Table of Contents

1. Some problems to solve: 1
2. Enum 2
2.1. Enum and Storage 3
2.2. Sorting 4
2.3. Inserts with not null attributes 5
2.4. Non strict mode 6
2.5. Should you use enums? 6
3. Set 7
3.1. Storage 8
4. Information on Enum and Set (optional) 10

A DBMS can define different data types; we have been using the more traditional data types of strings and numbers and dates. And even with these types, we have seen variations on how the types are defined and what operations are allowed.

MySQL supports two special types that support lists of values. One of the rules for a relational database is that a cell in a table should store a single atomic value. An atomic value is one that is treated as a whole unit. So how could we have a list of values considered to be an atomic value? If the dbms has a defined type to handle this type of value, then it can be treated as an atomic value. Other data that we consider atomic also have internal structure (such as a date value having a month component and a year component, etc). Other dbms also have defined types (or user defined types) that have internal structure.

The Enum type and the Set type let us handle several issues in MySQL.

1. Some problems to solve:

1. We have a table orders with an attribute ord_mode and we want to limit this attribute to three possible values: PHONE, ONLINE, and DIRECT. We do not expect the acceptable values to change.
2. We have a table products with an attribute catg_id and we want to limit this attribute to a set of values- right now we have 8 categories but we might have more, or fewer, or different categories in the near future.
3. We have a table employees and we want to store data regarding certifications the employee has. There is a list of 4 different certifications we care about and an employee could have earned 0 or more of these.

Problem 1: The traditional way to solve this is to use a check constraint in the table definition.

```
constraint ord_mode_ck check (ord_mode in ('PHONE', 'ONLINE', 'DIRECT'))
```

This is a good solution for a small set of possible values, an enumeration of values that are not expected to change. This constraint is part of the table definition and is validated on inserts and on updates. We usually do not want to include rules in a table definition that change frequently.

One issue with MySQL is that, although you can write that constraint, MySQL does not enforce it. We can use an Enum to handle this.

Problem 2: The traditional way to solve this is to use a lookup table and set a foreign key constraint. This is a good solution for a list of values where the values might change over time. MySQL uses this technique also.

Problem 3: This can also be solved with a lookup table and a foreign key constraint. MySQL has an alternate solution for this- the Set type.

2. Enum

First set up a small table d_enum_01 in the a_testbed database. This is using a check constraint.

Demo 01: Set up table d_enum_01 and insert rows

```
create table d_enum_01 (
  ord_id   int not null
, ord_mode varchar(15) not null
, constraint ord_mode_ck check (ord_mode in ('PHONE', 'ONLINE', 'DIRECT'))
)
Engine = innodb;
```

Insert two rows. One of these follows the check constraint and the other does not, but both rows can be inserted into the table.

```
Insert into d_enum_01 values (1, 'PHONE');
Insert into d_enum_01 values (2, 'INTERNET');
Select * from d_enum_01;
+--------+----------+
| ord_id | ord_mode |
+--------+----------+
|      1 | PHONE    |
|      2 | INTERNET |
+--------+----------+
```

Now create the second table d_enum_02. This time we use a type for the ord_mode attribute where the type is defined as an enum with three possible values.

Demo 02: Set up table d_enum_01 and insert rows

```
create table d_enum_02 (
  ord_id   int not null
, ord_mode enum('PHONE', 'ONLINE', 'DIRECT')
)
Engine = innodb;
```

Insert two rows. One of these follows the enum type and the other does not; only the first row is inserted into the table.

```
Insert into d_enum_02 values (1, 'PHONE');
Insert into d_enum_02 values (2, 'INTERNET');
ERROR 1265 (01000): Data truncated for column 'ord_mode' at row 1

Insert into d_enum_02 values (3, 'DIRECT');

Select * from d_enum_02;
+--------+----------+
| ord_id | ord_mode |
+--------+----------+
|      1 | PHONE    |
|      3 | DIRECT   |
+--------+----------+
```

Demo 03: Update with an enum. The enum definition controls the ability to update. We have not studied the update statement yet, but I think you can follow this. We are changing a value in the table based on a row filter- here we want to change the value of ord_mode for the row with ord_id of 3.

