CS W186 - Spring 2024
Exam Prep Section 1
SQL

Question 1: SQL

Consider the following schema for library books:

```
Books: bid INTEGER,
       title TEXT,
       library INTEGER REFERENCES Library,
       genre TEXT,
       PRIMARY KEY (bid)

Library: lid INTEGER,
         lname TEXT,
         PRIMARY KEY (lid)

Checkouts: book INTEGER REFERENCES Books,
           day DATETIME,
           PRIMARY KEY (book, day)
```

(a) Return the bid and genre of each book that has ever been checked out. Remove any duplicate rows with the same bid and genre.

```
SELECT DISTINCT b.bid, b.genre
FROM Books b, Checkouts c
WHERE b.bid = c.book
```

(b) Find all of the fantasy book titles that have been checked out and the date when they were checked out. Even if a book hasn't been checked out, we still want to output the title (i.e. the row should look like (title, NULL)).

```
SELECT title, day
FROM Books b LEFT OUTER JOIN Checkouts c ON c.book = b.bid
WHERE b.genre = 'Fantasy';
```

(c) Select the name of the book that has been checked out the most times and the corresponding checked out count. You can assume that each book was checked out a unique number of times, and that the titles of the books are all unique.

```
SELECT title, count(*) as cnt
FROM Books b, Checkouts c
WHERE b.bid = c.book
GROUP BY b.title
ORDER BY cnt DESC
LIMIT 1;
```

(d) Select the name of all of the pairs of libraries that have books with matching titles. Include the name of both libraries and the title of the book. There should be no duplicate rows. The first library name should be alphabetically less than the second library name, e.g. (‘East’, ‘West’, ‘Of Mice and Men’) not (‘West’, ‘East’, ‘Of Mice and Men’). **There may be zero, one, or more than one correct answer.**

```
(A) SELECT DISTINCT l.lname, l.lname, b.title
     FROM Library l, Books b
     WHERE b.library = l.lid
     ORDER BY l.lname
```

```
(B) SELECT DISTINCT l1.lname, l2.lname, b1.title
     FROM Library l1, Library l2, Books b1, Books b2
     WHERE l1.lname < l2.lname AND b1.title = b2.title
     AND b1.library = l1.lid AND b2.library = l2.lid
```

```
(C) SELECT DISTINCT first.l1, second.l2, b1
      FROM
          (SELECT lname l1, title b1
             FROM Library l, Books b
           WHERE b.library = l.lid) as first,
          (SELECT lname l2, title b2
             FROM Library l, Books b
           WHERE b.library = l.lid) as second
      WHERE first.l1 < second.l2
         AND first.b1 = second.b2;
```

Solution: **B, C**

A is incorrect because it does not return the books with matching titles but rather the names of the books at a specific library.

B is correct. C will perform a cross product of libraries with itself and books with itself matching books on title and matching that book to two different libraries.

C is correct. It performs two separate joins on libraries and books finding all of the book library pairs as inner subqueries. Then the outer query finds the matching book titles from the subqueries such that the library name for one is less than the second.

Question 2: More SQL

Consider the following schema for bike riders between cities:

```
Locations: lid INTEGER PRIMARY KEY,
           city_name VARCHAR

Riders: rid INTEGER PRIMARY KEY,
        name VARCHAR,
        home INTEGER REFERENCES locations (lid)

Bikes: bid INTEGER PRIMARY KEY
       owner INTEGER REFERENCES riders (rid)

Rides: rider INTEGER REFERENCES riders(rid)
       bike INTEGER REFERENCES bikes(bid)
       src INTEGER REFERENCES locations(lid)
       dest INTEGER REFERENCES locations(lid)
```

(a) Select all of the following queries which return the rid of Rider with the most bikes. Assume all Riders have a unique number of bikes.

(A) SELECT owner FROM bikes GROUP BY owner ORDER BY COUNT(*) DESC LIMIT 1;

(B) SELECT owner FROM bikes GROUP BY owner HAVING COUNT(*) >= ALL
(SELECT COUNT(*) FROM bikes GROUP BY owner);

(C) SELECT owner FROM bikes GROUP BY owner HAVING COUNT(*) = MAX(bikes);

Solution: **A, B**
C is incorrect syntax. MAX(bikes) does not make sense in this context.

(b) Select the bid of all Bikes that have never been ridden.

(A) SELECT bid FROM bikes b1 WHERE NOT EXISTS
(SELECT owner FROM bikes b2 WHERE b2.bid = b1.bid);

(B) SELECT bid FROM bikes WHERE NOT EXISTS
(SELECT bike FROM rides WHERE bike = bid);

(C) SELECT bid FROM bikes WHERE bid NOT IN
(SELECT bike FROM rides, bikes as b2 WHERE bike = b2.bid);

Solution: **B, C**
A finds all of the bikes with no owners which is not the question we are trying to answer.

(c) Select the name of the rider and the city name of the src and dest locations of all their journeys for all rides. Even if a rider has not ridden a bike, we still want to output their name (i.e. the output should be (name, null, null)).

(A) SELECT tmp.name, s.city_name AS src, d.city_name AS dst FROM
locations s, locations d,
(riders r LEFT OUTER JOIN rides ON r.rid = rides.rider) as tmp
WHERE s.lid = tmp.src AND d.lid = tmp.dest;

(B) SELECT r.name, s.city_name AS src, d.city_name as dst FROM riders r
LEFT OUTER JOIN rides ON r.rid = rides.rider
INNER JOIN locations s on s.lid = rides.src
INNER JOIN locations d on d.lid = rides.dest;

(C)
```
SELECT r.name, s.city_name AS src, d.city_name AS dst FROM rides
    RIGHT OUTER JOIN riders r ON r.rid = rides.rider
    INNER JOIN locations s on s.lid = rides.src
    INNER JOIN locations d on d.lid = rides.dest;
```

Solution: **None of these are correct**, because the subsequent inner joins/where clauses will filter out the null rows from the outer join.