CSE 344 Section 2 Worksheet Solutions

Joins Examples

Given tables created with these commands:

```
CREATE TABLE A (a INT);
INSERT INTO A VALUES (1), (2), (3), (4);
CREATE TABLE B (b INT);
INSERT INTO B VALUES (3), (4), (5), (6);
```

What's the output for each of the following:

```
SELECT * FROM A INNER JOIN B ON A.a=B.b;
```

```
a|b
3|3
4|4
```

```
SELECT * FROM A INNER JOIN B;
```

```
a|b
1|3
1|4
1|5
1|6
2|3
2|4
2|5
2|6
3|3
3|4
3|5
3|6
4|3
4|4
4|5
4|6
```

```
SELECT * FROM A LEFT OUTER JOIN B ON A.a=B.b;
a|b
3|3
4|4
1|
2|
 SELECT * FROM A RIGHT OUTER JOIN B ON A.a=B.b;
a|b
 |5
 |6
3|3
4|4
SELECT * FROM A FULL OUTER JOIN B ON A.a=B.b;
a|b
 |5
 |6
1|
2|
3|3
4|4
```

*** ADDITIONAL INFO BELOW ***

sqlite3 supports neither RIGHT OUTER JOIN nor FULL OUTER JOIN.

RIGHT OUTER JOIN can be implemented with SELECT * FROM B LEFT OUTER JOIN A ON A.a=B.b;

FULL OUTER JOIN can be implemented with (SELECT * FROM A LEFT OUTER JOIN B ON A.a=B.b) UNION (SELECT * FROM B LEFT OUTER JOIN A ON A.a=B.b);

UNION is a set union that eliminates duplicates;
UNION ALL is a multiset (or bag) union that keeps duplicates.

SQL Practice (Movie-Actor)

```
CREATE TABLE Movies ( id int, name varchar(30), budget int,
gross int, rating int, year int, PRIMARY KEY (id) );

CREATE TABLE Actors ( id int, name varchar(30), age int, PRIMARY
KEY (id) );

CREATE TABLE ActsIn ( mid int, aid int, FOREIGN KEY (mid)
REFERENCES Movies (id), FOREIGN KEY (aid) REFERENCES Actors (id)
);
```

(a) What is the number of movies, and the average rating of all movies that the actor ”Patrick Stewart” has appeared in?

SELECT count(*), avg(rating) FROM Movies as M, ActsIn as AI, Actors as A

WHERE M.id = AI.mid AND A.id = AI.aid AND A.name = “Patrick Stewart”;

(bonus) What movies have no actors? Return movie names of those movies.

select M.name from Movies M left outer join ActsIn I on M.id = I.mid where I.mid is null;

Or, for another approach,

select M.name from Movies M where not exists (select * from ActsIn I where I.mid = M.id);

(b) What is the minimum age of an actor who has appeared in a movie where the gross of the movie has been over $1,000,000,000?

SELECT min(age) FROM Movies as M, ActsIn as AI, Actors as A

WHERE M.id = AI.mid AND AI.aid = A.id AND M.gross > 1,000,000,000;

(c) What is the budget of each movie released in 2017 whose oldest actor is less than 30?

SELECT M.name, M.budget FROM Movies as M, ActsIn as AI, Actors as A

WHERE M.id = AI.mid AND AI.aid = A.id AND M.year = 2017

GROUP BY M.id, M.name, M.budget

```
HAVING max(A.age) < 30;
```

Some code for testing…

```
insert into Movies values (1,'MA',100,0,0,2017);
insert into Movies values (2,'MB',200,0,0,2017);
insert into ActsIn values (1,11);
insert into ActsIn values (1,12);
insert into ActsIn values (2,11);
insert into ActsIn values (2,13);
insert into Actors values (11,'AA',20);
insert into Actors values (12,'AB',35);
insert into Actors values (13,'AC',21);
```

Self Join

Consider the following over-simplified Employee table, listing employees and their boss (if any):

```
    CREATE TABLE Employees (id int, boss int);
```

Suppose all employees have an id which is not null. How would we find all distinct pairs of employees with the same boss?

```
SELECT E1.id, E2.id FROM Employee AS E1, Employee AS E2
WHERE E1.id > E2.id AND E1.boss = E2.boss;
```

Sidenote: The predicate “E1.id > E2.id” could also be written as “E1.id < E2.id”. We cannot use plain inequality as the predicate condition because this would lead to duplicate pairs.

Additional Movie & Director Practice

Movies and Directors

```
CREATE TABLE Director (
    id      INT PRIMARY KEY,
    name    VARCHAR(75),
    country VARCHAR(75));
```

```
CREATE TABLE Movie (
    id      INT PRIMARY KEY,
    name    VARCHAR(75),
    did     INT REFERENCES Director,
    year    INT,
    budget  INT);
```

Find the id and name of all directors who have directed more than 20 movies.

- We see that we want the property of the “count of movies” associated with that director being > 20. Associating movies to a director lends itself naturally to categorization over director_id so a GROUP BY is needed.
- The conditional property we want is over groups of movies (associated with a director), so a HAVING clause is also needed.

```
SELECT D.id, D.name
  FROM Director D, Movie M
 WHERE D.id = M.did
 GROUP BY D.name, D.id
HAVING COUNT(*) > 20;
```

Additional Self Join Practice

Consider the following over-simplified Employee table

```
CREATE TABLE Employees (
      id int NOT NULL,
      bossOf int
);
```

Note that all employees have an id that is not null, but they may have a null “bossOf” entry, or the bossOf entry may refer to employees already left the company. How do we find the id of all employees who are the boss of at least one other employee? Ensure that the bossOf value refers to a current employee in the Employees relation.

```
SELECT DISTINCT e1.id
FROM Employees AS e1, Employees AS e2
WHERE e1.bossOf=e2.id;
```

We need to use self-join in order to ensure that the “bossOf” id refers to an actual employee id, as opposed to an employee who left the company.

Consider the Employees relation {(1, NULL), (2, NULL), (5, 1), (5, 2), (5, NULL), (3, NULL)}. How many current employees is the employee with id=5 boss of? (i.e. how many employees works for employee with id=5)

Since we do not count the null entry, **two** employees.

Write a query that returns a relation with the id of each employee and the count of how many employees (i.e., any non-null id) they are the boss of.

```
SELECT id, COUNT(bossOf)
FROM Employees
GROUP BY id;
```

Note that we need to use COUNT(bossOf) as opposed to COUNT(*) so that we do not count null entries.