

Table of Contents

1. Theoretical concepts	1
1.1. Demo	1
2. SQL Concepts and Rules	3
3. Union All	6
4. Union All versus Union	8
5. Casting to handle syntax rules	9
6. Sorting	10
7. Venn Diagrams	12

We have seen a few ways to produce a result set based on data from more than one table. With a join of two tables, each row in the result set 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 set comes from one or the other of the two tables. The rows being "unioned" are returned combined into a single result set. (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 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.

Suppose we wanted to get the names of the cats and dogs from the vets animals table. We would write the query:

```
Select an_id, an_name
From a_vets.vt_animals
Where an_type in ( 'cat', 'dog' )
;
```

That is actually the best way to write this query, but we could write this as a Union query as in demo 01. Using the union would take the query about 4 times as long to run- but it is a simple example of a Union query.

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 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: Using the vets animals table, we could write the following and we would get the ID and names of the cats and dogs

```
Select an_id, an_name
From a_vets.vt_animals
Where an_type = 'cat'
UNION
Select an_id, an_name
From a_vets.vt_animals
Where an_type = 'dog'
;
+-----+-----+
| an_id | an_name |
+-----+-----+
| 10002 | Gutsy   |
| 16003 | Ursula  |
| 16004 | Napper  |
| 21314 | Adalwine|
| 21315 | Baldric |
| 21316 | Etta    |
| 21317 | Manfred |
| 21318 | Waldrom |
| 15165 | Burgess |
| 19845 | Pinkie  |
| 21003 | Calvin  |
+-----+-----+
```

Demo 02: The following should give us an error because an_name is a string and an_dob is a date value. MySQL is more flexible about type conversions than some other dbms. Understand- this is a bad query- the fact that mysql runs this does not make it a good query.

```

Select an_id, an_name
From a_vets.vt_animals
Where an_type = 'cat'
UNION
Select an_id, an_dob
From a_vets.vt_animals
Where an_type = 'dog';

```

```

+-----+-----+
| an_id | an_name |
+-----+-----+
| 10002 | Gutsy   |
| 16003 | Ursula  |
| 16004 | Napper  |
| 21314 | Adalwine|
| 21315 | Baldric |
| 21316 | Etta    |
| 21317 | Manfred |
| 21318 | Waldrom |
| 15165 | 2005-11-20 |
| 19845 | 2009-02-02 |
| 21003 | 2004-06-18 |
+-----+-----+

```

Demo 03: It would be more standard to do a type conversion. This is still a rather silly query.

```

Select an_id, an_name
From a_vets.vt_animals
Where an_type = 'cat'
UNION
Select an_id, cast(an_dob as char(10))
From a_vets.vt_animals
Where an_type = 'dog';

```

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.

ID	FirstName	MiddleName	LastName	DOB

ID	FirstName	LastName	Phone

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
```

[illegible]

- 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
```

ID	FirstName	MiddleNa	LastName	DOB	Phone
					null
					null
					null
					null
					null
					null
ID	FirstName	MiddleNa	LastName	DOB	Phone
		null		null	
		null		null	
		null		null	
		null		null	
		null		null	
		null		null	

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.

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. See the discussion in the document on Subqueries and Set Operations for ways to implement the other set operators.

3. Union All

Demo 04: Query to show relevant orders

```
Select ord_id, prod_id, catg_id, prod_name
From a_oe.order_details
Join a_prd.products using (prod_id)
Where catg_id in ('PET', 'SPG')
order by ord_id
;
```

selected rows

ord_id	prod_id	catg_id	prod_name
105	1020	SPG	Dartboard
105	1030	SPG	Basketball
105	1010	SPG	Weights
106	1060	SPG	Mountain bike
111	1150	PET	Cat exerciser
111	1141	PET	Bird cage- deluxe
117	1141	PET	Bird cage- deluxe
117	1030	SPG	Basketball
117	1150	PET	Cat exerciser
120	1010	SPG	Weights
121	1010	SPG	Weights
121	1040	SPG	Treadmill
124	1151	PET	Cat pillow

Demo 05: Union All to return orders with either a Sporting goods item or a pet supply item or both. This example is simply to show the Union technique. This also shows the use of the Order by clause at the very end of the query, where it applies to the final result set.

```
Select ord_id, prod_id, catg_id, prod_name
From a_prd.products
Join a_oe.order_details using (prod_id)
Where catg_id = 'PET'
```

