Ryan Inghilterra - DSE 201 - Final Exam - Winter 2018

Problem 1:

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

consider the case where there exists a NULL in the table

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

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

can use group by on column A

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

```
SELECT * FROM test
SELECT DISTINCT a FROM test WHERE a NOT IN
(
  SELECT t1.a FROM test AS t1 JOIN test AS t2 ON t1.a < t2.a
)
```

6. Let R(A,B) and S(B,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

EXCEPT filters the DISTINCT values from the left-hand table that do not appear in the right-hand table. It's essentially the same as doing a NOT EXISTS with a DISTINCT clause.

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).
TRUE

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

DDL -

```
CREATE TABLE teams (
 name  TEXT PRIMARY KEY,
 coach TEXT UNIQUE
);

CREATE TABLE matches (
 hTeam  TEXT REFERENCES Teams (name) NOT NULL,
 vTeam  TEXT REFERENCES Teams (name) NOT NULL,
 hScore INT                  NOT NULL,
 vScore INT                  NOT NULL,
 PRIMARY KEY (hTeam, vTeam)
);
```

Express the following in SQL:

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

```
SELECT count(*) AS wins
FROM matches
WHERE (hteam = 'San Diego Sockers' AND hscore > vscore)
    OR (vteam = 'San Diego Sockers' AND hscore < vscore);
```

(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.

(note: used descending order by points because that is standard in most sports (leaders at top))

```
WITH homeWinPoints AS (
  SELECT
    hteam as name,
    COUNT(*) * 2 AS points
  FROM  matches m
  WHERE hscore > vscore
  GROUP BY hteam
),
  awayWinPoints AS (
    SELECT
      vteam as name,
      COUNT(*) * 3 AS points
    FROM matches m
    WHERE hscore < vscore
    GROUP BY vteam
),
  homeTiePoints AS (
    SELECT
      hteam    AS name,
      COUNT(*) AS points
    FROM matches
    WHERE hscore = vscore
    GROUP BY hteam
),
  awayTiePoints AS (
    SELECT
      vteam    AS name,
      COUNT(*) AS points
    FROM matches
    WHERE hscore = vscore
    GROUP BY vteam
),
 allPoints AS (
   SELECT *
   FROM awayWinPoints
   UNION ALL
   SELECT *
   FROM awayTiePoints
   UNION ALL
   SELECT *
```

```
  FROM homeWinPoints
  UNION ALL
  SELECT *
  FROM homeTiePoints
)

SELECT t.name as name, COALESCE(SUM(a.points),0) as points
FROM teams t LEFT JOIN allPoints a
ON t.name = a.name
GROUP BY t.name
ORDER BY points DESC;
```

(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”.

```
WITH homeLosses AS (
  SELECT
    hteam    AS name,
    COUNT(*) AS losses
  FROM matches
  WHERE hscore < vscore
  GROUP BY hteam
),

  awayLosses AS (
    SELECT
      vteam    AS name,
      COUNT(*) AS losses
    FROM matches
    WHERE hscore > vscore
    GROUP BY vteam
),

  totalLosses AS (
   SELECT *
   FROM homeLosses
   UNION ALL
   SELECT *
   FROM awayLosses

),

  totalLosses2 AS (
    SELECT
      t.name,
      t.coach,
      COALESCE(l.losses, 0) AS losses
    FROM teams t LEFT JOIN totalLosses l
       ON t.name = l.name
)

SELECT coach
FROM totalLosses2
WHERE losses = 0;
```

