DATA PROTECTION AND SECURITY IN MYSQL DATABASE

Ivan ŠUŠTER
Darjan KARABAŠEVIĆ
Dragiša STANUJKIĆ
Gabrijela POPOVIĆ
Ana VELJIĆ
Marko MARKOVIĆ

Abstract: *Given the emphasis on protecting sensitive data within relational database management systems (RDBMS), data protection and security in MySQL databases are crucial factors in the modern information management landscape. A popular MySQL has a number of security and backup features that are necessary to preserve data availability and integrity. MySQL provide extensive data protection and security capabilities that are essential for businesses handling sensitive data. Organizations may significantly reduce the risks of data loss and breaches by putting in place strict access control, efficient backup plans, replication techniques, and secure communication protocols. MySQL's emphasis on improving data security and availability as it develops is in line with the rising demand for reliable data management solutions in a world that is becoming more and more digital. Therefore, aim of the paper is to provide insight regarding data protection and security in MySQL database.*
Keywords: *MySQL, security, database, protection*

INTRODUCTION

Ensuring the confidentiality, integrity, and availability of data in the database depends on its security. Security measures are necessary to prevent unauthorized access as well as other security risks. MySQL database security includes a number of activities, including the appropriate use of passwords, access and privilege controls, and the implementation of additional security mechanisms. The use of a password is necessary for access control and authentication. Granting users permissions according to their responsibilities is also a key factor in MySQL. Access control restricts access to specific databases and tables. The risk of accidental data change or unauthorized access can be reduced with access permissions.

MySQL's strong user authorization system is one of its key characteristics that are pertinent to data security. By using this approach, database administrators can specify particular access privileges for various users, guaranteeing that only authorized staff can access sensitive data. To prevent unwanted access and data breaches, it is essential to impose security rights and encrypt user connections and user access using passwords (Berski & Bilau, 2019). Furthermore, MySQL's binary logging makes data replication and recovery easier by enabling the production of backups that, in the event of a failure, can restore the database state (Sholihah & Darujati, 2022).

MySQL offers a number of backup techniques, such as logical backups using dump files and physical backups (image copies). Although the image copy approach is thought to be simple and dependable, the significant overheads involved in large data transmissions may make it resource-intensive (Nakamura et al., 2013). However, when dealing with massive amounts of data, logical backups provide flexibility and enable the backing up of particular tables or databases, which might be crucial (Petrov et al., 2022). Logical backups are frequently carried out using tools like mysqldump, which guarantee the preservation of important data while causing the least amount of disturbance to the database system (Ramesh et al., 2023).

Password management and password security, such as secure storage and use of passwords to prevent unauthorized access to passwords, are also part of the security of both MySQL and other databases.

When managing accounts, it is important to lock unused accounts as well as manage account resources, which can resolve multiple login attempts using the wrong password. In order to further improve the security of the MySQL database, additional security measures can be introduced. This includes using a firewall to restrict access to the server where the database is located, regularly updating the database management system, and performing analysis and assessing the security of the databases in order to eliminate deficiencies.

DATABASE SECURITY

Security, or database security, is a component that aims to protect databases from intentional and unintentional threats. This includes ensuring the confidentiality, integrity, and availability of data stored in databases (Paul & Aithal, 2019). In addition, operating system security plays a key role in protecting the integrity and confidentiality of databases in Windows and Linux environments. Windows and Linux operating systems represent the basis on which databases work, so any vulnerability and breach of the operating system can directly affect the security of databases.

As the volume of data increases, so do the dangers that come with it. Critical information stored in databases faces serious challenges to its security, integrity, and availability due to malicious attacks, illegal access, and data manipulation.

Ensuring data privacy in databases is essential to prevent unauthorized access. In addition, query integrity is essential for maintaining database security, as any compromise in query integrity can lead to data manipulation.

Encryption also significantly affects database security. Although encryption can improve data security, it is also important to pay attention to database performance when implementing a data encryption strategy. Effective selecting the data that needs to be encrypted, sensitive data can be protected without too much impact on the efficiency of the database.

Database security concerns cover a wide range of issues that must be addressed to ensure comprehensive protection. Factors such as access control, authentication mechanisms, and data encryption contribute to strengthening overall data security (Gupta et al., 2012). In addition, it is necessary to solve the problems of making backup copies of data, as well as to pay attention to the protection of physical servers on which databases are located from network and other attacks.

