13.7.1.6 GRANT Statement

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user_or_role [, user_or_role] ...
    [WITH GRANT OPTION]
    [AS user
        [WITH ROLE
            DEFAULT
          | NONE
          | ALL
          | ALL EXCEPT role [, role ] ...
          | role [, role ] ...
        ]
    ]
}

GRANT PROXY ON user_or_role
    TO user_or_role [, user_or_role] ...
    [WITH GRANT OPTION]

GRANT role [, role] ...
    TO user_or_role [, user_or_role] ...
    [WITH ADMIN OPTION]

object_type: {
    TABLE
  | FUNCTION
  | PROCEDURE
}

priv_level: {
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.routine_name
}

user_or_role: {
    user (see Section 6.2.4, “Specifying Account Names”)
  | role (see Section 6.2.5, “Specifying Role Names”)
}
```

The GRANT statement assigns privileges and roles to MySQL user accounts and roles. There are several aspects to the GRANT statement, described under the following topics:

- GRANT General Overview
- Object Quoting Guidelines
- Account Names
- Privileges Supported by MySQL
- Global Privileges
- Database Privileges
- Table Privileges
- Column Privileges
- Stored Routine Privileges
- Proxy User Privileges
- Granting Roles
- The AS Clause and Privilege Restrictions
- Other Account Characteristics
- MySQL and Standard SQL Versions of GRANT

GRANT General Overview

The GRANT statement enables system administrators to grant privileges and roles, which can be granted to user accounts and roles. These syntax restrictions apply:

- GRANT cannot mix granting both privileges and roles in the same statement. A given GRANT statement must grant either privileges or roles.
- The ON clause distinguishes whether the statement grants privileges or roles:
  - With ON, the statement grants privileges.
  - Without ON, the statement grants roles.
  - It is permitted to assign both privileges and roles to an account, but you must use separate GRANT statements, each with syntax appropriate to what is to be granted.

For more information about roles, see Section 6.2.10, “Using Roles”.

To grant a privilege with `GRANT`, you must have the `GRANT OPTION` privilege, and you must have the privileges that you are granting. (Alternatively, if you have the `UPDATE` privilege for the grant tables in the `mysql` system schema, you can grant any account any privilege.) When the `read_only` system variable is enabled, `GRANT` additionally requires the `CONNECTION_ADMIN` privilege (or the deprecated `SUPER` privilege).

`GRANT` either succeeds for all named users and roles or rolls back and has no effect if any error occurs. The statement is written to the binary log only if it succeeds for all named users and roles.

The `REVOKE` statement is related to `GRANT` and enables administrators to remove account privileges. See Section 13.7.1.8, “REVOKE Statement”.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. Each role name uses the format described in Section 6.2.5, “Specifying Role Names”. For example:

```
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
GRANT 'role1', 'role2' TO 'user1'@'localhost', 'user2'@'localhost';
GRANT SELECT ON world.* TO 'role3';
```

The host name part of the account or role name, if omitted, defaults to `'%'`.

Normally, a database administrator first uses `CREATE USER` to create an account and define its nonprivilege characteristics such as its password, whether it uses secure connections, and limits on access to server resources, then uses `GRANT` to define its privileges. `ALTER USER` may be used to change the nonprivilege characteristics of existing accounts. For example:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
GRANT SELECT ON db2.invoice TO 'jeffrey'@'localhost';
ALTER USER 'jeffrey'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;
```

From the **mysql** program, `GRANT` responds with `Query OK, 0 rows affected` when executed successfully. To determine what privileges result from the operation, use `SHOW GRANTS`. See Section 13.7.7.21, “SHOW GRANTS Statement”.

> **Important**
>
> Under some circumstances, `GRANT` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about the conditions under which this occurs for the server logs and

> how to control it, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, see Section 4.5.1.3, “mysql Client Logging”.

`GRANT` supports host names up to 255 characters long (60 characters prior to MySQL 8.0.17). User names can be up to 32 characters. Database, table, column, and routine names can be up to 64 characters.