(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”.

```
WITH homeWinPoints AS (
  SELECT
    hteam       AS name,
    COUNT(*) * 2 AS points
  FROM matches m
  WHERE hscore > vscore
  GROUP BY hteam
),
  awayWinPoints AS (
    SELECT
      vteam       AS name,
      COUNT(*) * 3 AS points
    FROM matches m
    WHERE hscore < vscore
    GROUP BY vteam
 ),
  homeTiePoints AS (
    SELECT
      hteam    AS name,
      COUNT(*) AS points
    FROM matches
    WHERE hscore = vscore
    GROUP BY hteam
 ),
  awayTiePoints AS (
    SELECT
      vteam    AS name,
      COUNT(*) AS points
    FROM matches
    WHERE hscore = vscore
    GROUP BY vteam
 ),
  allPoints AS (
  SELECT *
  FROM awayWinPoints
  UNION ALL
  SELECT *
  FROM awayTiePoints
  UNION ALL
  SELECT *
  FROM homeWinPoints
  UNION ALL
  SELECT *
  FROM homeTiePoints
```

```
),
  totalPoints2 AS (
    SELECT
      t.name                    AS name,
      COALESCE(SUM(a.points), 0) AS points
    FROM teams t LEFT JOIN allPoints a
       ON t.name = a.name
    GROUP BY t.name

),
  leaders AS (
    SELECT *
    FROM totalPoints2
    WHERE points = (SELECT MAX(points)
                    FROM totalPoints2)
),

  defeatedByLeaders AS (
  (SELECT hteam AS name
   FROM matches
   WHERE vscore > hscore
        AND vteam IN (
     SELECT name
     FROM leaders
   )
  )
  UNION
  (SELECT vteam AS name
   FROM matches
   WHERE vscore < hscore
        AND hteam IN (
     SELECT name
     FROM leaders
   )
  )
)

SELECT *
FROM defeatedByLeaders;
```

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

Query (i):
No Index:
Aggregate (cost=2507.99..2508.00 rows=1 width=8) (actual time=14.731..14.732 rows=1 loops=1)

```
CREATE INDEX q1 on matches (vteam);
```

With Index:
Aggregate (cost=407.39..407.40 rows=1 width=8) (actual time=0.022..0.022 rows=1 loops=1)

(Removes a sequential scan for the query plan and replaces with index lookup)

Query (ii)
No Index:
Sort (cost=7785.57..7788.09 rows=1005 width=39) (actual time=76.291..76.342 rows=1005 loops=1)

No indexes added, as none of them had any affect on the cost of the query plan. Looking at the query plan all of the sequential scans are on the CTE's. So possibly creating some precomputed tables instead of CTE's and adding indexes on those tables could be a good idea.

Query (iii)
No Index:
CTE Scan on totallosses2 (cost=4061.83..4106.88 rows=10 width=32) (actual time=43.101..43.114 rows=3 loops=1)

No indexes added, as none of them had any affect on the cost of the query plan. Looking at the query plan all of the sequential scans are on the CTE's. So possibly creating some precomputed tables instead of CTE's and adding indexes on those tables could be a good idea. (Same result as Query (ii)

Query (iv):
No Index:
CTE Scan on defeatedbyleaders (cost=9917.80..9924.12 rows=316 width=32) (actual time=81.480..81.515 rows=104 loops=1)

Same as query (ii) and (iii). The way I wrote my queries does not benefit from indexes on either the teams or matches tables.

(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).

```
CREATE FUNCTION update_scoreboard() RETURNS trigger AS $update_scoreboard$
  BEGIN


    UPDATE scoreboard SET points = points + 2 WHERE name = NEW.hteam and NEW.hscore > NEW.vscore;
    UPDATE scoreboard SET points = points + 3 WHERE name = NEW.vteam and NEW.hscore < NEW.vscore;
    UPDATE scoreboard SET points = points + 1 WHERE name = NEW.hteam and NEW.hscore = NEW.vscore;
    UPDATE scoreboard SET points = points + 1 WHERE name = NEW.vteam and NEW.hscore = NEW.vscore;

    RETURN NEW;

  END;
$update_scoreboard$ LANGUAGE plpgsql;

CREATE TRIGGER update_scoreboard AFTER INSERT ON matches
  FOR EACH ROW EXECUTE PROCEDURE update_scoreboard();
```