The MySQL Grant Tables

By W.J. Gilmore

Table of Contents

Introduction........1

Access Control........2

Tables_priv and columns_priv........5

References........8

Introduction

One of the most powerful aspects of the MySQL server (http://www.mysql.com) is the amazing amount of control the administrator has over each user's intended behavior. This control can restrict user privileges over a general part of the server, such as limited access to an entire database, but can also be as specific as limiting privileges for a specific table or even column. This article will serve to explain the process in which the MySQL server grants/revokes these user privileges, highlighting in particular the newest additions to the MySQL privilege system, the **tables_priv** and **columns_priv** tables. Please keep in mind that the GRANT/REVOKE commands detailed later in this article are only relevant to MySQL version 3.22.11 and up. These commands are irrelevant to any previous version.

The MySQL privilege system is controlled within the MySQL database. There are currently 5 tables that provide this control; the **user**, **db**, **host**, **tables_priv** and **columns_priv** tables. These tables all vary slightly in purpose, yet all serve the same function which is to verify that the user is doing what the user is allowed to do. Each table can be broken down into two categories of fields: scope fields and privilege fields. The scope fields identify a host, user or database. The privilege fields determine which actions can be performed in reference to that host, user or database. Let's take a brief look at each table's function:

- **user** − Determines whether or not the connecting user is allowed to connect to the server. Assuming the connection is allowable, the privilege fields contain the user's global privileges.
- **db** − Determines which users can access which databases from which hosts. The privilege contained within the db table apply to the database identified within this table.
- **host** − The host table is used when you want to extend an entry within the db table's range. For example, if a certain db is to be accessed by more than one host, then the superuser would leave the host column empty within the db table and fill the host table with all of the necessary hostnames.
- **tables_priv** − In principle works just like the db table, except that it is used for tables instead of databases. This table also contains one other field category (Other) in which a timestamp and grantor column is stored.The tables_priv table will be explained in further detail later in this article.
- **columns_priv** − Works just like the db and tables_priv tables, except that it provides access privileges for certain columns of certain tables. This table also contains one other field category (Other) in which a timestamp column is stored. The columns_priv table will be explained in further detail later in this article.

Let's move on to an explantion of MySQL's user−authorization procedure.

Part 1: The MySQL Access Control Process.
How do the MySQL Grant tables work anyway?

Part 2: The **tables_priv** and **columns_priv** grant tables.
An explanation and several examples relating to MySQL's tables_priv table.

Part 3: References
An explanation and several examples relating to MySQL's columns_priv grant table.

Access Control

The process in which the MySQL server controls user privileges, although seemingly daunting at first look, is actually a fairly simple, although secure, procedure. Let's take a look at some not−so−obvious properties of this process before working through an example.

Property#1:
The tables can be looked at as a sort of filter which works from most general to most specific. This filter, (working from general to specific), is as follows:

- User table.
- Db Table
- Host Table
- Tables_priv Table
- Columns_priv Table

Property#2:
Once a server−connection is made, there are two kinds of requests that a user can make:

- Administrative Request (shutdown, reload, process, etc...)
- Database−related Request (insert, delete, alter, update, etc...)

When a user makes an administrative request, the server only has to look in one specific location: the **user** table. This is because the **user** table is the only table containing privileges related to administrative processes. However, when the user makes a database request, the process is a slight bit more complicated.

You may have noticed that the grant tables are somewhat redundant (i.e. A 'select' privilege within the **user** table, and the same privilege repeated within the **host** and **user** tables); This is not without reason. One could consider the database−related privileges within the **user** table as global. That is, the privileges granted to the user within this table are good for **every** database on the server. These privileges could be considered superuser privileges. On the other hand, the database−related privileges contained within the **host** and **db** tables are specifically related to the host or database in question. Thus it would be a wise decision to leave all privileges within this table as 'N'.

