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. (MySQL currently supports only the Union and Union All operators. We will implement the other set operators using other techniques.)

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.

1.1. 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.

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   |
+-------+-------+---------+---------+
6 rows in set (0.00 sec)
```

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;
```

```
ERROR 1222 (21000): The used SELECT statements have a different number of columns
```

Demo 04: The following will give us an will work in MySQL since it is more "friendly" about casting data types. This will not work in some other dbms because an_type is a string (in the first part) and an_dob is a date value (in the second part)

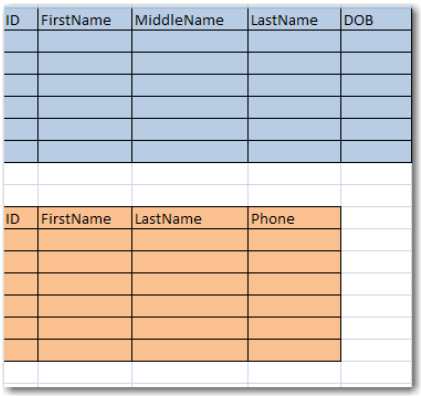
```
select cl_id, an_id, an_name, an_type
from vt_animals
where an_type = 'snake'
UNION
select cl_id, an_id, an_name, an_dob
from vt_animals
where an_type = 'lizard'
order by cl_id;
+-------+-------+---------+------------+
| cl_id | an_id | an_name | an_type    |
+-------+-------+---------+------------+
|   411 | 15401 | Pinkie  | 2010-03-15 |
|  1852 | 21007 | Pop 22  | snake      |
|  3561 | 17025 | 25      | 2013-01-10 |
|  3561 | 21004 | Gutsy   | snake      |
|  7152 | 17027 | 3P#_25  | 2010-01-10 |
|  7152 | 17026 | 3P#_26  | 2010-01-10 |
+-------+-------+---------+------------+
```

Demo 05: You can do an explicit case 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, cast(an_dob as char(10))
from vt_animals
where an_type = '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.

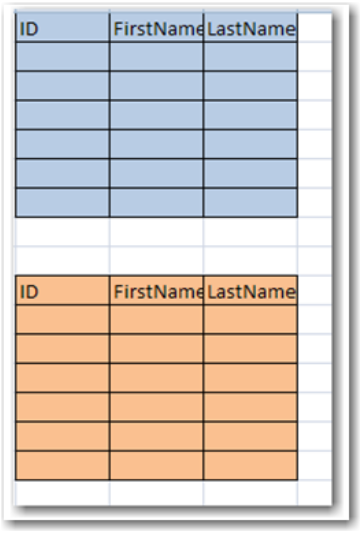


If I want to combine these tables with the set operators, I have several choices.

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

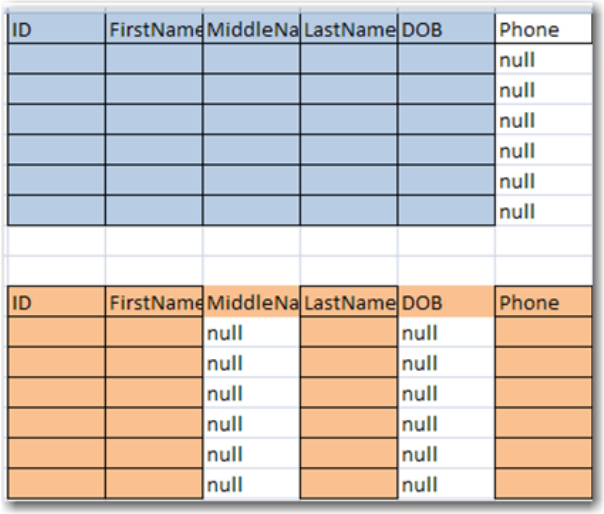
```
selectID, FirstName, LastName
from tblBlue
Union
```

```
selectID, FirstName, LastName
from tblOrange
```



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

```
selectID, FirstName, MiddleName, LastName , DOB, null
from tblBlue
Union
selectID, 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.

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.

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    |
| Monk        | Theo      | client   |
| Montgomery  | Wes       | client   |
```

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

UNION or UNION DISTINCT- returns all of the rows but removes any duplicates. These two terms are interchangeable. Since some other dbms do not allow the use of the word Distinct here and it is the default, you might wish to avoid it.

MySQL does not directly implement the other set operators. We will discuss ways to implement the other set operators.

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 |
+-------+
```

Demo 09: 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 |
+-------+
```

3.3. Implementing an Intersect

Demo 10: The Intersect operation returns the rows that are in both query results. Here these are people with both a lizard and a snake. 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 |
+-------+
```

3.4. Implementing an Except

Demo 11: The Except operation returns rows that are in one query result, but not in the other query result. This is not 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. is similar to an IN and Not IN- clients who have a snake and do not have a lizard.

Snake - no 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 |
+-------+
```

Demo 12: Lizard- no snake

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

```
|   411 |
|  7152 |
+-------+
```

4. Casting to handle syntax rules

MySQL does a lot of automatic casting of data types. The following union query has the first column as an integer in the first subquery and as a string in the second subquery. Since MySQL handles this cast for you, you can run the query shown below.

