DSE 201 Final – Winter 2018

Problem 1 True or False (no justification required)? User-defined functions (UDFs) are not allowed in any of your solutions.

1. Consider a schema in which each pair of distinct tables has disjoint column names. Then every SQL query Q with aliases (tuple variables) over this schema can be reformulated to a query Q' without aliases, over the same schema, such that Q' always returns the same answer as Q on every input database.

   False

2. SELECT * FROM T WHERE T.A <= 39 OR T.A > 39 always returns the same result as SELECT * FROM T

   False

3. NATURAL LEFT JOIN is SQL-expressible without the JOIN keyword.

   True

4. SELECT DISTINCT T.A FROM T is SQL-expressible without the DIS-TINCT keyword

   True

5. SELECT MAX (R.A) FROM R can be expressed without the MAX built-in aggregate, ORDER BY, LIMIT, TOP K, WINDOW and without UDFs.

   True

6. Let R(<u>A</u>,B) and S(<u>B</u>,C) be tables whose underlined attributes are primary keys. Attribute R.B is not null, and it is a foreign key referencing S.
   SELECT r.A FROM R r, S s WHERE r.B = s.B
   always returns the same answer as SELECT A FROM R.

   True

7. EXCEPT can be expressed in SQL without using the EXCEPT keyword or UDFs

   True

8. In SQL, all nested queries without correlated variables can be unnested (without creating views or auxiliary tables).

   False

9. Consider tables R(A,B) and S(A,B). Then
   SELECT A FROM (R UNION S)
   always returns the same result as
   (SELECT A FROM R) UNION (SELECT A FROM S).

   False

10. Let R(A,B) be a relation with primary key A and numeric, not-null B. Then
    SELECT A, MAX(B) FROM R GROUP BY A returns R

    True

Problem 2 In a soccer league each team has a home stadium, and each pair of teams faces each other twice during the season (once at the home stadium of each team). For a given match, the team whose stadium hosts the match is the home team, while the other team is the visitors team. We model information about the league using the schema
Teams (name, coach)
Matches (hTeam, vteam, hScore, vScore)
where name is the primary key for table Teams and coach is a candidate key for the same table. Attributes hTeam and vteam denote the home, respectively visitors team. They are foreign keys referencing the Teams table. Their value cannot be null. The pair hTeam, vteam is the primary key for table Matches. hScore/vScore denote the score of the home/visitors team, respectively. The Matches table refers only to completed matches, listing their final scores which are not null.
Express the following in SQL:

SCHEMA:

```
drop table if exists teams cascade;
CREATE TABLE teams
(
  name character varying(50) primary key NOT NULL,
  coach character varying(50) NOT NULL unique
);

drop table if exists matches cascade;
CREATE TABLE matches
(
  hteam character varying(50) references teams (name) NOT NULL,
  vteam character varying(50) references teams (name) NOT NULL,
  hscore int NOT NULL,
  vscore int NOT NULL,
  primary key (hteam,vteam)
);
```

(i) Count the victories of team ″San Diego Sockers″. Return a single column called ″wins″.

```
prepare Q1(character) as
select count(*) as wins from matches m
        where (m.hteam=$1 and m.hscore>m.vscore) or (m.vteam=$1 and
m.vscore>m.hscore);

Execute Q1('San Diego Sockers');
```

(ii) According to league rules, a defeat results in 0 points, a tie in 1 point, a victory at home in 2 points, and a victory away in 3 points.
For each team, return its name and total number of points earned. Output a table with two columns: name and points.

```
SELECT t.NAME,
       COALESCE(f.points, 0) AS points
FROM   teams t
       LEFT JOIN (SELECT u.team            AS team,
                         Sum(u.win_points) AS points
                  FROM   (SELECT h.hteam            AS team,
                                 Count(h.hteam) * 2 AS win_points
                          FROM   matches h
                          WHERE  h.hscore > h.vscore
                          GROUP  BY h.hteam
                          UNION ALL
                          SELECT v.vteam            AS team,
                                 Count(v.vteam) * 3 AS win_points
                          FROM   matches v
                          WHERE  v.vscore > v.hscore
```

```
                    GROUP  BY v.vteam
                    UNION ALL
                    SELECT h.hteam           AS team,
                           Count(h.hteam) * 1 AS win_points
                    FROM   matches h
                    WHERE  h.hscore = h.vscore
                    GROUP  BY h.hteam
                    UNION ALL
                    SELECT v.vteam           AS team,
                           Count(v.vteam) * 1 AS win_points
                    FROM   matches v
                    WHERE  v.vscore = v.hscore
                    GROUP  BY v.vteam) u
            GROUP  BY u.team) f
        ON t.NAME = f.team
```

