use the singleton
instance
        setLayout(new BorderLayout());

        JLabel title = new JLabel("Drop Tables", SwingConstants.CENTER);
```

```
        title.setFont(new Font("Times New Roman", Font.BOLD, 24));
        add(title, BorderLayout.NORTH);

        JButton executeButton = new JButton("Execute Drop Tables (Default)");
        executeButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        executeButton.addActionListener(e -> {
            boolean allSuccessful = true;
            StringBuilder errors = new StringBuilder();

            try {
                String[] tables = { "score", "goal", "game", "coach", "player",
"team", "stadium" };

                for (String table : tables) {
                    try {
                        dbConnection.executeUpdate("DROP TABLE " + table);
                    } catch (SQLException ex) {
                        allSuccessful = false;
                        errors.append("Failed to drop table ").append(table).append(":
").append(ex.getMessage()).append("\n");
                    }
                }

                if (allSuccessful) {
                    dbConnection.commit();
                    JOptionPane.showMessageDialog(this, "Tables dropped
successfully!");
                } else {
                    dbConnection.rollback();
                    JOptionPane.showMessageDialog(this, "Some tables could not be
dropped:\n" + errors, "Partial Success", JOptionPane.WARNING_MESSAGE);
                }
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(this, "Unexpected error: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            }
        });
        add(executeButton, BorderLayout.CENTER);

        JButton backButton = new JButton("Back");
        backButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        backButton.addActionListener(e -> mainGUI.showMainMenu());
        add(backButton, BorderLayout.SOUTH);
    }
}
```

Figure 9.0.b: Drop Tables Page of Soccer League Application (DropTables.java)

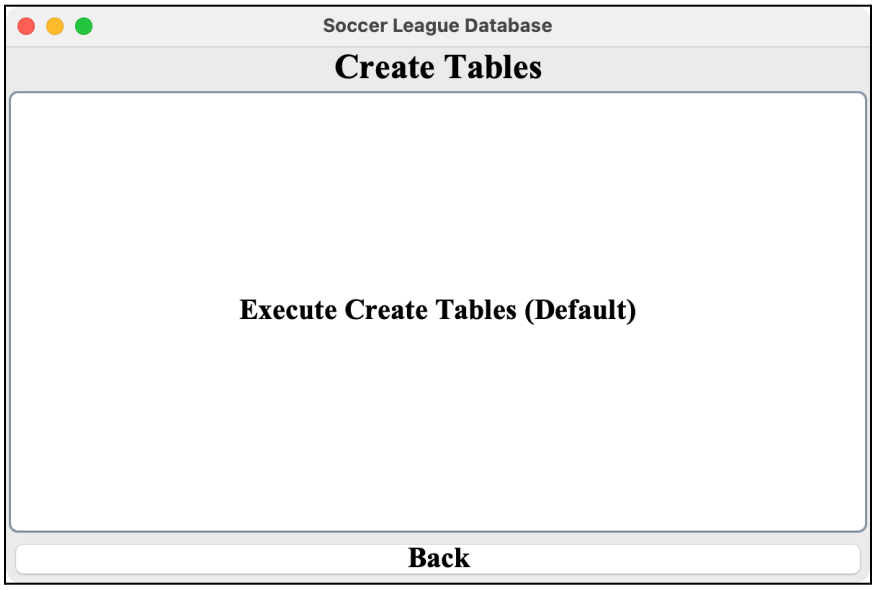


CreateTables.java

```
import javax.swing.*;
import java.awt.*;
import java.sql.*;

public class CreateTables extends JPanel {
    private final DatabaseConnection dbConnection;

    public CreateTables(MainGUI mainGUI) {
        this.dbConnection = DatabaseConnection.getInstance(); // Use the singleton
instance
        setLayout(new BorderLayout());

        JLabel title = new JLabel("Create Tables", SwingConstants.CENTER);
        title.setFont(new Font("Times New Roman", Font.BOLD, 24));
        add(title, BorderLayout.NORTH);

