

Chapter 27: MySQL User & Security Management

27.1 MySQL User Accounts

- * MySQL uses a **user-account system** to allow clients to connect to the server and access databases.
- * Each user account has a **username**, **host** and a **password**.
- * The **host** part specifies from where the user can connect.
- * MySQL checks **user privileges** to control what the user can do.



Key Points :

- ✓ Every user has a unique username and password.
- ✓ Host name determines where the user can connect from.
- ✓ Privileges define what actions a user can perform.
- ✓ Root user has all privileges by default.



Default MySQL User Accounts:

Username	Host	Description
root	localhost	Super user, full access to all databases.
mysql.session	localhost	Used internally by MySQL.
mysql.sys	localhost	Used internally by MySQL.
debian-sys-maint	localhost	System maintenance user (on some systems).



27.2 CREATE USER

- * The **CREATE USER** statement is used to create new MySQL user accounts.
- * You can specify the **username**, **host** and **password**.
- * Syntax :

```
CREATE USER 'username'@'host' IDENTIFIED BY 'password';
```



Examples :

- `CREATE USER 'john'@'localhost' IDENTIFIED BY 'john123';` -- User can connect only from this server.
- `CREATE USER 'alice'@'%' IDENTIFIED BY 'alice123';` -- User can connect from any host.
- `CREATE USER 'report'@'192.168.1.%' IDENTIFIED BY 'report123';` -- User can connect from local network.

Key Takeaway :



CREATE USER is the first step in MySQL security. Always create users with strong passwords and proper host restrictions.



27.3 ALTER USER

- * The **ALTER USER** statement is used to **modify** the properties or privileges of an existing MySQL user.
- * You can change the **password**, **host** or other attributes.
- * Syntax :



```
ALTER USER 'username'@'host' IDENTIFIED BY 'new_password';
```

- * You can also change other attributes like **account lock** or **privilege**.

Examples :

- ```
ALTER USER 'john'@'localhost' IDENTIFIED BY 'john@123';
```

 -- Change password for user 'john'.
- ```
ALTER USER 'alice'@'%' IDENTIFIED BY 'alice@123';
```

 -- Change password for user 'alice'.
- ```
ALTER USER 'report'@'192.168.1.%' IDENTIFIED BY 'newpass123';
```

 -- Change password for user 'report'.

### Key Points :

- ✓ **ALTER USER** is used to modify an existing user.
- ✓ You can change password, host or other attributes.
- ✓ Use **FLUSH PRIVILEGES**; after making changes to apply them.



## 27.4 DROP USER

- \* The **DROP USER** statement is used to **remove** a user account from MySQL.
- \* When a user is dropped, the user can no longer connect to the MySQL server.
- \* Syntax :

```
DROP USER 'username'@'host';
```

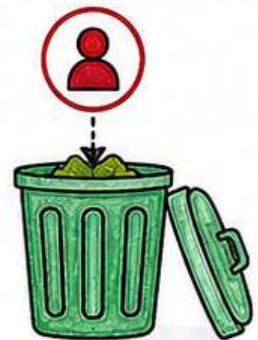
### Examples :

- ```
DROP USER 'john'@'localhost';
```

 -- Drop user 'john' from localhost.
- ```
DROP USER 'alice'@'%';
```

 -- Drop user 'alice' from any host.
- ```
DROP USER 'report'@'192.168.1.%';
```

 -- Drop user 'report' from local network.



Key Takeaway :



DROP USER permanently removes the user from the MySQL server. Use it carefully because the user will lose all access.



27.5 GRANT

- * The **GRANT** statement is used to **give privileges** to users.
- * Privileges allow users to perform specific actions on databases and tables.
- * You can grant privileges on all databases, a specific database or a specific table.
- * After granting privileges, always use **FLUSH PRIVILEGES** to apply changes.
- * Syntax :

```
GRANT privilege_list ON database_name.table_name  
TO 'username'@'host' IDENTIFIED BY 'password';
```



Examples :

- `GRANT SELECT ON student_db.* TO 'john'@'localhost';` -- Grant SELECT on all tables in database student_db.
- `GRANT INSERT, UPDATE ON student_db.marks TO 'alice'@'%';` -- Grant INSERT and UPDATE on table marks.
- `GRANT ALL PRIVILEGES ON *.* TO 'admin'@'localhost';` -- Grant all privileges on all databases and tables.
- `FLUSH PRIVILEGES;` -- Apply privilege changes.

Key Points :

- ✓ GRANT is used to **give privileges** to a user.
- ✓ Specify the **exact privileges** a user needs.
- ✓ Always use **FLUSH PRIVILEGES** after granting privileges.



