“Division” in SQL

“Division” in SQL is not the typical mathematical idea. Instead it means to **find members of one set which participate with *every* member of another set.**

In the first example below, we will use the Professors database to **find professors who have an appointment in every department**.

Consider the following example tables:

Prof
pid
A
B
C
D

Dept
did
1
2
3

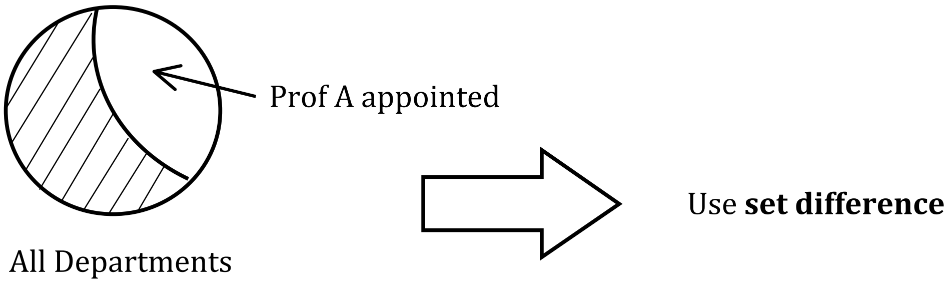
Appointment	
pid	did
A	2
B	1
B	2
B	3
C	1
C	3

By inspection of the tables, we can see that only professor B is appointed in all the departments.

The steps below show how to construct the SQL.

1) Suppose we want to know **only for professor A**, is A appointed in every department.

    a. Get a list of all departments where A is **NOT** appointed (shaded region):



```
SELECT did
FROM Department
WHERE did NOT IN (SELECT did
                  FROM Appointment
                  WHERE pid = ‘A’)
```

The answer is (1,3)

b. Determine if the answer to a) is empty using WHERE NOT EXISTS:

```
WHERE NOT EXISTS  (SELECT did
                   FROM Department
                   WHERE did NOT IN (SELECT did
                                     FROM Appointment
                                     WHERE pid = ‘A’))
```

This translates to WHERE NOT EXISTS (1,3), which returns FALSE, because the list exists.

2) Now generalize to all professors (note the use of p.pid):

```
SELECT p.pid
FROM Professor p
WHERE NOT EXISTS  (SELECT did
                   FROM Department
                   WHERE did NOT IN (SELECT did
                                     FROM Appointment
                                     WHERE pid = p.pid))
```

This query will process for each pid in Professor the query in part b) above. It is called a **correlated subquery** because the inner query uses a value from the outer query.

Alternate method (won’t always apply)

In this case, we could use a simpler method which just counts, for each professor, the number of departments where they have an appointment:

```
SELECT pid
FROM Appointment
GROUP BY pid
HAVING COUNT(*) = (SELECT COUNT (did)
                   FROM Department)
```

In the second example below, we will use the Editors database to **find reviewers who were asked to review in every year from 2011-2013.**

Consider the following example tables:

Name

id
A
B
C
D

Reviewers

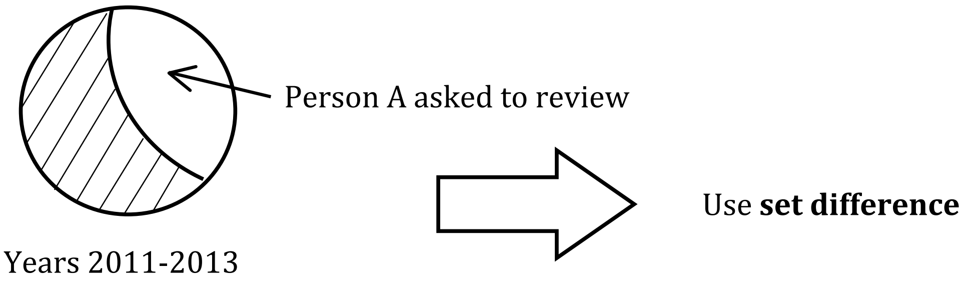
reviewerid,	year,	narid
A	2011	1
B	2011	2
C	2011	3
C	2011	4
A	2012	5
B	2012	6
C	2012	7
D	2013	9
A	2013	10
B	2013	11

By inspection of the tables, we can see that only reviewers A and B were asked to review in all three years. However, note that C was asked to review three times, so in this case, simple counting won't give the correct answer.

The steps below show how to construct the SQL.

2) Suppose we want to know **only for person A**, was A asked to review in each of the three years 2011-2013.

    a. Get a list of all years where A is **NOT** asked to review (shaded region):



```
SELECT year
FROM Papers
WHERE year in (2011, 2012, 2013)
     AND year NOT IN (SELECT year
                                   FROM Reviewers
                                   WHERE reviewerid = 'A')
```

The answer is the empty list ( ).

b. Determine if the answer to a) is empty using WHERE NOT EXISTS:

```
WHERE NOT EXISTS  (SELECT year
                   FROM Papers
                   WHERE year in (2011, 2012, 2013)
                     AND year NOT IN (SELECT year
                                      FROM Reviewers
                                      WHERE reviewerid = ‘A’))
```

This translates to WHERE NOT EXISTS ( ), which returns TRUE, because the list does not exist (is empty).

2) Now generalize to all professors (note the use of n.id):

```
SELECT n.id
FROM Name n
WHERE NOT EXISTS  (SELECT year
                   FROM Papers
                   WHERE year in (2011, 2012, 2013)
                       AND year NOT IN (SELECT year
                                        FROM Reviewers
                                        WHERE reviewerid = n.id))
```

This query will process for each id in Name the query in part b above. It is called **correlated subquery** because the inner query uses a value from the outer query.