# SET OPERATOR PRACTICE

The location table has 23 rows in it.

1 Using the location table, find entries where values in the city column also appear in the state column. Use a set operator.

SELECT city

FROM locations

INTERSECT

SELECT state

FROM locations;

CITY

-------------------------

Oxford

Sao Paulo

Utrecht

2 Write another query but do NOT use SET operator.

SELECT city

FROM locations

WHERE city = state;

3 Show the same queries, but show city and state names

4 Write a query that shows the number of city and state names there are. You will need to add up the number of each. Since you know there are 23 rows, then the answer should be 46.

SELECT count(\*)

FROM

(

SELECT city

FROM locations

UNION ALL

SELECT state

FROM locations

);

5 Write the query so that only unique names are counted.

SELECT count(\*)

FROM

(

SELECT city

FROM locations

UNION

SELECT state

FROM locations

);

The result is 38, but there was only 3 where city and state were the same

23 cities and 17 states

Why 38 from a total of 46.

There are 6 nulls in state and no nulls in city.

46 -6nulls – 2 duplicates (--- there were 3 entries the same so only 2 are duplicates) = 38

# CODE EXAMPLES with and without SET operators

Using PRODUCT\_CATEGORIES and PRODUCTS

select count(category\_id)

from product\_categories; ------ 5

select count(category\_id)

from products; ------ 288

This next example using the MINUS operator is equivalent to using an antijoin (NOT IN clause) between the two tables.

**select category\_id**

**from product\_categories**

**MINUS**

**select category\_id**

**from products**;

CATEGORY\_ID

-----------

3 Category 3 is not in both

**select category\_id**

**from product\_categories**

**WHERE category\_id NOT IN (SELECT category\_id**

**FROM products)**

This next example using the INTERSECT operator. The equivalent query that selects distinct rows from the tables. .

**select category\_id**

**from product\_categories**

**INTERSECT**

**select category\_id**

**from products;**

CATEGORY\_ID

-----------

1

2

4

5

Same example using a JOIN and a DISTINCT

**select distinct category\_id**

**from product\_categories JOIN products**

**using(category\_id)**

OTHER EXAMPLES  
This next example using the UNION operator, which does not preserve duplicates, is shown in comparison to an analogous query that uses a full outer join and the distinct operator to remove duplicates.

## Code Sample:

Using-Set-Operators/Demos/union.sql

SELECT distinct job\_id

FROM jobs

UNION

SELECT distinct job\_id

FROM job\_history;

-- This is equivalent to a full outer join with a distinct

-- operation to eliminate duplicates

SELECT DISTINCT j.job\_id

FROM jobs j

FULL OUTER JOIN job\_history jh

ON j.job\_id = jh.job\_id;

The UNION ALL operator preserves duplicates between both of the tables. Because it preserves all duplicates and does not involve a join, there is no simple similar query using a join to provide the same results:

## Code Sample:

Using-Set-Operators/Demos/union\_all.sql

SELECT distinct job\_id

FROM jobs

UNION ALL

SELECT distinct job\_id

FROM job\_history

ORDER BY job\_id;