Homework 3: SQL

7 March 2003
Due: 14 March 2003
To be done individually!!

1.0 Introduction

In this assignment, you will write SQL queries that answer questions about a database containing information about players, teams and games from the National Basketball Association (NBA). Yahoo Sports (http://sports.yahoo.com/nba/) and ESPN (http://sports.espn.go.com/nba/) are examples of web sites that use such a database. We have simplified the schema by maintaining information for only the *current season*[1].

Some notes to keep in mind as you go:

- While we have tried to limit the possible number of correct solutions to each question, keep in mind that there still may be multiple correct solutions. For each of the questions, there is a relatively short solution.
- Unless specified in the question, your queries should produce the correct answer for any instance of the relations. We have provided a script that will contain DDL for creating a sample database and some test data, but your queries should work over any legitimate instance of the relation.
- Unless specified in the question, return the entire tuple. E.g. "all games" would mean return the entire Game tuple.
- Unless specified in the question, do not remove duplicates.
- Return the fields in the order we specify. E.g. question 2 would require you to output the fields in *Player.position, average_height, average_weight* order.

The scripts are available /home/ff/cs186/sp03/Hw3/Hw3.tar.gz. Copy the tar ball over to your own directory, apply `gtar zxvf Hw3.tar.gz`. We have provided the following scripts:

Script	**Description**
initdb.sh	Initialize database directory at $PGDATA_HW3. To reinitialize, you have to delete existing $PGDATA_HW3 first.
startpg.sh	Starts Postgres master process for database directory at $PGDATA_HW3
loadnba.sh loadbignba.sh	Creates database 'hw3', and loads in some test data. The test data from loadbignba.sh is based on real data from this year's NBA roster. We have also provided a smaller data set that you can load using loadnba.sh. Feel free to insert in your own data for your own testing. Read loaddata.sql to see how to use the *copy* command to bulk load data.

[1] As a fun exercise, think about how to extend the schema we provide to support multiple seasons.

startpsql.sh	Starts psql for hw3
runquery.sh <QUERYFILE>	Runs query stored in <QUERYFILE> using psql and outputs to screen. We have provided a sample query file query0.sql.
stoppg.sh	Stops Postgres master process for the database directory at $PGDATA_HW3 before you log off.

To set up the initial database for the very first time, you should run initdb.sh, followed by startpg.sh and loadnba.sh. Some of these scripts are simply there for convenience to ensure that you run our version of pg_ctl, psql, initdb, createdb, etc and not your compiled version for hw2. You can choose not to use our scripts (at your own risk!) and create your own database (using our schema.sql) as you may have done in hw0.

You are strongly encouraged to read our DDL (schema.sql), and draw the ER diagram as practice for the midterm!

2.0 Schema

There are 4 relations in the schema, which are described below along with their integrity constraints. Columns in the primary key are underlined.

Player (playerID: integer, name : varchar(50), position : varchar(10), height : integer, weight : integer, team: varchar(30))
Each `Player` is assigned a unique *playerID*. The position of a player can either be *Guard*, *Center* or *Forward*. The height of a player is in inches while the weight is in pounds. Each player plays for only one team. The *team* field is a foreign key to Team.

Team (name: varchar(30), city : varchar(20))
Each `Team` has a unique name associated with it. There can be multiple teams from the same city.

Game (gameID: integer, homeTeam: varchar(30), awayTeam : varchar(30), homeScore : integer, awayScore : integer)
Each `Game` has a unique *gameID*. The fields *homeTeam* and *awayTeam* are foreign keys to Team. Two teams may play each other multiple times each season. There is a check constraint to ensure that homeTeam and awayTeam are different.

GameStats (playerID : integer, gameID: integer, points : integer, assists : integer, rebounds : integer)
`GameStats` records the performance statistics of a player within a game. A player may not play in every game, in which case it will not have its statistics recorded for that game. *gameID* is a foreign key to Game. *playerID* is a foreign key to Player. Assume that two assertions are in place. The first is to ensure that the player involved belongs to either the involving home or away teams, and the second is to ensure that the total score obtained

by a team recorded (in `Game`) is consistent with the total sum (in `GameStats`) of individual players in the team playing in the game[2].

3.0 Queries

Make sure you include the order by clause as stated in our queries. If your query output differs from ours because of the lack of the order by clause, you do not get any credit! Name the aggregate columns in the select list as we have indicated. E.g., in query 4, you can use 'avg(assists) AS average_assists' in your query to rename the aggregate column.

Each query is worth 1.5 points. The entire assignment is worth 15 points.

1. Find distinct names of players who play the "Guard" Position and have name containing the substring "Jordan". (ORDER BY Player.name)
    select list: Player.name
    ordering : Player.name ascending

2. For each different position, find the average height and average weight of players that play that position for the "ChicagoBulls". Output the position first followed by the averages. (ORDER BY Player.position)
    select list: Player.position, average_height, average_weight
    ordering: Player.position ascending

3. Find the team(s) with the most away wins compared to other teams. Output the team name and the number of away wins. (ORDER BY team name) If none of the teams have won any away games, do not output any rows.
    select list: Team.name, number_of_away_wins
    ordering: Team.name ascending

4. List all players' playerIDs and their average number of assists in all home games that they played in, ignoring players who did not play in any home games. (ORDER BY Player.playerID)
    select list: Player.playerID, average_assists
    ordering: Player playerID

5. List distinct cities that have more than 1 team playing there. (ORDER BY Team.city)
    select list: Team.city
    ordering: Team.city ascending

6. Find the tallest player(s) from each team. (ORDER BY Player.playerID)
    select list: Player.*
    ordering: Player.playerID ascending

[2] A good exercise for you (outside the scope of this assignment) is to figure out how to add in these checks.

7. List the playerIDs and names of players who have played in at least ten games so far during the season. Output the playerID and name followed by the number of games played. (ORDER BY Player.playerID)

select list: Player.playerID, Player.name, number_of_games
ordering: Player.playerID ascending

8. Find the number of games in which both "Larry Hughes" and "Michael Jordan" play, and Hughes scores more points than Jordan

select list: Number_of_games

9. List the playerIDs, names and teams of players who played in all away games for their team. Do not output a player if their team played 0 away games. (ORDER BY Player.playerID)

select list: Player.playerID, Player.name, Player.team
ordering: Player.playerID ascending

10. List players who have obtained at least one "triple double". For each of these players, output their playerID, name, team and the number of "triple doubles" they earned. (A "triple double" is a game in which the player's number of assists, rebounds, and points are all in the double-digit range). (ORDER BY Player.playerID)

select list: Player.playerID, Player.name, Player.team, number_of_3_2s
ordering: Player.playerID ascending

We have provided a sample query query0.sql that returns players' playerIDs and names, sorted in ascending order of name.

4.0 Submission instructions

Submit each of your queries inside a separate file. Query X should go into a file called 'queryX.sql'. E.g. query 1 goes into query1.sql, query 2 goes into query2.sql, and so on. You should be able to run './runquery.sh queryX.sql' to see the output of your query X. You are welcome to use view definitions within your query, as long as you ensure that runquery.sh command generates the correct output. This output will be used to be compared with our solutions to see if your query is written correctly.

You should submit the following files:
query[1,2,3,4,5,6,7,8,9,10].sql for each of your answers
README – Your name and login

Make sure all the above files are in a directory called Project3/. `cd` into that directory and type `submit Project3` to submit the files.