        JButton executeButton = new JButton("Execute Create Tables (Default)");
        executeButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        executeButton.addActionListener(e -> {
            boolean allSuccessful = true;
            StringBuilder errors = new StringBuilder();

            try {
                // Get the current connection object from dbConnection
                Connection conn = dbConnection.getConnection();

                try {
                    // Table creation statements
                    dbConnection.executeUpdate("  Insert SQL statements from Figure 3.0  ");
                    // Commit the transaction if all tables are created successfully
                    conn.commit();
```

```
                    JOptionPane.showMessageDialog(this, "Tables created
successfully!");

                } catch (SQLException ex) {
                    allSuccessful = false;
                    conn.rollback(); // Rollback the transaction in case of error
                    errors.append("Failed to create table:
").append(ex.getMessage()).append("\n");
                    JOptionPane.showMessageDialog(this, "Some tables could not be
created:\n" + errors, "Error", JOptionPane.ERROR_MESSAGE);
                }

            } catch (SQLException ex) {
                JOptionPane.showMessageDialog(this, "Error with database connection: "
+ ex.getMessage(), "Database Error", JOptionPane.ERROR_MESSAGE);
            }
        });
        add(executeButton, BorderLayout.CENTER);

        JButton backButton = new JButton("Back");
        backButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        backButton.addActionListener(e -> mainGUI.showMainMenu());
        add(backButton, BorderLayout.SOUTH);
    }
}
```

Figure 9.0.c: Create Tables Page of Soccer League Application (CreateTables.java)

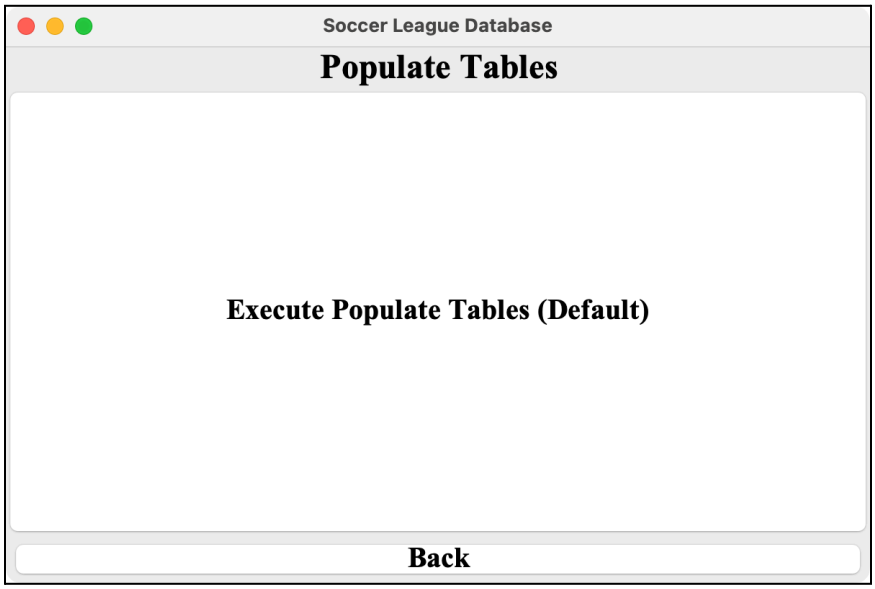


PopulateTables.java

```
import javax.swing.*;
import java.awt.*;
import java.sql.*;

public class PopulateTables extends JPanel {
    private final DatabaseConnection dbConnection;
```

```
    public PopulateTables(MainGUI mainGUI) {
        this.dbConnection = DatabaseConnection.getInstance(); // Use the singleton
instance
        setLayout(new BorderLayout());

        JLabel title = new JLabel("Populate Tables", SwingConstants.CENTER);
        title.setFont(new Font("Times New Roman", Font.BOLD, 24));
        add(title, BorderLayout.NORTH);

