Replicating User Account Management Statements

Account management statements (CREATE USER, DROP USER, RENAME USER, SET PASSWORD, GRANT, REVOKE) are considered modifications to the system database.

Best Practices

Here are five best practices that can be followed to log account management/system level operations:

1. Create an initial RBR binlog.

```
master> CREATE BINLOG 'b1' FORMAT='ROW'; /* b1 will record all database operations and system level
operations */
```

2. If you're going to have multiple binlogs then identify the ones for which system level logging is required and add system to the binlog explicitly.

```
master> CREATE BINLOG 'b2' LOG('system'), FORMAT='ROW'; /* b2 will record only account management
statements */
master> CREATE BINLOG 'b3' LOG('system', 'db3'), FORMAT='ROW'; /* b3 will record all system level
operations and all operations on database 'db3'*/
```

3. If there is already a binlog and you want to log system level operations then ALTER the binlog to include it.

```
master> CREATE BINLOG 'b4' LOG('db4'), FORMAT='ROW';
master> ALTER BINLOG 'b4' ADD LOG('system') /* b4 will record all system level operations and all
operations on database 'db4'*/
```

4. If you want to exclude some database(s)/table(s) from the binlog scope which was defined to include system level operations, IGNORE the specific database.

```
master> CREATE BINLOG 'b5' IGNORE('db5'), FORMAT='ROW'; /* b5 will record all system level operations and
all database operations except 'db5' */
```

5. This binlog is for a specific table. Note that GRANT and REVOKE statements that refer to the specific databases or tables modify the system database so will be excluded.

```
master> CREATE BINLOG 'b6' LOG('db6.tbl6'), FORMAT='ROW'; /* b6 will not record all system level
operations. It will only record specific system level and database operations pertaining to table db6.
tbl6 */
```

Examples

Here are four examples of multi-binlog configurations that would safely replicate account management statements (in addition to normal database changes):

1. A database-scope binlog and a general binlog:

```
master> CREATE BINLOG foo LOG(`foo`), FORMAT='ROW';
master> CREATE BINLOG everything_else IGNORE(`foo`), FORMAT='ROW';
```

   The everything_else binlog will log changes to system (i.e. account management statements), along with changes to every other database besides foo.

2. A table-scope binlog and a general binlog:

```
master> CREATE BINLOG foo_tables LOG(`foo`.`a`, `foo`.`b`, `foo`.`c`), FORMAT='ROW';
master> CREATE BINLOG everything_else IGNORE(`foo`.`a`, `foo`.`b`, `foo`.`c`), FORMAT='ROW';
```

   The everything_else binlog will log changes to system and everything else besides those three tables in foo.

3. A database-scope binlog and a table-scope binlog:

```
master> CREATE BINLOG foo_tables LOG(`foo`.`a`), FORMAT='ROW';
master> CREATE BINLOG foo_plus_system LOG(`foo`, `system`), IGNORE(`foo`.`a`), FORMAT='ROW';
```

We have to add system to one of the binlogs, because there is no general binlog. In this case, we added it to the database-scope binlog. We could have added it to the table-scope binlog instead or created a separate binlog just for system (which we do in the next example).

4. Multiple database-scope binlogs:

```
master> CREATE BINLOG foo LOG(`foo`), FORMAT='ROW';
master> CREATE BINLOG bar LOG(`bar`), FORMAT='ROW';
master> CREATE BINLOG system LOG(`system`), FORMAT='ROW';
```

There is no general binlog, so we have to explicitly include system in one of the binlogs. In this case, we just give it its own binlog. FORMAT='ROW' does not make a difference for this binlog, but it's a good habit to always use RBR.