**Section 4 Worksheet**

**Part 1: Interpreting SQL and Relational Data**

For each SQL query, find

1) what the SQL statement is querying for (a short description) and
2) an equivalent relational algebra (RA) expression/tree

A. (Midterm 12AU)

**`Clinic(cid, name, street, state)`**
**`Equipment(eid, type, model)`**
**`Assignment(cid, eid)`**

Finds the count of clinics that do not have a fridge (of model 1004) assigned to it.

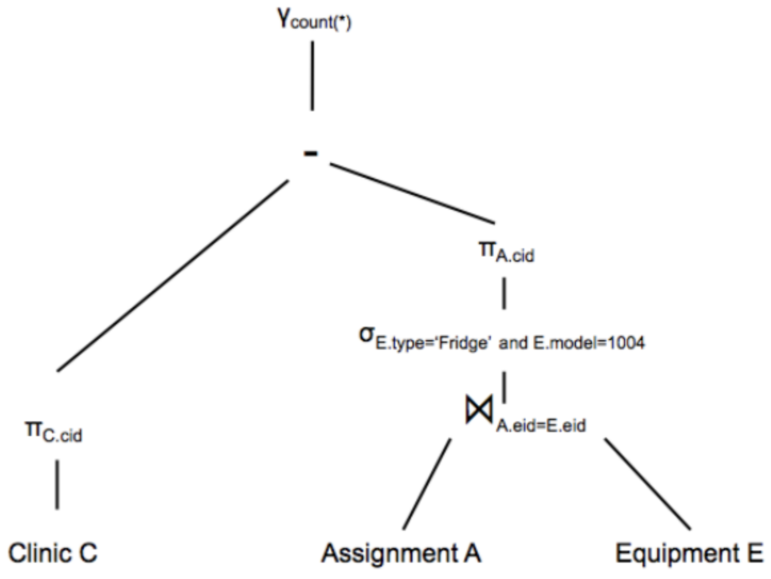
1)

Our problem is a negated existential. Thus a subquery might help very much. Below is one possible solution to compute the set difference between all clinics and clinics that have a fridge of model 1004.

```
SELECT COUNT(*)
  FROM Clinic AS C
 WHERE NOT EXISTS (SELECT *
                     FROM Assignment AS A, Equipment AS E
                    WHERE C.cid = A.cid AND
                          A.eid = E.eid AND
                          E.type = 'Fridge' AND
                          E.model = 1004);
```

2)

We cannot easily express a subquery in RA so we must remember what the problem is asking. We wish to model a set difference, so we can use the minus operation.



B. (Midterm 15AU)

```
Item(oid, category, price)
Gift(pid, rid, oid) -- pid gifts oid to rid

SELECT O1.category, max(abs(O1.price - O2.price))
  FROM Gift AS G1, Gift AS G2, Item AS O1, Item AS O2
 WHERE G1.pid = G2.rid AND
       G2.pid = G1.rid AND
       O1.oid = G1.oid AND
       O2.oid = G2.oid AND
       O1.category = O2.category
 GROUP BY O1.category
HAVING count(*) > 5;
```

1)

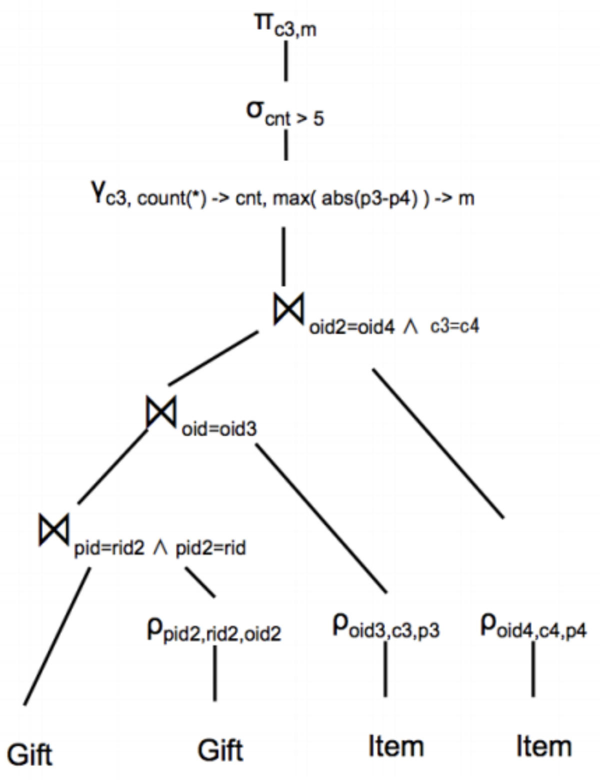
Things to note:

- “G1.pid = G2.rid AND G2.pid = G1.rid” indicates mutually exchanged gifts.
- “max(abs(O1.price – O2.price))” indicates the largest absolute price difference
- “HAVING count(*) > 5” indicates there must be at least 5 tuples for this category (however, the query will effectively double count pairs that pass the WHERE clause)

Finds item categories that have been mutually gifted over 5 times and the corresponding maximum price difference between mutually exchanged items (of said category).

