DSE 201 Final – Winter 2020

This take-home final is to be performed individually, no collaboration allowed. Good luck!

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(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

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

   TRUE

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 *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:

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

```sql
-- count number of wins for specified team
select count(*) as wins
from Matches
-- find wins as a home or visitor team
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.

```sql
-- combine home and visitor subqueries and sum points
select name, sum(points) as points
from (
    -- count points for home team
    select hteam as name,
        (case
            when hscore = vscore then 1
            when hscore > vscore then 2
            else 0
        end) as points
    from Matches
    union all
    -- count points for visitor team
    select vteam as name,
        (case
            when hscore = vscore then 1
            when hscore < vscore then 3
            else 0
        end) as points
    from Matches
) as pts_tb
-- aggregate by name and order points descending
group by 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".

```sql
-- find coach from the teams with no defeat
select coach
from (
    -- find the number of defeats for each team
    select name, sum(defeat) as num_defeats
    from (
        -- count defeats for home teams
        select hteam as name,
            (case
                when hscore < vscore then 1
                else 0
            end) as defeat
        from Matches
        union all
        -- count defeats for visitor teams
        select vteam as name,
            (case
                when hscore > vscore then 1
                else 0
            end) as defeat
        from Matches
    ) as defeats_tb
    group by name
) as num_defeats_tb
-- join teams data to get coach's name
join teams t
on num_defeats_tb.name = t.name
where num_defeats = 0 -- keep teams with no loses
;
```

(iv) Return the teams defeated only by the scoreboard leaders (i.e. "if de- feated, 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".

Note: Materialized table "Scoreboard" from part (vi) is used here to take advantage of the pre-computed query.

```sql
-- get teams defeated only by scoreboard leaders and those undefeated
select name
from teams
where name not in (
    -- find teams that lost against others but not the leaders
    select t.name
    from teams t, Matches m
    where t.name = m.vteam and m.vScore < m.hScore and m.hteam not in (
        -- get scoreboard leaders
        select name
        from Scoreboard s1
        where not exists(
            select *
            from scoreboard s2
            where s1.points < s2.points
        )
    ) or t.name = m.hteam and m.hScore < m.vScore and m.vteam not in (
        -- get scoreboard leaders
        select name
        from Scoreboard s1
        where not exists(
            select *
            from scoreboard s2
            where s1.points < s2.points
        )
    )
)
;
```

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

Query Benefited	Column	Table	Clause Where Possibly Useful	SQL Statement
ii, iii	name	Teams	GROUP BY	Created by default
None	(hTeam, vteam)	Matches		Created by default
i, iv	hTeam (Foreign Key)	Matches	WHERE	CREATE INDEX home_team ON Matches(hTeam);
i, iv	vteam (Foreign Key)	Matches	WHERE	CREATE INDEX visitor_team ON Matches(vteam);

- Summary of observations
  - Part (i)
    - Having an index on the foreign key columns "hTeam" and "vteam" (from Matches table) is very useful for the query as it helps the optimizer find a smaller subset of tuples faster. For example, "San Diego Sockers" in this query.
    - The query cannot take advantage of the current primary key (pk) pair "hTeam and vteam" because the statement is finding one column at a time with the specified team. Also, the pair cannot be defined as the opponent information is not provided.
  - Part (ii)
    - Additional indices are not useful in this query. For example, creating indices on columns "hScore" and "vScore" (for Matches) table won't help the optimizer as much for two main reasons: for the inequality (i.e. hscore > vscore) it still has to check every entry, and in the case for the equality (i.e. hscore = vscore) it still has to compare many entries as the integer values might repeat multiple times.
    - Is possible that the default index for primary key "name" (on Teams) table is beneficial for the "GROUP BY name" statement.
  - Part (iii)
    - Creating additional indices is not very useful in this query because instead of finding a specific tuple value, the statement is finding inequalities and it still has to visit every entry for the scores.
    - By default there is an index on the primary key "name" for the Teams table. This index is most likely beneficial when the inner join is performed at the end of the query. Also, is possible that the same index is useful when the aggregate function "SUM" is used together with the "GROUP BY name" statement.
  - Part (iv)
    - Indices created on "hTeam" and "vteam" are potentially useful as there are equality computations performed (on the WHERE clause) between team names. However, the computations take place on a large dataset that might still include most of the tuples from the Matches table (the part where is commented with "— find the teams that lost against others but not the leaders"), therefore, the benefit will probably be small.

(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).

```sql
-- create materialized view Scoreboard
create table Scoreboard (name, points) as
    -- combine home and visitor subqueries and sum points
    select name, sum(points) as points
    from (
        -- count points for home team
        select hteam as name,
            (case
                when hscore = vscore then 1
                when hscore > vscore then 2
                else 0
            end) as points
        from Matches
        union all
        -- count points for visitor team
        select vteam as name,
            (case
                when hscore = vscore then 1
                when hscore < vscore then 3
                else 0
            end) as points
        from Matches
    ) as pts_tb
    -- aggregate by name and order points descending
    group by name
    order by points desc
;
```

```
-- create trigger procedure
create or replace function process_match_scoreboard() returns trigger as $scoreboard$
    begin
        -- add points when tied, one to each
        if (new.hScore = new.vScore) then
            update scoreboard
            set points = points + 1
            where (name = new.hTeam) or (name = new.vteam)
            ;
        -- add points when home wins, add 2 points to winner
        elsif (new.hScore > new.vScore) then
            update scoreboard
            set points = points + 2
            where name = new.hTeam
            ;
        -- add points when visitor wins, add 3 points to winner
        elsif (new.hScore < new.vScore) then
            update scoreboard
            set points = points + 3
            where name = new.vTeam
            ;
        end if;
        return null;
    end;
$scoreboard$ language plpgsql
;

-- create trigger
create trigger tr_match_scoreboard
after insert on matches
    for each row execute procedure process_match_scoreboard()
;
```