> **Warning**
>
> *Do not attempt to change the permissible length for user names by altering the `mysql.user` system table. Doing so results in unpredictable behavior which may even make it impossible for users to log in to the MySQL server*. Never alter the structure of tables in the `mysql` system schema in any manner except by means of the procedure described in Section 2.11, “Upgrading MySQL”.

Object Quoting Guidelines

Several objects within `GRANT` statements are subject to quoting, although quoting is optional in many cases: Account, role, database, table, column, and routine names. For example, if a ***`user_name`*** or ***`host_name`*** value in an account name is legal as an unquoted identifier, you need not quote it. However, quotation marks are necessary to specify a ***`user_name`*** string containing special characters (such as -), or a ***`host_name`*** string containing special characters or wildcard characters such as `%` (for example, `'test-user'@'%.com'`). Quote the user name and host name separately.

To specify quoted values:

- Quote database, table, column, and routine names as identifiers.
- Quote user names and host names as identifiers or as strings.
- Quote passwords as strings.

For string-quoting and identifier-quoting guidelines, see Section 9.1.1, “String Literals”, and Section 9.2, “Schema Object Names”.

The `_` and `%` wildcards are permitted when specifying database names in `GRANT` statements that grant privileges at the database level (`GRANT ... ON` ***`db_name`*****`.*`**). This means, for example, that to use a `_` character as part of a database name, specify it using the `\` escape character as `\_` in the `GRANT` statement, to prevent the user from being able to access additional databases matching the wildcard pattern (for example, `GRANT ... ON` `` `foo\_bar`.* TO ... ``).

In privilege assignments, MySQL interprets occurrences of unescaped _ and % SQL wildcard characters in database names as literal characters under these circumstances:

- When a database name is not used to grant privileges at the database level, but as a qualifier for granting privileges to some other object such as a table or routine (for example, `GRANT ... ON db_name.tbl_name`).
- Enabling `partial_revokes` causes MySQL to interpret unescaped _ and % wildcard characters in database names as literal characters, just as if they had been escaped as `\_` and `\%`. Because this changes how MySQL interprets privileges, it may be advisable to avoid unescaped wildcard characters in privilege assignments for installations where `partial_revokes` may be enabled. For more information, see Section 6.2.12, “Privilege Restriction Using Partial Revokes”.

Account Names

A *`user`* value in a `GRANT` statement indicates a MySQL account to which the statement applies. To accommodate granting rights to users from arbitrary hosts, MySQL supports specifying the *`user`* value in the form `'user_name'@'host_name'`.

You can specify wildcards in the host name. For example, `'user_name'@'%.example.com'` applies to *`user_name`* for any host in the `example.com` domain, and `'user_name'@'198.51.100.%'` applies to *`user_name`* for any host in the `198.51.100` class C subnet.

The simple form `'user_name'` is a synonym for `'user_name'@'%'`.

MySQL does not support wildcards in user names. To refer to an anonymous user, specify an account with an empty user name with the `GRANT` statement:

```
GRANT ALL ON test.* TO ''@'localhost' ...;
```

In this case, any user who connects from the local host with the correct password for the anonymous user is permitted access, with the privileges associated with the anonymous-user account.

For additional information about user name and host name values in account names, see Section 6.2.4, “Specifying Account Names”.

> **Warning**
>
> If you permit local anonymous users to connect to the MySQL server, you should also grant privileges to all local users as `'user_name'@'localhost'`. Otherwise, the anonymous user account for `localhost` in the `mysql.user` system table is

used when named users try to log in to the MySQL server from the local machine. For details, see Section 6.2.6, “Access Control, Stage 1: Connection Verification”.

