

Desenvolvimento de sistemas

Segurança do banco de dados: usuários e permissões de acesso (GRANT e REVOKE, *roles*), encriptação de dados, SQL *injection*; integridade de dados

Introdução

O conceito de segurança de computadores está diretamente relacionado ao de segurança da informação, incluindo não apenas a proteção de dados e informações, mas também a proteção de sistemas e bancos de dados.

São características básicas da segurança da informação os atributos de **confidencialidade**, **integridade** e **disponibilidade**. Atualmente, o conceito de segurança da informação está padronizado pela norma ABNT ISO/IEC 17799:2005.

Por definição, a segurança da informação se refere à proteção de informações de uma determinada pessoa ou empresa. Em outras palavras, o conceito se aplica a informações tanto pessoais quanto corporativas. Entende-se por **informação** todo e qualquer dado que tenha valor para uma pessoa ou organização.

A tríade **CIA** (*confidentiality, integrity and availability*) representa os principais atributos que orientam a análise, o planejamento e a implementação da segurança de um determinado grupo de informações o qual se deseja proteger. Apesar de conter a mesma sigla, não se está falando sobre a Agência Central de Inteligência (Central Intelligence Agency). Em segurança cibernética, CIA refere-se à tríade que concentra o equilíbrio entre a **confidencialidade** (*confidentiality*), **integridade** (*integrity*) e **disponibilidade** (*availability*) de dados sob a proteção de informações.

Clique ou toque para visualizar o conteúdo.

Disponibilidade

Integridade

Confidencialidade



Figura 1 – Tríade CIA Fonte: <<https://www.ibm.com/blogs/cloud-computing/wp-content/uploads/2018/01/TRIAD.png>>. 25 mar. 2022.

Confidencialidade

Em sua essência, o princípio da confidencialidade é o de manter privado o que precisa ser privado. Na prática, a confidencialidade consiste em controlar o acesso aos dados para que apenas usuários autorizados possam acessá-los ou modificá-los.

Integridade

A integridade se concentra em manter os dados limpos e imaculados, não somente quando são carregados, mas também quando são armazenados. Isso significa garantir que apenas aqueles que têm permissão para modificar esses dados possam fazê-lo.

Disponibilidade

Significa essencialmente que, quando um usuário autorizado precisa acessar uma ou mais informações, ele pode fazê-lo. Às vezes, o conceito de disponibilidade pode ser confundido ou até mesmo parecer contradizer o conceito de confidencialidade. Enquanto a confidencialidade é garantir que apenas as pessoas que precisam acessar os dados possam obtê-los, a disponibilidade é garantir que seja fácil acessar esses dados caso uma pessoa autorizada precise. Isso pode incluir garantir que as redes e os aplicativos estejam funcionando como deveriam, que os protocolos de segurança não estão prejudicando a produtividade ou que um recurso está disponível quando surge um problema e precisa ser corrigido.

Confidencialidade

Em sua essência, o princípio da confidencialidade é o de manter privado o que precisa ser privado. Na prática, a confidencialidade consiste em controlar o acesso aos dados para que apenas usuários autorizados possam acessá-los ou modificá-

los.

Integridade

A integridade se concentra em manter os dados limpos e imaculados, não somente quando são carregados, mas também quando são armazenados. Isso significa garantir que apenas aqueles que têm permissão para modificar esses dados possam fazê-lo.

Disponibilidade

Significa essencialmente que, quando um usuário autorizado precisa acessar uma ou mais informações, ele pode fazê-lo. Às vezes, o conceito de disponibilidade pode ser confundido ou até mesmo parecer contradizer o conceito de confidencialidade. Enquanto a confidencialidade é garantir que apenas as pessoas que precisam acessar os dados possam obtê-los, a disponibilidade é garantir que seja fácil acessar esses dados caso uma pessoa autorizada precise. Isso pode incluir garantir que as redes e os aplicativos estejam funcionando como deveriam, que os protocolos de segurança não estão prejudicando a produtividade ou que um recurso está disponível quando surge um problema e precisa ser corrigido.

Quando se planeja aplicar esses conceitos na segurança em um banco de dados, é preciso considerar uma ampla gama de tópicos possíveis e como eles afetam a segurança de um servidor MySQL. Realmente, não é uma tarefa fácil. Neste conteúdo, você aprenderá como tornar um ambiente de banco de dados mais seguro e conhecerá os principais problemas de segurança que podem ser enfrentados.

Usuários e permissões de acesso

Assim que você finalizar a instalação do MySQL, receberá um usuário padrão com acesso total às bases de dados e às tabelas do banco de dados. Esse será o usuário **root**. Com esse usuário, você tem acesso total a todos os recursos do MySQL.

Em algum momento, o administrador de banco de dados pode precisar conceder acesso ao banco de dados à outra pessoa. Um exemplo dessa situação é quando uma empresa contrata um programador para desenvolver um sistema que consome os dados do banco de dados. Como o acesso ao banco de dados é necessário para o desenvolvimento do sistema, cabe ao administrador de banco de dados passar as credenciais de acesso para a equipe de desenvolvimento. Porém, fornecer as credenciais do usuário **root** pode comprometer todos os registros e as configurações do banco de dados da empresa, caso, por exemplo, haja a modificação ou exclusão acidental ou intencional de registros. Para evitar esse problema, são criadas contas de usuário no MySQL com permissões específicas para manter o controle do que os desenvolvedores podem ou não fazer com os dados e evitar o comprometimento de qualquer informação do banco de dados.

Aprenda agora passo a passo como criar uma conta de usuário no MySQL e conceder os privilégios para esse usuário. Para começar, abra o MySQL Workbench e conecte-se ao MySQL Community.

Criação de usuários

Para criar um novo usuário, utiliza o seguinte comando:

```
CREATE USER 'usuario'@'endereço' IDENTIFIED BY 'senha';
```

Essa é a sintaxe do comando de criação de usuários, cujos elementos são:

Clique ou toque para visualizar o conteúdo.

Usuário

É o nome da nova conta de usuário que está sendo criada.

Endereço

É o endereço do banco de dados. Se você estivesse fazendo a criação do usuário remotamente, seria necessário informar o IP do servidor do banco de dados.

Senha

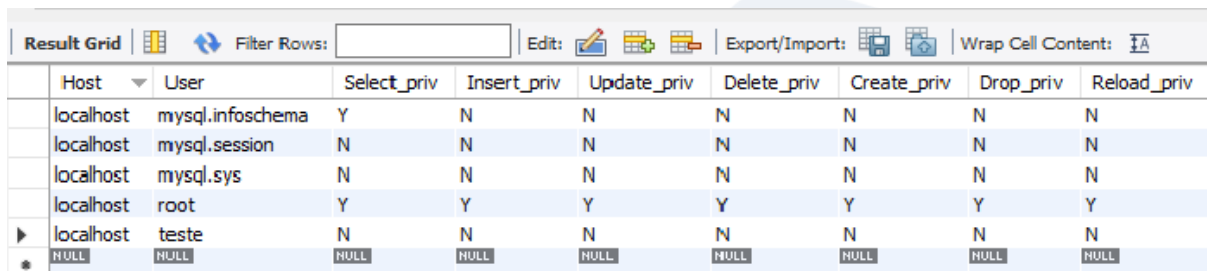
Aqui, você define a senha do usuário. Apesar de ser possível deixar a senha em branco, essa prática não é recomendada, pois pode comprometer a segurança do banco de dados.

Para exemplificar a criação de usuários, crie o usuário **teste** com a senha **q1w2e3r4**. Como o seu banco de dados está instalado e sendo executado no seu computador local, o endereço que informado será o **localhost**. Logo, seu comando ficará com a seguinte sintaxe:

```
CREATE USER 'teste'@'localhost' IDENTIFIED BY 'q1w2e3r4';
```

Após a execução desse comando, o usuário será adicionado à tabela de usuários interna do banco de dados MySQL. Para exibir os dados dessa tabela, execute este comando:

```
SELECT * FROM mysql.user;
```



The screenshot shows the MySQL Workbench interface with the 'Result Grid' tab selected. The grid displays the results of the SQL query 'SELECT * FROM mysql.user;'. The table has 10 columns: Host, User, Select_priv, Insert_priv, Update_priv, Delete_priv, Create_priv, Drop_priv, and Reload_priv. There are 6 rows of data, including a row with NULL values at the bottom.