        JButton executeButton = new JButton("Execute Populate Tables (Default)");
        executeButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        executeButton.addActionListener(e -> {
            boolean allSuccessful = true;
            StringBuilder errors = new StringBuilder();

            try {
                // Get the current connection object from dbConnection
                Connection conn = dbConnection.getConnection();

                try {
                    // Populating Stadium Table
                    dbConnection.executeUpdate(" Insert SQL statements from Figure 4.0 ");

                    conn.commit();
                    JOptionPane.showMessageDialog(this, "Tables populated
successfully!");
                } catch (SQLException ex) {
                    allSuccessful = false;
                    conn.rollback(); // Rollback the transaction in case of error
                    errors.append("Failed to populate table:
").append(ex.getMessage()).append("\n");
                    JOptionPane.showMessageDialog(this, "Some tables could not be
populated:\n" + errors, "Error", JOptionPane.ERROR_MESSAGE);
                    ex.printStackTrace();
                }

            } catch (SQLException ex) {
                JOptionPane.showMessageDialog(this, "Error with database connection: "
+ ex.getMessage(), "Database Error", JOptionPane.ERROR_MESSAGE);
            }
        });
        add(executeButton, BorderLayout.CENTER);

        JButton backButton = new JButton("Back");
        backButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        backButton.addActionListener(e -> mainGUI.showMainMenu());
        add(backButton, BorderLayout.SOUTH);
    }
}
```

Figure 9.0.d: Populate Tables Page of Soccer League Application (PopulateTables.java)

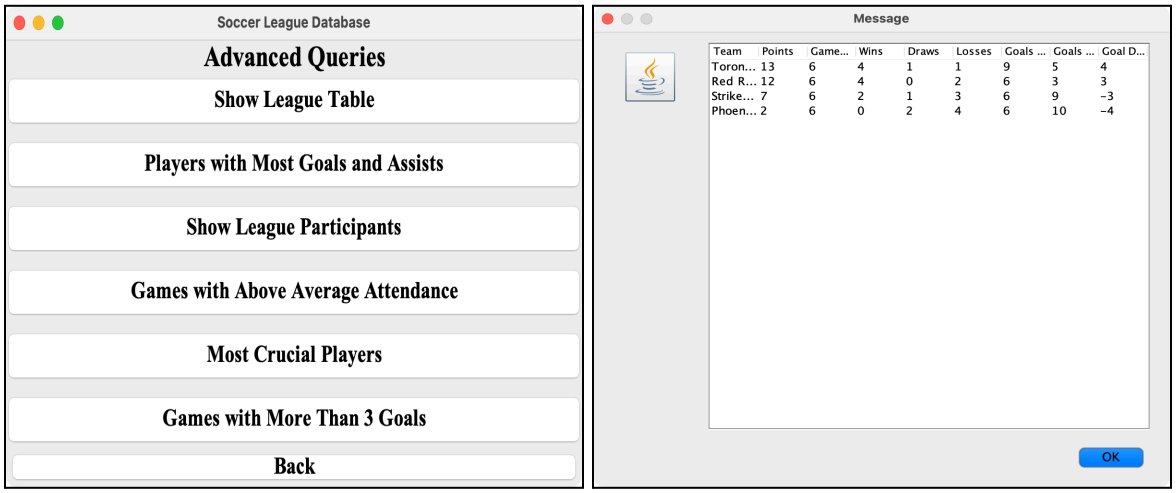


QueryTables.java

```
import javax.swing.*;
import java.awt.*;

public class QueryTables extends JPanel {
    private DatabaseConnection dbConnection;

    public QueryTables(MainGUI mainGUI) {
        this.dbConnection = DatabaseConnection.getInstance(); // Use the singleton
instance
        setLayout(new BorderLayout());

        JLabel title = new JLabel("Advanced Queries", SwingConstants.CENTER);
        title.setFont(new Font("Times New Roman", Font.BOLD, 24));
        add(title, BorderLayout.NORTH);

        //Panel for buttons
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(0, 1, 10, 10)); // 1 column, multiple
rows with spacing

        // Button to show League Table
        JButton leagueTableButton = new JButton("Show League Table");
        leagueTableButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryLeagueTable = " Insert SQL Query 9 from Figure 4.2 ";
        MainGUI.executeButtonActionEvent(leagueTableButton, dbConnection,
queryLeagueTable);
        buttonPanel.add(leagueTableButton);