This one works:

```
Update d_enum_02 set ord_mode = 'ONLINE' where ord_id = 3;
```

This one fails:

```
Update d_enum_02 set ord_mode = 'DOORTODOOR' where ord_id = 3;
ERROR 1265 (01000): Data truncated for column 'ord_mode' at row 1
```

Demo 04: Stored data versus displayed data

This one is interesting. I have entered the value in mixed case.

```
Insert into d_enum_02 values (4, 'Phone');
```

But it is displayed in upper case letters.

```
+--------+----------+
| ord_id | ord_mode |
+--------+----------+
|      4 | PHONE    |
+--------+----------+
```

We can enter nulls into that attribute

```
Insert into d_enum_02 values (5, null);
Insert into d_enum_02 values (6, 'DIRECT');
```

We can see that the enum type gives us some control over the values that can be entered into our table.

Concerns:

- There is an upper limit of 65,536 values in the enumeration- that generally is not a problem. It is more of a problem if you try to use an enum that big- but that upper limit should suggest something to you about storage
- You can alter the enumeration with an alter table statement, but you need to include the full set of values, nullability, default etc.

```
Alter table d_enum_02 modify ord_mode enum('PHONE', 'ONLINE', 'DIRECT',
'DOORTODOOR');
Insert into d_enum_02 values (7, 'DOORTODOOR');
```

- The enum is defined within the table definition; this is not a type definition saved on the server. You need to redefine it each time you want to use this enumeration.

2.1. Enum and Storage

Demo 05: Display the data in the table sorted by the ord_mode.

```
Select *
From d_enum_02
Order by ord_mode;
+--------+------------+
| ord_id | ord_mode   |
+--------+------------+
|      5 | NULL       |
|      1 | PHONE      |
|      4 | PHONE      |
|      3 | ONLINE     |
|      6 | DIRECT     |
|      7 | DOORTODOOR |
+--------+------------+
```

This did a sort on the ord_mode column but the rows are not in alphabetic order. This is due to the actual storage used for these values. The row order corresponds to the order of the literals in the enumeration.

The Enum values are stored as integers. We can demonstrate that with another select. First remember that if we multiple a string literal by 1 we get a zero.

```
Select  'DIRECT' * 1;
```

```
+--------------+
| 'DIRECT' * 1 |
+--------------+
|            0 |
+--------------+
```

Demo 06: Showing a numeric values associated with the enum

```
Select ord_id, ord_mode, ord_mode * 1
From d_enum_02;
+--------+------------+--------------+
| ord_id | ord_mode   | ord_mode * 1 |
+--------+------------+--------------+
|      1 | PHONE      |            1 |
|      3 | ONLINE     |            2 |
|      4 | PHONE      |            1 |
|      5 | NULL       |         NULL |
|      6 | DIRECT     |            3 |
|      7 | DOORTODOOR |            4 |
+--------+------------+--------------+
```

PHONE is stored as 1, ONLINE as 2, DIRECT as 3 and DOORTODOOR as 4. You will occasionally see that the way that MySQL stores data is not the same as we might think of the data (for example dates). This is true of any dbms- it has the right to store the data internally in any way that lets the data be accessed correctly. MySQL is different in that it exposes the storage pattern more frequently. That opens a door for a developer to write code based on that storage pattern- this can end up with confusing code but also with efficient code in terms of execution.

2.2. Sorting

Demo 07: To get the rows sorted in alphabetic order we can use a cast

```
Select *
From d_enum_02
Order by cast(ord_mode as char);
+--------+------------+
| ord_id | ord_mode   |
+--------+------------+
|      5 | NULL       |
|      6 | DIRECT     |
|      7 | DOORTODOOR |
|      3 | ONLINE     |
|      1 | PHONE      |
|      4 | PHONE      |
+--------+------------+
```

We could modify the enum to get an alphabetic ordering if that is desired.

```
Alter table d_enum_02 modify ord_mode enum('DIRECT', 'DOORTODOOR', 'ONLINE',
'PHONE');

Select ord_id, ord_mode, ord_mode * 1 from d_enum_02 order by ord_mode;
```

This could be useful if you have a list of string values that you want sorted in a non-alphabetic sort (such as Jan, Feb, Mar, etc). Set up an enum with the names of the months in the sorting order you prefer.