Securing databases through network security involves an approach that addresses various critical aspects to ensure the protection of sensitive data. One of the main aspects is the use of stored procedures to improve database security (Ahmad & Karim, 2021). They play a key role in protecting databases from threats like SQL injection, thereby improving overall security. In addition, network security measures such as firewalls, access control, and web server request filtering play an important role in ensuring the security of databases. These basic network-level security measures are essential components of database security, contributing to overall database protection (Hang et al., 2024).

SECURITY IN MYSQL DATABASE

When using MySQL on a server, it is necessary to implement security measures to prevent common errors. Security measures should not only focus on protecting the MySQL server but also on protecting the entire server from various types of attacks.

MySQL includes security measures that use Access Control Lists (ACLs) for user operations and supports SSL encrypted connections between client and server. These security principles are not exclusive to MySQL and are relevant to most applications.

To ensure secure use of MySQL, it is crucial to restrict access to the user table in the system database to the MySQL root account (https://dev.mysql.com/doc/mysql-security-excerpt/8.0/en/security-guidelines.html). Understanding the access privilege system in MySQL and using commands such as GRANT and REVOKE to control access is very important. In addition, it is recommended to store the hashed password instead of plain text in the database in order to prevent the violation of data privacy in the event of an attack on the database. Regularly checking and adjusting user privileges using commands such as SHOW GRANT to check access, as well as REVOKE to remove access rights, is advised to maintain a secure system.

Furthermore, it is important to create strong passwords with a combination of upper and lowercase letters, numbers, and special characters to deter brute-force attempts and other attacks. Passwords should also be long, unpredictable, and easy-to-guess password components such as common words or phrases should be avoided, and the use of salt appended to an encrypted password is also recommended to provide additional security. Using a firewall as well as configuring MySQL so that it is not directly accessible to untrusted hosts are additional security measures.

Applications that communicate with MySQL should include appropriate techniques to ensure the security of the database, and this mostly applies to the transmission of unencrypted data over the Internet. It is also recommended to use encrypted protocols such as SSL and SSH for communication to improve data security.

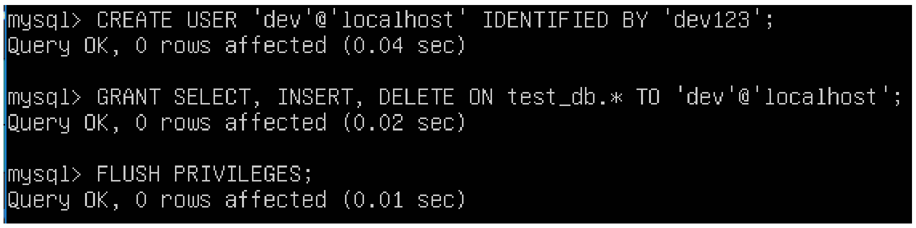
A tool commonly used for port scanning is nmap, and this tool is necessary to assess the accessibility of specific ports on a host. Nmap in particular is a widely recognized and popular port scanning tool. By scanning ports, it provides valuable information for assessing and configuring the network to be secure from attacks.

In addition, MySQL recommends quickly checking the availability of the port and telnet, with which it is possible to try to connect via the IP address of the host and the port, in this case 3306.

MANAGEMENT OF ACCESS RIGHTS AND ACCOUNTS IN MySQL

The MySQL access rights system plays a key role in managing user access to the server and data. By creating accounts, MySQL allows client users to connect to a server and perform various operations on the databases hosted on that server. The basic function of the MySQL access rights system is to authenticate users based on their host and associate them with certain privileges over databases, such as SELECT, INSERT, UPDATE, and DELETE.

To control access and determine which users can connect to the server, each account is assigned authentication information, usually in the form of a password. MySQL account management commands include SQL statements such as CREATE USER, GRANT, and REVOKE, which allow administrators to efficiently define and modify user privileges. The MySQL privilege system ensures that users can only perform operations for which they are explicitly granted permissions, maintaining the security and integrity of the data inside the database.