        // Button to show players with most goals and assists
        JButton mostGoalsAndAssistsButton = new JButton("Players with Most Goals and
Assists");
        mostGoalsAndAssistsButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
```

```
        String queryGoalsAssists = " Insert SQL Query 12 from Figure 5.0 ";
        MainGUI.executeButtonActionEvent(mostGoalsAndAssistsButton, dbConnection,
queryGoalsAssists);
        buttonPanel.add(mostGoalsAndAssistsButton);

        // Button to show all participants in a league (Players and Coaches)
        JButton participantsButton = new JButton("Show League Participants");
        participantsButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryParticipants = " Insert SQL Query 13 from Figure 5.0 ";
        MainGUI.executeButtonActionEvent(participantsButton, dbConnection,
queryParticipants);
        buttonPanel.add(participantsButton);

        // Button to show games with attendance above average
        JButton gamesAboveAverageButton = new JButton("Games with Above Average
Attendance");
        gamesAboveAverageButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryGamesAboveAvg = " Insert SQL Query 14 from Figure 5.0 ";
        MainGUI.executeButtonActionEvent(gamesAboveAverageButton, dbConnection,
queryGamesAboveAvg);
        buttonPanel.add(gamesAboveAverageButton);

        // Button to show most crucial players
        JButton crucialPlayersButton = new JButton("Most Crucial Players");
        crucialPlayersButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryCrucialPlayers = " Insert SQL Query 15 from Figure 5.0 ";
        MainGUI.executeButtonActionEvent(crucialPlayersButton, dbConnection,
queryCrucialPlayers);
        buttonPanel.add(crucialPlayersButton);

        // Button to show games with more than 3 goals
        JButton gamesMoreThan3GoalsButton = new JButton("Games with More Than 3
Goals");
        gamesMoreThan3GoalsButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryGamesMoreThan3Goals = " Insert SQL Query 16 from Figure 5.0 ";
        MainGUI.executeButtonActionEvent(gamesMoreThan3GoalsButton, dbConnection,
queryGamesMoreThan3Goals);
        buttonPanel.add(gamesMoreThan3GoalsButton);

        // Adding all buttons to the center
        add(buttonPanel, BorderLayout.CENTER);

        JButton backButton = new JButton("Back");
        backButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        backButton.addActionListener(e -> mainGUI.showMainMenu());
        add(backButton, BorderLayout.SOUTH);
    }
}
```

Figure 9.0.e: Query Page of Soccer League Application (QueryTables.java)

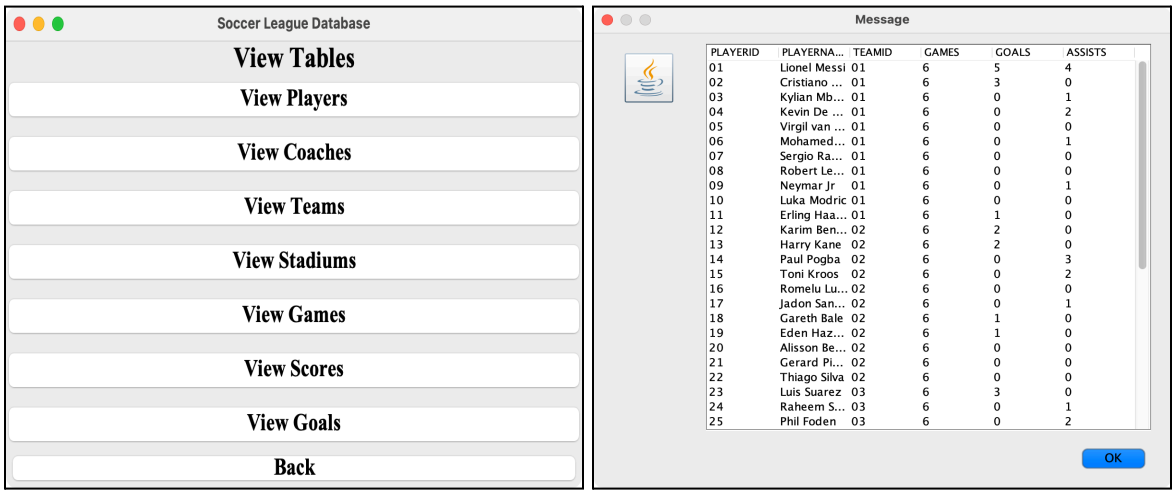


ViewTables.java