27.6 REVOKE

- * The **REVOKE** statement is used to **remove privileges** from users.
- * It does not delete the user; it only removes the given privileges.
- * After revoking privileges, always use **FLUSH PRIVILEGES** to apply changes.
- * Syntax :

```
REVOKE privilege_list ON database_name.table_name  
FROM 'username'@'host';
```



Examples :

- `REVOKE SELECT ON student_db.* FROM 'john'@'localhost';` -- Revoke SELECT on all tables.
- `REVOKE INSERT, UPDATE ON student_db.marks FROM 'alice'@'%';` -- Revoke INSERT and UPDATE on table marks.
- `REVOKE ALL PRIVILEGES ON *.* FROM 'admin'@'localhost';` -- Revoke all privileges.
- `FLUSH PRIVILEGES;` -- Apply privilege changes.

Key Points :

- ✓ **REVOKE** is used to **remove specific privileges** from a user.
- ✓ The user remains, but the removed privileges cannot be used.
- ✓ Always use **FLUSH PRIVILEGES** after revoking privileges.



Key Takeaway :



GRANT gives permissions, **REVOKE** removes permissions. Use them carefully to manage **MySQL security** effectively.



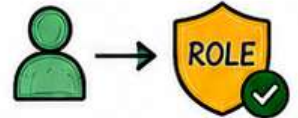
27.7 MySQL Roles

- * Roles are named **collections of privileges**.
- * Roles make it easier to manage permissions for multiple users.
- * You can **grant privileges to roles** and then **assign roles to users**.
- * After making changes, always use **FLUSH PRIVILEGES** to apply.
- * Syntax :

```
CREATE ROLE 'role_name'; -- Create a new role.
GRANT privilege_list ON database.table TO 'role_name'; -- Grant privileges to role.
GRANT 'role_name' TO 'username'@'host'; -- Assign role to user.
REVOKE 'role_name' FROM 'username'@'host'; -- Remove role from user.
DROP ROLE 'role_name'; -- Drop a role.
```



ROLE



Examples :

- CREATE ROLE 'data_reader'; -- Create role named data_reader.
- GRANT SELECT ON sales.* TO 'data_reader'; -- Grant SELECT privilege on sales database to role.
- GRANT 'data_reader' TO 'john'@'localhost'; -- Assign role to user john.
- REVOKE 'data_reader' FROM 'john'@'localhost'; -- Remove role from user john.
- DROP ROLE 'data_reader'; -- Drop role data_reader.

Key Points :

- ✓ Roles help in **grouping privileges** and simplifying permission management.
- ✓ Privileges are **granted to roles**, not directly to users.
- ✓ Roles can be assigned to **multiple users**.



27.8 Privileges

- * Privileges define what **actions** a user can perform.
- * Privileges can be applied at **various levels**.
- * Some of the common privileges are shown below.



Privilege	Description	Example
SELECT	Allows reading data from tables.	GRANT SELECT ON db.table TO user;
INSERT	Allows inserting data into tables.	GRANT INSERT ON db.table TO user;
UPDATE	Allows updating data in tables.	GRANT UPDATE ON db.table TO user;
DELETE	Allows deleting data from tables.	GRANT DELETE ON db.table TO user;
CREATE	Allows creating databases and tables.	GRANT CREATE ON database.* TO user;
DROP	Allows dropping databases or tables.	GRANT DROP ON database.* TO user;
ALL PRIVILEGES	Grants all available privileges.	GRANT ALL PRIVILEGES ON *.* TO user;
USAGE	No privileges, allows connection only.	GRANT USAGE ON *.* TO user;

Levels at which privileges can be applied:

- Global Level (*.*)
- Database Level (database.*)
- Table Level (database.table)
- Column Level (database.table.column)

Key Takeaway :



Privileges control what users can do in MySQL. Use the **minimum necessary** privileges to maintain a **secure** database environment.



27.9 Authentication

- * Authentication is the process of **verifying a user's identity**.
- * MySQL uses **authentication plugins** to authenticate users.
- * The default authentication plugin in MySQL 8.0 is **caching_sha2_password**.
- * Authentication plugins can be specified while creating or altering users.
- * After making changes, always use **FLUSH PRIVILEGES** to apply.

Syntax :

```
CREATE USER 'username'@'host' IDENTIFIED WITH plugin_name BY 'password';  
ALTER USER 'username'@'host' IDENTIFIED WITH plugin_name BY 'password';
```



Examples :

- `CREATE USER 'john'@'localhost' IDENTIFIED WITH caching_sha2_password BY 'john123';` -- Create user with default authentication plugin.
- `CREATE USER 'alice'@'%' IDENTIFIED WITH mysql_native_password BY 'alice123';` -- Create user with mysql_native_password plugin.
- `ALTER USER 'john'@'localhost' IDENTIFIED WITH mysql_native_password BY 'newpass123';` -- Change authentication plugin and password for user.
- `FLUSH PRIVILEGES;` -- Apply changes.