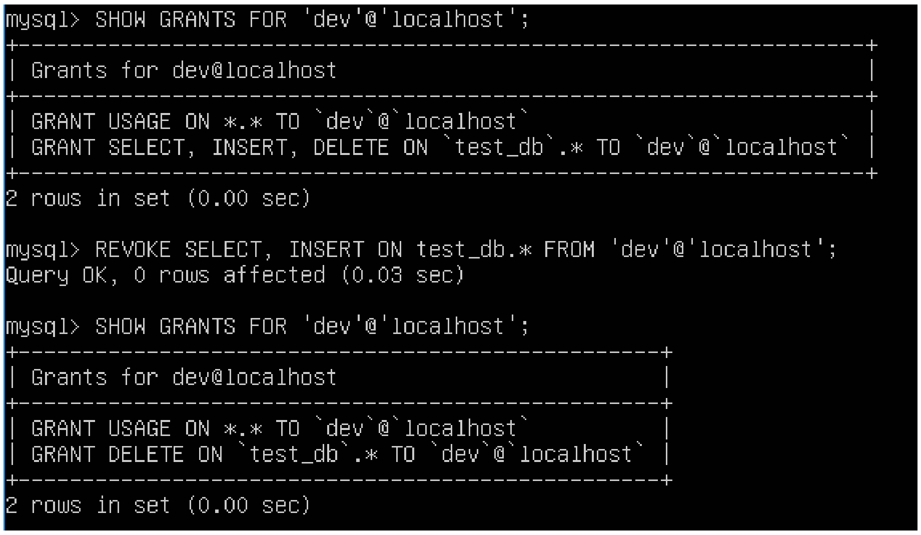
```
mysql> CREATE USER 'dev'@'localhost' IDENTIFIED BY 'dev123';
Query OK, 0 rows affected (0.04 sec)

mysql> GRANT SELECT, INSERT, DELETE ON test_db.* TO 'dev'@'localhost';
Query OK, 0 rows affected (0.02 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.01 sec)
```

Figure 1Creating users and assigning read, add and delete privileges in the test_db database over all tables

Figure 1 shows an example of creating a dev user who was then assigned read, add and delete rights in the test_db database over all tables. The command FLUSH PRIVILEGES, which is used at the end, serves to reload the tables into memory and is recommended if a direct change is made in the system tables using the UPDATE command.

When a user establishes a connection to a MySQL server, his identity is determined based on the host he is connecting from and the specified username and password. Subsequent requests made by the user are evaluated by permissions or privileges in accordance with their identity and requested actions. MySQL considers both hostname and username to uniquely identify users, recognizing that the same username can represent different users connecting from different hosts. This flexibility allows administrators to assign different privileges to users based on their connection, thereby providing additional control over access permissions.

To view the privileges associated with a particular account, the SHOW GRANTS command can be used; which is given below in Figure 2, in order to obtain this information from the permission tables that are stored in the mysql system database. These permission tables contain privilege information that the MySQL server reads into memory at startup, using it to make privilege control decisions during user interaction with the server. The image below also shows an example of REVOKE for revocation or removal of user rights.



```
mysql> SHOW GRANTS FOR 'dev'@'localhost';
+---------------------------------------------------------------------+
| Grants for dev@localhost                                            |
+---------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `dev`@`localhost`                             |
| GRANT SELECT, INSERT, DELETE ON `test_db`.* TO `dev`@`localhost`    |
+---------------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> REVOKE SELECT, INSERT ON test_db.* FROM 'dev'@'localhost';
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW GRANTS FOR 'dev'@'localhost';
+---------------------------------------------------+
| Grants for dev@localhost                          |
+---------------------------------------------------+
| GRANT USAGE ON *.* TO `dev`@`localhost`           |
| GRANT DELETE ON `test_db`.* TO `dev`@`localhost`  |
+---------------------------------------------------+
2 rows in set (0.00 sec)
```

Figure 2 Command to show user privileges and command to revoke privileges

MySQL privileges also involve two key stages when a user connects to a server. In the first phase, identity and credential verification:

• The server checks the user's credentials, such as password and name, to verify identity.

• Examines account lock status to determine if the account is locked or unlocked.

• If there is a problem in the previous two steps, the server refuses the connection, on the contrary, it moves to the second stage.

The second stage is user authentication and access control:

• The server verifies the user's identity and credentials using information stored in the user system table.

• Conditions for accepting a connection include matching the hostname (or IP address) and the username, with the corresponding columns in the user table.

• Account lock status is indicated by the account_locked column, where N represents an unlocked account.

• A user's identity is based on their MySQL username and the hostname they are connecting from.

• The server encrypts and compares passwords during the authentication process.

• Matching in the user table is done by sorting based on host values and usernames.