To determine whether this issue applies to you, execute the following query, which lists any anonymous users:

```
SELECT Host, User FROM mysql.user WHERE User='';
```

To avoid the problem just described, delete the local anonymous user account using this statement:

```
DROP USER ''@'localhost';
```

Privileges Supported by MySQL

The following tables summarize the permissible static and dynamic ***priv_type*** privilege types that can be specified for the `GRANT` and `REVOKE` statements, and the levels at which each privilege can be granted. For additional information about each privilege, see Section 6.2.2, “Privileges Provided by MySQL”. For information about the differences between static and dynamic privileges, see Static Versus Dynamic Privileges.

Table 13.11 Permissible Static Privileges for GRANT and REVOKE

Privilege	Meaning and Grantable Levels
`ALL [PRIVILEGES]`	Grant all privileges at specified access level except `GRANT OPTION` and `PROXY`.
`ALTER`	Enable use of `ALTER TABLE`. Levels: Global, database, table.
`ALTER ROUTINE`	Enable stored routines to be altered or dropped. Levels: Global, database, routine.
`CREATE`	Enable database and table creation. Levels: Global, database, table.
`CREATE ROLE`	Enable role creation. Level: Global.
`CREATE ROUTINE`	Enable stored routine creation. Levels: Global, database.
`CREATE TABLESPACE`	Enable tablespaces and log file groups to be created, altered, or dropped. Level: Global.
`CREATE TEMPORARY TABLES`	Enable use of `CREATE TEMPORARY TABLE`. Levels: Global, database.
`CREATE USER`	Enable use of `CREATE USER`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES`. Level: Global.

Privilege	Meaning and Grantable Levels
CREATE VIEW	Enable views to be created or altered. Levels: Global, database, table.
DELETE	Enable use of DELETE. Level: Global, database, table.
DROP	Enable databases, tables, and views to be dropped. Levels: Global, database, table.
DROP ROLE	Enable roles to be dropped. Level: Global.
EVENT	Enable use of events for the Event Scheduler. Levels: Global, database.
EXECUTE	Enable the user to execute stored routines. Levels: Global, database, routine.
FILE	Enable the user to cause the server to read or write files. Level: Global.
GRANT OPTION	Enable privileges to be granted to or removed from other accounts. Levels: Global, database, table, routine, proxy.
INDEX	Enable indexes to be created or dropped. Levels: Global, database, table.
INSERT	Enable use of INSERT. Levels: Global, database, table, column.
LOCK TABLES	Enable use of LOCK TABLES on tables for which you have the SELECT privilege. Levels: Global, database.
PROCESS	Enable the user to see all processes with SHOW PROCESSLIST. Level: Global.
PROXY	Enable user proxying. Level: From user to user.
REFERENCES	Enable foreign key creation. Levels: Global, database, table, column.
RELOAD	Enable use of FLUSH operations. Level: Global.
REPLICATION CLIENT	Enable the user to ask where source or replica servers are. Level: Global.
REPLICATION SLAVE	Enable replicas to read binary log events from the source. Level: Global.
SELECT	Enable use of SELECT. Levels: Global, database, table, column.
SHOW DATABASES	Enable SHOW DATABASES to show all databases. Level: Global.
SHOW VIEW	Enable use of SHOW CREATE VIEW. Levels: Global, database, table.
SHUTDOWN	Enable use of **mysqladmin shutdown**. Level: Global.
SUPER	Enable use of other administrative operations such as CHANGE REPLICATION SOURCE TO, CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and **mysqladmin debug** command. Level: Global.
TRIGGER	Enable trigger operations. Levels: Global, database, table.
UPDATE	Enable use of UPDATE. Levels: Global, database, table, column.
USAGE	Synonym for “no privileges”