In some cases you might need to use a cast function to get compatible types.

Demo 13:

```
   select prod_id  AS "Product ID"
   , prod_list_price as "List Price"
   from Product.products
   where catg_id = 'APL'
UNION  ALL
   select '---- avg Price for all Appliances ----'
   , avg(prod_list_price)
   from Product.products
   where catg_id = 'APL' ;
+---------------------------------------+------------+
| Product ID                            | List Price |
+---------------------------------------+------------+
| 1120                                  | 549.990000 |
| 1125                                  | 500.000000 |
| 1126                                  | 850.000000 |
| 1130                                  | 149.990000 |
| 4569                                  | 349.950000 |
| ---- Avg Price for all Appliances ---- | 479.986000 |
+---------------------------------------+------------+
```

5. Sorting

MySQL supports an extension that gives you the choice of sorting the individual subqueries or the overall subquery but the results might not be what you expect.

Demo 14: Set up the following two tables in the a_testbed database.

```
Create table a_testbed.z_tst_set_emp ( E_id int, E_name varchar(10), E_city
varchar(10));

Create table a_testbed.z_tst_set_cust ( C_id int, C_name varchar(10), C_city
varchar(10));

Insert into a_testbed.z_tst_set_emp values ( 101, 'Jones',     'Chicago');
Insert into a_testbed.z_tst_set_emp values ( 102, 'Anderson',  'Chicago');
Insert into a_testbed.z_tst_set_emp values ( 103, 'Baxter',    'Chicago');
Insert into a_testbed.z_tst_set_emp values ( 104, 'Johnson',   'Chicago');
Insert into a_testbed.z_tst_set_emp values ( 105, 'Miller',    'Chicago');

Insert into a_testbed.z_tst_set_cust values ( 201, 'Oliver',   'Boston');
Insert into a_testbed.z_tst_set_cust values ( 202, 'Athena',   'Boston');
Insert into a_testbed.z_tst_set_cust values ( 203, 'Sanders',  'Boston');
Insert into a_testbed.z_tst_set_cust values ( 204, 'Baxter',   'Boston');
```

Demo 15: Do a plain union query; this has not specified any ordering so we should not assume any row order.

```
select E_id, E_name, E_city
from a_testbed.z_tst_set_emp
Union all
select C_id, C_name, C_city
from a_testbed.z_tst_set_cust;
+------+----------+---------+
| E_id | E_name   | E_city  |
+------+----------+---------+
|  101 | Jones    | Chicago |
|  102 | Anderson | Chicago |
|  103 | Baxter   | Chicago |
|  104 | Johnson  | Chicago |
|  105 | Miller   | Chicago |
|  201 | Oliver   | Boston  |
|  202 | Athena   | Boston  |
|  203 | Sanders  | Boston  |
|  204 | Baxter   | Boston  |
+------+----------+---------+
```

Demo 16: Now do a sort at the end for the query and the final result set is sorted.

```
select E_id, E_name, E_city
from a_testbed.z_tst_set_emp
Union all
select C_id, C_name, C_city
from a_testbed.z_tst_set_cust
order by E_name;
+------+----------+---------+
| E_id | E_name   | E_city  |
+------+----------+---------+
|  102 | Anderson | Chicago |
|  202 | Athena   | Boston  |
|  103 | Baxter   | Chicago |
|  204 | Baxter   | Boston  |
|  104 | Johnson  | Chicago |
|  101 | Jones    | Chicago |
|  105 | Miller   | Chicago |
|  201 | Oliver   | Boston  |
|  203 | Sanders  | Boston  |
+------+----------+---------+
```

Demo 17: Now add a sort to each subquery. You need to enclose the subqueries within parentheses. But the final result is not sorted. That is because the Union operator does not produce a sorted result so that operator removed the ordering of the rows.

```
(select E_id, E_name, E_city
 from a_testbed.z_tst_set_emp
 order by E_name)
Union all
(select C_id, C_name, C_city
 from a_testbed.z_tst_set_cust
 order by c_name);
+------+----------+---------+
| E_id | E_name   | E_city  |
+------+----------+---------+
|  101 | Jones    | Chicago |
|  102 | Anderson | Chicago |
```

```
|  103 | Baxter   | Chicago |
|  104 | Johnson  | Chicago |
|  105 | Miller   | Chicago |
|  201 | Oliver   | Boston  |
|  202 | Athena   | Boston  |
|  203 | Sanders  | Boston  |
|  204 | Baxter   | Boston  |
+------+----------+---------+
```

Demo 18: The use of order by in the subqueries is used mainly with a limit clause. Now I get the first two sorted names lists for each of the subqueries.

```
(select E_id, E_name, E_city
 from a_testbed.z_tst_set_emp
 order by E_name Limit 2)
Union all
(select C_id, C_name, C_city
 from a_testbed.z_tst_set_cust
 order by c_name Limit 2)
;
+------+----------+---------+
| E_id | E_name   | E_city  |
+------+----------+---------+
|  102 | Anderson | Chicago |
|  103 | Baxter   | Chicago |
|  202 | Athena   | Boston  |
|  204 | Baxter   | Boston  |
+------+----------+---------+
```