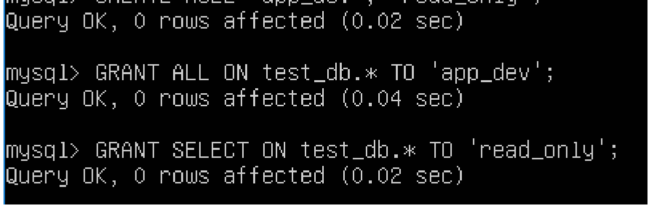
• As a rule, priority is given to specific host IP addresses and host names with '%' representing any host.

Although the MySQL system provides strong control over user privileges, there are certain limitations to be aware of. The administrator cannot explicitly deny access to a specific user or specify popout permissions for manipulation of tables without the right to create databases.

In addition to privileges for working with databases, administrators can grant other rights, such as server administration rights: CREATE_ROLE, DROP_ROLE, CREATE_USER, DROP_USER, etc. In addition to these, there are also privileges for creating views, indexes, etc. All of these privileges are granted using the GRANT command and revoked using the REVOKE command.

MYSQL ROLES

Roles in MySQL also represent a form of privilege management within the database system. A MySQL role is essentially a named collection of privileges that can be granted or revoked, similar to user accounts. By assigning roles to user accounts, the privileges associated with each role can be inherited. This simplifies the privilege management process and provides a more efficient way to manage access control. Managing roles in MySQL includes several key commands: CREATE ROLE and DROP ROLE are used to create and remove roles within the database system.

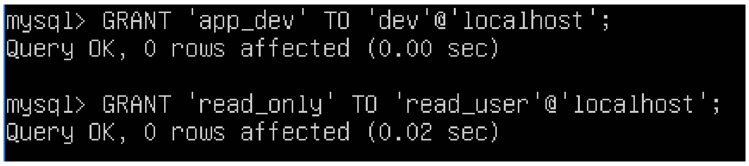


```
mysql> CREATE ROLE 'app_dev', 'read_only';
Query OK, 0 rows affected (0.02 sec)

mysql> GRANT ALL ON test_db.* TO 'app_dev';
Query OK, 0 rows affected (0.04 sec)

mysql> GRANT SELECT ON test_db.* TO 'read_only';
Query OK, 0 rows affected (0.02 sec)
```

Figure 3 Example of creating app_dev and read_only roles and assigning privileges to those roles

In the example in Figure 3, two roles were created, as well as assigning all privileges to the app_dev role in the test_db database over all tables and assigning the SELECT statement, read_only to the test_db database over all tables.

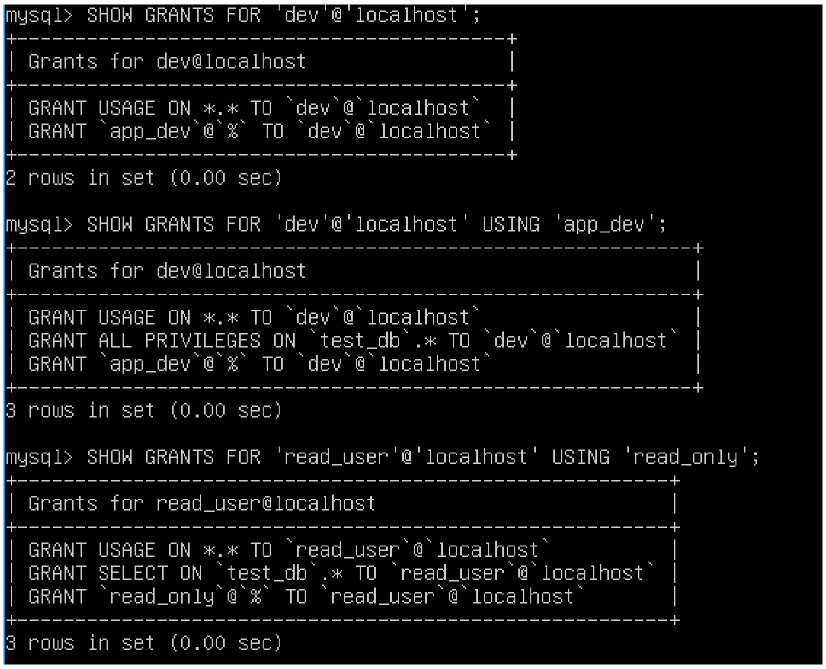
GRANT and REVOKE are key for granting privileges to both users and roles and for revoking privileges. The image below shows the assignment of roles to users. That is, the app_dev role is assigned to the dev user, while the read_only role is assigned to the read_user.



```
mysql> GRANT 'app_dev' TO 'dev'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT 'read_only' TO 'read_user'@'localhost';
Query OK, 0 rows affected (0.02 sec)
```

Figure 4 Assigning roles to users

SHOW GRANTS is used to display granted privileges and roles to both users and roles. In Figure 5, it can be seen that by assigning a role, the user inherited the privileges of that role, which is the goal of roles.