Table 13.12 Permissible Dynamic Privileges for GRANT and REVOKE

Privilege	Meaning and Grantable Levels
APPLICATION_PASSWORD_ADMIN	Enable dual password administration. Level: Global.
AUDIT_ABORT_EXEMPT	Allow queries blocked by audit log filter. Level: Global.
AUDIT_ADMIN	Enable audit log configuration. Level: Global.
AUTHENTICATION_POLICY_ADMIN	Enable authentication policy administration. Level: Global.
BACKUP_ADMIN	Enable backup administration. Level: Global.
BINLOG_ADMIN	Enable binary log control. Level: Global.
BINLOG_ENCRYPTION_ADMIN	Enable activation and deactivation of binary log encryption. Level: Global.
CLONE_ADMIN	Enable clone administration. Level: Global.
CONNECTION_ADMIN	Enable connection limit/restriction control. Level: Global.
ENCRYPTION_KEY_ADMIN	Enable InnoDB key rotation. Level: Global.
FIREWALL_ADMIN	Enable firewall rule administration, any user. Level: Global.
FIREWALL_EXEMPT	Exempt user from firewall restrictions. Level: Global.
FIREWALL_USER	Enable firewall rule administration, self. Level: Global.
FLUSH_OPTIMIZER_COSTS	Enable optimizer cost reloading. Level: Global.
FLUSH_STATUS	Enable status indicator flushing. Level: Global.
FLUSH_TABLES	Enable table flushing. Level: Global.
FLUSH_USER_RESOURCES	Enable user-resource flushing. Level: Global.
GROUP_REPLICATION_ADMIN	Enable Group Replication control. Level: Global.
INNODB_REDO_LOG_ENABLE	Enable or disable redo logging. Level: Global.
INNODB_REDO_LOG_ARCHIVE	Enable redo log archiving administration. Level: Global.
NDB_STORED_USER	Enable sharing of user or role between SQL nodes (NDB Cluster). Level: Global.
PASSWORDLESS_USER_ADMIN	Enable passwordless user account administration. Level: Global.
PERSIST_RO_VARIABLES_ADMIN	Enable persisting read-only system variables. Level: Global.
REPLICATION_APPLIER	Act as the PRIVILEGE_CHECKS_USER for a replication channel. Level: Global.
REPLICATION_SLAVE_ADMIN	Enable regular replication control. Level: Global.
RESOURCE_GROUP_ADMIN	Enable resource group administration. Level: Global.
RESOURCE_GROUP_USER	Enable resource group administration. Level: Global.
ROLE_ADMIN	Enable roles to be granted or revoked, use of WITH ADMIN OPTION. Level: Global.
SESSION_VARIABLES_ADMIN	Enable setting restricted session system variables. Level: Global.

Privilege	Meaning and Grantable Levels
SET_USER_ID	Enable setting non-self DEFINER values. Level: Global.
SHOW_ROUTINE	Enable access to stored routine definitions. Level: Global.
SYSTEM_USER	Designate account as system account. Level: Global.
SYSTEM_VARIABLES_ADMIN	Enable modifying or persisting global system variables. Level: Global.
TABLE_ENCRYPTION_ADMIN	Enable overriding default encryption settings. Level: Global.
VERSION_TOKEN_ADMIN	Enable use of Version Tokens functions. Level: Global.
XA_RECOVER_ADMIN	Enable XA RECOVER execution. Level: Global.

A trigger is associated with a table. To create or drop a trigger, you must have the TRIGGER privilege for the table, not the trigger.

In GRANT statements, the ALL [PRIVILEGES] or PROXY privilege must be named by itself and cannot be specified along with other privileges. ALL [PRIVILEGES] stands for all privileges available for the level at which privileges are to be granted except for the GRANT OPTION and PROXY privileges.

MySQL account information is stored in the tables of the mysql system schema. For additional details, consult Section 6.2, “Access Control and Account Management”, which discusses the mysql system schema and the access control system extensively.

If the grant tables hold privilege rows that contain mixed-case database or table names and the lower_case_table_names system variable is set to a nonzero value, REVOKE cannot be used to revoke these privileges. It is necessary in such cases to manipulate the grant tables directly. (GRANT does not create such rows when lower_case_table_names is set, but such rows might have been created prior to setting that variable. The lower_case_table_names setting can only be configured at server startup.)

Privileges can be granted at several levels, depending on the syntax used for the ON clause. For REVOKE, the same ON syntax specifies which privileges to remove.

