CSE 344 Section 4 Worksheet

```
CREATE TABLE Class (
    dept   VARCHAR(50),
    number INT,
    title  VARCHAR(50),
    PRIMARY KEY (dept, number));

CREATE TABLE Instructor (
    username   VARCHAR(50) PRIMARY KEY,
    fname      VARCHAR(50),
    lname      VARCHAR(50),
    started_on CHAR(10));

CREATE TABLE Teaches(
    username VARCHAR(50) REFERENCES Instructor,
    dept     VARCHAR(50),
    number   INT,
    FOREIGN KEY (dept, number) REFERENCES Class);
```

a. How many classes are currently being taught by at least one instructor?

By the nature of our data, we know that any class that appears in Teaches must be taught by at least 1 teacher. Thus, if we categorize the tuples in Teaches by dept and coursenum (the primary key), we can get our answer by counting the number of groups. The sticking point of this query is how to count the number of groups. The easy solution is to wrap the grouping query in a count(*) query.

```
SELECT COUNT(*)
  FROM (SELECT DISTINCT dept, coursenum
          FROM Teaches);
```

b. Return the first name and last name of the instructors who teach the most number of classes. If there are multiple instructors, list all of them.

This is another example of the witnessing problem. There are multiple approaches - see below.

SOLUTION 1:

```
WITH (
   SELECT username, COUNT(*) AS count
     FROM Teaches
    GROUP BY username
) AS ClassCounts
SELECT I.fname, I.lname
  FROM ClassCounts AS C1, ClassCounts AS C2, Instructor AS I
 WHERE C1.username = I.username
 GROUP BY I.username, I.fname, I.lname, C1.count
HAVING C1.count = MAX(C2.count);
```

SOLUTION 2:

```
WITH (
    SELECT username, COUNT(*) AS count
      FROM Teaches
     GROUP BY username
) AS ClassCounts,
(
   SELECT MAX(count) AS max
     FROM ClassCounts
) AS MaxCounts
SELECT I.fname, I.lname
  FROM ClassCounts AS C, MaxCounts AS M, Instructor AS I
 WHERE C.username = I.username AND C.count = M.max;
```

SOLUTION 3:

```
SELECT I.fname, I.lname
  FROM Instructor AS I, Teaches AS T
 WHERE I.username = T.username
 GROUP BY I.username, I.fname, I.lname
HAVING COUNT(*) >= ALL (
                     SELECT COUNT(*)
                       FROM Teaches
                      GROUP BY username
                  );
```

```
CREATE TABLE Members (
    member_id   INT PRIMARY KEY,
    name        VARCHAR(100) NOT NULL,
    city        VARCHAR(100),
    join_date   DATE);

CREATE TABLE Fish (
    fish_id         INT PRIMARY KEY,
    species         VARCHAR(100) NOT NULL,
    water_type      VARCHAR(20), -- e.g. 'Freshwater', 'Saltwater'
    adult_size      INT);        -- average adult size (e.g. length in cm)

CREATE TABLE Ownership (
    member_id           INT REFERENCES Members,
    fish_id             INT REFERENCES Fish,
    date_acquired       DATE,
    PRIMARY KEY(member_id, fish_id));
```

There are often multiple ways to solve problems like these!

a. Find all fish whose `adult_size` is bigger than that of *any* fish owned by Carol (i.e. bigger than her smallest fish). Output their `fish_id` and `adult_size`.

```
SELECT f.fish_id, f.adult_size
FROM Fish f
WHERE f.adult_size > ANY (
   SELECT f2.adult_size
   FROM Ownership o2
   JOIN Fish f2 ON o2.fish_id = f2.fish_id
   JOIN Members mc ON o2.member_id = mc.member_id
   WHERE mc.name = 'Carol'
);
```

NOTE: SQLite does not support the ANY keyword. Instead, you can usually slightly modify your query:

```
SELECT f.fish_id, f.adult_size
FROM Fish f
WHERE f.adult_size > (
      SELECT MIN(f2.adult_size)
      FROM Ownership o2
      JOIN Fish f2 ON o2.fish_id = f2.fish_id
      JOIN Members mc ON o2.member_id = mc.member_id
      WHERE mc.name = 'Carol'
);
```

b. Find all members who own every species of freshwater fish, and output their names

```
SELECT m.name
FROM Members m
WHERE NOT EXISTS (
   SELECT 1
   FROM Fish f
   WHERE f.water_type = 'Freshwater'
     AND f.fish_id NOT IN (
         SELECT o.fish_id
         FROM Ownership o
         WHERE o.member_id = m.member_id
     )
);
```

c. Find all rare fish (we define rare fish as those owned either by only 1 member or by no members). Output Challenge: can you solve this both using and without using subqueries?

With subqueries:

```
SELECT f.fish_id, f.species
FROM Fish f
WHERE f.fish_id IN (
   SELECT o.fish_id
   FROM Ownership o
   GROUP BY o.fish_id
   HAVING COUNT(o.member_id) = 1
)
OR f.fish_id NOT IN (
   SELECT DISTINCT o.fish_id FROM Ownership o
);
```

Without subqueries:

```
SELECT f.fish_id, f.species
FROM Fish f
LEFT JOIN Ownership o ON f.fish_id = o.fish_id
GROUP BY f.fish_id
HAVING COUNT(o.member_id) = 1 OR COUNT(o.member_id) = 0;
```