Table of Contents

We have seen a few ways to produce a result based on data from more than one table. With a join of two tables, each row in the result can have data from the two tables being joined. The set operators provide a different way to associate data from two tables. The set operators are UNION ALL, UNION, INTERSECT and EXCEPT. With a Union operator, each row in the result comes from one or the other of the two tables. The rows being "unioned" are returned combined into a single result.

1. Theoretical concepts

First we should consider the terms Set and Bag

A Set is a collection of data values, without any ordering and with no repeated values. A Bag (or MultiSet). is a collection of data values, without any ordering but it can have repeated values.

Demo

Suppose we have the following collections: (You could think of this as two children with their bug collections)

Collection_1 (ant, ant, ant, beetle, cricket, cricket)

Collection_2 (ant, cricket, earwig, flea, cricket)

These are multisets/bags since they have duplicates.

One way to combine these two collections is just to dump them all into one bag

Result_Collection_3 (ant, ant, ant, ant, beetle, cricket, cricket, cricket, cricket, earwig, flea)

Another way to combine these two collections is just to get one of each type and put them together

Result_Collection_4 (ant, beetle, cricket, earwig, flea)

Result_Collection_3 is called a Union All collection and Result_Collection_4 is a Union Distinct; Distinct suppresses duplicates.

We could make a collection of all values that are in both sets

Result_Collection_5 (ant, cricket)

That is called an Intersection.

But we could think of this in a slightly different way and come up with the following

Result_Collection_6 (ant, cricket, cricket)

because there are two crickets in Collection_1 and two crickets in Collection_2. This is also an Intersection. We go to collection _1 and match ant (1) and ant(2), then we match 2-crickets(1) and 2 crickets(2)

We can classify Result_Collection_5 as an Intersection Distinct (no duplicates) and Result_Collection_6 as an Intersection All.

So far all of these operations have been commutative- that means that Collection_1 Union Collection_2 has the same meaning as Collection_2 Union Collection_1. Intersection is also commutative. There is another way to work with these collections and that is shown here

Result_Collection_7 (beetle) This is the values in Collection_1 that are not in Collection_2. This is not commutative.

Result_Collection_8 (earwig, flea) is the values in Collection_2 that are not in Collection_1. This uses the Except Distinct operator.

Result_Collection_9 (ant, ant, beetle) is the non-distinct set of values in Collection_1 that are not in Collection_2; this uses Except All.

Now consider what should happen if there are nulls in the original collections. (maybe the children have unidentified bugs.)

Collection_1v2 (ant, ant, ant, beetle, cricket, cricket, unknown-insect)

Collection_2v2 (ant, cricket, earwig, flea, cricket, unknown-insect, unknown-insect)

A Union Distinct result set contains only a single Null. Although we know that nulls are not equal to each other, for many situations SQL lumps the nulls together. A Union All results set contain a null for each null in one of the original collections.

If you have used Venn Diagrams before and found them helpful, you might want to look at the end of this document next and then come back to the SQL.

2. SQL Concepts and Rules

To combine the result sets of two Select statements with a set operator, the two result sets must be **union compatible**. The result sets

- must have the same number of attributes and
- must have the same (or convertible) data types for the corresponding columns.

Although it is not a syntax requirement that the corresponding columns have the same domain, you have the responsibility to make the output meaningful.

Demo 01: Suppose we wanted to get some data for the snakes and lizards from the vets animals table. We would write the query:

```
select cl_id, an_id, an_name, an_type
from vt_animals
where an_type in ('snake', 'lizard');
```

CL_ID	AN_ID	AN_NAME	AN_TYPE
411	15401	Pinkie	lizard
3561	17025	25	lizard
7152	17026	3P#_26	lizard
7152	17027	3P#_25	lizard
3561	21004	Gutsy	snake
1852	21007	Pop 22	snake

Demo 02: We could write this as a Union query. There is no particular advantage in using the Union here- but SQL often has more than one way to solve a task

```
select cl_id, an_id, an_name, an_type
from vt_animals
where an_type = 'snake'
UNION
select cl_id, an_id, an_name, an_type
from vt_animals
where an_type = 'lizard'
order by cl_id;
```

Demo 03: A common mistake in using a set operator is not have the same number of columns.

```
select cl_id, an_id, an_name, an_type
from vt_animals
where an_type = 'snake'
UNION
select cl_id, an_id, an_name
from vt_animals
where an_type = 'lizard'
```

```
order by cl_id;
```

```
SQL Error: ORA-01789: query block has incorrect number of result columns
```