For the global, database, table, and routine levels, GRANT ALL assigns only the privileges that exist at the level you are granting. For example, GRANT ALL ON ***db_name*.*** is a database-level statement, so it does not grant any global-only privileges such as FILE. Granting ALL does not assign the GRANT OPTION or PROXY privilege.

The ***object_type*** clause, if present, should be specified as TABLE, FUNCTION, or PROCEDURE when the following object is a table, a stored function, or a stored procedure.

The privileges that a user holds for a database, table, column, or routine are formed additively as the logical OR of the account privileges at each of the privilege levels, including the global level. It is not

possible to deny a privilege granted at a higher level by absence of that privilege at a lower level. For example, this statement grants the `SELECT` and `INSERT` privileges globally:

```
GRANT SELECT, INSERT ON *.* TO u1;
```

The globally granted privileges apply to all databases, tables, and columns, even though not granted at any of those lower levels.

As of MySQL 8.0.16, it is possible to explicitly deny a privilege granted at the global level by revoking it for particular databases, if the `partial_revokes` system variable is enabled:

```
GRANT SELECT, INSERT, UPDATE ON *.* TO u1;
REVOKE INSERT, UPDATE ON db1.* FROM u1;
```

The result of the preceding statements is that `SELECT` applies globally to all tables, whereas `INSERT` and `UPDATE` apply globally except to tables in `db1`. Account access to `db1` is read only.

Details of the privilege-checking procedure are presented in Section 6.2.7, “Access Control, Stage 2: Request Verification”.

If you are using table, column, or routine privileges for even one user, the server examines table, column, and routine privileges for all users and this slows down MySQL a bit. Similarly, if you limit the number of queries, updates, or connections for any users, the server must monitor these values.

MySQL enables you to grant privileges on databases or tables that do not exist. For tables, the privileges to be granted must include the `CREATE` privilege. *This behavior is by design*, and is intended to enable the database administrator to prepare user accounts and privileges for databases or tables that are to be created at a later time.

> **Important**
>
> *MySQL does not automatically revoke any privileges when you drop a database or table*. However, if you drop a routine, any routine-level privileges granted for that routine are revoked.

Global Privileges

Global privileges are administrative or apply to all databases on a given server. To assign global privileges, use `ON *.*` syntax:

```
GRANT ALL ON *.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON *.* TO 'someuser'@'somehost';
```

The CREATE TABLESPACE, CREATE USER, FILE, PROCESS, RELOAD, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES, SHUTDOWN, and SUPER static privileges are administrative and can only be granted globally.

Dynamic privileges are all global and can only be granted globally.

Other privileges can be granted globally or at more specific levels.

The effect of GRANT OPTION granted at the global level differs for static and dynamic privileges:

- GRANT OPTION granted for any static global privilege applies to all static global privileges.
- GRANT OPTION granted for any dynamic privilege applies only to that dynamic privilege.

GRANT ALL at the global level grants all static global privileges and all currently registered dynamic privileges. A dynamic privilege registered subsequent to execution of the GRANT statement is not granted retroactively to any account.

MySQL stores global privileges in the mysql.user system table.

Database Privileges

Database privileges apply to all objects in a given database. To assign database-level privileges, use ON ***db_name*.*** syntax:

```
GRANT ALL ON mydb.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.* TO 'someuser'@'somehost';
```

If you use ON * syntax (rather than ON *.*), privileges are assigned at the database level for the default database. An error occurs if there is no default database.

The CREATE, DROP, EVENT, GRANT OPTION, LOCK TABLES, and REFERENCES privileges can be specified at the database level. Table or routine privileges also can be specified at the database level, in which case they apply to all tables or routines in the database.

MySQL stores database privileges in the mysql.db system table.