What about select and filters?

Demo 08: We can filter on the numeric value and on the name value

```
Select ord_id, ord_mode
From d_enum_02
Where ord_mode = 'PHONE'
;
+--------+----------+
| ord_id | ord_mode |
+--------+----------+
|      1 | PHONE    |
|      4 | PHONE    |
+--------+----------+

Select ord_id, ord_mode
From d_enum_02
Where ord_mode = 3;
+--------+----------+
| ord_id | ord_mode |
+--------+----------+
|      3 | ONLINE   |
+--------+----------+
```

2.3. Inserts with not null attributes

Make two more versions of this table. And test the following inserts.

Demo 09:

```
create table d_enum_03 (
  ord_id   int not null
, ord_mode enum('PHONE', 'ONLINE', 'DIRECT')  not null default 'DIRECT'
);

Insert into d_enum_03(ord_id) Values (33);
select * from d_enum_03;
+--------+----------+
| ord_id | ord_mode |
+--------+----------+
|     33 | DIRECT   |
+--------+----------+
```

That should make sense- we did not supply a value for ord_mode so we got the default.

But you might not expect the next one. The ord_mode is declared not null, we do not declare a default. What do we get?

Demo 10:

```
create table d_enum_04 (
  ord_id   int not null
, ord_mode enum('PHONE', 'ONLINE', 'DIRECT')  not null
);

Insert into d_enum_04(ord_id) Values (44);
select * from d_enum_04;
+--------+----------+
| ord_id | ord_mode |
+--------+----------+
|     44 | PHONE    |
+--------+----------+
```

Demo 11: Compare this to the way MySQL works with a not null attribute of another type such as varchar or int.

```
create table d_enum_05 (
  ord_id   int not null
, ord_mode varchar(15)  not null
);
Insert into d_enum_05(ord_id) Values (55);
ERROR 1364 (HY000): Field 'ord_mode' doesn't have a default value
```

2.4. Non strict mode

We do not use non strict mode in this class. With non strict mode there are some changes in how enums are handled. If you are supporting table in non-strict mode, consult the manual and test extensively.

2.5. Should you use enums?

It all depends--You will find enums being used so you need to be aware of them

To handle the lack of a check constraint: If you have an attribute that should store only one of two values ('Y', 'N') it seems inefficient to set up a lookup table and a relationship for this. Until MySQL gets a check constraint, enums can be helpful in keeping your data clean.

To handle data that comes from a web page that has a series of radio buttons: For example, you might give a user a choice of shipping methods and want an attribute to store that choice. My concern here is that web pages often change- we have more options or fewer options and then you would have to change the structure of the table. You should always be concerned about systems which require changes to the table structure.

To save space: Storing the number 1 takes less space than the literal for ord_modes. Enums might also be more efficient when it comes to handling indexes.

You will find code that takes advantage of the numeric nature. The following are to show you rather silly queries that hopefully you will not see.

Demo 12:

```
Select sum(ord_mode) from d_enum_02;
+---------------+
| sum(ord_mode) |
+---------------+
|            11 |
+---------------+

 Select ord_mode, sqrt(ord_mode) from d_enum_02;
+------------+------------------+
| ord_mode   | sqrt(ord_mode)   |
+------------+------------------+
| PHONE      |                1 |
| ONLINE     |  1.4142135623731 |
| PHONE      |                1 |
| NULL       |             NULL |
| DIRECT     | 1.73205080756888 |
| DOORTODOOR |                2 |
+------------+------------------+
```

Enums are not common in other dbms so there are other ways to handle this. If you do decide to use Enums, or inherit databases that use Enums, Test- test- Test.

3. Set

The set data type is similar to the Enum type in that it allows for defining a type with multiple values. It differs in that you could store more than one of those values in an attribute in a table row.