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv
localhost	mysql.infoschema	Y	N	N	N	N	N	N
localhost	mysql.session	N	N	N	N	N	N	N
localhost	mysql.sys	N	N	N	N	N	N	N
localhost	root	Y	Y	Y	Y	Y	Y	Y
localhost	teste	N	N	N	N	N	N	N
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 2 – Resultado da consulta SQL na tabela “mysql.user”

Fonte: MySQL Workbench (2022)

A instrução **SELECT** recuperará todos os dados dos usuários cadastrados no banco de dados MySQL. A tabela “user” contém diversas colunas, mas nem todas são relevantes neste momento. Não é necessário um aprofundamento sobre o papel de cada coluna, pois seu foco agora deve ser a criação e configuração das contas de usuário. Porém, é importante você saber quais informações pode ou não extrair aqui. Então, para facilitar o entendimento da estrutura dessa tabela, ela será dividida em quatro grupos diferentes:

Colunas de escopo Definem o usuário e o servidor no qual ele é válido.	Host User
--	--------------

Senac

<p>Colunas de privilégios</p> <p>Define permissão de acesso ou execução de operações no banco de dados, desde ações simples (como <i>select</i>, <i>insert</i>, <i>update</i>) até ações mais administrativas (<i>shutdown</i> [desligar], <i>file</i> [arquivos], <i>grant</i> [dar permissões]). Note que, como padrão, utiliza-se o nome da ação em inglês, seguido de _priv.</p>	<p>Select_priv,</p> <p>Insert_priv,</p> <p>Update_priv,</p> <p>Delete_priv,</p> <p>Create_priv,</p> <p>Drop_priv,</p> <p>Reload_priv,</p> <p>Shutdown_priv,</p> <p>Process_priv,</p> <p>File_priv,</p> <p>Grant_priv,</p> <p>References_priv,</p> <p>Index_priv,</p> <p>Alter_priv,</p> <p>Show_db_priv,</p> <p>Super_priv,</p> <p>Create_tmp_table_priv,</p> <p>Lock_tables_priv,</p> <p>Execute_priv,</p> <p>Repl_slave_priv,</p> <p>Repl_client_priv,</p> <p>Create_view_priv,</p> <p>Show_view_priv,</p> <p>Create_routine_priv,</p> <p>Alter_routine_priv,</p> <p>Create_user_priv,</p> <p>Event_priv,</p>
--	---

	Trigger_priv, Create_tablespace_priv, Create_role_priv, Drop_role_priv
Colunas de segurança Contém informações sobre senha e criptografia associadas ao usuário.	ssl_type, ssl_cipher, x509_issuer, x509_subject, plugin, authentication_string, password_expired, password_last_changed, password_lifetime, account_locked, Password_reuse_history, Password_reuse_time, Password_require_current, User_attributes
Colunas de controle de recursos Contém limitações impostas ao usuário, como o número de consultas ou de conexões que ele poderá emitir por hora.	max_questions, max_updates, max_connections, max_user_connections

Como foi executado o comando **SELECT * FROM mysql.user**, serão exibidas todas as colunas no resultado da pesquisa e, como se pode ver, a maioria delas se refere a permissões do usuário e outras configurações. Nesse momento, as colunas relevantes são as de escopo: “Host” e “User”.

Colunas de escopo		
Nome da coluna	Descrição	Tipo
Host	Endereço do servidor da conta do usuário	char(60)
User	Nome da nova conta de usuário	char(60)

Logo, quando observados os registros dessas duas colunas, tem-se os seguintes resultados na pesquisa:

Host	User
localhost	mysql.infoschema
localhost	mysql.session
localhost	mysql.sys
localhost	root
localhost	teste

Como se pode ver, o usuário de teste foi criado com sucesso exatamente como especificado no comando **CREATE USER**. É importante lembrar que, ao se conectar ao MySQL com o MySQL Workbench, você se autenticou com o usuário **root** e é com essa autenticação que você está operando o banco de dados neste momento.

Para utilizar o novo usuário criado, você precisará estabelecer uma nova conexão com o banco de dados usando as novas credenciais (pode ser necessário, na tela inicial do Workbench, criar uma nova configuração de conexão informando no campo **Username** o *login* do novo usuário).

Apesar de utilizar apenas os usuários *root* e teste, o MySQL contém outros usuários internos (“mysql.infoschema”, “mysql.session” e “mysql.sys”), que são responsáveis pelo funcionamento de determinados recursos no banco de dados. A exclusão ou modificação desses usuários pode resultar no comprometimento do funcionamento do MySQL como um todo. Portanto, evite qualquer alteração nas configurações desses usuários.

Que tal realizar um desafio?

Agora que você já sabe como criar novos usuários no MySQL, crie um usuário com o seu nome, utilizando o comando **CREATE USER**.

Exclusão de usuários

Apesar de não ser recomendado, é possível fazer a exclusão de usuários criados no MySQL. Para isso, utilize o seguinte comando:

```
DROP USER 'usuario'@'endereco';
```

Observe que a sintaxe é bem semelhante ao comando **CREATE USER**. A diferença está na primeira palavra reservada (**DROP** em vez de **CREATE**) e também por não ser necessário informar a senha do usuário.

Como exemplo de uso desse comando, crie um usuário “teste2” com a senha em branco e, em seguida, remova esse usuário. Inicie criando o usuário, com o seguinte comando:

```
CREATE USER 'teste2'@'localhost' IDENTIFIED BY '';
SELECT host, user FROM mysql.user;
```

Ao executar a instrução **SELECT**, você terá o seguinte resultado:

Host	User
localhost	mysql.infoschema
localhost	mysql.session
localhost	mysql.sys
localhost	root
localhost	teste
localhost	teste2

O usuário foi criado com sucesso. Agora, remova-o com o comando a seguir:

```
DROP USER 'teste2'@'localhost';
SELECT host, user FROM mysql.user;
```

No final, você terá o seguinte resultado:

Host	User
localhost	mysql.infoschema
localhost	mysql.session
localhost	mysql.sys
localhost	root
localhost	teste

Como esperado, o usuário foi removido com sucesso.

Que tal realizar alguns desafios?

Agora que você já sabe como remover os usuários no MySQL, crie dois novos usuários e depois remova-os, utilizando apenas uma instrução **DROP USER**.

Como mencionado anteriormente, a exclusão de usuários não é uma prática recomendada, pois ela desvinculará esse usuário de algum registro de *logs* importante, caso haja. Imagine que, por exemplo, em uma tabela de banco de dados, foi registrado o nome do usuário que fez o cadastro de cada item de um estoque. Se esse usuário for excluído, não haverá mais registros sobre quem fez aquele cadastro. Considerando esse cenário, a melhor solução seria alterar a senha desse usuário e remover todas as suas permissões de uso do banco de dados. Inicialmente, todo usuário criado no MySQL começa sem nenhuma permissão na sua configuração. Logo, se você tentasse operar o MySQL com as credenciais do novo usuário, não conseguiria fazê-lo dentro do ambiente.

Permissões de usuários

Além da criação e da remoção de contas, o gerenciamento de usuários do banco de dados é um processo que define as permissões que cada usuário terá para manipular tabelas, base de dados etc. Definir uma permissão significa determinar se um usuário pode ou não executar uma ação específica.

Na tabela “mysql.user”, estão as seguintes colunas de privilégios:

Nome da coluna	Privilégio	Descrição
Select_priv	SELECT	O usuário pode executar instruções SELECT .
Insert_priv	INSERT	O usuário pode executar instruções INSERT .
Update_priv	UPDATE	O usuário pode executar instruções UPDATE .
Delete_priv	DELETE	O usuário pode executar instruções DELETE .
Create_priv	CREATE	O usuário pode criar bancos de dados e tabelas.
Drop_priv	DROP	O usuário pode remover bancos de dados e tabelas.
Reload_priv	RELOAD	O usuário pode executar instruções FLUSH ou comandos equivalentes a mysqladmin .
Shutdown_priv	SHUTDOWN	O usuário pode desligar o servidor com SHUTDOWN ou mysqladmin shutdown .
Process_priv	PROCESS	O usuário pode mostrar informações sobre processos ativos, via SHOW PROCESSLIST ou mysqladmin processlist .

File_priv	FILE	O usuário pode ler e gravar arquivos no servidor, utilizando instruções como LOAD DATA INFILE ou funções como LOAD_FILE() . Também é necessário criar tabelas externas CONNECT . O servidor MariaDB deve ter permissão para acessar esses arquivos.
Grant_priv	GRANT	O usuário pode conceder os privilégios que tem.
References_priv	REFERENCES	O usuário pode relacionar tabelas com o uso de chave estrangeira.
Index_priv	INDEX	O usuário pode criar um índice em uma tabela usando a instrução CREATE INDEX . Sem o privilégio, o usuário ainda pode criar índices ao desenvolver uma tabela usando a instrução CREATE TABLE se tiver o privilégio CREATE , e criar índices usando a instrução ALTER TABLE se tiver o privilégio ALTER .
Alter_priv	ALTER	O usuário pode executar instruções ALTER TABLE .

Show_db_priv	SHOW DATABASES	O usuário pode listar todos os bancos de dados usando a instrução SHOW DATABASES . Sem este privilégio, ele ainda pode emitir a instrução SHOW DATABASES , mas apenas listará bancos de dados contendo tabelas nas quais ele tem privilégios.
Super_priv	SUPER	O usuário pode executar instruções de superusuário.
Create_tmp_table_priv	CREATE TEMPORARY TABLES	O usuário pode criar tabelas temporárias com a instrução CREATE TEMPORARY TABLE .
Lock_tables_priv	LOCK TABLES	O usuário pode adquirir bloqueios explícitos usando a instrução LOCK TABLES , mas também precisa ter o privilégio SELECT em uma tabela para bloqueá-la.
Execute_priv	EXECUTE	O usuário pode executar procedimento armazenado ou funções.