Table Privileges

Table privileges apply to all columns in a given table. To assign table-level privileges, use ON ***db_name.tbl_name*** syntax:

```
GRANT ALL ON mydb.mytbl TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.mytbl TO 'someuser'@'somehost';
```

If you specify ***tbl_name*** rather than ***db_name.tbl_name***, the statement applies to ***tbl_name*** in the default database. An error occurs if there is no default database.

The permissible ***priv_type*** values at the table level are `ALTER`, `CREATE VIEW`, `CREATE`, `DELETE`, `DROP`, `GRANT OPTION`, `INDEX`, `INSERT`, `REFERENCES`, `SELECT`, `SHOW VIEW`, `TRIGGER`, and `UPDATE`.

Table-level privileges apply to base tables and views. They do not apply to tables created with `CREATE TEMPORARY TABLE`, even if the table names match. For information about `TEMPORARY` table privileges, see Section 13.1.20.2, “CREATE TEMPORARY TABLE Statement”.

MySQL stores table privileges in the `mysql.tables_priv` system table.

Column Privileges

Column privileges apply to single columns in a given table. Each privilege to be granted at the column level must be followed by the column or columns, enclosed within parentheses.

```
GRANT SELECT (col1), INSERT (col1, col2) ON mydb.mytbl TO 'someuser'@'somehost';
```

The permissible ***priv_type*** values for a column (that is, when you use a ***column_list*** clause) are `INSERT`, `REFERENCES`, `SELECT`, and `UPDATE`.

MySQL stores column privileges in the `mysql.columns_priv` system table.

Stored Routine Privileges

The `ALTER ROUTINE`, `CREATE ROUTINE`, `EXECUTE`, and `GRANT OPTION` privileges apply to stored routines (procedures and functions). They can be granted at the global and database levels. Except for `CREATE ROUTINE`, these privileges can be granted at the routine level for individual routines.

```
GRANT CREATE ROUTINE ON mydb.* TO 'someuser'@'somehost';
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'someuser'@'somehost';
```

The permissible ***priv_type*** values at the routine level are `ALTER ROUTINE`, `EXECUTE`, and `GRANT OPTION`. `CREATE ROUTINE` is not a routine-level privilege because you must have the privilege at the global or database level to create a routine in the first place.

MySQL stores routine-level privileges in the `mysql.procs_priv` system table.

Proxy User Privileges

The PROXY privilege enables one user to be a proxy for another. The proxy user impersonates or takes the identity of the proxied user; that is, it assumes the privileges of the proxied user.

```
GRANT PROXY ON 'localuser'@'localhost' TO 'externaluser'@'somehost';
```

When PROXY is granted, it must be the only privilege named in the GRANT statement, and the only permitted WITH option is WITH GRANT OPTION.

Proxying requires that the proxy user authenticate through a plugin that returns the name of the proxied user to the server when the proxy user connects, and that the proxy user have the PROXY privilege for the proxied user. For details and examples, see Section 6.2.19, “Proxy Users”.

MySQL stores proxy privileges in the mysql.proxies_priv system table.

Granting Roles

GRANT syntax without an ON clause grants roles rather than individual privileges. A role is a named collection of privileges; see Section 6.2.10, “Using Roles”. For example:

```
GRANT 'role1', 'role2' TO 'user1'@'localhost', 'user2'@'localhost';
```

Each role to be granted must exist, as well as each user account or role to which it is to be granted. As of MySQL 8.0.16, roles cannot be granted to anonymous users.

Granting a role does not automatically cause the role to be active. For information about role activation and inactivation, see Activating Roles.

These privileges are required to grant roles:

- If you have the ROLE_ADMIN privilege (or the deprecated SUPER privilege), you can grant or revoke any role to users or roles.
- If you were granted a role with a GRANT statement that includes the WITH ADMIN OPTION clause, you become able to grant that role to other users or roles, or revoke it from other users or roles, as long as the role is active at such time as you subsequently grant or revoke it. This includes the ability to use WITH ADMIN OPTION itself.
- To grant a role that has the SYSTEM_USER privilege, you must have the SYSTEM_USER privilege.

