Access Granted

By icarus

Table of Contents

Turning The Tables

If you've been working with PHP for a while, you probably already know the basics of connecting your Web application to a database and using it to dynamically generate Web pages. Along the way, you'll probably have picked up the basics of SQL (if you didn't know it already) and perhaps even worked a little with mySQL, one of the most popular open−source database engines.

Most users concentrate on mySQL's databases and tables − after all, that's where most of the action takes place − and don't usually look under the hood to see the engine beneath. This is usually more than adequate for most development activities − unless you happen to be a database administrator whose job involves setting up and securing the databases against unauthorized usage or malicious mischief.

Over the next few pages, I'm going to examine the mySQL access control system, and throw some light on the mySQL "grant tables". These tables, which are an integral part of the server's security system, offer database administrators a great deal of power and flexibility in deciding the rules which govern access to the system.

I'll be assuming that you know the basics of SQL, and have a mySQL database server up and running. If you're not familiar with SQL, you can find a tutorial at http://www.devshed.com/Server_Side/MySQL/Speak/

Meet Joe User

Modifying the mySQL grant tables requires superuser access to the mySQL database server. So the first order of business is to ensure that you have this level of access, and can alter table records

If you've installed the server yourself, on your own development machine, you would have been told to enter a root password at the time of installation. Hopefully, you did this − too many people leave the password blank, thereby opening up a gaping security hole − and still remember the password you used.

To verify that you have the required access, log into the server as the "root" user

```
$ mysql −u root −p
Enter password: ****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 3.22.32

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql>
```

and ensure that you can view the contents of the tables in the "mysql" database − this is the database that contains all the grant tables.

```
mysql> USE mysql;
Reading table information for completion of table and column
names
You can turn off this feature to get a quicker startup with −A

Database changed
mysql> SHOW TABLES;
+−−−−−−−−−−−−−−−−+
| Tables in mysql |
+−−−−−−−−−−−−−−−−+
| columns_priv |
| db |
| host |
| tables_priv |
| user |
+−−−−−−−−−−−−−−−−+
5 rows in set (0.00 sec)
```

Of course, root−level access is typically available only to the system administrator − other users have a lower security rating and, consequently, limited access. Each of these "ordinary" users will typically connect to the database by supplying his or her own user name and password − like this:

```
$ mysql −h localhost −u joe −p
Enter password: *********
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 3.22.32

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql>
```

The purpose of the mySQL grant tables is to make it possible to manipulate security settings for these "ordinary" users, and customize each user's level of access to a very fine degree.

Beeping Turkeys

As you will have seen from the example above, the entire mySQL security system consists of five tables.

```
+-----------------+
| columns_priv |
| db |
| host |
| tables_priv |
| user |
+-----------------+
```

Each of these has a different role to play in deciding whether a user has access to a specific database, table or table column. Access rules may be set up on the basis of username, connecting host, or database requested.

When a user requests a connection to the database server from a specific host, mySQL will first check whether there is an entry for the user in the "user" table, if the user's password is correct, and if the user is allowed to connect from that specific host. If the check is successful, a connection will be allowed to the server.

Once a connection is allowed, every subsequent request to the server − SELECT, DELETE, UPDATE and other queries − will first be vetted to ensure that the user has the security privileges necessary to perform the corresponding action. A number of different levels of access are possible − some users may only have the ability to SELECT from the tables, while others may have INSERT and UPDATE capabilities, but not DELETE capabilities.

Of the three tables above, the most important one is the "user" table − let's take a closer look at the fields it contains:

```
+-----------------+--------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| Host | char(60) | | PRI | | |
| User | char(16) | | PRI | | |
| Password | char(16) | | | | |
| Select_priv | enum('N','Y') | | | N | |
| Insert_priv | enum('N','Y') | | | N | |
| Update_priv | enum('N','Y') | | | N | |
| Delete_priv | enum('N','Y') | | | N | |
| Create_priv | enum('N','Y') | | | N | |
```

```
| Drop_priv | enum('N','Y') | | | N | |
| Reload_priv | enum('N','Y') | | | N | |
| Shutdown_priv | enum('N','Y') | | | N | |
| Process_priv | enum('N','Y') | | | N | |
| File_priv | enum('N','Y') | | | N | |
| Grant_priv | enum('N','Y') | | | N | |
| References_priv | enum('N','Y') | | | N | |
| Index_priv | enum('N','Y') | | | N | |
| Alter_priv | enum('N','Y') | | | N | |
+-----------------+--------------+------+-----+---------+-------+
```