Demo 04: The following will give us an error because an_type is a string (in the first part) and an_dob is a date value (in the second part)

```
select cl_id, an_id, an_name, an_type
from vt_animals
where an_type = 'snake'
UNION
select cl_id, an_id, an_name, an_dob
from vt_animals
where an_type = 'lizard'
order by cl_id;
```

```
SQL Error: ORA-01790: expression must have same datatype as corresponding expression
```

Demo 05: This is not a very sensible query but I now have two string columns, so this works. The output is not very meaningful to most people.

```
select cl_id, an_id, an_name, an_type
from vt_animals
where an_type = 'snake'
UNION
select cl_id, an_id, an_name , to_char(an_dob, 'YYYY-MM-DD')
from vt_animals
```

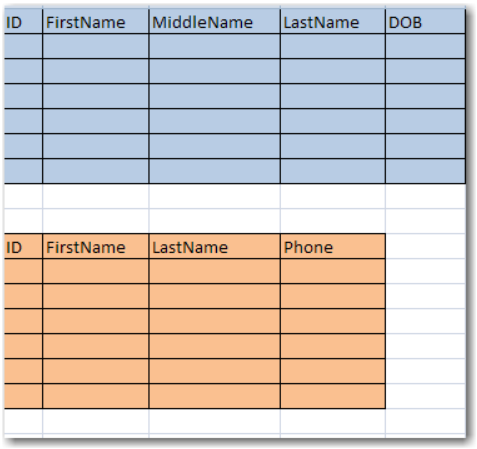
```
where an_type     CL_ID      AN_ID AN_NAME                   AN_TYPE
---------- ---------- ---------- ------------------------- --------------------------
       411      15401 Pinkie                    2010-03-15
      1852      21007 Pop 22                    snake
      3561      17025 25                        2013-01-10
      3561      21004 Gutsy                     snake
      7152      17026 3P#_26                    2010-01-10
      7152      17027 3P#_25                    2010-01-10

 6 rows selected
```

```
= 'lizard'
order by cl_id;
```

Suppose we have different numbers of columns to display in each part of the set query. We can always play tricks such as adding a literal column to one part of the union.

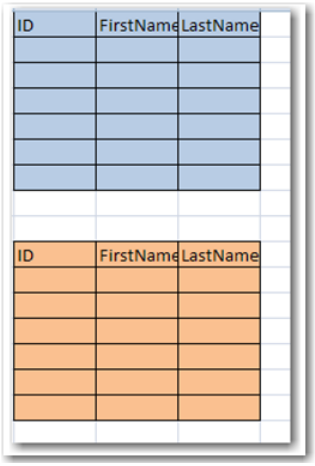
Suppose I had two tables (the blue table and the orange table) with some differences in the attributes.



If I want to combine these tables with the set operators, I have several choices.

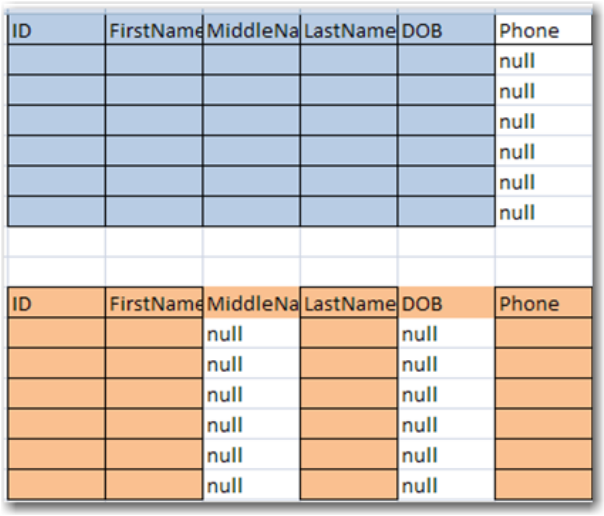
- I could select only the columns found in both tables.

```
   select ID, FirstName, LastName
   from tblBlue
Union
   select ID, FirstName, LastName
   from tblOrange
```



- I could return a default value- probably nulls, for the columns found in one table or the other.

```
   select ID, FirstName, MiddleName, LastName , DOB, null
   from tblBlue
Union
   select ID, FirstName, null, LastName , null, Phone
   from tblOrange
```



In the return table, the column headers and the Order By clause are based on the first Select columns and aliases. Oracle follows the standards in that the individual selects cannot be ordered- but the final result can be sorted.

If you have a query that uses more than one of these operators, the order of operations is top to bottom. You can use parentheses to change this order. Be careful with this- it is nonstandard and may change with future Oracle versions- use parentheses.