A set is a string with zero or more values separated by commas; the values have to be chosen from a list of values. You can have at most 64 values. You cannot have a comma inside a value and you cannot have duplicates. The values are stored internally as numbers (see section 3.1 below).

The display of a set values is based on the definition of the set in the table create statement.

Problem 3: We have a table employees and we want to store data regarding certifications the employee has. There is a list of 5 different certifications we care about and an employee could have earned 0 or more of these.

The traditional solution is to define a lookup table and a foreign key and then join the two tables. Using the Set type means that you do not have do that.

Demo 13: Create a table with a Set type

```
create table d_set_01(
  emp_id int
, emp_cert SET ('Java','Visual Basic', 'C++','Database')
)
Engine = innodb;
```

Demo 14: Inserts

I allowed nulls in the emp_cert attribute

```
Insert into d_set_01 values (1, null);
```

Insert with a single element for emp_cert.

```
Insert into d_set_01 values (2, 'Java');
```

Insert with two elements for emp_cert; note that this is a single string value- we are entering this into a single attribute. Use a comma between the individual elements.

```
Insert into d_set_01 values (3, 'Java,Database');
```

Insert with two elements for emp_cert;

```
Insert into d_set_01 values (4, 'Database,Java');
```

Invalid entry- since that item is not on the allowed list.

```
Insert into d_set_01 values (5, 'Set Theory');
ERROR 1265 (01000): Data truncated for column 'emp_cert' at row 1
```

It might not be obvious why the next is an error- there is a space after the comma between the two elements

```
Insert into d_set_01 values (6, 'Database, Java');
ERROR 1265 (01000): Data truncated for column 'emp_cert' at row 1
```

Demo 15: What do we have in the table? Note the difference in rows 4 with the insert value and the display value

```
select * from d_set_01;
+--------+---------------+
| emp_id | emp_cert      |
+--------+---------------+
|      1 | NULL          |
|      2 | Java          |
|      3 | Java,Database |
|      4 | Java,Database |
+--------+---------------+
```

3.1. Storage

This part will be confusing if you are not familiar with the binary system. But it does help you understand some issues with ordering and with demos you might see on the internet.

Demo 16: Remove the current rows and do these inserts. Truncate removes all the rows from the table.

```
Truncate d_set_01;
Insert into d_set_01 values (1, 'Java');
Insert into d_set_01 values (2, 'Visual Basic');
Insert into d_set_01 values (3, 'C++');
Insert into d_set_01 values (4, 'Database');


Insert into d_set_01 values (5, 'Java,Visual Basic' );
Insert into d_set_01 values (6, 'Java,C++');
Insert into d_set_01 values (7, 'Java,Database');


Insert into d_set_01 values (8, 'Visual Basic,C++');
Insert into d_set_01 values (9, 'Visual Basic,Database');
Insert into d_set_01 values (10, 'C++,Database');
Insert into d_set_01 values (11, 'Visual Basic,C++,Database');
Insert into d_set_01 values (12, 'Java,C++,Database');
Insert into d_set_01 values (13, 'Java,Visual Basic,Database');
Insert into d_set_01 values (13, 'Java,Visual Basic,C++');

Insert into d_set_01 values (14, 'Java,Visual Basic,C++,Database');
```

Looking at the last column: we start with multiplying by 1 to get a numeric value; then we use the **bin** function which returns a string corresponding to the binary value of that number; I want these valued right aligned so I left padded them with '0' to a column width of 4.

Demo 17: Showing the binary values of the attribute

```
Select emp_id, emp_cert
, emp_cert *1 as Intgr
, lpad(bin(emp_cert*1),4,'0') as bnry
From d_set_01
Order by bnry;
```

emp_id	emp_cert	Intgr	bnry
1	Java	1	0001
2	Visual Basic	2	0010
5	Java,Visual Basic	3	0011
3	C++	4	0100
6	Java,C++	5	0101
8	Visual Basic,C++	6	0110
13	Java,Visual Basic,C++	7	0111
4	Database	8	1000
7	Java,Database	9	1001
9	Visual Basic,Database	10	1010
13	Java,Visual Basic,Database	11	1011
10	C++,Database	12	1100
12	Java,C++,Database	13	1101
11	Visual Basic,C++,Database	14	1110
14	Java,Visual Basic,C++,Database	15	1111