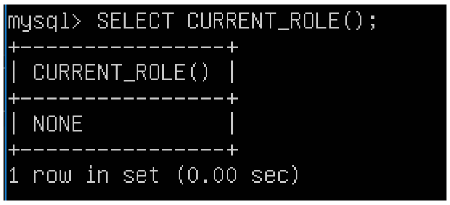


```
mysql> SHOW GRANTS FOR 'dev'@'localhost';
+-----------------------------------------+
| Grants for dev@localhost                |
+-----------------------------------------+
| GRANT USAGE ON *.* TO `dev`@`localhost`  |
| GRANT `app_dev`@`%` TO `dev`@`localhost` |
+-----------------------------------------+
2 rows in set (0.00 sec)

mysql> SHOW GRANTS FOR 'dev'@'localhost' USING 'app_dev';
+-----------------------------------------------------------+
| Grants for dev@localhost                                  |
+-----------------------------------------------------------+
| GRANT USAGE ON *.* TO `dev`@`localhost`                   |
| GRANT ALL PRIVILEGES ON `test_db`.* TO `dev`@`localhost`  |
| GRANT `app_dev`@`%` TO `dev`@`localhost`                  |
+-----------------------------------------------------------+
3 rows in set (0.00 sec)

mysql> SHOW GRANTS FOR 'read_user'@'localhost' USING 'read_only';
+--------------------------------------------------------+
| Grants for read_user@localhost                         |
+--------------------------------------------------------+
| GRANT USAGE ON *.* TO `read_user`@`localhost`          |
| GRANT SELECT ON `test_db`.* TO `read_user`@`localhost` |
| GRANT `read_only`@`%` TO `read_user`@`localhost`       |
+--------------------------------------------------------+
3 rows in set (0.00 sec)
```

Figure 5 Display of rights assigned to users

When creating roles and assigning privileges in MySQL, it is important to consider the specific requirements of the application. To avoid manually assigning privileges to individual users, roles can be created and used to assign privileges and thereby simplify the privilege management process.

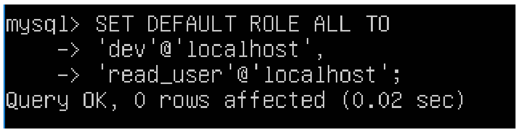
After creating and assigning roles, it is necessary to activate the roles because they can be active or inactive during the session. If the created and assigned roles are active in the session then the privileges are applied, otherwise they are not valid for the users. To check if the roles are active, use the CURRENT_ROLE() function. Below in the picture as read_user with the function checks if the roles are active.



```
mysql> SELECT CURRENT_ROLE();
+----------------+
| CURRENT_ROLE() |
+----------------+
| NONE           |
+----------------+
1 row in set (0.00 sec)
```

Figure 6 View role status for user read_user

Given that the roles are not active, it is necessary to activate them using the command SET DEFAULT ROLE as in the picture below.

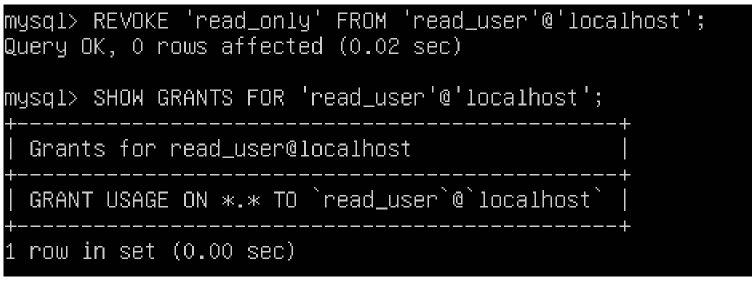


```
mysql> SET DEFAULT ROLE ALL TO
    -> 'dev'@'localhost',
    -> 'read_user'@'localhost';
Query OK, 0 rows affected (0.02 sec)
```

Figure 7 Activating roles for users

When read_user logs in again and uses the CURRENT_ROLE() function, he will see that the role is active and will be able to access the database, i.e. read data from the database.