Repl_slave_priv	REPLICATION SLAVE	As contas usadas por servidores escravos no mestre precisam desse privilégio. Isso é necessário para obter as atualizações feitas no mestre.
Repl_client_priv	REPLICATION CLIENT	O usuário pode executar as instruções SHOW MASTER STATUS e SHOW SLAVE STATUS .
Create_view_priv	CREATE VIEW	O usuário pode criar uma exibição usando a instrução CREATE_VIEW .
Show_view_priv	SHOW VIEW	O usuário pode mostrar a instrução CREATE VIEW para criar uma exibição usando a instrução SHOW CREATE VIEW .
Create_routine_priv	CREATE ROUTINE	O usuário pode criar programas armazenados usando as instruções CREATE PROCEDURE e CREATE FUNCTION .
Alter_routine_priv	ALTER ROUTINE	O usuário pode alterar as características de uma função armazenada usando a instrução ALTER FUNCTION .

Create_user_priv	CREATE USER	O usuário pode criar um usuário usando a instrução CREATE USER ou criar implicitamente um usuário com a instrução GRANT .
Event_priv	EVENT	O usuário pode criar, descartar e alterar eventos.
Trigger_priv	TRIGGER	O usuário pode executar <i>triggers</i> .
Create_tablespace_priv	CREATE TABLESPACE	O usuário pode criar TABLESPACES .

Obviamente, não é necessário decorar todas essas permissões, basta que você saiba que elas existem e recorra a essa tabela ou a uma busca na *web* quando de fato necessitar. É possível, inclusive, que você nunca tenha que lidar com algumas dessas permissões.

Note que são muitas as permissões que podem ser concedidas ou revogadas a um usuário, e a definição dessas permissões dependerá do tipo de usuário com o qual você estará trabalhando. Se você estiver criando uma conta de um administrador, por exemplo, o ideal é que ela tenha a maioria das permissões concedidas. Já para um usuário comum, algumas permissões de manipulação de dados e de manipulação de tabela já seriam suficientes.

Para conceder privilégios a um usuário MySQL, utiliza-se o comando **GRANT** (que significa “conceder”, em português), apontando quais privilégios esse usuário terá. Caso você precise revogar algum privilégio, utilize o comando **REVOKE** (que significa “revogar”, em português).

Considerando a segurança do banco de dados, é comum administradores desses bancos revogarem privilégios do usuário *root* e criarem um novo usuário com todas as permissões.

GRANT

A instrução **GRANT** tem a seguinte sintaxe:

```
GRANT privileges ON nome_banco.nome_tabela TO usuario@endereco;
```

Para garantir todos os privilégios do banco de dados a um usuário, execute o seguinte comando:

```
GRANT ALL PRIVILEGES ON *.* TO 'teste'@'localhost';
```

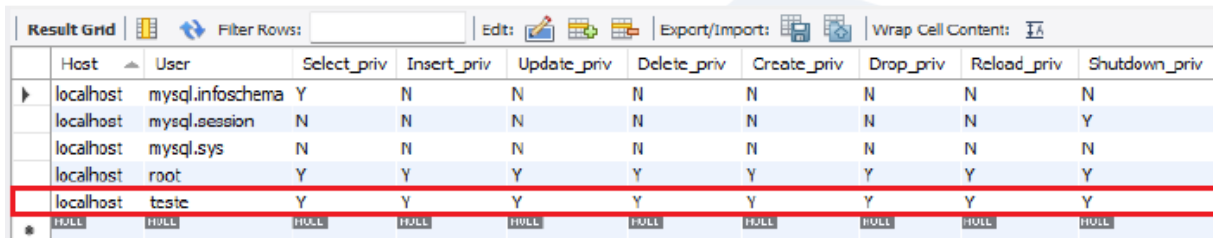
Perceba que foi utilizado o caractere *** no lugar em que deveriam estar informados o nome da base de dados e o nome da tabela. Com esse caractere, você estará indicando à instrução que deseja atribuir os privilégios para todas as bases de dados e todas as tabelas, o que inclui as bases e tabelas já criadas e as que ainda serão criadas. Dessa forma, o usuário “teste” poderá manipular qualquer base de dados ou tabela. Além disso, para não ser necessário especificar cada um dos privilégios manualmente, o MySQL contém as palavras reservadas **ALL PRIVILEGES**, que remetem a todos os privilégios de manipulação de dados e de tabelas.

Apesar de já estarem definidos os privilégios para o usuário no qual você está trabalhando agora, ainda será preciso executar o comando **FLUSH** para que as mudanças tenham efeito. Então, execute o seguinte comando:

```
FLUSH PRIVILEGES;
```

Então, se você executar o comando **SELECT * FROM mysql.user;** e observar as colunas de privilégios, verá que as permissões que anteriormente estavam com valor **N** (*no*, em inglês, traduzido para “não”, em português) agora estão com o valor

Y (yes, em inglês, que significa “sim”, em português).



Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdown_priv
localhost	mysql.infoschema	Y	N	N	N	N	N	N	N
localhost	mysql.session	N	N	N	N	N	N	N	Y
localhost	mysql.sys	N	N	N	N	N	N	N	N
localhost	root	Y	Y	Y	Y	Y	Y	Y	Y
localhost	teste	Y	Y	Y	Y	Y	Y	Y	Y
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 3 – Resultado da pesquisa na tabela “User” no MySQL Workbench

Fonte: MySQL Workbench (2022)

Outro modo de definir permissões para as contas de usuário é especificar cada uma das permissões que o seu usuário terá em cada tabela. Na maioria das vezes, as permissões especificadas são de manipulação de dados e de manipulação de tabelas, já que tendem a ser destinadas a usuários com acessos restritos.

As permissões de manipulação de dados são:

Clique ou toque para visualizar o conteúdo.

Permite que o usuário recupere dados.

Permite que o usuário adicione novas entradas em tabelas.

Permite que o usuário modifique entradas existentes em tabelas.

Permite que o usuário exclua registros de tabelas.

SELECT

Permite que o usuário recupere dados.

INSERT

Permite que o usuário adicione novas entradas em tabelas.

UPDATE

Permite que o usuário modifique entradas existentes em tabelas.

DELETE

Permite que o usuário exclua registros de tabelas.

As permissões de manipulação de tabelas são:

Clique ou toque para visualizar o conteúdo.

Permite que o usuário crie bases de dados ou tabelas.

Permite a modificação de tabelas e colunas.

Permite que o usuário exclua tabelas inteiras.

CREATE

Permite que o usuário crie bases de dados ou tabelas.

ALTER

Permite a modificação de tabelas e colunas.

DROP

Permite que o usuário exclua tabelas inteiras.

Agora que você já conhece a estrutura do comando **GRANT**, basta acompanhar o que já foi feito até aqui. A única mudança será especificar a permissão que deseja conceder no espaço **privilégios** da sintaxe (logo após **GRANT**). Caso haja mais de uma permissão, separe cada uma delas utilizando vírgula. Por exemplo, serão atribuídos os privilégios de **CREATE** para manipulação de tabelas e **SELECT**, **INSERT** e **UPDATE** para manipulação de dados em todas as tabelas de todas as bases de dados para um novo usuário chamado “teste3”.

Então, inicialmente, crie o usuário com o comando **CREATE USER**:

```
CREATE USER 'teste3'@'localhost' IDENTIFIED BY '';
```

Depois, defina as permissões específicas com o comando **GRANT**:

```
GRANT CREATE, SELECT, INSERT, UPDATE ON *.* TO 'teste3'@'localhost';
```

Agora, solicite ao servidor para recarregar as configurações de privilégios, pois você acabou de atualizar as permissões do usuário “teste3”. Utilize este comando:

FLUSH PRIVILEGES;

Por fim, confira na tabela “mysql.user” se está tudo correto:

```
SELECT * FROM mysql.user;
```

Crie uma nova conexão no MySQL Workbench associada ao usuário “teste3” e experimente fazer uma operação de **SELECT**, uma de **INSERT** e depois uma de **DELETE** sobre alguma tabela de algum banco de dados em sua máquina. As duas primeiras operações serão bem-sucedidas, mas a última gerará um erro semelhante ao seguinte:

Error Code: 1142. DELETE command denied to user 'teste3'@'localhost' for table 'nome_tabela'

A tradução desse erro seria “comando DELETE recusado para o usuário teste3 na tabela”.

