Experimenting with bags (tables and query answers with duplicate rows):

Write an SQL query (and run it against the sailors database) that does the following:

1. List every row in the boats table.

   ```
   select * from boats
   ```

   ```
   bid   bname   color
   102   Interlake   red
   103   Clipper   green
   104   Marine   red
   105   Majestic   green
   101   Interlake   blue
   101   Interlake   blue
   ```

   6 row(s)

   Are there any duplicate rows? (You may think that this is a mistake; but there are some situations where a table may very well have duplicate rows. This table allows us to experiment with duplicate rows.)

   Yes, the last two rows are identical (with identical bid values). Note: this table used to have bid as a primary key. I dropped the primary key constraint and then inserted the final row shown here for this exercise.

   This allows you to see that a basic SQL query does NOT eliminate duplicates. It shows you the duplicate rows in the query answer. (There is a way to ask SQL to eliminate duplicates from a query answer using the word distinct; we’ll see more about that down below.)

2. List every row from the boats table where the color is not red. Are there any duplicate rows in the query answer? (This shows you what SQL does when you issue a query that is equivalent to a single select operator (from relational algebra on bags).

   ```
   select * from boats where color != 'red'
   ```

   ```
   bid   bname   color
   103   Clipper   green
   ```
4 row(s)

This seems logical; the where clause is evaluated for each row in the original table. If the where clause evaluated to true, then the row is placed in the query answer. The last two rows show (the duplicate rows) each allow the where clause to evaluate to true; therefore they are both placed in the query answer.

3. List every combination of one boat and one reservation. Before you run the query, estimate the number of rows that you will have in the query answer. After you run the query, notice the names of all the attributes. Is there an attribute name that appears more than once? (This shows you what SQL does with a simple cross product – even if there is ambiguity in the attribute names.)

Let’s find out how many rows are in the boats table. With this query we see that there are six.

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
<tr>
<td>105</td>
<td>Majestic</td>
<td>green</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
</tbody>
</table>

6 row(s)

Let’s find out how many rows are in the reserves table with this query.

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>102</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>1998-10-08</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>1998-10-07</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>1998-11-10</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>1998-11-06</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>1998-11-12</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>1998-09-05</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>1998-09-08</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>1998-09-08</td>
</tr>
</tbody>
</table>
Since we are asked to compute the cross product, we would expect 66 rows in the query answer. If we run this query:

```sql
select * from boats, reserves
```

We see this as the final part of the query answer:

```
102 Interlake red  55 6 1999-10-10
103 Clipper green  55 6 1999-10-10
104 Marine red     55 6 1999-10-10
105 Majestic green 55 6 1999-10-10
101 Interlake blue 55 6 1999-10-10
101 Interlake blue 55 6 1999-10-10
```

66 row(s)

Does this query answer have any duplicate rows? (This shows you what SQL does with the cross product operator when there are bags, i.e., tables with duplicates, as input tables.)

Yes, we can see, for example, that the final two rows are identical. This seems logical because we had two identical rows in the boats table; and they each appear here – in the cross product – with each reservation. The final two rows of the query answer show the two duplicate boat rows with the final reservation. (Earlier in the query answer, these two duplicate boat rows appear with each of the other reservation.)

4. Extend the query from item 3 so that the only combinations (of a boat and a reservation) shown in the query answer are where the boat id in the reservation matches the boat id in the boat table. (Do not use a join clause.)

```sql
select *  
from boats b, reserves r
where b.bid = r.bid
```

Try writing this query without using any correlations names (e.g., aliases or short names) for the tables.

```sql
select *  
from boats, reserves
```
where bid = bid

SQL error:
ERROR: column reference "bid" is ambiguous

LINE 2: where bid = bid

^  

In statement:
select * from boats, reserves
where bid = bid

We get this error because the first (and the second) bid in the where clause is ambiguous. The query parser has no idea which (of the two) bid attribute we are using.