UNION ALL

```
Select ord_id, prod_id, catg_id, prod_name
From a_prd.products
Join a_oe.order_details using (prod_id)
Where catg_id = 'SPG'
order by ord_id;
```

selected rows

ord_id	prod_id	catg_id	prod_name
105	1010	SPG	Weights
105	1020	SPG	Dartboard
105	1030	SPG	Basketball
106	1060	SPG	Mountain bike
111	1141	PET	Bird cage- deluxe
111	1150	PET	Cat exerciser
117	1150	PET	Cat exerciser
117	1141	PET	Bird cage- deluxe
117	1030	SPG	Basketball

Demo 06: The attempt to do Union All that has a syntax problem. The various selects need to have the same number of columns of compatible types

```
Select ord_id, prod_id, catg_id, prod_name
From a_prd.products
Join a_oe.order_details using (prod_id)
Where catg_id = 'PET'
```

UNION ALL

```
Select ord_id, prod_id
From a_prd.products
Join a_oe.order_details using (prod_id)
Where catg_id = 'SPG'
order by ord_id;
```

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

Demo 07: Union All from two tables

```
Select name_last, name_first, 'Employee' as "Pers Type"
From a_emp.employees
```

UNION ALL

```
Select cust_name_last, cust_name_first, 'Customers'
From a_oe.customers;
```

Selected rows

name_last	name_first	Pers Type
King	Steven	Employee
Koch	Karen	Employee
D'Haa	Helen	Employee

Hunol	Maria	Employee
Ernst	Oliver	Employee
Green	Oliver	Employee
Fiet	Robert	Employee
Chen	Helen	Employee
Russ	Oliver	Employee

... rows removed

McCoy	Tyner	Customers
Jones	Elton John	Customers
Williams	Sally	Customers
Otis	Elisha	Customers
Hamilton	Alexis	Customers
Stevenson	James	Customers
Stevenson	JAMES	Customers
O'Leary	Mary	Customers
O'Leary	Mary	Customers
Olmsted	Frederick	Customers
Button	D. K.	Customers

... rows removed

4. Union All versus Union

Demo 08: Union ALL to return HW items purchased with either a Nov or Dec date. Union all is more efficient since it does not have to remove duplicates. These queries ignore years of the order date.

```

Select prod_id, catg_id, prod_name
From a_prd.products
Join a_oe.order_details using (prod_id)
Join a_oe.order_headers using (ord_id)
Where catg_id = 'HW'
and extract(month From ord_date) = 11

```

UNION ALL

```

Select prod_id, catg_id, prod_name
From a_prd.products
Join a_oe.order_details using (prod_id)
Join a_oe.order_headers using (ord_id)
Where catg_id = 'HW'
and extract(month From ord_date) = 12
order by prod_id;

```

prod_id	catg_id	prod_name
1000	HW	Hand Mixer
1070	HW	Iron
1071	HW	Iron
1080	HW	Cornpopper
1080	HW	Cornpopper
1080	HW	Cornpopper
1090	HW	Gas grill
1100	HW	Blender
1100	HW	Blender
1100	HW	Blender
1110	HW	Pancake griddle
1110	HW	Pancake griddle

Demo 09: Union to return HW items purchased with either a Nov or Dec date eliminating duplicates.

```

Select prod_id, catg_id, prod_name
From a_prd.products
Join a_oe.order_details using (prod_id)
Join a_oe.order_headers using (ord_id)
Where catg_id = 'HW'
and extract(month From ord_date) = 11
UNION
Select prod_id, catg_id, prod_name
From a_prd.products
Join a_oe.order_details using (prod_id)
Join a_oe.order_headers using (ord_id)
Where catg_id = 'HW'
and extract(month From ord_date) = 12 ;

```

prod_id	catg_id	prod_name
1000	HW	Hand Mixer
1070	HW	Iron
1071	HW	Iron
1080	HW	Cornpopper
1100	HW	Blender
1110	HW	Pancake griddle
1090	HW	Gas grill

5. 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 10:

```

Select prod_id AS "Product ID"
, prod_list_price as "List Price"
From a_prd.products
Where catg_id = 'APL'
UNION ALL
Select '---- Avg Price for all Appliances ----'
, Avg(prod_list_price)
From a_prd.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

6. 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 11: Set up the following two tables in the a_testbed database.

```
Create table d_set_emp ( E_id int, E_name varchar(10), E_city varchar(10));

Create table d_set_cust ( C_id int, C_name varchar(10), C_city varchar(10));

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

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

Demo 12: 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 d_set_emp
Union all
select C_id, C_name, C_city
From d_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 13: 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 d_set_emp
Union all
select C_id, C_name, C_city
From d_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

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.