The first three fields (referred to as "scope fields") are used to define which users are allowed to connect to the database server, their passwords, and the hosts from which they may connect − mySQL uses a combination of both user and host identification as the basis for its security system. Consider the following extract from this table:

```
+--------------------+-------+----------+
| Host | User | Password |
+--------------------+-------+----------+
| turkey.domain.com | john | |
+--------------------+-------+----------+
```

This implies that the user "john" (password null) is allowed to connect from the host "turkey.domain.com"

It's also possible to specify wildcards − the following example would allow access to a user named "john", regardless of the host from which the connection is requested.

```
+-----------------+--------+
| Host | User | Password |
+-----------------+--------+
| % | john | |
+-----------------+--------+
```

The % character is used as a wildcard − the following example would match any user named "john" connecting from the "loudbeep.com" domain.

```
+-----------------+--------+-----------+
| Host | User | Password |
+-----------------+--------+-----------+
| %.loudbeep.com | john | |
+-----------------+--------+-----------+
```

Born Privileged

Once the users, passwords and hosts are specified, it becomes necessary to specify the privileges each user has − which is where the other fourteen columns (or "privilege fields") come in. Here's what each of those fields represents:

Select_priv − execute a SELECT query

Insert_priv − execute an INSERT query

Update_priv − execute an UPDATE query

Delete_priv − execute a DELETE query

Create_priv − CREATE databases and tables

Drop_priv − DROP databases and tables

Reload_priv − Reload/refresh the mySQL server

Shutdown_priv − Shut down a running mySQL server

Process_priv − track activity on a mySQL server

File_priv − read and write files on the server

Grant_priv − GRANT other users privileges

Index_priv − Create, edit and delete indices

Alter_priv − execute an ALTER query

It is important to note at this point that the security privileges assigned to each user in the "user" table are globally valid − they apply to each and every database on the system. The following record

---

```
+-----------------+---------------+
| Field | Value |
+-----------------+---------------+
| Host | apple.pie.com |
| User | joe |
| Password | gf64us |
| Select_priv | Y |
| Insert_priv | N |
| Update_priv | N |
| Delete_priv | Y |
```

```
| Create_priv | N |
| Drop_priv | N |
| Reload_priv | N |
| Shutdown_priv | N |
| Process_priv | N |
| File_priv | N |
| Grant_priv | N |
| References_priv | N |
| Index_priv | N |
| Alter_priv | N |
+-----------------+--------------+
```

would imply that user "joe" has the ability to DELETE records from any table in any database on the server − not a Good Thing if Joe happens to be in a bad mood. It is for this reason that most administrators (and the mySQL manual) recommends leaving all privileges in this table to "N" (the default value) for every user, and using the "host" and "db" tables to assign more focused levels of access.

Similarly, the following record would create a super−user named "god", with complete access to all mySQL privileges.

```
+-----------------+--------------+
| Field | Value |
+-----------------+--------------+
| Host | apple.pie.com |
| User | god |
| Password | hjgj4j34 |
| Select_priv | Y |
| Insert_priv | Y |
| Update_priv | Y |
| Delete_priv | Y |
| Create_priv | Y |
| Drop_priv | Y |
| Reload_priv | Y |
| Shutdown_priv | Y |
| Process_priv | Y |
| File_priv | Y |
| Grant_priv | Y |
| References_priv | Y |
| Index_priv | Y |
| Alter_priv | Y |
+-----------------+--------------+
```

It is instructive at this point to look at the default "user" table that ships with mySQL, in order to better understand the implications of running an out−of−the−box mySQL setup.

```
+-----------+------+----------+----------------+
| Host | User | Password | all_privileges |
+-----------+------+----------+----------------+
| localhost | root | | Y |
| % | | | N |
+-----------+------+----------+----------------+
```

In other words, the user connecting as "root" from "localhost" has complete access, while any other user, connecting from any other host, would not be able to perform any actions at all.