This might be clearer if we filtered for various certificates
All of the rows that include database have a 1 in the 4th position of the binary string(count from the right)

Demo 18: Filtering for database

```
Select emp_id, emp_cert
, emp_cert *1 as Intgr
, lpad(bin(emp_cert*1),4,'0') as bnry
From d_set_01
Where emp_cert like '%database%'
Order by bnry;
+--------+--------------------------------+-------+------+
| emp_id | emp_cert                       | Intgr | bnry |
+--------+--------------------------------+-------+------+
|      4 | Database                       |     8 | 1000 |
|      7 | Java,Database                  |     9 | 1001 |
|      9 | Visual Basic,Database          |    10 | 1010 |
|     13 | Java,Visual Basic,Database     |    11 | 1011 |
|     10 | C++,Database                   |    12 | 1100 |
|     12 | Java,C++,Database              |    13 | 1101 |
|     11 | Visual Basic,C++,Database      |    14 | 1110 |
|     14 | Java,Visual Basic,C++,Database |    15 | 1111 |
+--------+--------------------------------+-------+------+
```

All of the rows that include java have a 1 in the 1st position of the binary string(count from the right)

Demo 19: Filtering for java

```
Select emp_id, emp_cert
, emp_cert *1 as Intgr
, lpad(bin(emp_cert*1),4,'0') as bnry
From d_set_01
Where emp_cert like '%Java%'
Order by bnry;
+--------+--------------------------------+-------+------+
| emp_id | emp_cert                       | Intgr | bnry |
+--------+--------------------------------+-------+------+
|      1 | Java                           |     1 | 0001 |
|      5 | Java,Visual Basic              |     3 | 0011 |
|      6 | Java,C++                       |     5 | 0101 |
|     13 | Java,Visual Basic,C++          |     7 | 0111 |
|      7 | Java,Database                  |     9 | 1001 |
|     13 | Java,Visual Basic,Database     |    11 | 1011 |
|     12 | Java,C++,Database              |    13 | 1101 |
|     14 | Java,Visual Basic,C++,Database |    15 | 1111 |
+--------+--------------------------------+-------+------+
```

All of the rows that include Basic have a 1 in the 2nd position of the binary string(count from the right)

Demo 20: Filtering for basic

```
Select emp_id, emp_cert
, emp_cert *1 as Intgr
, lpad(bin(emp_cert*1),4,'0') as bnry
From d_set_01
Where emp_cert like '%Basic%'
Order by bnry;
+--------+--------------------------------+-------+------+
| emp_id | emp_cert                       | Intgr | bnry |
+--------+--------------------------------+-------+------+
|      2 | Visual Basic                   |     2 | 0010 |
|      5 | Java,Visual Basic              |     3 | 0011 |
|      8 | Visual Basic,C++               |     6 | 0110 |
|     13 | Java,Visual Basic,C++          |     7 | 0111 |
|      9 | Visual Basic,Database          |    10 | 1010 |
```

```
|     13 | Java,Visual Basic,Database     |    11 | 1011 |
|     11 | Visual Basic,C++,Database      |    14 | 1110 |
|     14 | Java,Visual Basic,C++,Database |    15 | 1111 |
+--------+--------------------------------+-------+------+
```

All of the rows that include C++ have a 1 in the 3rd position of the binary string(count from the right)

Demo 21: Filtering for C++

```
Select emp_id, emp_cert
, emp_cert *1 as Intgr
, lpad(bin(emp_cert*1),4,'0') as bnry
From d_set_01
Where emp_cert like '%C++%'
Order by bnry
;
+--------+--------------------------------+-------+------+
| emp_id | emp_cert                       | Intgr | bnry |
+--------+--------------------------------+-------+------+
|      3 | C++                            |     4 | 0100 |
|      6 | Java,C++                       |     5 | 0101 |
|      8 | Visual Basic,C++               |     6 | 0110 |
|     13 | Java,Visual Basic,C++          |     7 | 0111 |
|     10 | C++,Database                   |    12 | 1100 |
|     12 | Java,C++,Database              |    13 | 1101 |
|     11 | Visual Basic,C++,Database      |    14 | 1110 |
|     14 | Java,Visual Basic,C++,Database |    15 | 1111 |
+--------+--------------------------------+-------+------+
```

Another way to look at this is

- Java has a weight of 1
- Visual Basic has a weight of 2
- C++ has a weight of 4
- Database has a weight of 8

Hopefully, you recognize those as the decimal values of the place system for binary.