Demo 06: A somewhat more useful example of a Union query. Suppose we want the name of all the people the vet clinic deals with- both staff and clients. You can also see that the column aliases come from the first query.

```
select stf_name_last as LastName, stf_name_first as FirstName
, 'staff' as Position
from vt_staff
UNION
select cl_name_last, cl_name_first
, 'client'
from vt_clients
order by LastName, FirstName
```

Sample rows

```
LASTNAME                 FIRSTNAME                POSITION
------------------------ ------------------------ --------
. . .
Dolittle                 Eliza                    staff
Drake                    Donald                   client
Gordon                   Dexter                   staff
Halvers                  Pat                      staff
Harris                   Eddie                    client
Hawkins                  Coleman                  client
Helfen                   Sandy                    staff
Horn                     Shirley                  staff
```

UNION ALL- returns all of the rows from each of the queries.

UNION- returns all of the rows but removes any duplicates.

INTERSECT- returns rows that are part of both of the return sets for the component queries.

MINUS- returns rows that were returned by the first Select and not by the second. (The ansi standard term is Except; Oracle uses the keyword Minus)

Oracle does not implement Intersect All and Except All

3. Some Simple Set operations

In these demos we only want to get the client id for clients with different types of animals.

3.1. Union

Demo 07: This uses a Union; we get 4 rows returned. (In the earlier demos we got 6 rows for snakes and lizards)

```
select cl_id
from vt_animals
where an_type = 'snake'
UNION
select cl_id
from vt_animals
where an_type = 'lizard'
order by cl_id;
```

```
      CL_ID
-----------
        411
       1852
       3561
       7152
```

3.2. Union All

Demo 08: the Union operator removes duplicate rows; the Union All operators leaves in duplicate rows. Union All is more efficient if having duplicate rows is not a problem for the desired result.

```
select cl_id
from vt_animals
where an_type = 'snake'
UNION ALL
select cl_id
from vt_animals
where an_type = 'lizard'
order by cl_id;
```

```
      CL_ID
-----------
        411
       1852
       3561
       3561
       7152
       7152
```

3.3. Intersect

Demo 09: The Intersect operator returns the rows that are in both query results. Here these are people with both a lizard and a snake. The intersect operator is commutative- we get the same result if we ask for people who have a snake and have a lizard as when we ask for people who have a lizard and have a snake.

```
select cl_id
from vt_animals
where an_type = 'snake'
INTERSECT
select cl_id
from vt_animals
where an_type = 'lizard'
order by cl_id;
```

```
      CL_ID
-----------
       3561
```

3.4. Minus

Demo 10: The Minus operator returns rows that are in one query result, but not in the other query result. This is no commutative. We get different results if we ask for people who have a snake and do not have a lizard then when we ask for people who have a lizard and do not have a snake.
Snake - no lizard

```
select cl_id
from vt_animals
where an_type = 'snake'
MINUS
select cl_id
from vt_animals
where an_type = 'lizard'
order by cl_id;
```

```
      CL_ID
-----------
       1852
```

Demo 11: Lizard- no snake

```
select cl_id
from vt_animals
where an_type = 'lizard'
MINUS
select cl_id
from vt_animals
where an_type = 'snake'
order by cl_id;
```

```
      CL_ID
-----------
        411
       7152
```

4. Different ways to accomplish a task

Demo 12: A union is similar to an OR- clients who have a snake or a lizard

```
select Distinct cl_id
from vt_animals
where cl_id in (
   select cl_id
   from vt_animals
   where an_type = 'snake'
   )
OR cl_id in (
   select cl_id
   from vt_animals
   where an_type = 'lizard'
   )
order by cl_id;
```

```
      CL_ID
-----------
        411
       1852
       3561
       7152
```

Demo 13: An intersect is similar to an AND- clients who have a snake and a lizard.

```
select Distinct cl_id
from vt_animals
where cl_id in (
   select cl_id
   from vt_animals
   where an_type = 'snake'
   )
AND cl_id in (
   select cl_id
   from vt_animals
   where an_type = 'lizard'
   )
```

```
order by cl_id;
```

```
      CL_ID
----------
      3561
```

Demo 14: The MINUS is similar to an IN and Not IN- clients who have a snake and do not have a lizard

```
select Distinct cl_id
from vt_animals
where cl_id in (
   select cl_id
   from vt_animals
   where an_type = 'snake'
   )
AND cl_id NOT  in (
   select cl_id
   from vt_animals
   where an_type = 'lizard'
   )
order by cl_id;
```

```
      CL_ID
----------
      1852
```

You are not limited to one set operator. You can write queries to find who has a cat and a dog and a dormouse. Who has a cat but not a bird and not a hedgehog.

This document has dealt with very simple set queries to establish the logic patterns. We will look at a few more complex queries in the next document.