MySQL Tutorial 2: Views and Joins

Getting to know the MySQL database engine

— P. Lutus — Message Page —

(double-click any word to see its definition)

— Navigate this multi-page article with the arrows and drop-down lists at the top and bottom of each page —

Note: In this article, footnotes are marked with a light bulb over which one hovers .

Introduction

In early database practice, if one wanted a different query result one would retype the query, or perhaps create a new table that contained the result for a given query. But in modern database design, there are ways to store queries separate from table contents, so the same query design can be applied to a table whose data has changed.

In MySQL, a stored query is called a "view" — it's a distinct way of looking at a table's data that doesn't depend on changes in the data. The advantage of this is that, as the table's data changes, the original view is automatically applied to the new data. The official MySQL literature describes a view as a "virtual table".

In the first section of this article, we typed in queries to meet certain requirements, and as it turns out, any of those queries can be encapsulated in views for later use.

Let's start writing some views.

Views

Let's create a view that defines a group called "youngsters". Start a MySQL command-line session as explained on the earlier page, and make these entries:

```
mysql> use tutorial;
mysql> create or replace view `youngsters` as select * from people where Age <= 20;
mysql> select count(*) from `youngsters`;

+----------+
| count(*) |
+----------+
|      132 |
+----------+
```

That was easy. Let's create another view --- this one will define a group that likes rock music:

```
mysql> create or replace view `rock_lovers` as select * from people where Music = 'Rock';
mysql> select count(*) from `rock_lovers`;

+----------+
| count(*) |
+----------+
|      211 |
+----------+
```

Joining Views

Now let's try this:

```
mysql> select count(*) from rock_lovers as x natural join youngsters as y;
```

```
+----------+
| count(*) |
+----------+
|       23 |
+----------+
```

The meaning of "natural join" may not be obvious. From the "youngsters" and "rock_lovers" groups, a natural join creates a new group that are *both* youngsters *and* rock lovers (see Figure 1, blue area).

NOTE: in the above MySQL command, notice the expressions "as x" and "as y". These are used to create temporary tables required for logical joins. Joins don't operate on queries, they operate on tables, so we must create them.

Database logical operations can be loosely mapped to the terminology of set theory . In set theory terms, a MySQL "natural join" equals an "intersection": (A ∩ B).

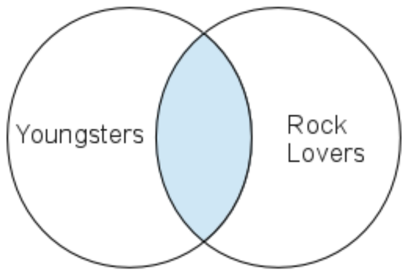

Figure 1: "natural join" (A ∩ B)

The same result could be gotten this way:

```
mysql> select count(*) from people where Age <= 20 and Music = 'Rock';

+----------+
| count(*) |
+----------+
|       23 |
+----------+
```

But the advantage of using views instead of explicit queries is that we don't need to know the specifics of the views to perform logical operations on them — we're free to operate at a more abstract level, with confidence that the same logic will apply regardless of how the views are defined.

Also, equally important, by using views we give a group a name that's easier to understand than the underlying query. While reading a complex expression, I think "youngster" is easier to understand than "age <= 20".

This leads us to the insight that *a view is to a database what a variable is to algebra* — it allows us to manipulate a symbolic value rather than a specific one, with assurance that any view should obey the same logical rules.

Now let's create a group that's the *logical inverse* of the natural join:

```
mysql> select count(*) from (select * from youngsters as x union select * from rock_lovers as y) as z;

+----------+
| count(*) |
+----------+
|      320 |
+----------+
```

This result creates a group composed of youngsters, or rock lovers, but not both (see Figure 2, blue area).

In set theory terms, a MySQL "union" equals the inverse of an intersection: ~(A ∩ B).

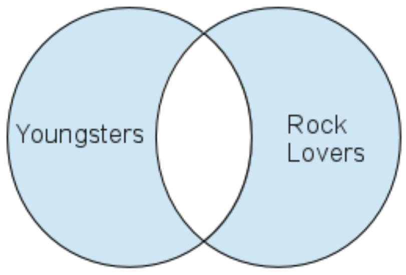

Figure 2: "union" ~(A ∩ B)

There is one more possibility — a group composed of youngsters, rock lovers, and those who are both youngsters and rock lovers:

```
mysql> select count(*) from (select * from youngsters as x union all select * from rock_lovers as y) as z;

+----------+
| count(*) |
+----------+
|      343 |
+----------+
```

The difference between "union" and "union all" is that "union" joins members of the groups except those who are members of both groups, but "union all" includes those who are members of both groups (see Figure 3).