The Belly Of The Beast

The "host" and "db" tables are used together − they control which databases are available to which users, and the operations possible on those databases. Take a look at the fields in a typical "db" table:

```
+-----------------+---------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-----------------+---------------+------+-----+---------+-------+
| Host | char(60) | | PRI | | |
| Db | char(32) | | PRI | | |
| User | char(16) | | PRI | | |
| Select_priv | enum('N','Y') | | | N | |
| Insert_priv | enum('N','Y') | | | N | |
| Update_priv | enum('N','Y') | | | N | |
| Delete_priv | enum('N','Y') | | | N | |
| Create_priv | enum('N','Y') | | | N | |
| Drop_priv | enum('N','Y') | | | N | |
| Grant_priv | enum('N','Y') | | | N | |
| References_priv | enum('N','Y') | | | N | |
| Index_priv | enum('N','Y') | | | N | |
| Alter_priv | enum('N','Y') | | | N | |
+-----------------+---------------+------+-----+---------+-------+
```

Again, the first three fields are scope fields, which link a specific user and host to one or more databases. The remaining fields are used to specify the type of operations the user can perform on the named database.

A record like this would imply that the user "bill", connecting from host "cranberry.domain.com", would be able to use database "darkbeast"

```
+----------------------+------+-----------+----------------+
| Host | User | Db | all_privileges |
+----------------------+------+-----------+----------------+
| cranberry.domain.com | bill | darkbeast | Y |
+----------------------+------+-----------+----------------+
```

while this would imply that any user, connecting from any host, would have complete access to the "test" database.

```
+------+------+---------+----------------+
| Host | User | Db | all_privileges |
+------+------+---------+----------------+
| % | | test | Y |
+------+------+---------+----------------+
```

The Perfect Host

A blank entry under the "Host" column in the "db" table implies that the list of allowed hosts should be obtained from the third table, the "hosts" table − which looks like this:

```
+-----------------+--------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| Host | char(60) | | PRI | | |
| Db | char(32) | | PRI | | |
| Select_priv | enum('N','Y') | | | N | |
| Insert_priv | enum('N','Y') | | | N | |
| Update_priv | enum('N','Y') | | | N | |
| Delete_priv | enum('N','Y') | | | N | |
| Create_priv | enum('N','Y') | | | N | |
| Drop_priv | enum('N','Y') | | | N | |
| Grant_priv | enum('N','Y') | | | N | |
| References_priv | enum('N','Y') | | | N | |
| Index_priv | enum('N','Y') | | | N | |
| Alter_priv | enum('N','Y') | | | N | |
+-----------------+--------------+------+-----+---------+-------+
```

This separation is more useful than you might think. If you would like to connect and use a database from several different hosts, you would usually have to create a separate entry naming each host in the "db" table. However, with the introduction of the "host" table, you can place the host names in the "host" table while retaining only a single entry (with a blank "Host" field) in the "user" table.

The "host" table also has privilege fields − these allow you to control the level of access for each database, with the connecting host name as the criteria for operation.

Here's an example of how the relationship between the "host" and "db" table can be exploited for maximum benefit:

```
mysql> SELECT Host, User, Password FROM user;
+-----------+------+----------+
| Host | User | Password |
+-----------+------+----------+
| | jim | h35472k |
+-----------+------+----------+
```

```
mysql> SELECT Host, User, Db FROM db
+------+------+-------+
| Host | User | Db |
+------+------+-------+
| | jim | title |
+------+------+-------+


mysql> SELECT Host, Db, Select_priv, Insert_priv FROM host
+-------------------+-------+---------------+---------------+
| Host | Db | Select_priv | Insert_priv |
+-------------------+-------+---------------+---------------+
| turkey.ix6.com | title | Y | Y |
+-------------------+-------+---------------+---------------+
| blackbox.glue.net | title | Y | N |
+-------------------+-------+---------------+---------------+
| fireball.home.net | title | Y | Y |
+-------------------+-------+---------------+---------------+
```

In this case, "jim" will be able to connect to the mySQL server from any of the hosts listed in the "hosts" table, and the privileges assigned can differ on the basis of the host.