Regardless of you decide to set the **user** table, please take note of the following example. Let's assume our **user** and **db** tables are as follows:

User Table		Db Table	
Host	%.pi.com	Host	%.pi.com
User	wj	Db	oats
Password	34ghyT	User	wj
Select_priv	'N'	Select_priv	'Y'
Insert_priv	'Y'	Insert_priv	'Y'
Update_priv	'N'	Update_priv	'Y'
Delete_priv	'N'	Delete_priv	'N'

Index_priv	'N'	Index_priv	'N'
Alter_priv	'N'	Alter_priv	'N'
Create_priv	'N'	Create_priv	'N'
Drop_priv	'N'	Drop_priv	'N'
Grant_priv	'N'	Grant_priv	'Y'
Reload_priv	'N'		
Shutdown_priv	'N'		
Process_priv	'N'		
File_priv	'N'		

Scenario #1: Failed Connection Attempt

1. User 'alessia' connection attempt failed. − Host, User and/or password does not match up with those contained within the **user** table. User is denied access.

Scenario #2: 'N' db−privilege in **user** table, 'Y' db−privilege in **db** table.

1. User 'wj' connection attempt successful.
2. User 'wj' attempts to perform a 'Select' command on the 'oats' database.
3. Server looks towards the **user** table. There is a 'N' (denied) entry for the 'Select' command.
4. Server then looks toward the **db** table. There is a 'Y' (allowed) entry for the 'Select' command.
5. Request is successful, because there is a 'Y' within the SELECT column of the user's db table insertion.

Scenario #3: 'Y' db−privilege in **user** table, 'N' db−privilege in **db** table.

1. User 'wj' connection attempt successful.
2. User 'wj' attempts to perform a 'Select' command on the 'oats' database.
3. Server looks towards the **user** table. There is a 'Y' (allowed) entry for the 'Select' command. Since the privileges granted within the **user** table are global, the request is successfully carried out.

Scenario #4: 'N' db−privilege in **user** table, 'N' db−privilege in **db** table.

1. User 'wj' connection attempt successful.
2. User 'wj' attempts to perform a 'Select' command on the 'oats' database.
3. Server looks towards the **user** table. There is a 'N' (denied) entry for the 'Select' command.
4. Server now looks towards the **db** table. There is a 'N' (denied= entry for the 'Select' command.
5. Server now looks towards the **tables_priv** and **columns_priv** tables. If the privileges are in accordance with the user's request, access is granted. Otherwise, access is denied.

The **tables_priv**and **columns_priv** tables are discussed in further detail later on in this article.

Scenario #5: Let's assume the following is true:

- The 'host' column for user'wj' is '%' within the user table.

* the 'host' column for user 'wj' was blank within the db table.

What happens?

1. User 'wj' connection via a given host is attempted.
2. Assuming the password is correct, the attempt is successful, because the user table states that **any** ('%') host can connect if the connection is via username 'wj' and the given password.
3. The MySQL server looks to the **db** table. However, there is no host given.
4. The MySQL server now looks to the **host** table. IF the db in which the user is connecting to is listed within the host table along with the particular host name that the user is connecting from, then the user is free to carry out commands in accordance with the privileges listed within the **host** table. If the db/host are not in accordance with those in which the user is connecting from, the user cannot carry out commands and is in essence denied of connection.

The above scenarios should give the reader at least a bit of insight into the privilege system. We will now move on to the latest additions to the privilege system, the **tables_priv** and **columns_priv** tables.

Tables_priv and columns_priv

The **tables_priv** and **columns_priv** grant tables is two of the more recent additions to the MySQL database server. They are intended to provide the user with even greater control over the user's actions while connected to the server. Both are very similar to the **db** grant table, but with an even more specified range of purpose; a given table contained within a given database. Whereas the superuser could previously limit a user's actions within a database via the **db** grant table, the superuser can now limit a user's actions on a per−table basis and per−column basis. Understandably, this provides the superuser with a very flexible array of options to work with.

Before we look further into each table, please read carefully the following characteristics:

- Wildcards are permitted within the host field of both tables, but are not permitted within the Db, Table_name and Column_name fields.
- Both tables are sorted similarly to the db table, but is much easier since only the host column can hold wildcards.
- The privilege fields are declared as 'SET' fields.
- The **tables_priv** and **columns_priv** tables should ONLY be modified via GRANT/REVOKE commands. Attempts to insert data into these tables using 'INSERT' commands will result in a problematic server!
- The table_priv column within the tables_priv table allows the following: 'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter'.
- The column_priv column within the tables_priv table allows the following: 'Select', 'Insert', 'Update', 'References'.
- The type column within the columns_priv table allows the following: 'Select', 'Insert', 'Update', 'References'.