2)

For the bottom of the tree, joins can be done in any order, although standard RA dictates that each join have two children operations.



## Part 2: Misc. SQL Practice

A. (Difference Techniques/3-Value Logic) Imagine we have the table R(v), such that R holds the tuples {(NULL), (1)}, and we also have the table S(v) such that S holds the tuples {(NULL), (2)}. What are the outputs of the following queries?

```
-- 1
SELECT *
  FROM R
 WHERE R.v NOT IN
           (SELECT v
              FROM S);
```

```
-- 2
SELECT *
  FROM R
 WHERE NOT EXISTS
       (SELECT *
          FROM S
         WHERE R.v=S.v);
```

```
-- 3
SELECT *
  FROM R
EXCEPT
SELECT *
  FROM S;
```

1. {} (empty set)
2. {(NULL), (1)}
3. {(1)}

This question explores how SQL deals with NULL in logical set operations. For all pairings of R and S we have (R.v, S.v) each R.v as {(NULL, NULL), (NULL, 2)} and {(1, NULL), (1, 2)}. Note, if we can guarantee that there are no NULLs, all three of the above are logically equivalent.

1.
The semantics of "NOT IN" are such that the value being checked (R.v) is compared to every value in in the temporary table for equivalence. This follows standard 3-value logic outputting, so NULLs are not output when compared to.

- For R.v as NULL, NULL = x for all x will have the truth value of 0.5/NULL, so it is not output.
- For R.v as 1, 1 = x will have the truth value of 0.5/NULL on the instance 1 = NULL and a values of 0/FALSE on the instance 1 = 2, so it is not output.

2.
"NOT EXISTS" will just check if any tuples are returned from the subquery. In this case we are generating tuples in the subquery on the predicate R.v = S.v.

- For R.v as NULL, NULL = x for all x will have the truth value of 0.5/NULL, so the subquery outputs nothing, and thus NULL is output.
- For R.v as 1, 1 = x will have the truth value of 0.5/NULL on the instance 1 = NULL and a values of 0/FALSE on the instance 1 = 2, so the subquery outputs nothing, and thus 1 is output.

3.
The semantics of "EXCEPT" are not as closely tied to the prior two. This can be thought of the set difference where NULLs are treated as if they were a distinct constant value. It is also worth noting in future use that EXCEPT outputs a set, or in other words it automatically applies DISTINCT.

- NULL in R will be removed from the output because a NULL is in S.
- 1 in R is not removed because no value matches it in S.

The moral of the story is that SQL has many keywords that do very similar things. Their differences are how they handle 3-value logic.

B. (Set Operations and ALL keyword) Say we have the R and S again but now
R has tuples {(1), (2), (2), (2), (3), (4), (4)} and
S has tuples {(2), (3), (4), (4), (4), (5)}.
What are the outputs of the following queries?

| | | | | |
|---|---|---|---|---|
| `SELECT *` `FROM R` **`UNION`** `SELECT *` `FROM S;` | `SELECT *` `FROM R` **`UNION ALL`** `SELECT *` `FROM S;` | `SELECT *` `FROM R` **`INTERSECT`** `SELECT *` `FROM S;` | `SELECT *` `FROM R` **`INTERSECT ALL`** `SELECT *` `FROM S;` | `SELECT *` `FROM R` **`EXCEPT ALL`** `SELECT *` `FROM S;` |

| | | | | |
|---|---|---|---|---|
| {(1), (2), (3), (4), (5)} | {(1), (2), (2), (2), (3), (4), (4), (2), (3), (4), (4), (4), (5)} | {(2), (3), (4)} | {(2), (3), (4), (4)} | {(1), (2), (2)} |

The main idea is that including ALL switches set operations from set-semantics to bag-semantics. In some SQL engines, INTERSECT ALL and EXCEPT ALL are not normally supported. UNION ALL is widely supported.