(iii) Return the names of undefeated coaches (that is, coaches whose teams have lost no match). Output a table with a single column called "coach".

```
/*a team has played at least one game.*/
SELECT t.coach
FROM   teams t
WHERE  t.NAME NOT IN (SELECT h.hteam
                      FROM   matches h
                      WHERE  h.hscore < h.vscore)
       AND t.NAME NOT IN (SELECT v.vteam
                          FROM   matches v
                          WHERE  v.vscore < v.hscore)
       AND t.NAME IN (SELECT DISTINCT hteam
                      FROM   matches
                      UNION
                      SELECT DISTINCT vteam
                      FROM   matches)
```

(iv) Return the teams defeated only by the scoreboard leaders (i.e. "if defeated, then the winner is a leader"). The leaders are the teams with the highest number of points (several leaders can be tied). Output a single column called "name".

```
DROP VIEW IF EXISTS points_table;

CREATE VIEW points_table(name, points)
AS
```

```
  SELECT t.name,
       Coalesce(f.points, 0) AS points
  FROM   teams t
       LEFT JOIN (SELECT u.team           AS team,
                         Sum(u.win_points) AS points
                  FROM   (SELECT h.hteam           AS team,
                                 COUNT(h.hteam) * 2 AS win_points
                          FROM   matches h
                          WHERE  h.hscore > h.vscore
                          GROUP  BY h.hteam
                          UNION ALL
                          SELECT v.vteam           AS team,
                                 COUNT(v.vteam) * 3 AS win_points
                          FROM   matches v
                          WHERE  v.vscore > v.hscore
                          GROUP  BY v.vteam
                          UNION ALL
                          SELECT h.hteam           AS team,
                                 COUNT(h.hteam) * 1 AS win_points
                          FROM   matches h
                          WHERE  h.hscore = h.vscore
                          GROUP  BY h.hteam
                          UNION ALL
                          SELECT v.vteam           AS team,
                                 COUNT(v.vteam) * 1 AS win_points
                          FROM   matches v
                          WHERE  v.vscore = v.hscore
                          GROUP  BY v.vteam) u
                  GROUP  BY u.team) f
              ON t.name = f.team;

DROP VIEW IF EXISTS defeated_team_and_defeated_by_team;

CREATE VIEW defeated_team_and_defeated_by_team(defeated_team,
defeated_by_team)
AS
  SELECT h.hteam AS defeated_team,
       h.vteam AS defeated_by_team
  FROM   matches h
  WHERE  h.hscore < h.vscore
  UNION ALL
  SELECT v.vteam AS defeated_team,
       v.hteam AS defeated_by_team
  FROM   matches v
  WHERE  v.vscore < v.hscore;

SELECT C.defeated_team AS name
FROM   (SELECT DISTINCT A.defeated_team
       FROM   defeated_team_and_defeated_by_team A
       WHERE  A.defeated_by_team IN (SELECT p.name
                                     FROM   points_table p
                                     WHERE  p.points IN (SELECT Max(points)
                                                         FROM   points_table)))
      C
WHERE  C.defeated_team NOT IN (SELECT DISTINCT B.defeated_team
                               FROM   defeated_team_and_defeated_by_team B
                               WHERE  B.defeated_by_team NOT IN (SELECT p.name
                                                                 FROM
                                     points_table p
                                                                 WHERE
                                     p.points IN (SELECT Max(points)
                                                  FROM   points_table)))
```

(v) For each query in Problems (i) through (iv), create useful indexes or explain why there are none.