```
import javax.swing.*;
import java.awt.*;

public class ViewTables extends JPanel {
    private DatabaseConnection dbConnection;

    public ViewTables(MainGUI mainGUI) {
        this.dbConnection = DatabaseConnection.getInstance(); // Use the singleton
instance
        setLayout(new BorderLayout());
        JLabel title = new JLabel("View Tables", SwingConstants.CENTER);
        title.setFont(new Font("Times New Roman", Font.BOLD, 24));
        add(title, BorderLayout.NORTH);

        //Panel for buttons
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(0, 1, 10, 10)); // 1 column, multiple
rows with spacing

        // Button to show PLAYER table
        JButton viewPlayerButton = new JButton("View Players");
        viewPlayerButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryPlayers = "SELECT * FROM PLAYER";
        MainGUI.executeButtonActionEvent(viewPlayerButton, dbConnection,
queryPlayers);
        buttonPanel.add(viewPlayerButton);

        // Button to show COACH table
        JButton viewCoachButton = new JButton("View Coaches");
        viewCoachButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryCoaches = "SELECT * FROM COACH";
        MainGUI.executeButtonActionEvent(viewCoachButton, dbConnection, queryCoaches);
        buttonPanel.add(viewCoachButton);

        // Button to show TEAM table
```

```
        JButton viewTeamButton = new JButton("View Teams");
        viewTeamButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryTeams = "SELECT * FROM TEAM";
        MainGUI.executeButtonActionEvent(viewTeamButton, dbConnection, queryTeams);
        buttonPanel.add(viewTeamButton);

        // Button to show STADIUM table
        JButton viewStadiumButton = new JButton("View Stadiums");
        viewStadiumButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryStadiums = "SELECT * FROM STADIUM";
        MainGUI.executeButtonActionEvent(viewStadiumButton, dbConnection,
queryStadiums);
        buttonPanel.add(viewStadiumButton);

        // Button to show GAME table
        JButton viewGameButton = new JButton("View Games");
        viewGameButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryGames = "SELECT * FROM GAME";
        MainGUI.executeButtonActionEvent(viewGameButton, dbConnection, queryGames);
        buttonPanel.add(viewGameButton);

        // Button to show SCORE table
        JButton viewScoreButton = new JButton("View Scores");
        viewScoreButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryScores = "SELECT * FROM SCORE";
        MainGUI.executeButtonActionEvent(viewScoreButton, dbConnection, queryScores);
        buttonPanel.add(viewScoreButton);

        // Button to show GOAL table
        JButton viewGoalButton = new JButton("View Goals");
        viewGoalButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        String queryGoals = "SELECT * FROM GOAL";
        MainGUI.executeButtonActionEvent(viewGoalButton, dbConnection, queryGoals);
        buttonPanel.add(viewGoalButton);

        //Adding all buttons to the center
        add(buttonPanel, BorderLayout.CENTER);