Além da consulta à tabela “mysql.user”, outra forma de visualizar os privilégios atribuídos para um usuário é por meio do comando **SHOW GRANTS**. Se você executar o comando exatamente assim, ele buscará as informações do usuário que está executando esse comando. Se você estiver autenticado na sessão com usuário *root*, o resultado deve ser semelhante ao seguinte:

Grants for root@localhost

```
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP,
RELOAD, SHUTDOWN, PROCESS, FILE, REFERENCES, INDEX,
ALTER, SHOW DATABASES, SUPER, CREATE TEMPORARY
TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE,
REPLICATION CLIENT,
CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER
ROUTINE,
CREATE USER, EVENT, TRIGGER, CREATE TABLESPACE,
CREATE ROLE, DROP ROLE ON *.* TO `root`@`localhost` WITH
GRANT OPTION
```

```
GRANT APPLICATION_PASSWORD_ADMIN,
AUDIT_ABORT_EXEMPT, AUDIT_ADMIN,
AUTHENTICATION_POLICY_ADMIN, BACKUP_ADMIN,
BINLOG_ADMIN, BINLOG_ENCRYPTION_ADMIN,
CLONE_ADMIN, CONNECTION_ADMIN,
ENCRYPTION_KEY_ADMIN, FLUSH_OPTIMIZER_COSTS,
FLUSH_STATUS, FLUSH_TABLES, FLUSH_USER_RESOURCES,
GROUP_REPLICATION_ADMIN,
GROUP_REPLICATION_STREAM,
INNODB_REDO_LOG_ARCHIVE,
INNODB_REDO_LOG_ENABLE, PASSWORDLESS_USER_ADMIN,
PERSIST_RO_VARIABLES_ADMIN, REPLICATION_APPLIER,
REPLICATION_SLAVE_ADMIN,
RESOURCE_GROUP_ADMIN, RESOURCE_GROUP_USER,
ROLE_ADMIN, SERVICE_CONNECTION_ADMIN,
SESSION_VARIABLES_ADMIN, SET_USER_ID,
SHOW_ROUTINE,
SYSTEM_USER, SYSTEM_VARIABLES_ADMIN,
TABLE_ENCRYPTION_ADMIN, XA_RECOVER_ADMIN ON *.* TO
`root`@`localhost` WITH GRANT OPTION
```

```
GRANT PROXY ON ``@`` TO `root`@`localhost` WITH GRANT
OPTION
```

Agora, se você quiser especificar outro usuário, basta adicionar FOR 'usuario'@'endereço' na instrução. Por exemplo, recupere as informações de privilégio do usuário "teste3" com a seguinte instrução:

```
SHOW GRANTS FOR 'teste3'@'localhost';
```


Grants for teste3@localhost
GRANT SELECT, INSERT, UPDATE, CREATE ON *.* TO 'teste3'@'localhost'

Que tal realizar alguns desafios?

Agora que você já sabe como conceder privilégios aos usuários do MySQL, conceda todas as permissões para o usuário que você criou com o seu nome.

Após isso, cria uma nova base de dados com duas tabelas com, pelo menos, três colunas cada. Depois, crie um novo usuário chamado **programador** e atribua os privilégios de manipulação de dados para que ele possa selecionar, inserir e atualizar os dados de apenas uma dessas tabela.

REVOKE

A instrução **REVOKE** contém a seguinte sintaxe:

```
REVOKE privileges ON nome_banco.nome_tabela FROM usuario@endereco;
```

Note que ela é semelhante à sintaxe do comando **GRANT**. A única diferença é que a palavra reservada **TO** foi substituída pela palavra **FROM**. Portanto, a construção dos seus comandos será bem similar ao que você já tem feito.

Para revogar todos os privilégios de um usuário, execute o seguinte comando:

```
REVOKE ALL PRIVILEGES ON *.* FROM 'teste'@'localhost';
```

Como o MySQL Community não permite que usuários sejam desabilitados, a instrução **REVOKE** é muito utilizada para manter o usuário criado sem privilégio algum. É comum também a senha do usuário ser trocada para que as credenciais não sejam utilizadas para autenticação no MySQL.

Assim como na instrução **GRANT**, aqui também é preciso utilizar o comando **FLUSH** para atualizar as alterações. Então, assim que você terminar de conceder ou revogar permissões, execute o seguinte comando:

```
FLUSH PRIVILEGES;
```

Por fim, confira se as permissões realmente foram removidas com o comando:

```
SHOW GRANTS FOR 'teste'@'localhost';
```

Grants for teste@localhost
'GRANT USAGE ON *.* TO `teste`@`localhost`'

Anteriormente, tinham sido concedidas as permissões **CREATE**, **SELECT**, **INSERT** e **UPDATE** ao usuário “teste3”. Agora, remova essas permissões. Para revogar privilégios específicos, especifique no comando quais são os privilégios exatamente como no **GRANT**. Seu comando ficará assim:

```
REVOKE CREATE, SELECT, INSERT, UPDATE ON *.* FROM  
'teste3'@'localhost';
```

Após revogar as permissões, execute o comando **FLUSH** para aplicar as alterações e verificar se está tudo correto com o comando **SHOW GRANTS**:

```
FLUSH PRIVILEGES;  
SHOW GRANTS FOR 'teste3'@'localhost';
```

Grants for teste3@localhost
'GRANT USAGE ON *.* TO `teste3`@`localhost`'

Roles

Em português, a tradução da palavra **role** (“papel” ou “função”) refere-se a um papel empenhado por alguém em uma situação particular, uma responsabilidade.

Role no MySQL é uma coleção nomeada de privilégios. Assim como as contas de usuário, as **roles** podem ter privilégios concedidos e revogados. Uma conta de usuário pode receber **roles**, o que concede à conta os privilégios associados a cada **role**. Isso permite a atribuição de conjuntos de privilégios a várias contas e fornece uma alternativa conveniente para conceder privilégios individualmente.

Considere o seguinte cenário:

- ◆ Um aplicativo usa um banco de dados chamado “app_db”.
- ◆ Associado ao aplicativo, pode haver contas para desenvolvedores que criam e mantêm o aplicativo e para usuários que interagem com ele.
- ◆ Os desenvolvedores precisam de acesso total ao banco de dados.
Alguns usuários precisam apenas de acesso de leitura; outros precisam de acesso de leitura/gravação.

Para que não seja necessário conceder privilégios individualmente para cada conta de usuário, podem-se ser criadas as **roles** como nomes para os conjuntos de privilégios necessários. Isso facilitará a definição dos privilégios necessários para as contas de usuário que serão criadas agora e poderá ser aproveitado no futuro, caso surjam novas contas para serem criadas.

Para criar as **roles**, utilize a instrução **CREATE ROLE**:

```
CREATE ROLE 'app_developer', 'app_read', 'app_write';
```

Nessa instrução foram definidas então três *roles*: **app_developer**, destinada para desenvolvedores, **app_read**, para usuários que poderão realizar apenas leitura no BD, e **app_write**, para usuários que poderão realizar escrita.

Agora, para definir as permissões de cada *role*, utilize a seguinte instrução **GRANT**:

```
GRANT ALL ON app_db.* TO 'app_desenvolvedor';  
GRANT SELECT ON app_db.* TO 'app_leitura';  
GRANT INSERT, UPDATE, DELETE ON app_db.* TO 'app_escrita';
```

Com as *roles* criadas e as permissões atribuídas, vincule agora a conta de usuário à *role* desejada. Primeiro, crie os seguintes usuários:

```
CREATE USER 'usuario_desenvolvedor'@'localhost';  
CREATE USER 'usuario_leitura'@'localhost';  
CREATE USER 'usuario_escrita'@'localhost';
```

Agora, vincule cada usuário a uma *role* diferente:

```
GRANT 'app_desenvolvedor' TO 'usuario_desenvolvedor'@'localhost';  
GRANT 'app_leitura' TO 'usuario_leitura'@'localhost';  
GRANT 'app_leitura', 'app_escrita' TO 'usuario_escrita'@'localhost';
```

A instrução **GRANT** do usuário “usuario_escrita” (terceira linha do *script* anterior) concede as *roles* de leitura e gravação, que se combinam para fornecer os privilégios de leitura e gravação necessários.

Após a associação de *roles* a um usuário, pode ser necessário incluir uma *role* padrão a ele, especificando qual *role* deve ser ativada por padrão para um usuário específico quando ele se conectar ao servidor. Isso afeta os privilégios que são concedidos ao usuário durante a sessão. Para isso, usa-se o comando **SET DEFAULT ROLE**, como no exemplo a seguir:

```
SET DEFAULT ROLE 'app_escrita' TO 'usuario_escrita'@'localhost';  
FLUSH PRIVILEGES;
```

Que tal realizar alguns desafios?

Experimente autenticar-se com cada um dos usuários criados anteriormente e realizar operações de leitura, escrita e atualização em tabelas de um banco de dados. Verifique os possíveis erros emitidos.

Caso você precise descartar uma *role*, use a instrução **DROP ROLE**:

```
DROP ROLE 'app_leitura', 'app_escrita';
```

A eliminação de uma *role* revogará todas as permissões concedidas a todas as contas.