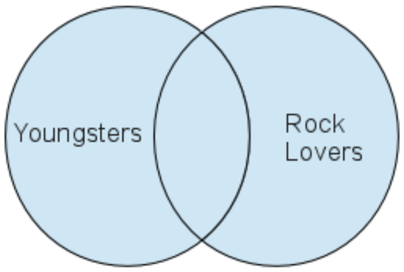

Figure 3: "union all" (A ∪ B)

In set theory terms, a MySQL "union all" equals a union: (A ∪ B).

When designing joins and unions, it's useful to look at the numeric totals as these examples do — it helps to detect logical errors:

- From above we see that the "youngsters" group has 132 members: Y = 132.
- And the "rock lovers" group has 211 members: R = 211.
- The "natural join" of youngsters and rock lovers created a group who are *both* youngsters and rock lovers: B = 23.
- This means that the "union" operation, which should include youngsters and rock lovers, *but not members of both groups*, should have R + Y - B = 320 members, and it does.
- And the "union all" operation, which includes youngsters, rock lovers, and member of both groups, should have R + Y = 343 members, and it does.

One final twist. From the above it might seem as though logical joins are limited to two groups. Not so — enter this new view:

```
mysql> create or replace view modern_art_lovers as select * from people where Art = 'Modern';
mysql> select count(*) from modern_art_lovers;

+----------+
| count(*) |
+----------+
|      178 |
+----------+
```

Okay, in our population of 1000, there are 178 modern art lovers. But I can't resist asking — how many people are modern art lovers AND youngsters AND rock lovers?

```
mysql> select count(*) from rock_lovers as x natural join youngsters as y natural join modern_art_lovers as z;

+----------+
| count(*) |
+----------+
|        3 |
+----------+
```

Just to be sure, let's check the actual records to verify that the logical constraints have been met:

```
mysql> select * from rock_lovers as x natural join youngsters as y natural join modern_art_lovers as z;

+------------+-----------+--------+-----+-------+--------+-----+
| First Name | Last Name | Gender | Age | Music | Art    | pk  |
+------------+-----------+--------+-----+-------+--------+-----+
| Patricia   | Jackson   | F      |  16 | Rock  | Modern | 438 |
| Mark       | Moore     | M      |  16 | Rock  | Modern | 632 |
| Gary       | Taylor    | M      |  13 | Rock  | Modern | 811 |
+------------+-----------+--------+-----+-------+--------+-----+
```

All the members appear to meet the join's logical requirements.

A complex example like this shows the value of abstracting queries into views, for a number of reasons including simply being able to understand what's going on (see Figure 4).

In set theory terms, this example equals an intersection of three sets: (A ∩ B ∩ C).

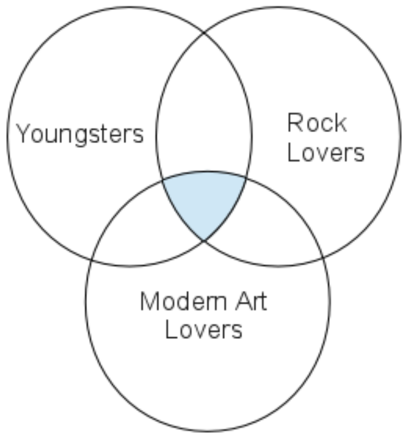

Figure 4: "natural join" (A ∩ B ∩ C)

I hope it hasn't escaped my readers' attention that one can construct a view from a join of views:

```
mysql> create or replace view unlikely_group as select * from rock_lovers as x natural join youngsters as y natural join modern_art_lovers as z;
mysql> select * from unlikely_group;

+------------+-----------+--------+-----+-------+--------+-----+
| First Name | Last Name | Gender | Age | Music | Art    | pk  |
+------------+-----------+--------+-----+-------+--------+-----+
| Patricia   | Jackson   | F      |  16 | Rock  | Modern | 438 |
| Mark       | Moore     | M      |  16 | Rock  | Modern | 632 |
| Gary       | Taylor    | M      |  13 | Rock  | Modern | 811 |
+------------+-----------+--------+-----+-------+--------+-----+
```

Summary

Let's review, and add to, what we've learned:

- A MySQL view is a stored query with a name, an abstraction not unlike a variable in algebra.
- Views can be combined using logical rules similar to those in set theory.
- Views don't contain any data, only rules for processing data. This means if the underlying data tables change, so does the result created by the view.
- Views that are created become part of the database they refer to, and during MySQL backup operations, are backed up along with their database.
- The examples on this page are only a small subset of all possible joins and logical combinations, but the most common and useful ones. Also, most joins involve more than one table — these examples involve one table only for the sake of descriptive simplicity. Click here for a more technical description of joins.

In the next article section, we'll use views to create dynamic content not present in the source tables.

Topical Links

- MySQL Documentation / Reference Manuals
- MySQL Create View Syntax
- MySQL Join Syntax
- Join (SQL) (Wikipedia)

— Navigate this multi-page article with the arrows and drop-down lists at the top and bottom of each page —