Try writing this query with just one correlation name (for one of the two tables).

select * from boats, reserves r
where bid = r.bid

SQL error:
ERROR: column reference "bid" is ambiguous
LINE 2: where bid = r.bid

^  

In statement:
select * from boats, reserves r
where bid = r.bid

We get this error because the first “bid” in the where clause is ambiguous; the query parser has no idea which bid we are talking about.

Try writing this query with a correlation name for both of the two tables.

My first query listed above used two correlation names. Here’s the result:

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>22</td>
<td>102</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
<td>22</td>
<td>103</td>
<td>1998-10-08</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>22</td>
<td>104</td>
<td>1998-10-07</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>31</td>
<td>102</td>
<td>1998-11-10</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
<td>31</td>
<td>103</td>
<td>1998-11-06</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>31</td>
<td>104</td>
<td>1998-11-12</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>64</td>
<td>101</td>
<td>1998-09-05</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>64</td>
<td>101</td>
<td>1998-09-05</td>
</tr>
</tbody>
</table>
Notice the attributes that appear in the query answer. Are they any different from the attributes that appeared in your answer to query from item 3?

We see all of the attributes for boat and we see all of the attributes for reserves. This is the same set of attributes that we see for the cross product above. (You can see that the values for bid in the first column and in the fifth column are identical. This is not surprising; the where clause requires that these two values be identical. This query is, in fact, a join query. We can say that it is written the “old fashioned” way – with a cross product (in the from clause) followed by a select (in the where clause).)

Try writing this query where you list the two tables involved in the opposite order. Is there any difference in the attributes that appear in the query answer? Do the same rows appear in these two variations of the query? Note: SQL makes no promise about the order that the tuples appear in a query answer. As long as the proper rows are returned in the query answer – in any order – then the query answer is deemed to be correct. (There is a way to control the order of the rows in the query answer – using the order by clause; we’ll talk about that next week.)

```
select * from reserves r, boats b
where b.bid = r.bid
```
Here we see that the attributes from the reserves table are shown first followed by the attributes of the boats table. So, (if you don’t do something in your query to change the order of the attributes), the attributes of the first table in the where clause are shown first in the query answer.

Notice the duplicates in the query answer. Does it make sense – what you see as duplicates (knowing which rows are in the boats table and in the reserves table)?

In a similar way to the cross product, each duplicate boat row is processed through the join. So, we see that both of the “101 Interlake blue” boats join with the reservation by sid 64 on September 5 and they both join with the reservation by sid 22 on October 10. This seems logical; each row in each table participates in the cross product (as we saw above) and then each such result is evaluated using the where clause.

5. Write an SQL query (without using a join clause in the from clause) that lists the sailor id, the sailor name, the boat id, and the boat name where the sailor has reserved the boat. Explain in English what it means if there are duplicate rows in the query answer.

```sql
select s.sid, s.sname, b.bid, b.bname
from sailors s, reserves r, boats b
where r.bid = b.bid and s.sid = r.sid
```

Here’s an equivalent query using two join clauses:

```sql
select s.sid, s.sname, b.bid, b.bname
from sailors s join reserves r on s.sid = r.sid
    join boats b on r.bid = b.bid
```

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>bid</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>101</td>
<td>Interlake</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>101</td>
<td>Interlake</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>104</td>
<td>Marine</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>103</td>
<td>Clipper</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>102</td>
<td>Interlake</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>104</td>
<td>Marine</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>103</td>
<td>Clipper</td>
</tr>
</tbody>
</table>
If I describe this query in English, I would say that this query lists the sailor id and name with the boat id and name if the sailor has ever reserved the boat.

If we DIDN’T have duplicate rows in the boats table, it might make more sense.

But since we have duplicate rows in the boats table, we seen that if a sailor has reserved the boat with bid of 101, then the sailor is shown to have reserved each of the duplicate representations of a boat with bid of 101. It’s still the case that this query shows the sailor id and name with the boat id and name if the sailor has ever reserved the boat.