Key Points :

- ✓ Authentication **verifies who the user is**.
- ✓ Authentication plugins determine **how the password is verified**.
- ✓ Use **FLUSH PRIVILEGES** after changing authentication settings.
- ✓ Use strong plugins like **caching_sha2_password** for better security.



27.10 Password Management

- * MySQL provides commands to **change and manage** user passwords.
- * Passwords should be **strong and changed periodically**.
- * You can change passwords for your own account or other users (if you have permission).

Syntax :

```
ALTER USER 'username'@'host' IDENTIFIED BY 'new_password';  
SET PASSWORD FOR 'username'@'host' = PASSWORD('new_password');
```

Examples :

- `ALTER USER 'john'@'localhost' IDENTIFIED BY 'john@123';` -- Change password for user john.
- `SET PASSWORD FOR 'alice'@'%' = PASSWORD('alice@123');` -- Change password for user alice.
- `ALTER USER 'root'@'localhost' IDENTIFIED BY 'Root@1234';` -- Change password for root user.
- `FLUSH PRIVILEGES;` -- Apply changes.



Key Points :

- ✓ Use **ALTER USER** or **SET PASSWORD** to change passwords.
- ✓ Always use **strong passwords** with letters, numbers and symbols.
- ✓ Regular password changes **improve security**.
- ✓ Use **FLUSH PRIVILEGES** after changing passwords.

Strong Password Tips

- Use at least 8-12 characters.
- Use a mix of uppercase, lowercase, numbers and symbols.
- Avoid dictionary words and personal information.

Key Takeaway :



Authentication ensures the **right user** connects, and password management keeps the database **secure**. Use strong authentication plugins and **strong** passwords for a safe MySQL environment.

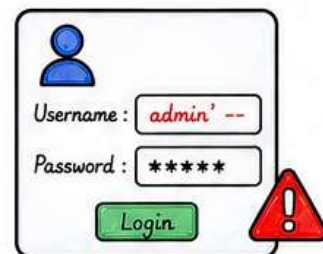


27.11 SQL Injection

- * SQL Injection is a **code injection technique** that might destroy your database.
- * It occurs when **user input** is not validated and is directly used in SQL queries.
- * Attackers can execute unwanted SQL commands through input fields.
- * Always **validate, sanitize** inputs and use **prepared statements** to prevent SQL Injection.

Example :

Normal Query	Injected Query
<code>SELECT * FROM users WHERE username = 'john' AND password = 'pass123';</code>	<code>" OR '1'='1' ; --</code> Bypasses login and returns all rows.
<code>SELECT * FROM users WHERE id = 1;</code>	<code>1; DROP TABLE users; --</code> Drops the users table.



Preventive Measures :

- ✓ Use **prepared statements** with parameterized queries.
- ✓ **Validate and sanitize** all user inputs.
- ✓ Use **least privilege** accounts.
- ✓ Avoid displaying SQL errors to users.
- ✓ Keep database and applications updated.



Example (Safe Query)

Prepared Statement in PHP (using MySQLi)

```
$stmt = $conn->prepare("SELECT * FROM users  
WHERE username=? AND password=?");  
$stmt->bind_param("ss", $username, $password);  
$stmt->execute();
```

-- User input is treated as data, not as code.

Key Takeaway :



SQL Injection can lead to **data theft, data loss** and complete **system compromise**.
Always use **secure coding practices** to protect your database.



27.12 Secure MySQL Applications

- * Securing MySQL applications is essential to protect data from unauthorized access.
- * Follow **best practices** to keep your database and application safe.
- * Security should be implemented at both database and application levels.
- * Regular **monitoring and updates** help in maintaining a secure environment.



Best Practices :

Practice	Description
Use Strong Passwords	Use strong, complex passwords and change them periodically.
Least Privilege	Grant only the minimum privileges needed for a user to perform tasks.
Use Prepared Statements	Use prepared statements to prevent SQL Injection.
Input Validation	Validate and sanitize all user inputs.
Regular Backups	Take regular backups and test restore process.
Keep Updated	Keep MySQL server and applications up to date.
Monitor and Audit	Monitor logs and audit access regularly.
Disable Unused Features	Disable unnecessary services and remove unused accounts.

Key Points :

- ✓ Security is everyone's responsibility.
- ✓ Follow best practices consistently.
- ✓ Regularly review users and privileges.
- ✓ Always use secure coding techniques.
- ✓ Backup your data regularly.



Key Takeaway :



A secure MySQL application protects your **data, users** and **business**.
Use **strong passwords, least privileges, validation, backups** and **monitoring** to stay secure.