A consulta à tabela “mysql.user” mostrará todas as permissões diretamente associadas a um usuário do banco de dados, independentemente das *roles* associadas a ele. Para verificar as permissões garantidas a um usuário a partir de uma *role* específica, use o comando **SHOW GRANTS FOR <usuario> USING <nome_role>** ;. Por exemplo:

```
SHOW GRANTS FOR 'usuario_desenvolvedor'@'localhost' USING  
'app_desenvolvedor';
```

Que tal realizar alguns desafios?

A instrução **REVOKE** também pode ser aplicada para criar *roles*. Considerando isso, crie uma *role* chamada **app_desativado** e negue todas as permissões do usuário “usuario_desenvolvedor”.

Encriptação de dados

A encriptação é uma técnica utilizada para a proteção de informações em sistemas computacionais, que adiciona uma camada adicional de segurança. Essa camada pode ser usada para proteger os dados contra violações realizadas por qualquer pessoa que não seja usuário autorizado.

A encriptação envolve a conversão de um texto simples e legível para humanos em um texto incompreensível, que é conhecido como texto cifrado. Essencialmente, isso significa pegar dados legíveis e alterá-los para que pareçam aleatórios.

Criptografia simétrica

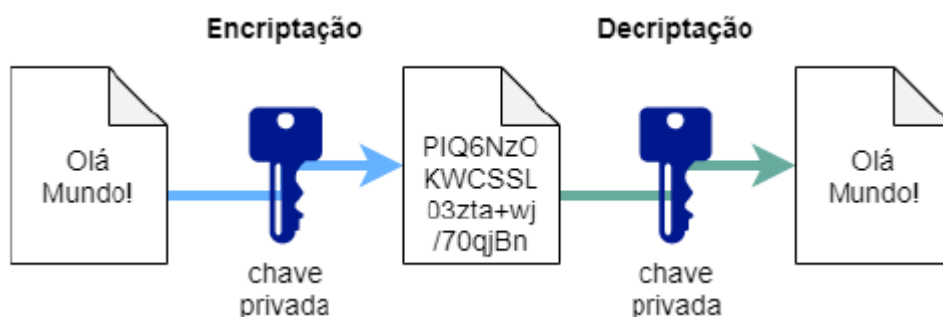


Figura 4 – Criptografia simétrica: uma única chave codifica e decodifica a informação
Fonte: Senac EAD (2022)

A criptografia simétrica, ou criptografia de chave privada, usa uma única chave para criptografar dados. A chave é um valor textual que funciona como uma senha para codificar e decodificar uma informação.

Nas criptografias simétricas, tanto a encriptação quanto a deciptação usam a mesma chave, tornando esta a forma mais fácil de criptografia. Em outras palavras, a criptografia simétrica utiliza uma chave única para cifrar e decifrar os dados.

Apesar de ser considerada menos segura devido ao uso de uma única chave para criptografia, a criptografia de chave simétrica funciona com baixo uso de recursos computacionais. O tamanho da chave pode variar de 128 *bits* até 256 *bits*.

Confira alguns algoritmos que usam a criptografia simétrica:

- ◆ AES (*advanced encryption standard*)
- ◆ DES (*data encryption standard*)
- ◆ IDEA (*international data encryption algorithm*)
- ◆ Blowfish (*drop-in replacement for DES or IDEA*)

O algoritmo de criptografia simétrica mais utilizado é o AES (que significa “padrão de criptografia avançada”, em português).

Criptografia assimétrica

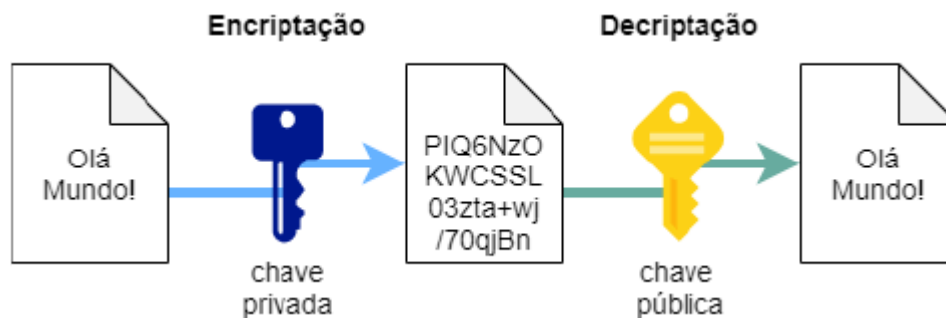


Figura 5 – Criptografia assimétrica: chaves diferentes são usadas para codificação e decodificação Fonte: Senac EAD (2022)

A criptografia assimétrica, ou criptografia de chave pública, usa duas chaves para criptografar dados. Uma é usada para realizar a criptografia, enquanto a outra chave é usada para decifrar os dados. Ao contrário da criptografia simétrica, se uma chave é usada para criptografar, essa mesma chave não pode ser usada para descriptografar.

Apesar de ser mais lenta que a criptografia simétrica, a criptografia assimétrica é considerada a criptografia mais segura, já que são necessárias duas chaves para criptografia e descryptografia. Ela contém chaves com tamanhos a partir de 2048 *bits*, que, por isso, requerem alto consumo de recursos computacionais para serem utilizadas.

Alguns algoritmos que usam a criptografia assimétrica são:

- ◆ RSA (Rivest Shamir Adleman)
- ◆ Diffie-Hellman
- ◆ Elliptical Curve Cryptography (ECC)

O algoritmo de criptografia assimétrica mais utilizado é o **RSA** (Rivest-Shamir-Adleman).

Funções de *hash*



Figura 6 – *Hash*: não depende de chave, mas não pode ser revertido

Fonte: Senac EAD (2022)

As funções de *hash* são funções irreversíveis e unidirecionais, que protegem os dados ao custo de não ser possível recuperar a mensagem original. *Hashing* é uma maneira de transformar um determinado texto em um texto de comprimento fixo. Um bom algoritmo de *hash* produzirá saídas exclusivas para cada entrada fornecida. A única maneira de quebrar uma *hash* é tentar todas as entradas possíveis até obter exatamente a mesma *hash*.

Muitos algoritmos de criptografia utilizam *hash* para aprimorar a segurança cibernética. Dados criptografados com *hash* não têm sentido para *hackers* sem uma chave de descryptografia.

Confira alguns exemplos de funções *hash*:

- ◆ MD5 (*message digest 5*): essa função tem o tamanho de 128 *bits*
- ◆ SHA-1 (*secure hash algorithm 1*): essa função tem o tamanho de 160 *bits*
- ◆ SHA-2 (*secure hash algorithm 2*): essa função tem o tamanho variado de 224 *bits* até 512 *bits*

Como o conteúdo das funções *hash* não pode ser descryptografado, elas se tornam um grande aliado para proteger dados sensíveis, como a senha de usuários, e são frequentemente implementadas em bancos de dados.

Criptografia no MySQL

Pensando na segurança dos dados que serão armazenados, o MySQL desenvolveu diversas funções internas para aumentar a segurança das informações, incluindo algoritmos de criptografia.

As funções disponíveis são:

Nome	Descrição
AES_DECRYPT()	Descriptografar usando AES
AES_ENCRYPT()	Criptografar usando AES
COMPRESS()	Retorna o resultado como uma <i>string</i> binária
MD5()	Calcular soma de verificação MD5
RANDOM_BYTES()	Retorna um vetor de <i>bytes</i> aleatório
SHA1(),SHA()	Calcular uma soma de verificação SHA-1 de 160 <i>bits</i>
SHA2()	Calcular uma soma de verificação SHA-2
STATEMENT_DIGEST()	Valor de <i>hash</i> de resumo de instrução de computação
STATEMENT_DIGEST_TEXT()	Computar resumo de instrução normalizado
UNCOMPRESS()	Descompactar uma <i>string</i> compactada
UNCOMPRESSED_LENGTH()	Retorna o comprimento de uma <i>string</i> antes da compressão
VALIDATE_PASSWORD_STRENGTH()	Determinar a força da senha

As funções mais utilizadas são as de *hash* **MD5**, **SHA-1** e **SHA-2**, e de criptografia simétrica **AES**.

Clique ou toque para acessar o conteúdo.



MD5

O algoritmo de *hash* **MD5** criará um texto criptografado de 128 *bits*, que equivale a 32 caracteres (16 *bytes*), independentemente do tamanho do texto original.

Por exemplo, se você quiser usar o **MD5** para criptografar a senha **SenacEAD_2022** (13 caracteres), no final terá o texto criptografado como **bfacf2674edde9a5558e5108b97e5732** (16 caracteres). Portanto, caso você queira guardar esse registro em uma tabela, é importante que a coluna tenha pelo menos 16 caracteres disponíveis para guardar essa informação.

Para exemplificar, crie uma tabela de usuário que será utilizada para fazer a autenticação em um sistema. Com os conhecimentos que você adquiriu até aqui, você já deve ter em mente como ficaria a estrutura dessa tabela. Será preciso uma *Id*, um nome de usuário e uma senha. Outros campos poderiam estar presentes nesse exemplo, mas trabalhe agora apenas com esses, para focar nos recursos de criptografia.