Demo 14:

```
(select E_id, E_name, E_city
From d_set_emp
order by E_name)
Union all
(select C_id, C_name, C_city
From d_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

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.

Demo 15:

```
(select E_id, E_name, E_city
From d_set_emp
order by E_name Limit 2)
Union all
(select C_id, C_name, C_city
From d_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

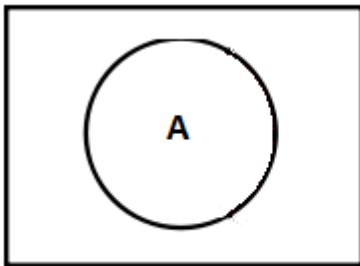
Demo 16:

```
(select E_id, E_name, E_city
From d_set_emp
Order by E_name Limit 200)
Union all
(select C_id, C_name, C_city
From d_set_cust
order by c_name Limit 200);
```

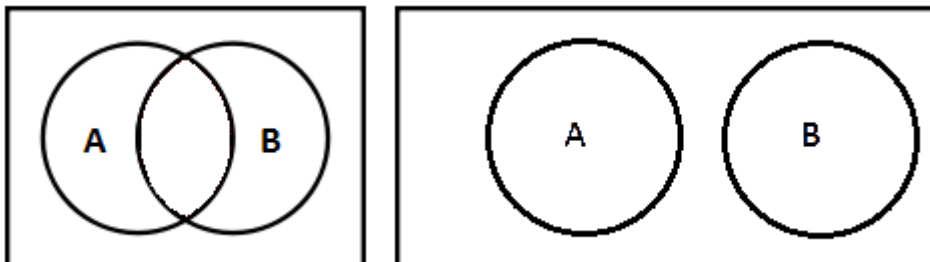
E_id	E_name	E_city
102	Anderson	Chicago
103	Baxter	Chicago
104	Johnson	Chicago
101	Jones	Chicago
105	Miller	Chicago
202	Athena	Boston
204	Baxter	Boston
201	Oliver	Boston
203	Sanders	Boston

7. Venn Diagrams

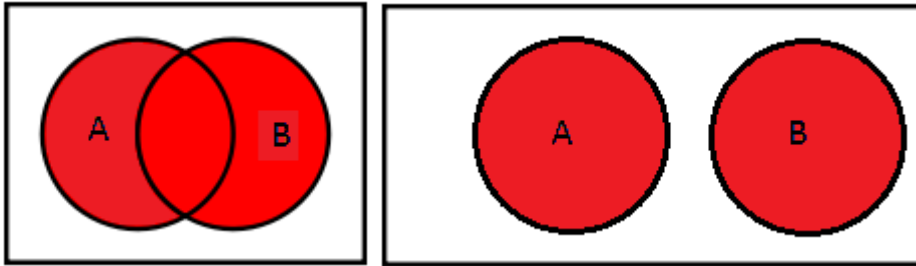
Some people understand concepts from Venn diagrams and some people do not. It is OK to ignore this section if you want. Suppose we think of the result set of a query as a circle. All of the rows in the result set are inside the circle.



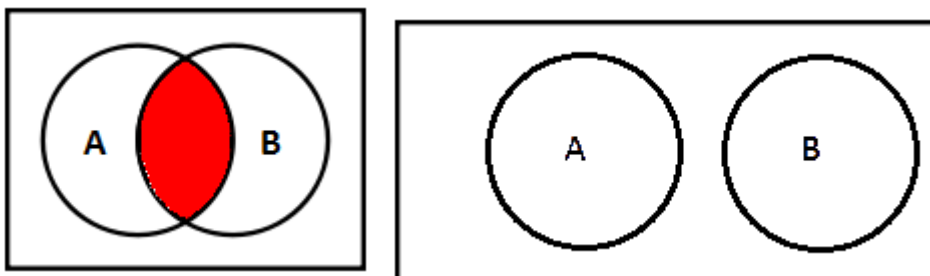
If we have two result sets it is possible that some rows might be in more than one of the result sets- the two circle would overlap. Or it might be that there is no overlap.



If we want all of the rows that are in either of the two results sets- that is the UNION



If we want all of the rows that are in both of the two results sets- that is the INTERSECTION (If we tried to do an Intersection of all of the cats and all of the dogs, the result would be empty.)



If we want all of the rows that are one of the results sets, but not in the other - that is the Except/Minus operation