An important point to be noted is that, in the hierarchy of mySQL grant tables, the "user" table comes first, with the "db" and "host" tables below it, and the "tables_priv" and "columns_priv" tables at the bottom. A table at a lower level is referred to only if a higher−level table fails to provide the necessary scope or privileges.

When deciding whether or not to allow a particular database operation, mySQL takes the privilege fields in all three tables into account. It starts with the "user" table and checks to see if the user has appropriate privileges for the operation being attempted; if not, the "db" and "host" tables are checked to see if privileges are available. It is only after logically parsing the privileges in the different tables that mySQL allows or disallows a specific database request.

Cream Of The Crop

In addition to the three tables already discussed, newer versions of mySQL also come with two additional tables, the "tables_priv" and "columns_priv" tables. These allow a database administrator to restrict access to specific tables in a database, and specific columns of a specific table, respectively.

Here's what these two tables look like:

```
tables_priv:

+-------------+--------------------------+
| Field | Type |
+-------------+--------------------------+
| Host | char(60) |
| Db | char(60) |
| User | char(16) |
| Table_name | char(60) |
| Grantor | char(77) |
| Timestamp | timestamp(14) |
| Table_priv | set('Select','Insert'...) |
| Column_priv | set('Select','Insert'...) |
+-------------+--------------------------+

columns_priv:

+-------------+-----------------------------------------------+
| Field | Type |
+-------------+-----------------------------------------------+
| Host | char(60) |
| Db | char(60) |
| User | char(16) |
| Table_name | char(60) |
| Column_name | char(60) |
| Timestamp | timestamp(14) |
| Column_priv | set('Select','Insert','Update','References') |
+-------------+-----------------------------------------------+
```

The following example would restrict user to performing SELECT operations on table "cream" *only* − any attempt to run a SELECT query on another table within the same database would result in an error.

```
+-----------------+-------+------+------------+--------------+-----------
| Host | Db | User | Table_name | Table_priv | Column_priv |
+-----------------+-------+------+------------+--------------+-----------
|lost.soul.com | db563 | john | cream | Select | |
+-----------------+-------+------+------------+--------------+-----------
```

The Mechanics

Now that you know how the grant tables work, the final item on the agenda is the mechanics of implementing changes to the tables. MySQL offers two methods of altering access rights in the grant tables − you can either use INSERT, UPDATE and DELETE queries to alter the information in the tables, or use the GRANT and REVOKE commands.

Personally, I prefer the former, since it's much easier to understand and remember − although typing in long−winded SQL queries is sometimes a little tedious. Power users would do well to learn GRANT and REVOKE command syntax − details are available in the mySQL manual. For the moment, I'll simply take you through a couple of examples, using both methods, so that you have some insight into the differences between the two methods.

The first example sets up a user "tom", password "tommygun", who has permission to access the "recipes" database only from "localhost"

```
mysql> INSERT INTO user (Host, User, Password)
VALUES('localhost','tom',PASSWORD('tommygun'));
mysql> INSERT INTO db (Host, Db, User, Select_priv,
Insert_priv,
Update_priv, Delete_priv, Create_priv, Drop_priv) VALUES
('localhost','recipes','tom','Y','Y','Y','Y','N','N');
```

The equivalent GRANT command is:

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, ON recipes.* TO
tom@localhost
IDENTIFIED BY 'tommygun';
```

You could set up an equivalent of the "root" user with

```
mysql> GRANT ALL PRIVILEGES ON *.* TO god@localhost IDENTIFIED
BY 'master';
```

or

```
mysql> INSERT INTO user (Host, User, Password, Select_priv,
Insert_priv,
Update_priv, Delete_priv, Create_priv, Drop_priv, Reload_priv,
Shutdown_priv, Process_priv, File_priv, Grant_priv,
References_priv,
Index_priv, Alter_priv) VALUES ('localhost', 'god',
PASSWORD('master'),
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y')
```

It should be noted that privileges set using GRANT and REVOKE are immediately activated; however, privileges set via regular SQL queries require a server reload to come into effect. A server reload can be accomplished via the "mysqladmin" command

```
$ mysqladmin reload
```

or with the

```
mysql> FLUSH PRIVILEGES;
```

command.

And that's about it. I hope you find this information useful, and that you can use it when maintaining your own databases. Ciao!