O *script* de criação da base de dados e tabela ficará assim:

```
CREATE DATABASE criptografia_exemplos;  
USE criptografia_exemplos;  
  
CREATE TABLE cadastro_usuarios (  
  id int PRIMARY KEY AUTO_INCREMENT,  
  usuario varchar(30),  
  senha text  
);
```

Você estará criando a tabela com a coluna “senha”, com o tipo **TEXT**, para ter um limite de caracteres a fim de aproveitar essa mesma tabela nos exemplos seguintes. Em um cenário real, melhor seria que a coluna “senha” tivesse apenas o número de caracteres necessários para guardar as informações. Caso essa coluna fosse usada para armazenar *hashs* **MD5**, por exemplo, ela deveria ser criada para armazenar apenas 16 caracteres, a fim de evitar desperdício de *bits* no armazenamento em disco do servidor de banco de dados.

Agora, efetue uma inserção nessa tabela. A instrução de inserção (**INSERT**) terá a seguinte sintaxe:

```
INSERT INTO cadastro_usuarios (usuario, senha) VALUES ('Usuário Normal', 'SenacEAD_2022');
```

Porém, quando cadastrados os usuários dessa maneira, a senha fica amostra nos registros da tabela. Se, de alguma forma, os dados dessa tabela forem acessados por alguém não autorizado, todos os usuários estarão comprometidos. Isso sem considerar que muitos usuários tendem a utilizar a mesma senha para todo tipo de autenticação (por exemplo, contas de *e-mail*, redes sociais, serviços de *streaming* etc.), o que aumenta ainda mais a necessidade de essas informações serem protegidas.

Portanto, utilize o **MD5** para criptografar a senha. Para isso, chame a função **md5()** e, como argumento, informe a *string* da senha que você quer cadastrar:

```
INSERT INTO cadastro_usuarios (usuario, senha) VALUES ('Usuário MD5', md5('SenacEAD_2022'));
```

Se você executou os dois comandos de inserção, verá que existem os seguintes dados na tabela:

```
SELECT * FROM cadastro_usuarios;
```

ID	Usuario	Senha
1	Usuário Normal	SenacEAD_2022
2	Usuário MD5	bfacf2674edde9a5558e5108b97e5732

Como o **MD5** faz a criptografia de uma *string*, é importante ter atenção ao cadastrar os registros. Para os algoritmos de *hash*, os textos **Senac EAD** e **Senad EAD**, por exemplo, são totalmente diferentes, mesmo que só se tenha um espaço a mais no final do texto. A conversão desses dois textos em **MD5** ficaria da seguinte maneira:

Texto original	Texto MD5
Senac EAD	ee28ebb1b042898d0ef3e8088a49d5ca
Senac EAD	c9b139522db6d4104b44c9fb84738cba

Com o uso de *hash*, os textos não podem ser descriptografados. Logo, não será possível recuperar o texto original. Por esse motivo, não é possível descobrir qual é a senha cadastrada de um usuário, por exemplo. Caso o usuário perca a senha, a senha deve ser redefinida com uma nova *hash* **MD5**.

SHA-1

A *hash* **SHA-1** produz um texto criptografado de 160 *bits*, equivalente a 40 caracteres (20 *bytes*). Por conter o número de caracteres maior que o **MD5**, ela pode ser considerada mais segura.

No MySQL, assim como a função **md5()**, a função **sha1()** receberá um texto como argumento e gerará um novo texto criptografado. Logo, seu uso é idêntico ao da função **md5()**, mudando apenas o nome da função que está sendo chamada:

```
INSERT INTO cadastro_usuarios (usuario, senha) VALUES ('Usuário SHA-1', sha1('SenacEAD_2022'));
```

Como o **SHA1** é a primeira versão da *hash* SHA, você também tem a opção de chamar a função no MySQL como **sha()**:

```
INSERT INTO cadastro_usuarios (usuario, senha) VALUES ('Usuário SHA', sha('SenacEAD_2022'));
```

Por fim, consulte se está tudo certo na tabela:

```
SELECT * FROM cadastro_usuarios;
```

ID	Usuario	Senha
1	Usuário Normal	SenacEAD_2022
2	Usuário MD5	bfacf2674edde9a5558e5108b97e5732
3	Usuário SHA-1	2bda860b76d9a0343ca46353546f428ae617b310
4	Usuário SHA	2bda860b76d9a0343ca46353546f428ae617b310

SHA-2

A *hash* **SHA-2** é uma evolução da **SHA-1**. A principal diferença entre elas está no número de caracteres criptografados. Enquanto o **SHA-1** contém 20 caracteres, o **SHA-2** oferece diferentes opções de tamanhos, sendo elas:

- ◆ 224 *bits*: 28 caracteres
- ◆ 256 *bits*: 32 caracteres
- ◆ 384 *bits*: 48 caracteres
- ◆ 512 *bits*: 64 caracteres

Como se pode definir o número de caracteres que se quer na criptografia **SHA-2**, é preciso passar dois argumentos para a função **sha2()** do MySQL. O primeiro argumento será o texto que se quer criptografar e o segundo será o tamanho da criptografia em *bits*:

```
INSERT INTO cadastro_usuarios (usuario, senha) VALUES ('Usuário SHA-2 ', sha2('SenacEAD_2022', 224));
```

Consulte a tabela “cadastro_usuarios”:

```
SELECT * FROM cadastro_usuarios;
```

ID	Usuario	Senha
1	Usuário Normal	SenacEAD_2022
2	Usuário MD5	bfacf2674edde9a5558e5108b97e5732
3	Usuário SHA-1	2bda860b76d9a0343ca46353546f428ae617b310
4	Usuário SHA	2bda860b76d9a0343ca46353546f428ae617b310
5	Usuário SHA-2	426e82068b898becc8d3635fee736526730178f9eb983f3d7c269

Essa *hash* pode ser considerada mais segura que o **MD5** e o **SHA-1**. Porém, ela consome mais espaço no armazenamento, já que contém um número de caracteres maior, e pode levar mais tempo para ser processada a comparação entre a entrada do usuário de um sistema e o dado armazenado, por exemplo.

AES

O MySQL contém as funções **AES_ENCRYPT()** e **AES_DECRYPT()** para implementar a criptografia e a descriptografia de dados usando o algoritmo AES, anteriormente conhecido como “Rijndael”. O padrão AES permite vários comprimentos de chave. Por padrão, essas funções implementam **AES** com um comprimento de chave de 128 *bits* (equivalente a 16 caracteres). Pode-se utilizar, ainda, comprimentos de chave de 196 ou 256 *bits*. O comprimento da chave é uma troca entre desempenho e segurança.

Para utilizar a encriptação **AES** no MySQL, chama-se a função **AES_ENCRYPT** e passam-se dois argumentos: o texto que será criptografado e a senha para descriptografar, respectivamente. A instrução ficará, então, da seguinte maneira:

```
INSERT INTO cadastro_usuarios (usuario, senha) VALUES ('Usuário AES',  
AES_ENCRYPT('SenacEAD_2022', 'Minha senha secreta' ));
```

Se você consultar a tabela “cadastro_usuarios”, terá os seguintes resultados:

```
SELECT * FROM cadastro_usuarios;
```

ID	Usuario	Senha
1	Usuário Normal	SenacEAD_2022
2	Usuário MD5	bfacf2674edde9a5558e5108b97e5732
3	Usuário SHA-1	2bda860b76d9a0343ca46353546f428ae617b310
4	Usuário SHA	2bda860b76d9a0343ca46353546f428ae617b310
5	Usuário SHA-2	426e82068b898becc8d3635fee736526730178f9eb983f3d7c269

Repare que a instrução **INSERT** não funcionou. Isso ocorreu porque a criptografia gerada pelo AES não combina com o tipo de dado da coluna “senha”. Então, para utilizar a criptografia, crie uma nova coluna chamada “senha_aes” do tipo **VARBINARY(100)**, e faça a inserção nessa coluna da senha criptografada.

```
ALTER TABLE cadastro_usuarios ADD COLUMN senha_aes  
varbinary(100);
```

```
INSERT INTO cadastro_usuarios (usuario, senha_aes) VALUES ('Usuário  
AES', AES_ENCRYPT('SenacEAD_2022', 'Minha senha secreta' ));
```

Agora, se você executar o comando **SELECT * FROM cadastro_usuarios;**, terá o seguinte resultado:

ID	Usuario	Senha
1	Usuário Normal	SenacEAD_2022
2	Usuário MD5	bfacf2674edde9a5558e5108b97e5732
3	Usuário SHA-1	2bda860b76d9a0343ca46353546f428ae617b310
4	Usuário SHA	2bda860b76d9a0343ca46353546f428ae617b310
5	Usuário SHA-2	426e82068b898becc8d3635fee736526730178f9eb983f3d7c269
6	Usuário AES	Null



Agora sim, a instrução **INSERT** foi concluída com sucesso. Caso seja a primeira vez que você note, talvez estranhe a coluna “senha_aes” estar retornando um **BLOB**. Um campo **BLOB** (*binary large object*, ou “grande objeto binário, em português) é um campo criado para armazenar qualquer tipo de informação em formato binário. Para visualizar a informação nesse dado, clique com o botão direito do *mouse* sobre ele e selecione a opção **Open Value in Editor**.