It is possible to create circular references with GRANT. For example:

```
CREATE USER 'u1', 'u2';
CREATE ROLE 'r1', 'r2';

GRANT 'u1' TO 'u1';   -- simple loop: u1 => u1
GRANT 'r1' TO 'r1';   -- simple loop: r1 => r1

GRANT 'r2' TO 'u2';
GRANT 'u2' TO 'r2';   -- mixed user/role loop: u2 => r2 => u2
```

Circular grant references are permitted but add no new privileges or roles to the grantee because a user or role already has its privileges and roles.

The AS Clause and Privilege Restrictions

As of MySQL 8.0.16, GRANT has an AS ***user*** [WITH ROLE] clause that specifies additional information about the privilege context to use for statement execution. This syntax is visible at the SQL level, although its primary purpose is to enable uniform replication across all nodes of grantor privilege restrictions imposed by partial revokes, by causing those restrictions to appear in the binary log. For information about partial revokes, see Section 6.2.12, “Privilege Restriction Using Partial Revokes”.

When the AS ***user*** clause is specified, statement execution takes into account any privilege restrictions associated with the named user, including all roles specified by WITH ROLE, if present. The result is that the privileges actually granted by the statement may be reduced relative to those specified.

These conditions apply to the AS ***user*** clause:

- AS has an effect only when the named ***user*** has privilege restrictions (which implies that the partial_revokes system variable is enabled).
- If WITH ROLE is given, all roles named must be granted to the named ***user***.
- The named ***user*** should be a MySQL account specified as '***user_name***'@'***host_name***', CURRENT_USER, or CURRENT_USER(). The current user may be named together with WITH ROLE for the case that the executing user wants GRANT to execute with a set of roles applied that may differ from the roles active within the current session.
- AS cannot be used to gain privileges not possessed by the user who executes the GRANT statement. The executing user must have at least the privileges to be granted, but the AS clause can only restrict the privileges granted, not escalate them.
- With respect to the privileges to be granted, AS cannot specify a user/role combination that has more privileges (fewer restrictions) than the user who executes the GRANT statement. The AS user/role combination is permitted to have more privileges than the executing user, but only if the statement does not grant those additional privileges.

- `AS` is supported only for granting global privileges (`ON *.*`).
- `AS` is not supported for `PROXY` grants.

The following example illustrates the effect of the `AS` clause. Create a user `u1` that has some global privileges, as well as restrictions on those privileges:

```
CREATE USER u1;
GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u1;
REVOKE INSERT, UPDATE ON schema1.* FROM u1;
REVOKE SELECT ON schema2.* FROM u1;
```

Also create a role `r1` that lifts some of the privilege restrictions and grant the role to `u1`:

```
CREATE ROLE r1;
GRANT INSERT ON schema1.* TO r1;
GRANT SELECT ON schema2.* TO r1;
GRANT r1 TO u1;
```

Now, using an account that has no privilege restrictions of its own, grant to multiple users the same set of global privileges, but each with different restrictions imposed by the `AS` clause, and check which privileges are actually granted.

- The `GRANT` statement here has no `AS` clause, so the privileges granted are exactly those specified:

```
mysql> CREATE USER u2;
mysql> GRANT SELECT, INSERT, UPDATE ON *.* TO u2;
mysql> SHOW GRANTS FOR u2;
+-----------------------------------------------+
| Grants for u2@%                               |
+-----------------------------------------------+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u2`@`%` |
+-----------------------------------------------+
```

- The `GRANT` statement here has an `AS` clause, so the privileges granted are those specified but with the restrictions from `u1` applied:

```
mysql> CREATE USER u3;
mysql> GRANT SELECT, INSERT, UPDATE ON *.* TO u3 AS u1;
mysql> SHOW GRANTS FOR u3;
+-----------------------------------------------+
| Grants for u3@%                               |
+-----------------------------------------------+
```

```
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u3`@`%`    |
| REVOKE INSERT, UPDATE ON `schema1`.* FROM `u3`@`%` |
| REVOKE SELECT ON `schema2`.* FROM `u3`@`%`         |
+----------------------------------------------------+
```

As mentioned previously, the AS clause can only add privilege restrictions; it cannot escalate privileges. Thus, although u1 has the DELETE privilege, that is not included in the privileges granted because the statement does not specify granting DELETE.

