CSE 462 Homework #3 (Optional): Relational Algebra and SQL

Name: _____ Date: March 05, 2011

***** Due on March 25, 2011 at the beginning of class. *****

Instructions. This problem set is optional and worth 200 points. Your answers **must be typed**. All problems reference the NBA player statistics schema below:

`Team`(`tId` : `int`, `name` : `string`, `homeCity` : `string`)

`Game`(`gId` : `int`, `homeId` : `int`, `awayId` : `int`, `isPlayoff` : `boolean`, `year` : `int`)

`Player`(`pId` : `int`, `lastName` : `string`, `firstName` : `string`, `height` : `int`, `tId` : `int`)

`Stats`(`gId` : `int`, `pId` : `int`, `points` : `int`, `rebounds` : `int`, `assists` : `int`, `seconds` : `int`)

Observations:

- Keys are underlined.
- No attributes allow `NULL`s.
- Table `Team` stores all NBA teams.
- Table `Player` stores every player of every team.
- Table `Game` stores every regular season and playoff game.
- Table `Stats` records individual player statistics for every season and playoff game.
- Attribute `homeId` in `Game` has a foreign key referencing `tId` in `Team`.
- Attribute `awayId` in `Game` has a foreign key referencing `tId` in `Team`.
- Attribute `tId` in `Player` has a foreign key to `tId` in `Team`.
- Attribute `gId` in `Stats` has a foreign key to `gId` in `Game`.
- Attribute `pId` in `Stats` has a foreign key to `pId` in `Player`.
- A player achieves a *double-double* in a game when he obtains a number total of 10 or more in exactly two out of the three statistics (points, rebounds, assists). He achieves a *triple-double* when he obtains a number total of 10 or more in all three statistics.
- **Important:** if a player does not play a game, there is no entry for *that* player in the `Stats` table for *that* game.

Problem 1. [80 points] Write relational algebra expressions to answer the problems below.

a) [10 points] Find the tallest player(s) of each team. List the respective `tId` and `pId` pairs.

$P := \pi_{tId,pId,height}(Player)$

$Bad := \pi_{tId,pId,height}(P \bowtie_{tId=tId2 \wedge height<h2} P(tId2, pId2, h2))$

$Answer := \pi_{tId,pId}(P - Bad)$

b) [20 points] Find player(s) who, for some year, did not play all regular season games neither all away games that his team played. List their respective `pId`s.

$AllAway := \pi_{year,pId,gId}(Game \bowtie_{tId=awayId} Player)$

$AllRegular := \pi_{year,pId,gId}(\sigma_{isPlayoff=FALSE}(Game \bowtie_{tId=homeId \vee tId=awayId} Player))$

$PlayedAway := \pi_{year,pId,gId}(AllAway \bowtie Stats)$

$PlayedRegular := \pi_{year,pId,gId}(AllRegular \bowtie Stats)$

$Answer := \pi_{year,pId}(AllAway - PlayedAway) \cap \pi_{year,pId}(AllRegular - PlayedRegular)$

c) [20 points] For every year, find the player(s) who scored more points than all other players in his team in every playoff game that he played. List the respective `year` and `pId` pairs.

$AllPlayoff := \pi_{year,tId,pId,gId,points}(\sigma_{isPlayoff=TRUE}(Game \bowtie_{tId=homeId \vee tId=awayId} Player \bowtie Stats))$

$LowScorers := \pi_{year,pId}(AllPlayoff \bowtie_{year=y2 \wedge tId=t2 \wedge gId=g2 \wedge points<pts2} AllPlayoff(y2, g2, t2, p2, pts2))$

$Answer := \pi_{year,pId}(AllPlayoff) - LowScorers$

d) [30 points] Find the player(s) who achieved at least two triple-doubles in playoff games in at least two consecutive years. List the respective `pId`, `lastName`, and `firstName`.

$PlayoffTD := \pi_{year,gId,pId}(\sigma_{points>=10 \wedge rebounds>=10 \wedge assists>=10 \wedge isPlayoff=TRUE}(Stats \bowtie Game))$

$TwicePlayoffTD := \pi_{year,pId}(PlayoffTD \bowtie_{year=y2 \wedge gId<>g2 \wedge pId=p2} PlayoffTD(y2, g2, p2)))$

$ConsecutiveTwicePlayoffTD := \pi_{year,pId}(TwicePlayoffTD \bowtie_{year=y2+1 \wedge pId=p2} TwicePlayoffTD(y2, p2)))$

$Answer := \pi_{pId,lastName,firstName}(ConsecutiveTwicePlayoffTD \bowtie Player)$

Problem 2. [120 points] Create SQL views to answer the problems below.

a) [10 points] View `RegularSeasonAverages(pId, avgPoints)` that computes, for every player of every team, the average points scored by that player for the games he played in the regular season. If a player played no regular season game, his average should appear as zero.

```
CREATE VIEW RegularSeasonAverages(pId, avgPoints) AS
  SELECT P.pId, AVG(COALESCE(S.points, 0.0))
  FROM Game G
       INNER JOIN Players P ON (G.homeId = P.tId OR G.awayId = P.tId)
       LEFT OUTER JOIN Stats S ON (G.gId = S.gId AND P.pId = S.pId)
  GROUP BY P.pId;
```

b) [10 points] Using the view in (a), create a view `MinMaxDelta(minAvg, maxAvg, delta)` that returns a single tuple, where `minAvg` and `maxAvg` are the minimum and maximum average points computed by the view in (a), and `delta = maxAvg - minAvg` is the difference between the largest and smallest average points.

```
CREATE VIEW MinMaxDelta(minAvg, maxAvg, delta) AS
  SELECT MIN(avgPoints), MAX(avgPoints), MAX(avgPoints) - MIN(avgPoints)
  FROM RegularSeasonAverages;
```

c) [10 points] You are given the task of "grading" all players according to their average scored points during the

regular season. Using the views in (a) and (b), create a view `RegularSeasonGrades(pId, grade)` that classifies each player, based on his average score `s`, as follows:

- if ($s \geq minAvg + 4 * delta$) then his grade is "A"
- if ($minAvg + 3 * delta \leq s < minAvg + 4 * delta$) then his grade is "B"
- if ($minAvg + 2 * delta \leq s < minAvg + 3 * delta$) then his grade is "C"
- if ($minAvg + 1 * delta \leq s < minAvg + 2 * delta$) then his grade is "D"
- if ($s < minAvg + 1 * delta$) then his grade is "F"

```
CREATE VIEW RegularSeasonGrades(pId, grade) AS
  SELECT pId, CASE
                WHEN avgPoints >= minAvg + 4*delta THEN 'A'
                WHEN avgPoints >= minAvg + 3*delta THEN 'B'
                WHEN avgPoints >= minAvg + 2*delta THEN 'C'
                WHEN avgPoints >= minAvg + 1*delta THEN 'D'
              ELSE 'F' END
  FROM RegularSeasonAverages RSA, MinMaxDelta MMD;
```

d) [20 points] Using the view in (c), create a view `Frequencies(grade, absFreq, relFreq)` that computes the absolute and relative frequencies of the grades of each player. The absolute frequency is the number of players having a particular grade and the relative frequency is the absolute frequency divided by the total number of players. Note that the sum of absolute frequencies must be equal to the total number of players and the sum of all relative frequencies must be equal to ONE.

```
CREATE VIEW Frequencies(grade, absFreq, relFreq) AS
  SELECT G1.grade,
         COUNT(DISTINCT G1.pId),
         1.0 * COUNT(DISTINCT G1.pId) / COUNT(DISTINCT G2.pId)
  FROM RegularSeasonGrades G1,
       RegularSeasonGrades G2
  GROUP BY G1.grade;
```

e) [20 points] View `GameScores(gId, homePoints, awayPoints)` that returns the final score of every game. *Hint:* create helper views `HomeScores(gId, homePoints)` and `AwayScores(gId, AwayPoints)` to compute the total number of points scored by the home and away teams in each game.

```
CREATE VIEW HomeScore(gId, homePoints) AS
  SELECT gId, SUM(points)
  FROM Stats NATURAL JOIN Game NATURAL JOIN Player
  WHERE tId = homeId
  GROUP BY gId;

CREATE VIEW AwayScore(gId, awayPoints) AS
  SELECT gId, SUM(points)
  FROM Stats NATURAL JOIN Game NATURAL JOIN Player
  WHERE tId = awayId
  GROUP BY gId;

CREATE VIEW GameScores(gId, homePoints, awayPoints) AS
  SELECT gId, homePoints, awayPoints
  FROM HomeScore NATURAL JOIN AwayScore;
```

f) [10 points] Using the view in (e), create a view `RepeatedScores(gId1, gId2)` that returns pairs of distinct `gIds` such that the respective games occurred in different years, had the same home and away teams, and also the same final score. If your query returns the pair (A,B), it must not return the pair (B,A).

```
CREATE VIEW RepeatedScores(gId1, gId2) AS
  SELECT G1.gId, G2.gId
  FROM
    GameScores AS GS1 NATURAL JOIN Game AS G1,
    GameScores AS GS2 NATURAL JOIN Game AS G2
  WHERE
    G1.gId < G2.gId AND G1.year <> G2.year AND
    G1.homeId = G2.homeId AND G1.awayId = G2.awayId AND
    GS1.homePoints = GS2.homePoints AND GS1.awayPoints = GS2.awayPoints;
```

g) [20 points] View `AwayYear(year)` that returns the years in which away teams won more often than home teams. *Hint:* create a helper view `AwayWins(year, wins)` that computes, for each year, the total number of games won by away teams.

```
CREATE VIEW AwayWins(year, wins) AS
  SELECT year, COUNT(*)
  FROM Game G NATURAL JOIN GameScores
  WHERE awayPoints > homePoints
  GROUP BY year;

CREATE VIEW GamesPerYear(year, games) AS
  SELECT year, COUNT(*)
  FROM Game
  GROUP BY year;

CREATE VIEW AwayYear(year) AS
  SELECT year
  FROM GamesPerYear NATURAL JOIN AwayWins WHERE games < 2*wins;
```

h) [20 points] View `ImprovingTeams(tId)` that returns the teams that, for each year, won more games in that year than in any preceding year. The first year that a team played satisfies the condition only if the team won at least one game in that year.

```
CREATE VIEW HomeTeamWins(year, tId, homeWins) AS
  SELECT year, homeId, COUNT(*)
  FROM Game AS G JOIN GameScore AS GS ON (GS.gId = G.gId)
  WHERE GS.homePoints > GS.awayPoints
  GROUP BY year, homeId;

CREATE VIEW AwayTeamWins(year, tId, awayWins) AS
  SELECT year, awayId, COUNT(*)
  FROM Game AS G JOIN GameScore AS GS ON (GS.gId = G.gId)
  WHERE GS.homePoints < GS.awayPoints
  GROUP BY year, awayId;

CREATE VIEW TeamWins(year, tId, wins) AS
  SELECT year, tId, homeWins + awayWins
  FROM AwayTeamWins NATURAL JOIN HomeTeamWins;

CREATE VIEW ImprovingTeams(tId) AS
  SELECT tId FROM TeamWins TW
  WHERE wins > 0 AND
        wins >= ALL (SELECT wins FROM TeamWins WHERE year < TW.year AND tId = TW.tId);
```