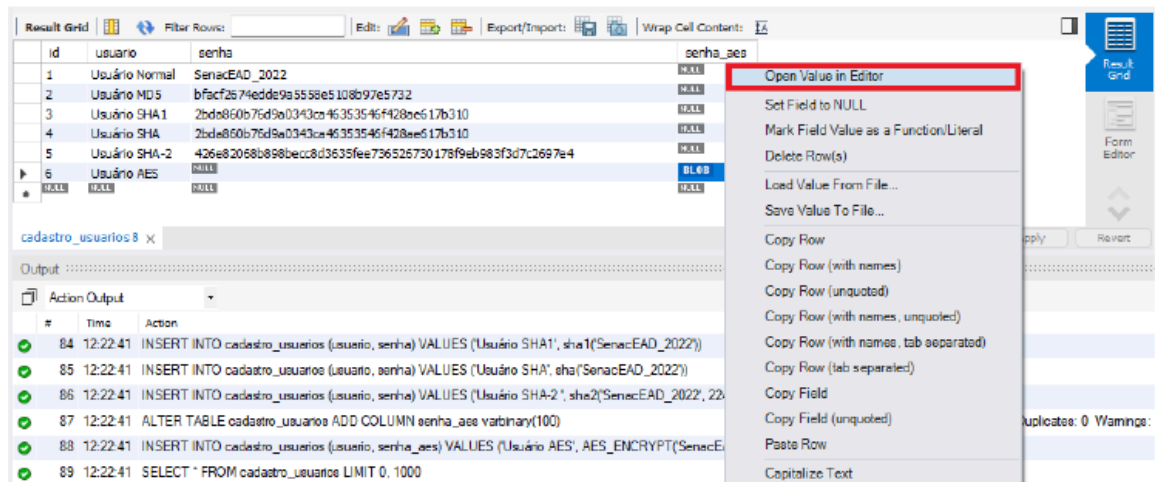


Figura 7 – Acessando a opção **Open Value in Editor** no MySQL Workbench

Fonte: MySQL Workbench (2022)

Com isso, uma nova janela será aberta e você verá o valor binário dessa informação. Clicando na guia **Text**, você visualizará a criptografia gerada pela função **AES**.

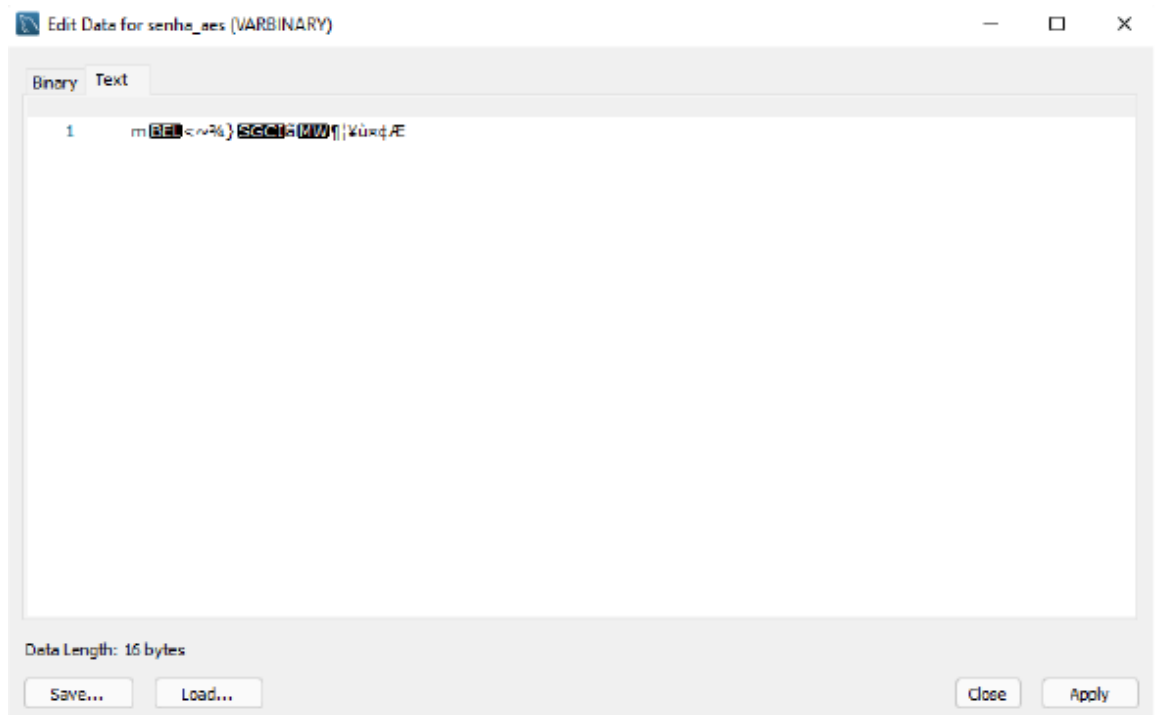


Figura 8 – Texto criptografado pela função **AES**

Fonte: MySQL Workbench (2022)

Diferentes das *hashs* **MD5**, **SHA-1** e **SHA-2**, a criptografia **AES** gera valores binários com caracteres que não se limitam ao alfanumérico. Por essa razão, essa criptografia é considerada mais segura que o uso de *hashs*.

Agora que você já finalizou o processo de criptografia, é preciso descriptografar o texto. Para isso, utilize a função **AES_DECRYPT()**, na qual você deve informar dois argumentos (qual dado você está tentando descriptografar e qual é a senha) para realizar o processo de descriptografia. A instrução SQL ficará do seguinte modo:

```
SELECT
AES_DECRYPT(senha_aes, 'Minha senha secreta')
FROM cadastro_usuarios
WHERE senha_aes is not null;
```

Ao executar essa instrução, porém, novamente haverá o resultado **BLOB**. Logo, o que você precisa fazer é converter esses dados binários para caracteres **CHAR**. Para efetuar essa conversão, utilize o método **CAST()**.

```
SELECT
CAST(AES_DECRYPT(senha_aes, 'Minha senha secreta') AS CHAR(255))
FROM cadastro_usuarios
WHERE senha_aes is not null;
```

CAST(AES_DECRYPT(senha_aes, 'Minha senha secreta') AS CHAR(255))

SenacEAD_2022

Criptografia de sistema e banco de dados

Apesar de o MySQL dispor de funções para encriptar os dados armazenados, essa encriptação também pode ser feita por meio do sistema que estiver utilizando o MySQL. Assim, os dados que estiverem sendo enviados para o banco de dados não correm o risco de serem interceptados e, conseqüentemente, lidos por outra pessoa. Mas isso não impede que os dados criptografados pelo sistema sejam criptografados novamente pelo banco de dados como uma camada extra de segurança.

Por exemplo, imagine que no sistema foi realizado o cadastro do usuário cuja senha seja “Password1”. O sistema pode encriptar essa informação com, por exemplo, o **SHA1** e o banco de dados encriptar esse texto criptografado como **MD5**.

Texto original	Texto criptografado em MD5 pelo sistema	Texto MD5 criptografado com
Password1	2ac9cb7dc02b3c0083eb70898e549b63	e0882992246aad37033110

Assim, caso os dados do banco de dados sejam roubados, o invasor precisaria descriptografar o **SHA-1** do banco de dados para chegar ao **MD5** do sistema e, só então, na senha original. Devido à complexidade do processo, é muito provável que o invasor desista de tentar descobrir a senha original ao deduzir que outras técnicas de segurança possam estar sendo aplicadas na política de senhas, por exemplo.

Existem diversas ferramentas *on-line* para fazer a criptografia e descriptografar textos em *hash* (**MD5**, **SHA-1**, **SHA-2**). Apesar de facilitar o processo de geração de *hash*, alguns sistemas costumam armazenar no banco de dados o texto original em uma coluna e o texto da criptografia em outra coluna. Assim, o processo de descriptografia acaba sendo apenas uma consulta no banco de dados pelo texto original da *hash* informada. Evite usar esse tipo de recurso, pois, caso você esteja

gerando uma *hash*, que será usada como senha, você estará facilitando o processo decriptografia para invasores mal-intencionados. Sempre gere suas *hashes* com funções internas do banco de dados ou funções da linguagem de programação utilizada na construção do sistema e nunca por meio de terceiros.

SQL *injection*

O **SQL *injection*** é um tipo de ameaça de segurança que se aproveita de falhas em sistemas que interagem com bancos de dados SQL. O SQL *injection* ocorre quando o atacante consegue inserir uma série de instruções SQL dentro de uma consulta (*query*) por meio da manipulação das entradas de dados de uma aplicação.