For each query in Problems (i) through (iv) None of the indexes can be created because
Matches.hscore and matches.vscore cannot be indexed since each values of hscore are compared with each values of vscore for less than or greater than or equality checks. Therefore sequential scan is preferred over index scan.

Teams.name, matches.hteam and matches.vteam are indexed by default since they are primary keys

(vi) Assume that the result of the query in Problem (ii) is materialized in a table called Scoreboard. Write triggers to keep the Scoreboard up to date when the Matches table is inserted into, respectively updated. The resulting Scoreboard updates should be incremental (i.e. do not recompute Scoreboard from scratch).

```
drop table if exists scoreboard cascade;
CREATE TABLE scoreboard
(
  name character varying(50) references teams (name) NOT NULL,
  points int NOT NULL
);

CREATE OR REPLACE FUNCTION scoreboard_update() RETURNS TRIGGER AS
$scoreboard$
   BEGIN
      IF (TG_OP = 'INSERT') THEN
      IF NEW.hteam NOT IN (SELECT scoreboard.name from scoreboard) THEN
            INSERT INTO scoreboard (name, points) VALUES (NEW.hteam, 0);
         END IF;
         IF NEW.vteam NOT IN (SELECT scoreboard.name from scoreboard) THEN
            INSERT INTO scoreboard (name, points) VALUES (NEW.vteam, 0);
                   END IF;
      IF NEW.hscore>NEW.vscore THEN
            UPDATE scoreboard SET points=points+2 where
scoreboard.name=NEW.hteam;
         ELSIF NEW.vscore>NEW.hscore THEN
            UPDATE scoreboard SET points=points+3 where
scoreboard.name=NEW.vteam;
         ELSIF NEW.vscore=NEW.hscore THEN
            UPDATE scoreboard SET points=points+1 where
scoreboard.name=NEW.hteam;
            UPDATE scoreboard SET points=points+1 where
scoreboard.name=NEW.vteam;
         END IF;

      ELSIF (TG_OP = 'UPDATE') THEN
      /*IF home team changed or visitior team changed or both changed or the
hscore changed or the vscore changed
            Then remove points on both both hteam and vteam based on win, lose
or draw*/
      IF OLD.hscore>OLD.vscore THEN UPDATE scoreboard SET points=points-2
where scoreboard.name=OLD.hteam;
         ELSIF OLD.hscore<OLD.vscore THEN UPDATE scoreboard SET
points=points-3 where scoreboard.name=OLD.vteam;
         ELSIF OLD.hscore=OLD.vscore THEN UPDATE scoreboard SET
points=points-1 where scoreboard.name=OLD.hteam;
                                               UPDATE scoreboard SET points=points-1
where scoreboard.name=OLD.vteam;
         END IF;
         /*After removing the points update the points table based on new data*/
         IF NEW.hteam NOT IN (SELECT scoreboard.name from scoreboard) THEN
            INSERT INTO scoreboard (name, points) VALUES (NEW.hteam, 0);
```

```
          END IF;
          IF NEW.vteam NOT IN (SELECT scoreboard.name from scoreboard) THEN
              INSERT INTO scoreboard (name, points) VALUES (NEW.vteam, 0);
                    END IF;
                    IF NEW.hscore>NEW.vscore THEN UPDATE scoreboard SET
points=points+2 where scoreboard.name=NEW.hteam;
          ELSIF NEW.vscore>NEW.hscore THEN UPDATE scoreboard SET
points=points+3 where scoreboard.name=NEW.vteam;
          ELSIF NEW.vscore=NEW.hscore THEN UPDATE scoreboard SET
points=points+1 where scoreboard.name=NEW.hteam;
                                                              UPDATE scoreboard SET
points=points+1 where scoreboard.name=NEW.vteam;
          END IF;
       END IF;
       RETURN NULL;
   END;
$scoreboard$ LANGUAGE plpgsql;

drop trigger if exists trigger_scoreboard_update on matches;
CREATE TRIGGER trigger_scoreboard_update
AFTER INSERT OR UPDATE ON matches
   FOR EACH ROW EXECUTE PROCEDURE scoreboard_update();


/*Query*/
Select * from scoreboard
```