University of Toronto
CSC343, Fall 2015

CDF IDs:
Names:

Nulls: Solutions

1. Suppose we have a table called Runnymede with the following content:

```
   name   | age | grade
----------+-----+-------
 diane    |     |     8
 will     |     |     8
 cate     |     |     1
 tom      |     |
 micah    |     |     1
 jamieson |     |     2
(6 rows)
```

   What is the output of each query below?

   (a) `select min(grade), max(grade), sum(grade), avg(grade), count(grade), count(*) from Runnymede;`

   (b) `select min(age), max(age), sum(age), avg(age), count(age), count(*) from Runnymede;`

   Solution:

```
csc343h-bogdan=> select min(grade), max(grade), sum(grade), avg(grade),
csc343h-bogdan-> count(grade), count(*)
csc343h-bogdan-> from Runnymede;
 min | max | sum |        avg         | count | count
-----+-----+-----+--------------------+-------+-------
   1 |   8 |  20 | 4.0000000000000000 |     5 |     6
(1 row)

csc343h-bogdan=> select min(age), max(age), sum(age), avg(age),
csc343h-bogdan-> count(age), count(*)
csc343h-bogdan-> from Runnymede;
 min | max | sum | avg | count | count
-----+-----+-----+-----+-------+-------
     |     |     |     |     0 |     6
(1 row)
```

   Aggregation ignores nulls. Notice that count(age) can report a value even though there is nothing but nulls in the age column.

2. We have tables R and T. Their contents are shown below.

```
   R            T
 a | b        b | c
```

```
---+---      ---+----
 1 | 2        2 |  5
 8 | 7        2 |  9
 5 |          1 |  4
   | 6          | 18
(4 rows)      6 | 88
             (5 rows)
```

What is the result of this query:

```
select * from R natural join T;
```

Solution:

```
csc343h-bogdan=> select * from R natural join T;
 b | a | c
---+---+----
 2 | 1 |  5
 2 | 1 |  9
 6 |   | 88
(3 rows)
```

Why are tuples (5, null) and (null, 18) excluded from the result, yet tuples (null, 6) and (6, 88) are included?

(5, null) and (null, 18) are excluded because when the natural join compares the 2 `null` values, they are not considered equal. It makes sense if you think of natural join as involving a Cartesian product and a WHERE, and if you remember that the truth value of a comparison involving nulls is always unknown and that WHERE is picky. In contrast, (null, 6) and (6, 88) are included because we are comparing on attribute `b`, which is 6 for both of these tuples, not `null`.

3. Suppose we have this table:

```
create table names(first text, last text, unique (first, last));
```

What will the result of the following be?

(a) Doing this twice: `insert into names values ('Diane', 'Horton');`

(b) Doing this twice: `insert into names values (null, 'Liu');`

Solution:

```
csc343h-bogdan=> insert into names values ('Diane', 'Horton');
INSERT 0 1
csc343h-bogdan=> insert into names values ('Diane', 'Horton');
ERROR:  duplicate key value violates unique constraint "names_pkey"
DETAIL:  Key (first, last)=(Diane, Horton) already exists.

csc343h-bogdan=> insert into names values (null, 'Liu');
INSERT 0 1
csc343h-bogdan=> insert into names values (null, 'Liu');
INSERT 0 1

csc343h-bogdan=> select * from names;
 first | last
-------+------
       | Liu
       | Liu
(2 rows)
```

The `unique` constraint will not allow two tuples with the same values for `first` and `last`. But in this context, two `null` values are considered to be different.

4. Suppose we have these tables:

```
csc343h-bogdan=> select * from R;
 a | b
---+---
 1 | 2
 8 | 7
 5 |
   | 6
(4 rows)

csc343h-bogdan=> select * from S;
 a | b
---+---
 3 | 4
 8 | 7
 5 |
   | 6
(4 rows)
```

   What will the result of the following be?

   (a) `(SELECT * from R) INTERSECT (SELECT * from S);`
   (b) `(SELECT * from R) EXCEPT (SELECT * from S);`

```
 a | b
---+---
 5 |
 8 | 7
   | 6
(3 rows)
```

   Since (5, null) appears in intersection, as does (null 6), we infer that the two nulls are considered to be equal in this context.

5.
```
 a | b
---+---
 1 | 2
(1 row)
```

   Ditto, (5, null) and (null, 6) don't appear in the set difference, implying that two nulls are considered to be equal for set operations.

Moral of the story

And the moral of this story is that you can never be too careful with `null`. It's extremely difficult to remember what exactly its impact will be in different contexts. Even when you have code you are confident in, never assume it will behave the same way in another DBMS.