Um ataque de SQL *injection* pode resultar no acesso não autorizado aos dados de tabela, na modificação e exclusão de tabelas inteiras e, em certos casos, na obtenção de direitos administrativos do banco de dados. Muitas violações de dados de perfil nos últimos anos foram resultado de ataques de SQL *injection*, manchando o nome de muitas empresas.

Há uma grande variedade de vulnerabilidades, ataques e técnicas de SQL *injection*, que surgem em diferentes situações. Você conhecerá agora os principais tipos de ataques usando SQL *injection* e como se pode prevenir o banco de dados desse tipo de ataque. Para isso, será utilizada como exemplo uma parte de um sistema *web*.

Considere uma página *web* na qual você precisa informar seu usuário e sua senha para fazer o *login* no sistema. Neste caso, o sistema está solicitando duas entradas: **usuário** e **senha**. Depois de preencher essas informações, o usuário clicará no botão **LOGIN** e, então, o sistema será autenticado ao banco de dados e executará a instrução **SELECT** para verificar se o usuário existe ou não nos registros.



Figura 9 – Exemplo de página *web* solicitando dados para autenticação

Subvertendo a lógica de um sistema

Considere um aplicativo que permite que os usuários efetuem *login* com um nome de usuário e uma senha. Se um usuário enviar o nome de usuário **teste** e a senha **teste123**, o sistema verificará as credenciais realizando a seguinte consulta SQL:

```
SELECT * FROM usuarios WHERE usuario = 'teste' AND senha = 'teste123';
```

Se a consulta retornar os detalhes de um usuário, o *login* foi bem-sucedido. Caso contrário, é rejeitado.

Um invasor pode efetuar *login* como qualquer usuário sem uma senha simplesmente utilizando a sequência de comentários SQL `--` para remover a verificação de senha da cláusula **WHERE** da consulta. Por exemplo, ao preencher o formulário de *login* com **root'--** e não informar senha nenhuma, pode ser possível adulterar a consulta.

Usuário: root'-- Senha:	Resultará na seguinte consulta SQL	SELECT * FROM usuarios WHERE usuario = 'root'-- ' AND senha = "
-------------------------------	--	---

Essa consulta retorna o usuário cujo nome é **root** e registra com êxito o invasor como esse usuário. Perceba que a instrução só é válida até o nome do usuário e tudo o que vem depois se torna um comentário. Logo, o invasor poderia utilizar qualquer nome de usuário com a senha em branco para conseguir se autenticar no sistema.

Tornando todas as consultas verdadeiras

Considere novamente o exemplo anterior. O objetivo original era criar uma instrução SQL para selecionar um usuário com um determinado nome e comentar qualquer outra verificação. Mas, nesse caso, o atacante ainda precisaria ter conhecimento do nome do usuário para ser bem-sucedido.

Outra forma de explorar o SQL *injection* nesse cenário seria adicionar um **OR 1=1** na verificação **WHERE**. A comparação **1=1** é sempre verdadeira. Se você combinar o operador lógico **OR** na instrução SQL, significa que basta uma condição ser verdadeira para a consulta ser executada.

Usuário: ' OR 1=1-- Senha:	Resultará na seguinte consulta SQL	SELECT * FROM usuarios WHERE usuario = " OR 1=1-- ' AND senha = "
----------------------------------	--	---

Note que a dica é fazer com que o SQL busque por um nome vazio de usuário e seguir com a comparação injetada. Dessa forma, o SQL recuperará o primeiro usuário cadastrado na tabela “usuarios” e é com ele que será feita a autenticação no sistema.

Essa é uma falha de segurança que só pode ser corrigida no código-fonte do sistema, implementando uma validação de dados ou utilizando parâmetros SQL. Neste caso, cada parâmetro deve ser verificado para garantir que esteja correto para a sua coluna e seja tratado literalmente e não como parte do SQL a ser executado.

Executando instruções SQL em lote

Um lote de instruções é um grupo de instruções SQL separadas por ponto e vírgula. Considerando o mesmo exemplo do sistema de *login*, a única instrução SQL sendo executada para validar o *login* do usuário é o **SELECT**. Porém, um invasor mal-intencionado pode executar outras instruções após essa como, por exemplo, excluir tabelas.

Usuário: '; DROP TABLE usuarios-- Senha:	Resultará na seguinte consulta SQL	SELECT * FROM usuarios WHERE usuario = "; DROP TABLE usuarios--' AND senha = "
---	--	---

Para proteger o banco de dados desse tipo de ataque *SQL injection*, é importante que o sistema esteja utilizando um usuário do MySQL com permissões limitadas. Portanto, se o usuário não tiver permissões para excluir tabelas, o atacante não terá sucesso em executar a instrução **DROP**. Além disso, é importante que o usuário que será usado pelo sistema para autenticar-se ao MySQL tenha suas permissões limitadas a tabelas específicas. Apesar de ser mais prático definir as permissões para todas as tabelas com **nome_do_banco.***, é nesse cenário que estão as consequências que isso pode causar.

Muitas vulnerabilidades ao *SQL injection* costumam estar presentes no código-fonte do sistema. Porém, ainda existem algumas boas práticas que podem ser aplicadas ao banco de dados para torná-lo mais seguro. Dois principais fatores que contribuem para um ataque de *SQL injection* bem-sucedido são a falta de validação dos dados informados pelo usuário e a utilização de um usuário com altos privilégios pelo sistema. Com essa informação, você já consegue saber quais medidas preventivas aplicar para que um banco de dados não esteja sujeito a esse tipo de vulnerabilidade.

Integridade de dados

A integridade de dados é um componente fundamental da segurança da informação. O seu princípio é garantir a precisão e consistência dos dados armazenados e está diretamente ligado ao banco de dados, pois é no banco de

dados que os dados estão guardados. Para isso, é definida uma série de procedimentos, regras e princípios de validação e verificação de erros baseada em regras de negócios predefinidas.

A integridade de dados é imposta em um banco de dados quando ele é projetado e autenticado por meio do uso contínuo de rotinas de verificação e validação de erros. Na prática, a integridade dos dados verifica se os dados permaneceram inalterados desde a sua criação. Eles passam por inúmeras operações que os manipulam, como criação (instrução **INSERT**), recuperação (instrução **SELECT**), atualização (instrução **UPDATE**), entre outras, e essas operações podem ocorrer diretamente no banco de dados ou por meio de sistemas que se comunicam com o banco de dados. Durante essas operações, os dados devem estar seguros de corrupção, modificação ou divulgação não autorizada, independentemente de essas situações ocorrerem acidentalmente (por exemplo, por meio de erros de programação) ou de forma maliciosa (por exemplo, por meio de violações de segurança e invasões).



Figura 10 – Alguns indícios de quebra na integridade dos dados

Fonte: Adaptado de <<https://techrrival.com/data-integrity-facts/>>. Acesso em: 28 mar. 2022.

Manter a integridade dos dados é uma tarefa que depende tanto dos administradores de banco de dados quanto dos desenvolvedores de sistemas. Durante a construção de um sistema, por exemplo, o desenvolvedor pode definir restrições, em que regras de negócio são impostas para o cadastro de novos dados ou para recuperação de dados já existentes. Já os profissionais de banco de dados devem adotar práticas para garantir a integridade dos dados, tais como:

- ◆ Controles de acesso, incluindo atribuição de privilégios de leitura/gravação
- ◆ Validação de entrada de dados para evitar entradas incorretas de dados
- ◆ Validação de dados para certificar a transmissão não corrompida
- ◆ *Backup* de dados para armazenar uma cópia dos dados em um local alternativo
- ◆ Criação de relacionamentos entre elementos de dados diferentes para garantir que dados transferidos de um estágio para outro sejam precisos e sem erros
- ◆ Uso de *logs* para monitorar quando os dados são inseridos, alterados ou apagados
- ◆ Realização de auditorias internas sistemáticas para garantir que as informações estejam atualizadas

Vários fatores podem afetar a integridade dos dados armazenados em um banco de dados. Entre esses fatores estão:

Clique ou toque para visualizar o conteúdo.

Erros humanos

A entrada manual de dados pode aumentar as chances de erros, duplicações ou exclusão. Muitas vezes, os erros na entrada manual podem se estender à execução de funções internas, que acabam corrompendo os resultados.

Erros de transferência

Um erro de transferência ocorre caso os dados não sejam transferidos com êxito de um *síte* dentro de um banco de dados para outro. Esses erros geralmente ocorrem quando um item de dados existe na tabela de destino, mas está ausente da tabela de origem em um banco de dados relacional.

Bugs e vírus

A integridade de dados também pode ser comprometida por *softwares* maliciosos, que podem invadir o servidor de banco de dados e comprometer a integridade, alterando, excluindo ou roubando os dados.

A integridade dos dados em um banco de dados abrange todos os aspectos da qualidade dessas informações e avança ainda mais ao executar várias regras e procedimentos que supervisionam como as informações são inseridas, recuperadas, transmitidas e muito mais. O objetivo final é proteger os dados contra violações externas ou internas.