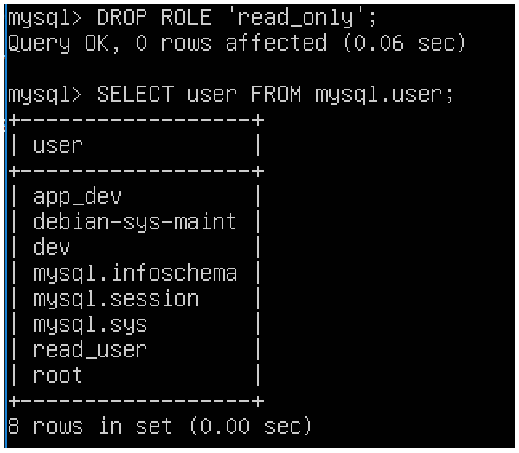
Removing a role from a user can be done using the REVOKE command and after that a check can be made to see that the role has been removed from read_user.



```
mysql> REVOKE 'read_only' FROM 'read_user'@'localhost';
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW GRANTS FOR 'read_user'@'localhost';
+-------------------------------------------+
| Grants for read_user@localhost            |
+-------------------------------------------+
| GRANT USAGE ON *.* TO `read_user`@`localhost` |
+-------------------------------------------+
1 row in set (0.00 sec)
```

Figure 8 Removing the read_user role

In addition to removing roles from users, REVOKE can also be used to remove privileges from roles. Later we will talk about how roles differ from users, since it can be seen that the commands are the same as for granting and removing privileges on users. Deleting a role is done with the DROP command as shown in the image below.



```
mysql> DROP ROLE 'read_only';
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT user FROM mysql.user;
+------------------+
| user             |
+------------------+
| app_dev          |
| debian-sys-maint |
| dev              |
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| read_user        |
| root             |
+------------------+
8 rows in set (0.00 sec)
```

Figure 9 Example of deleting a role

Roles differ from user accounts in that they are locked, have no password, and are assigned a default authentication plugin when created. These features can later be modified using the ALTER USER statement by a user with the necessary privileges.

Given that roles and users are similar in MySQL, it is possible to assign a user to another user, whereby he inherits the roles of the first user, the same is possible with roles, that is, it is possible to assign a user to a role.

PASSWORD MANAGEMENT IN MYSQL

MySQL provides a set of password management features to improve security and control over user accounts within the database system. These capabilities include prompting the user to enter a new password, password reuse restrictions, password verification, password strength assessment, random password generation, and temporary account locking.

Password expiration functionality in MySQL allows administrators to force users to periodically change their passwords for the accounts they use. This can be done using the ALTER USER command or automatically, globally using the default_password_lifetime system variable. It is necessary to determine the age of the password or the number of days that the password is valid, whereby the system marks passwords as expired when that number of days expires.

MySQL also supports password reuse restrictions to prevent users from choosing old passwords. This can be based on the number of password changes or the time since the password was last used. It is also possible to set a global variable that allows the user to be asked for the old password when changing the password, this mechanism provides protection against unauthorized password changes of a user.

In addition, MySQL has the ability to temporarily lock out an account after too many consecutive failed login attempts. This feature helps prevent attacks such as brute force attacks by blocking accounts for a certain period of time. The number of failed attempts is also configurable.

A clause to limit login attempts is also possible when using the user creation command, as well as all other commands listed in this chapter. Tracking failed login attempts and temporarily locking out accounts have specific features in MySQL. Both options must be non-zero to be enabled, successful logins reset the number of failed attempts.

LOCKING AND LIMITING ACCOUNT RESOURCES

In MySQL, the functionality of locking and unlocking user accounts is facilitated by using the ACCOUNT LOCK and ACCOUNT UNLOCK clauses within the CREATE USER and ALTER USER statements. These clauses serve different purposes when used when creating new accounts or modifying existing ones.

When used in conjunction with the CREATE USER statement, the ACCOUNT LOCK and ACCOUNT UNLOCK clauses define the initial lock status of a new user account. In situations where these clauses are not explicitly stated, the account is generated by default in an unlocked state. It is important to note that if the validate_password component is active, creating an account without a password is prohibited, regardless of whether the account is locked or not.