An employee with certificate in Java and Database gets a certificate weight of 1 + 8 = 9

An employee with a certificate weight of 13 is made up of 1 + 4 + 8 and there is no other sum of the base values (1,2,4,8) that adds up to 13.

Demo 22: Note that if we test as an equality for C++ we get only the employees who have that certificate and only that certificate.

```
Select emp_id, emp_cert
From d_set_01
Where emp_cert = 'C++';
+--------+----------+
| emp_id | emp_cert |
+--------+----------+
|      3 | C++      |
+--------+----------+
```

4. Information on Enum and Set (optional)

You can find the definitions of your enum and set types from a system view information_schema.columns

Demo 23: Display the column and their type for a particular table

```
Select  column_name, column_type
From information_schema.columns
Where table_schema='a_testbed' and table_name='d_enum_02'
;
+-------------+---------------------------------------------+
| column_name | column_type                                 |
+-------------+---------------------------------------------+
| ord_id      | int(11)                                     |
| ord_mode    | enum('PHONE','ONLINE','DIRECT','DOORTODOOR') |
+-------------+---------------------------------------------+
```

Demo 24: Display the column and their type for a particular table

```
Select  column_name, column_type
From information_schema.columns
Where table_schema='a_testbed' and table_name='d_set_01';
+-------------+-----------------------------------------+
| column_name | column_type                             |
+-------------+-----------------------------------------+
| emp_id      | int(11)                                 |
| emp_cert    | set('Java','Visual Basic','C++','Database') |
+-------------+-----------------------------------------+
```

Demo 25: Looking for a value that could be part of an enum or set

```
Select  table_name, column_name, column_type
From information_schema.columns
Where table_schema='a_testbed'
And Column_type like '%Basic%';
+------------+-------------+-----------------------------------------+
| table_name | column_name | column_type                             |
+------------+-------------+-----------------------------------------+
| d_set_01   | emp_cert    | set('Java','Visual Basic','C++','Database') |
+------------+-------------+-----------------------------------------+
```

Demo 26: Looking for a value that could be part of an enum or set

```
Select  table_name, column_name, column_type
From information_schema.columns
Where table_schema='a_testbed'
And Column_type like '%Phone%';
+------------+-------------+---------------------------------------------+
| table_name | column_name | column_type                                 |
+------------+-------------+---------------------------------------------+
| d_enum_02  | ord_mode    | enum('PHONE','ONLINE','DIRECT','DOORTODOOR') |
| d_enum_03  | ord_mode    | enum('PHONE','ONLINE','DIRECT')             |
| d_enum_04  | ord_mode    | enum('PHONE','ONLINE','DIRECT')             |
+------------+-------------+---------------------------------------------+
```

Demo 27: Looking for tables that use an enum

```
Select  table_name, column_name, column_type
from information_schema.columns
Where table_schema='a_testbed'
And Column_type like 'enum(%';
+------------+-------------+---------------------------------------------+
| table_name | column_name | column_type                                 |
+------------+-------------+---------------------------------------------+
| d_enum_02  | ord_mode    | enum('PHONE','ONLINE','DIRECT','DOORTODOOR') |
| d_enum_03  | ord_mode    | enum('PHONE','ONLINE','DIRECT')             |
| d_enum_04  | ord_mode    | enum('PHONE','ONLINE','DIRECT')             |
| tbl_001    | col2        | enum('true','false')                        |
| tbl_002    | col2        | enum('0','1')                               |
```

```
| tbl_003    | col2        | enum('0','1')                                |
+------------+-------------+----------------------------------------------+
```