Notice that when you list attributes in the select clause, you are setting the order that these attributes appear in the query answer. When you list the attributes you want to see in the final query answer in the select clause, it doesn’t matter what order you list the tables (or the joins) in the from clause; the select clause sets the order for the attributes in the query answer.

6.

**Experimenting with joins – including self-joins – without using the join clause**

7. Run the following query:

```sql
select b.bid, b.bname, r1.day, r2.day
from boats b, reserves r1, reserves r2
where b.bid = r1.bid and b.bid = r2.bid and r1.day < r2.day
```

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>day</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Interlake</td>
<td>1998-09-08</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>1998-09-08</td>
<td>1998-10-08</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>1998-09-08</td>
<td>1998-11-10</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>1998-10-10</td>
<td>1998-11-10</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>1998-09-08</td>
<td>1998-11-06</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>1998-10-08</td>
<td>1998-11-06</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>1998-10-07</td>
<td>1998-11-12</td>
</tr>
</tbody>
</table>
Describe in English what this query computes.

This query lists boat id and boat name for any boat that has at least two distinct reservations. The query answer shows the two dates (for the two reservations) that were found in the DB.

What happens if you leave out the second part of the where clause (i.e., leave out the “r1.day < r2.day”)? (You can run the query to see what happens.)

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>day</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>1998-09-05</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>1998-09-05</td>
<td>1998-10-10</td>
</tr>
</tbody>
</table>

9 row(s)
This query does not require that the two reservations found be distinct. This means that a boat that with just one reservation would appear in the query answer. (We don’t happen to have any boats like that in the current instance of the sailors database.) We also see that each boat that has at least one reservation is shown here – with the reservation repeated. See, for example, the third row in the query answer where the boat with id of 102 is shown with two (copies of) the one reservation on October 10.

What happens if you leave out the entire WHERE clause?

Then, you will be computing the cross product of boats, reserves, and sailors.

Contrast the following query with the one above:

```sql
select b.bid, b.bname, r1.day
from boats b, reserves r1
where b.bid = r1.bid
```

Describe in English what this query computes.

This query lists each reservation with the boat that has been reserved. (This is a straightforward join of two tables.)
8. Run the following query (which is a modification of the first query in item 8 – with fewer attributes in the final query answer):

```sql
select b.bid, b.bname
from boats b, reserves r1, reserves r2
where b.bid = r1.bid and b.bid = r2.bid and r1.day < r2.day
```

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Interlake</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
</tr>
</tbody>
</table>

9 row(s)

Describe in English what this query computes.

This query lists bid and boat name for any boat that has ever been reserved. Each boat is repeated as many times as it has been reserved.

Now, try this: (Note: I had a typo in the original activity; I correct it here in red.)
select distinct b.bid, b.bname
from boats b, reserves r1, reserves r2
where b.bid = r1.bid and b.bid = r2.bid and r1.day < r2.day

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>Marine</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
</tr>
</tbody>
</table>

4 row(s)

Explain in English what this query computes.

This query lists the boat id and name for all boats that have ever been reserved. You could also say that: This query lists the boat id and name for all boats with at least one reservation.

9. Write and run an SQL query (modeled after the query above) that finds all boats that have at least three reservations.

For this query, we need to join three copies of the reserves table with boats and we need to make sure that all three of the reservations are different. Note: you may have chosen to omit the “distinct” clause here; I wasn’t precise in my description of the query.

select distinct b.bid, b.bname
from boats b, reserves r1, reserves r2, reserves r3
where b.bid = r1.bid and b.bid = r2.bid and b.bid = r3.bid and r1.day < r2.day and r2.day < r3.day

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Interlake</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
</tr>
</tbody>
</table>