When used in the ALTER USER statement, these clauses play a key role in changing the lock status of an already existing account. In cases where these clauses are not specified, the current lock state of the account remains unchanged. It is important to note that using ALTER USER... UNLOCK has the ability to unlock any account that has been temporarily locked due to exceeding the number of failed login attempts.

The locked account status is stored in the account_locked column within the mysql.user system table. This information can be obtained using the SHOW CREATE USER command, which provides insight into whether the account is currently locked or unlocked.

In situations where a user tries to connect to a locked account, the connection attempt is unsuccessful. The server records this event by incrementing the locked_connects status variable, which indicates the number of failed attempts to connect to locked accounts. An error is displayed to the user and a corresponding message is printed.

It is important to note that locking an account does not prevent the use or execution of stored procedures and views associated with the locked account.

A means of limiting MySQL server resource usage can be achieved by setting the max_user_connection global variable to a non-zero value. This setting limits the number of simultaneous connections any account can make, but does not impose any restrictions on the actions a user can perform after connecting. Also, this method does not allow individual account management.

To provide fine-grained control over the use of server resources by an account, MySQL offers the ability to set limits on various aspects of resource usage for each account. These restrictions include:

• The number of queries that the account can execute in one hour.

• The number of updates an account can make in one hour.

• The frequency with which the account can connect to the server during one hour.

• Maximum number of simultaneous connections allowed for one account.

It is important to note that any command issued by the user counts as a query constraint, while only commands that involve modifications to the database or tables count as update constraints. Restrictions always apply to an account regardless of the host it connects from.

To set resource limits on an account when it is created, the CREATE USER command is used. To modify existing account limits, use the ALTER USER command, which includes a WITH clause that specifies resource limits. The default value for each limit is zero, which means no limit.

The WITH clause in the ALTER USER statement can include any combination of constraints, where each constraint represents a numerical value per hour. For the MAX_USER_CONNECTIONS constraint, setting it to zero refers to the global value of the max_user_connections system variable. If both limits are zero, there is no limit on the account.

The server tracks resource usage on a per account basis, not per client. If an account reaches its limit of connections, queries, or updates within an hour, the server refuses further operations until one hour has passed, displaying the appropriate errors.

Resource usage counters can be reset globally for all accounts using the FLUSH USER_RESOURCES command or individually for a specific account by redefining its limits. These resets do not affect the MAX_USER_CONNECTIONS limit and all counters are reset to zero on server startup without passing values.

ADDITIONAL GUIDELINES FOR DATABASE SECURITY

Protecting the MySQL database from the point of view of network security as well as application security is of great importance in order to prevent unauthorized access to the database. Protection against SQL injection attacks is crucial in order to preserve data integrity. In order to reduce the risk of SQL injection attacks, it is important to validate the values entered by the user before the input is applied to the query. This includes ensuring that input matches the expected data type, length, and format, so that input containing unexpected characters or patterns needs to be rejected.

Granting minimum privileges to database users and applications is critical, because using accounts with too many privileges for tasks that can be performed with some minimum privilege increases the impact of a successful SQL injection attack.

Also, keeping MySQL and plugins up-to-date is of great importance, because some flaws from previous versions can be used for unauthorized access to the database.

In addition, enforcing a limit on the length of input fields helps prevent buffer overflow attacks, and also implementing appropriate error handling mechanisms is crucial to avoid revealing sensitive information in error messages.

When talking about access to the server via the network, in order to protect the database, it is possible to allow access only from the local host, using the configuration file, where 127.0.0.1 is entered for the bind-address, which is usually entered as the default value. In addition, it is possible to use a firewall to filter incoming connections.

MySQL backup is essential to ensure data integrity, availability and data recovery in case of incidents such as data loss, data corruption or attack and unauthorized access to the database. Backing up your MySQL database helps protect against the above problems that may occur.

Despite MySQL's security features, vulnerabilities may still exist, making the database vulnerable to attack. Backups are therefore essential to mitigate the impact of such security breaches.

Backup strategies, such as data synchronization with centralized repositories or replication and external storage devices, play an important role in ensuring data security.

The mysqldump command offers a wide range of options that can be combined to achieve the desired results, for example, it is possible to use the option to connect to a remote database and make a backup copy, it is also possible to use the mysql_config_editor file to log in when copying data, and so on.