Note:

- 'References' is not yet implemented.
- 'usage' simply means a user with no privileges.

The tables_priv grant table

The following is a diagram of the **tables_priv** table:

Host	Host
Db	Db
User	User
Table_name	Table_name
	Column_name
Table_priv	Type
Column_priv	

Timestamp	Timestamp
Grantor	

Column definitions:

- **Host** − For what host does this apply?
- **Db** − For what db connected from the above host does this apply?
- **User** − For what user from the above host does this apply?
- **Table_name** − For which table within the above Db does this apply?
- **Table_priv** − What privileges are allowed for this table?
- **Column_priv** − What privileges are allowed for the columns contained within this table?
- **Timestamp** − When was this privilege granted?
- **Grantor** − Who granted the 'User' this privilege?

Perhaps the only way to truly understand how the tables_priv table is used is through examples. Let's take a look at a few of them.

Example #1:

```
%>GRANT SELECT ON italy TO wj@314interactive.com;
```

What does this accomplish?
The above command allows user 'wj' from host '314interactive.com' to perform a 'SELECT' statement on the table 'italy'. Remember that this table would be referred to only if there was a 'N' within the 'SELECT' column of the 'db' or 'host' table regarding the given database/host and given username. If there was a 'Y' within the 'SELECT' column of the 'db' or 'host' table regarding the given database/host and given username, then there would be no need to control the tables_priv table.

Example #2:

```
%>GRANT SELECT, INSERT ON oats.italy TO wj@314interactive.com;
```

What does this accomplish?
The above command allows user 'wj' from host '314interactive.com' to perform 'SELECT' and 'INSERT' statements on the table 'italy' residing within the 'oats' database.

Example #3:

```
%>REVOKE SELECT on oats.italy from wj@314interactive.com
```

What does this accomplish?
The above command revokes 'SELECT' privileges from user 'wj' from host '314interactive.com' pertaining to the table 'italy' contained within the database 'oats'.

It is important to understand that the information contained within the **tables_priv** only comes into effect when the host/db tables deny the user the necessary privileges to perform the requested function. If the given privilege were 'Y' within the host/db table, then there would be no need to even look at the tables_priv table.

Example #4: (A slight bit more complicated)

```
%>GRANT SELECT(id,name,address,phone),update(address,phone) ON
company.customers TO gilmore@314interactive.com;
```

What does this accomplish?
The above command grants SELECT privileges for the 'id', 'name', 'address', and 'phone' columns, and UPDATE privileges for the 'address' and 'phone' columns within the 'customers' table, contained within the 'company' database.
What does this affect?
This command modifies both the **tables_priv** table and the **columns_priv** tables. This is because it refers to both the table and specific columns residing within the table.

Example #5:

```
%>REVOKE UPDATE(address,phone) ON company.customers FROM
gilmore@314interactive.com;
```

What does this accomplish?
This revokes UPDATE privileges for the address and phone columns contained within the 'customers' table residing within the company database.
What does this affect?
Since the command makes direct references to the columns contained within the given table, the **columns_priv** table is updated as well as the **tables_priv** table.

Although stated previously within this article, it is of enough importance that it should be repeated; Grant tables are only used **if** needed. For example, if the table of higher precedence provides adequate privileges, than the lower table precedences will not be consulted. If the higher−precedence table contains 'N' within the requested command, then the lower−precedence table will be consulted. Simple as that.

Note from Monty: GRANT will create a new user if the user didn't exist for before and that one can add a password for a new user with the IDENTFIED BY 'password' syntax.

I have compiled a short list of references pertaining to the MySQL grant tables on the following page. Please feel free to review each.

References

Perhaps the most important MySQL resource out there is the MySQL documentation. The latest version can be found here.

The MySQL mailing−list archive can be found at: ListQuest.

Devshed provides a series of MySQL articles:

Beginning MySQL Tutorial − Introduction to MySQL

MySQL Administration Introduction − Introduction to configuration and administration of the MySQL server.

PHP and Web Database Introduction − Introduction to PHP and web database interfacing with MySQL.

If you have any questions/comments regarding this article, feel free to contact me at wj@314interactive.com.