        JButton backButton = new JButton("Back");
        backButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        backButton.addActionListener(e -> mainGUI.showMainMenu());
        add(backButton, BorderLayout.SOUTH);
    }
}
```

Figure 9.0.f: View Tables Page of Soccer League Application (ViewTables.java)



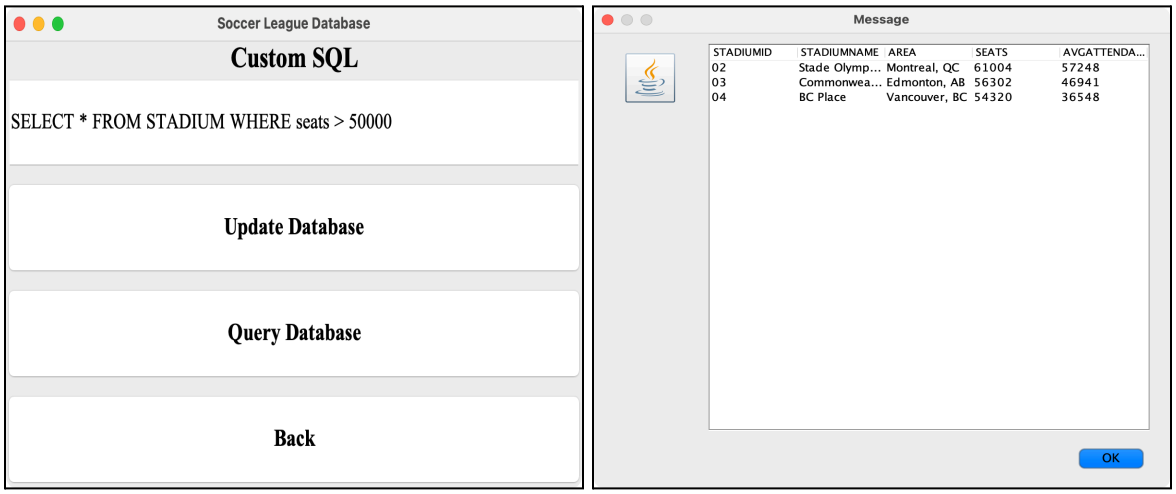
CustomSQL.java

```
import javax.swing.*;
import java.awt.*;
import java.sql.ResultSet;
import java.sql.SQLException;

public class CustomSQL extends JPanel {
    private final DatabaseConnection dbConnection;

    public CustomSQL(MainGUI mainGUI) {
        this.dbConnection = DatabaseConnection.getInstance(); // Use the singleton
instance
        setLayout(new BorderLayout());

        JLabel title = new JLabel("Custom SQL", SwingConstants.CENTER);
        title.setFont(new Font("Times New Roman", Font.BOLD, 24));
        add(title, BorderLayout.NORTH);

        // Panel for page elements
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(0, 1, 10, 10)); // 1 column, multiple rows with
spacing

        // Create a JTextField for user input
        JTextField sqlText = new JTextField();
        sqlText.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        panel.add(sqlText);

        // Add the button to Update DB
        JButton executeButton = new JButton("Update Database");
        executeButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        panel.add(executeButton);

        // Add the button to Update DB
        JButton queryButton = new JButton("Query Database");
```

```
        queryButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        panel.add(queryButton);

        // Add a "Back" button
        JButton backButton = new JButton("Back");
        backButton.setFont(new Font("Times New Roman", Font.BOLD, 20));
        panel.add(backButton);

        add(panel, BorderLayout.CENTER);

        // Action listener for the "Execute" button
        executeButton.addActionListener(e -> {
            String query = sqlText.getText().trim();
            if (query.isEmpty()) {
                JOptionPane.showMessageDialog(this, "SQL statement cannot be empty.",
"Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            try {
                    // Execute non-SELECT SQL statements
                    dbConnection.executeUpdate(query);
                    JOptionPane.showMessageDialog(this, "SQL executed successfully.",
"Success", JOptionPane.INFORMATION_MESSAGE);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(this, "Unexpected error: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                ex.printStackTrace();
            }
        });

        queryButton.addActionListener(e -> {
            String query = sqlText.getText().trim();
            if (query.isEmpty()) {
                JOptionPane.showMessageDialog(this, "SQL statement cannot be empty.",
"Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            if (!query.toLowerCase().startsWith("select")) {
                JOptionPane.showMessageDialog(this, "Only SELECT queries are allowed
for querying.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            try {
                ResultSet queryResult = dbConnection.executeQuery(query);
                JTable queryResultTable = new
JTable(MainGUI.buildTableModel(queryResult));
                JOptionPane.showMessageDialog(this, new
JScrollPane(queryResultTable));
            } catch (SQLException ex) {
                JOptionPane.showMessageDialog(this, "Unexpected error: " +
ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                ex.printStackTrace();
```