2 row(s)
We see two boats listed. We can manually check to see if that’s right by looking at all of the reservations – using this query:

```
select * from reserves
```

```
sid  bid   day
22  102  1998-10-10
22  103  1998-10-08
22  104  1998-10-07
31  102  1998-11-10
31  103  1998-11-06
31  104  1998-11-12
64  101  1998-09-05
64  102  1998-09-08
74  103  1998-09-08
22  101  1998-10-10
55   6   1999-10-10
```

11 row(s)

We see that 102 has three different reservations (Oct. 10, Nov. 10, and Sept. 8).
We see that 103 has three different reservations (Oct. 8, Nov. 6, and Sept. 8).
The other boats have fewer reservations: 101 has two reservations and 104 has two reservations.

**Experimenting with the join clause including left, right and full outer joins.**

10. Rewrite and run the query from item 4 using a join clause in the from clause. Your query answer should be identical to the answer you got for the query in item 4. (The order of the rows might be different, as explained above.)

Here’s the original query from item 4.

```
select *
from boats b, reserves r
where b.bid = r.bid
```

Here’s an equivalent query using the join clause:

```
select *
```
from boats b join reserves r on b.bid = r.bid

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>22</td>
<td>102</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
<td>22</td>
<td>103</td>
<td>1998-10-08</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>22</td>
<td>104</td>
<td>1998-10-07</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>31</td>
<td>102</td>
<td>1998-11-10</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
<td>31</td>
<td>103</td>
<td>1998-11-06</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>31</td>
<td>104</td>
<td>1998-11-12</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>64</td>
<td>101</td>
<td>1998-09-05</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>64</td>
<td>101</td>
<td>1998-09-05</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>64</td>
<td>102</td>
<td>1998-09-08</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
<td>74</td>
<td>103</td>
<td>1998-09-08</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>22</td>
<td>101</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>22</td>
<td>101</td>
<td>1998-10-10</td>
</tr>
</tbody>
</table>

12 row(s)

We see the same 12 rows as the original query.

11. Rewrite and run the query from item 5 using two join clauses in the from clause. Your query answer should be identical to the answer you got the query in item 5.

Here's the original query:
select s.sid, s.sname, b.bid, b.bname
from sailors s, reserves r, boats b
where r.bid = b.bid and s.sid = r.sid

Here's an equivalent query using two join clauses:

select s.sid, s.sname, b.bid, b.bname
from sailors s join reserves r on s.sid = r.sid
join boats b on r.bid = b.bid

Here are the two query answers – side by side; they are identical.
12. Rewrite and run the query from item 6 using the join clause in the from clause. Note: you’ll have to use two join clauses.

Since I had a typo above, you had no idea which item I was referring to. Here’s query from item 7 rewritten with a join clause.

Original query:
select b.bid, b.bname, r1.day, r2.day
from boats b, reserves r1, reserves r2
where b.bid = r1.bid and b.bid = r2.bid and r1.day < r2.day

Equivalent query using two join clauses:

select b.bid, b.bname, r1.day, r2.day
from boats b join reserves r1 on b.bid = r1.bid join reserves r2 on b.bid = r2.bid
where r1.day < r2.day
13. Item 6 and item 11 asked you to write a query to join reserves and boats.

Write and then run a similar query to compute the left outer join of reserves and boats, matching on bid. Explain in English what this query computes.

```
select *
from reserves r left join boats b on r.bid = b.bid
```

It has the same answer as item 7 above.
This query matches reservations with the boat that was reserved. All reservations are listed – even if there is no boat that corresponds to the reservation (e.g., the reservation for boat 55 above). Reservations without boats have Null values shown in the attributes from the boat table in the query answer.

Write and run a similar query to compute the right outer join of reserves and boats, matching on bid. Explain in English what this query computes.

```
select *
from reserves r right join boats b on r.bid = b.bid
```

This query lists boats with their corresponding reservations. Boats that have no reservations are also shown; such boats have Null values for the attributes from the reserves table.

