# 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;