```
            }
        });

        // Action listener for the "Back" button
        backButton.addActionListener(e -> mainGUI.showMainMenu());
    }
}
```

Figure 9.0.g: Custom Query/DB Update Page of Soccer League Application (CustomSQL.java)



10 - QUERIES IN RELATIONAL ALGEBRA

As part of this report, most queries used in this lab were written in relational algebra notation. Only queries that could be converted into relational algebraic expressions within the scope of this course are included. Refer to the corresponding query numbers to relate the relational algebra notation with the SQL.

Query 1: STADIUM - Show all stadiums with 50000+ Capacity

Relational Algebraic Expression:

$$\Pi_{stadiumname,\, seats}(\sigma_{seats > 50000}(STADIUM))$$

Query 2: TEAM - Select all teams and show their stadiums

Relational Algebraic Expression:

$$\Pi_{teamname,\, stadiumname}(TEAM \bowtie_{t.stadiumid = s.stadiumid} STADIUM)$$

Query 3: PLAYER - Show the top goal-scorers

Relational Algebraic Expression:

$$\Pi_{playername,\, goals}(\sigma_{goals > 0}(PLAYER))$$

Query 4: COACH - Show coach and the team they manage

Relational Algebraic Expression:

$$\Pi_{coachname,\, teamname}(COACH \bowtie_{c.teamid = t.teamid} TEAM)$$

Query 5: GAME - Show the teams with the most goals at home

Relational Algebraic Expression:

$$_{teamname}F_{SUM(homegoals)}(GAME \bowtie_{g.hometeam = t.teamid} TEAM)$$

Query 6: GOAL - Show the total number of goals scored in the league

Relational Algebraic Expression:

$$F_{COUNT(goals)}(GOAL)$$

Query 7: SCORE - Show all the winning results of Toronto FC

Relational Algebraic Expression:

$$\Pi_{sc.gameid,\, score,\, teamname}(\sigma_{t.teamname = 'Toronto\ FC'}(SCORE \bowtie_{sc.winner = t.teamid} TEAM))$$

Query 8: TOP PERFORMERS - Show the best performing players based on Goals/Assists

Relational Algebraic Expression:

$$\Pi_{t.teamname,\, p.playername,\, p.goals,\, p.assist,\, (p.goals + p.assists)}(\sigma_{p.goals > 0\ OR\ p.assists > 0}(PLAYER \bowtie_{p.teamid = t.teamid} TEAM))$$

Query 9: LEAGUE TABLE - Show the current league standings based on points

Relational Algebraic Expression:

$$_{teamname}F_{SUM('Points'),\ COUNT('Wins'),\ COUNT('Draws'),\ COUNT('Losses'),\ COUNT('Goals\ Scored'),\ COUNT('Goals\ Conceded'),\ 'Goals\ Scored' - 'Goals\ Conceded'}(GAME \bowtie_{g.hometeam = t.teamid\ OR\ g.awayteam = teamid} TEAM)$$

Query 10: MESSI_LOG - Show the games messi played in and the results

Relational Algebraic Expression:

$$\Pi_{m.gameid,\ m.hometeam\ OR\ m.awaytem,\ s.score}(GAME\ m \bowtie_{m.gameid = s.gameid} SCORE\ s,\ GAME \bowtie_{m.hometeam = t1.teamid} TEAM\ t1 \bowtie_{m.awayteam = t2.teamid} TEAM\ t2 \bowtie_{m.hometeam = p.teamid\ OR\ m.awayteam = p.teamid} PLAYER\ p)$$

Query 11: STADIUM_AVG - Show the attendance average for each stadium in each team

Relational Algebraic Expression:

$$_{t.teamname,\ s.stadiumname}F_{ROUND(AVG(m.attendance,\ 2))}(GAME\ m \bowtie_{m.stadiumid = s.stadiumid} STADIUM\ s,\ GAME\ m \bowtie_{m.hometeam = t.teamid} TEAM\ t)$$

Query 12: Show the players with the most goals and assists

Relational Algebraic Expression:

$$R1 \leftarrow \Pi_{p.playerid}(\sigma\ (GOAL \bowtie_{g.scorer = p.playerid} PLAYER))$$

$$R2 \leftarrow \Pi_{p.playerid}(\sigma\ (GOAL \bowtie_{g.assister = p.playerid} PLAYER))$$