CONCLUSION

The implementation of security measures and practices ensures the protection of data from unauthorized access and manipulation, but also provides the basis for the reliable operation of the MySQL database. Constant improvement of security practices and adaptation to changes and standards is the basis for maintaining data security. In addition, it is very important to pay attention to user access control, so as not to misuse any of the user accounts, as well as to familiarize database users with ways to properly handle passwords. Finally, organizations must be aware of the ongoing challenges posed by evolving cybersecurity threats. As SQL injection attacks continue to be a prevalent issue, it is crucial for MySQL users to employ best practices that safeguard against such vulnerabilities, including the use of parameterized queries and prepared statements in application development. The continuous evolution of security measures within MySQL is vital for organizations to remain resilient against both external and internal threats.

References

Ahmad, K., & Karim, M. (2021). A method to prevent SQL injection attack using an improved parameterized stored procedure. International Journal of Advanced Computer Science and Applications, 12(6).

Berski, S. and Bilau, M. (2019). Safety mechanisms in relational database as part of the it system of the enterprise. New Trends in Production Engineering, 2(2), 12-23. https://doi.org/10.2478/ntpe-2019-0068

Gupta, A., Bibhu, V., & Hussain, R. (2012). Security measures in data mining. International Journal of Information Engineering and Electronic Business, 4(3), 34.

Hang, F., Xie, L., Zhang, Z., Guo, W., & Li, H. (2024). Research on the application of network security defence in database security services based on deep learning integrated with big data analytics. International Journal of Intelligent Networks, 5, 101-109.

https://dev.mysql.com/doc/mysql-security-excerpt/8.0/en/security-guidelines.html

Nakamura, S., Zhao, X., & Nakagawa, T. (2013). Stochastic modeling of database backup policy for a computer system. Journal of Software Engineering and Applications, 06(02), 53-58. https://doi.org/10.4236/jsea.2013.62009

Paul, P., & Aithal, P. S. (2019). Database security: An overview and analysis of current trend. International Journal of Management, Technology, and Social Sciences (IJMTS), 4(2), 53-58.

Petrov, P., Kuyumdzhiev, I., Dimitrov, G., & Kremenska, A. (2022). Relative performance of various types of repositories for mysql archive backup and restore operations. International Journal of Online and Biomedical Engineering (Ijoe), 18(13), 152-159. https://doi.org/10.3991/ijoe.v18i13.33429

Ramesh, G., Logeshwaran, J., & Aravindarajan, V. (2023). A secured database monitoring method to improve data backup and recovery operations in cloud computing. Bohr International Journal of Computer Science, 2(1), 1-7. https://doi.org/10.54646/bijcs.019

Sholihah, I. and Darujati, C. (2022). Sistem replikasi basis data berdasarkan mysql menggunakan container docker. Majalah Ilmiah Teknologi Elektro, 21(2), 209. https://doi.org/10.24843/mite.2022.v21i02.p08

Notes on the authors:

Ivan ŠUŠTER, B.Sc., is a M.Sc. student at the Faculty of electronic engineering, University of Niš. E-mail: ivansu995@gmail.com

Darjan KARABAŠEVIĆ, Ph.D. is a Full Professor and Dean of the Faculty of Applied Management, Economics and Finance, University Business Academy in Novi Sad. E-mail: darjan.karabasevic@mef.edu.rs

Dragiša STANUJKIĆ, Ph.D. is a Full Professor at the Technical Faculty in Bor, University of Belgrade. E-mail: dstanujkic@tfbor.bg.ac.rs

Gabrijela POPOVIĆ, Ph.D. is a Full Professor and a Vice-dean for scientific research of the Faculty of Applied Management, Economics and Finance, University Business Academy in Novi Sad. E-mail: gabrijela.popovic@mef.edu.rs

Ana VELJIĆ, M.Sc., is a Ph.D. candidate at the Faculty of Technical Sciences Čačak, University of Kragujevac and Teaching Assistant at the Faculty of Applied Management, Economics and Finance, University Business Academy in Novi Sad. E-mail: ana.veljic@mef.edu.rs

Marko MARKOVIĆ, M.Sc., is a Ph.D. candidate at the Faculty of Economics and Engineering Management, University Business Academy in Novi Sad and a Teaching Assistant at the Faculty of Applied Management, Economics and Finance, University Business Academy in Novi Sad. E-mail: marko.markovic@mef.edu.rs