Notice that this query answer does not show the reservation for boat 55; it only shows the matches (e.g., from the regular join) plus rows from the RIGHT table (boat, in this case) that do not match. Contrast that with the left join shown above.

Write and run a similar query (as the one you just wrote) but change the order of the two tables.
listed in the from clause (but still use a right outer join). What is the difference between this query and the previous one? What is the difference between this query and the first one you wrote for this item (Item 12)?

```sql
select *
from boats b right join reserves r on r.bid = b.bid
```

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>22</td>
<td>102</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
<td>22</td>
<td>103</td>
<td>1998-10-08</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>22</td>
<td>104</td>
<td>1998-10-07</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>31</td>
<td>102</td>
<td>1998-11-10</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
<td>31</td>
<td>103</td>
<td>1998-11-06</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>31</td>
<td>104</td>
<td>1998-11-12</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>64</td>
<td>101</td>
<td>1998-09-05</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>64</td>
<td>101</td>
<td>1998-09-05</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>64</td>
<td>102</td>
<td>1998-09-08</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
<td>74</td>
<td>103</td>
<td>1998-09-08</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>22</td>
<td>101</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>22</td>
<td>101</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>55</td>
<td>6</td>
<td>1999-10-10</td>
</tr>
</tbody>
</table>

13 row(s)

So, this query answer is identical to the first, left outer join above.

Write a similar query to compute the full outer join of reserves and boats, matching on bid. Explain what this query computes in English.

Identify a row that appears in all three answers (the left, right, and full outer join) and explain why it appears in all three answers.

Is there a row in the query answer to the full outer join that does NOT appear in any of the other queries? Explain why this is so (or not so).

14. In general, what is the difference between computing the cross product of two tables and computing the full outer join of the same two tables (where you join for equality on some attribute that appears in both tables)? Explain.
The cross product contains every combination of one row from the second table and one from the first table.

A join (a regular or inner join) always contains a subset of the cross product of the two tables. We know this because a join is equivalent to a cross product followed by a select.

A full outer join always includes a join. So the part of a full outer join that has the “matches” or the result from the regular join – is a subset of the cross product of the two tables. But the additional rows in a full outer join always consist of one row from one of the two tables concatenated with a row consisting of all null values. These rows consisting of null values can appear on the left side or the right side of the resulting rows in a full outer join. But such rows NEVER appear in a cross product. The cross product never introduces null values.

(Try several queries out – if you’re not sure about this.)

Does a full outer join contain rows that do not appear in a cross product?

Yes, if the full outer join has any rows from the left table or the right table that do NOT join, then it will place that row into the query answer with the proper number of null values. Such tuples do NOT appear in the result of a cross product.

Does a cross product contain rows that do not appear in a join?

(Note: this following answer is the same – if I asked “Does a cross product contain rows that do not appear in a full outer join – or a left outer join – or a right outer join. The only rows that can appear in a cross product that is not in any kind of join, outer or inner, are rows that do NOT match. There may or may not be any rows that do not match.)

Not necessarily. In general, a join is a subset of the cross product. But consider the following tables:

<table>
<thead>
<tr>
<th>Student(id, name, adv)</th>
<th>Faculty(id, name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, John, 101</td>
<td>101, Waterson</td>
</tr>
<tr>
<td>2, Sue, 101</td>
<td></td>
</tr>
<tr>
<td>3, Wei, 101</td>
<td></td>
</tr>
</tbody>
</table>

In this case, the query answer for:
Student X Faculty
is identical to the query answer to:
Student s \*_{s.adv=f.id} Faculty f

Notice that these two queries are NOT equivalent. They only have the same answer some of the
time – depending on the current instance of the tables (that is, depending on the rows that are currently in the table).

Make sure you consider the case where the full outer join includes data in the original tables where at least one row from the first (left) table does not match any rows (in the second table) and where there is at least one row from the second (right) table does not match any rows (in the first table). For this experiment, you should insert data into the tables you created last week – to play around with left, right, and full outer join.