- The AS clause for the GRANT statement here makes the role r1 active for u1. That role lifts some of the restrictions on u1. Consequently, the privileges granted have some restrictions, but not so many as for the previous GRANT statement:

```
mysql> CREATE USER u4;
mysql> GRANT SELECT, INSERT, UPDATE ON *.* TO u4 AS u1 WITH ROLE r1;
mysql> SHOW GRANTS FOR u4;
+-----------------------------------------------+
| Grants for u4@%                               |
+-----------------------------------------------+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u4`@`%` |
| REVOKE UPDATE ON `schema1`.* FROM `u4`@`%`      |
+-----------------------------------------------+
```

If a GRANT statement includes an AS ***user*** clause, privilege restrictions on the user who executes the statement are ignored (rather than applied as they would be in the absence of an AS clause).

Other Account Characteristics

The optional WITH clause is used to enable a user to grant privileges to other users. The WITH GRANT OPTION clause gives the user the ability to give to other users any privileges the user has at the specified privilege level.

To grant the GRANT OPTION privilege to an account without otherwise changing its privileges, do this:

```
GRANT USAGE ON *.* TO 'someuser'@'somehost' WITH GRANT OPTION;
```

Be careful to whom you give the GRANT OPTION privilege because two users with different privileges may be able to combine privileges!

You cannot grant another user a privilege which you yourself do not have; the GRANT OPTION privilege enables you to assign only those privileges which you yourself possess.

Be aware that when you grant a user the `GRANT OPTION` privilege at a particular privilege level, any privileges the user possesses (or may be given in the future) at that level can also be granted by that user to other users. Suppose that you grant a user the `INSERT` privilege on a database. If you then grant the `SELECT` privilege on the database and specify `WITH GRANT OPTION`, that user can give to other users not only the `SELECT` privilege, but also `INSERT`. If you then grant the `UPDATE` privilege to the user on the database, the user can grant `INSERT`, `SELECT`, and `UPDATE`.

For a nonadministrative user, you should not grant the `ALTER` privilege globally or for the `mysql` system schema. If you do that, the user can try to subvert the privilege system by renaming tables!

For additional information about security risks associated with particular privileges, see Section 6.2.2, “Privileges Provided by MySQL”.

MySQL and Standard SQL Versions of GRANT

The biggest differences between the MySQL and standard SQL versions of `GRANT` are:

- MySQL associates privileges with the combination of a host name and user name and not with only a user name.
- Standard SQL does not have global or database-level privileges, nor does it support all the privilege types that MySQL supports.
- MySQL does not support the standard SQL `UNDER` privilege.
- Standard SQL privileges are structured in a hierarchical manner. If you remove a user, all privileges the user has been granted are revoked. This is also true in MySQL if you use `DROP USER`. See Section 13.7.1.5, “DROP USER Statement”.
- In standard SQL, when you drop a table, all privileges for the table are revoked. In standard SQL, when you revoke a privilege, all privileges that were granted based on that privilege are also revoked. In MySQL, privileges can be dropped with `DROP USER` or `REVOKE` statements.
- In MySQL, it is possible to have the `INSERT` privilege for only some of the columns in a table. In this case, you can still execute `INSERT` statements on the table, provided that you insert values only for those columns for which you have the `INSERT` privilege. The omitted columns are set to their implicit default values if strict SQL mode is not enabled. In strict mode, the statement is rejected if any of the omitted columns have no default value. (Standard SQL requires you to have the `INSERT` privilege on all columns.) For information about strict SQL mode and implicit default values, see Section 5.1.11, “Server SQL Modes”, and Section 11.6, “Data Type Default Values”.