$$R3\ =\ R1\ \cup\ R2$$

$$\Pi_{playername,\ goals,\ assists}(PLAYER\ p \bowtie_{p.playerid = R3.playerid} R3)$$

Query 13: Show all participants in a league: players and coaches

Relational Algebraic Expression:

$$R1 \leftarrow \Pi_{t.teamid,\ p.playername \rightarrow "Name",\ 'player' \rightarrow "Role"}(PLAYER \bowtie_{p.teamid = t.teamid} TEAM)$$

$$R2 \leftarrow \Pi_{t.teamid,\ c.coachname \rightarrow "Name",\ 'coach' \rightarrow "Role"}(COACH \bowtie_{c.teamid = t.teamid} TEAM)$$

$$R3\ =\ R1\ \cup\ R2$$

$$\Pi_{teamname,\ Name,\ Role}(R3)$$

Query 14:: Show the games where attendance was above average

Relational Algebraic Expression:

$$R1 \leftarrow \Pi_{m.gameid,\ t1.teamname \rightarrow "Home\ Team",\ t2.teamname \rightarrow "Away\ Team",\ s.stadiumname \rightarrow "Stadium",\ m.attendance}(GAME\ m \bowtie_{m.hometeam = t1.teamid} TEAM\ t1 \bowtie_{m.awayteam = t2.teamid} TEAM\ t2 \bowtie_{m.stadium = s.stadiumid} STADIUM\ s)$$

$$R2 \leftarrow \Pi_{m.gameid,\ t1.teamname \rightarrow "Home\ Team",\ t2.teamname \rightarrow "Away\ Team",\ s.stadiumname \rightarrow "Stadium",\ m.attendance}($$

$$\sigma_{m.attendance > s.avgattendance}$$

$$(GAME\ m \bowtie_{m.hometeam = t1.teamid} TEAM\ t1 \bowtie_{m.awayteam = t2.teamid} TEAM\ t2 \bowtie_{m.stadium = s.stadiumid} STADIUM\ s))$$

R3 = R1 - R2

$$\Pi_{gameid,\ "Home\ Team",\ "Away\ Team",\ "Stadium",\ attendance}(R3)$$

11 - USAGE INSTRUCTIONS

The application can be used by anyone with a connection to the Oracle 11g Database. The following instructions explain how users can interact with the Soccer League Database.

1. Download and unzip the compressed file: “SoccerDB_GUI.zip”
2. After unzipping the file, go to the src (Source Code) file
3. In section 9, the Java code for all classes is posted. In the actual code, go to each class and replace the highlighted text with you Oracle username and password (What you use to access Oracle DB)
4. Make sure you're able to connect to Oracle DB from your local computer. You may see error messages on the IDE terminal you’re using when running “DatabaseConnection.java”
5. Once there are no error messages from running DatabaseConnection.java, run Main.java
6. A window with 7 buttons should pop if. You can now access the Soccer League Database or even add your own data by clicking the “Custom SQL” button
7. Once done, press the exit button or close the window. (*NOTE: all changes to DB are committed)

12 - CONCLUSION

The Soccer League Database application successfully maintained a league system while storing, updating and providing data regarding player performance, match attendance, and game statistics. Additionally, it allowed users to update the database using their own SQL commands. The final result was the culmination of 12 weeks, each with a component of the project that required completion.

Overall, this project extensively reinforced integral concepts in relational database management such as data modelling, schema design, normalization, and SQL implementation. These are valuable concepts that could be applied to the real-world when working with relational databases in the future. Combining the teachings of this course with a passion for sports shared among all group members led to the design of a successful database application.