Introdução

Por que devemos manter a segurança em um banco de dados? Hoje em dia, possuir informação armazenada de forma correta e disponível é um diferencial que se torna uma vantagem competitiva entre as empresas.

Não adianta armazenar centenas de informações de uma empresa em um banco de dados se não existe controle sobre quem pode acessar essas informações, sobre a exigência de disponibilidade dessas informações.

Neste módulo, discutiremos técnicas para proteger os bancos de dados contra diversas ameaças, bem como formas de fornecer privilégios de acesso às informações apenas para usuários autorizados.

1. Introdução a questões de segurança

A segurança de um banco de dados é uma área muito extensa, que tenta resolver muitos problemas, como:

questões legais e éticas com relação a direito de acesso a informações;

questões políticas em nível governamental, institucional ou corporativo quanto às informações que não devem se tornar públicas;

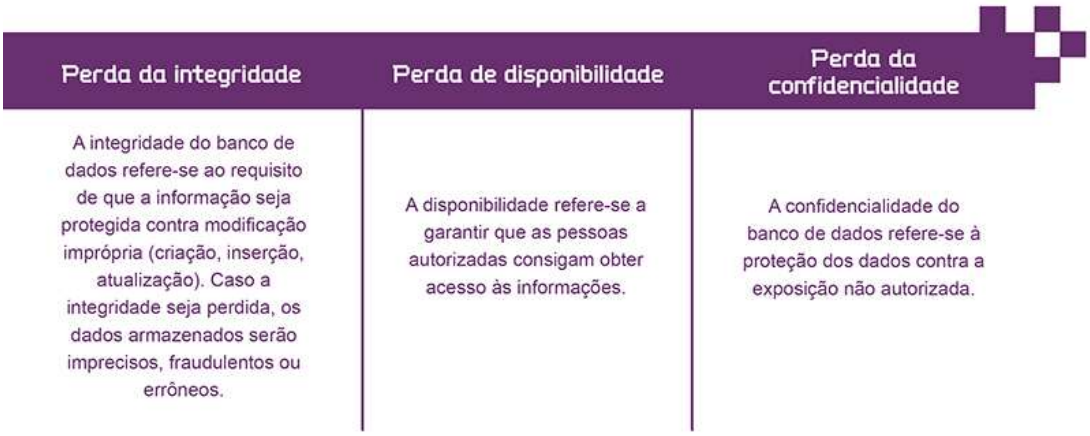
questões relacionadas ao sistema (SGBD);

necessidade de categorizar usuários e dados com base em níveis de segurança (informação secreta, confidencial, pública e não classificada, por exemplo).

Com a leitura dos próximos tópicos, você entenderá melhor a quais ameaças o banco de dados está sujeito, como implantar formas de proteção e quem será o responsável por garantir a segurança do banco de dados.

1.1 Ameaças aos bancos de dados

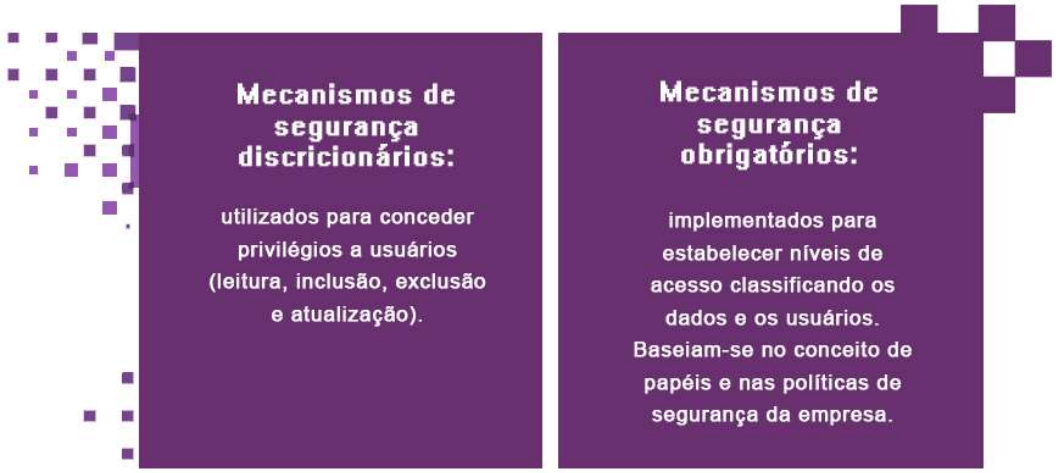
As ameaças que podem afetar os bancos de dados resultam na perda ou na degradação de alguns ou de todos os objetivos de segurança comumente aceitos por um banco de dados: integridade, disponibilidade e confidencialidade.



Perda da integridade:A integridade do banco de dados refere-se ao requisito de que a informação seja protegida contra modificação imprópria (criação, inserção, atualização). Caso a integridade seja perdida, os dados armazenados serão imprecisos, fraudulentos ou errôneos. Perda de disponibilidade: A disponibilidade refere-se a garantir que as pessoas autorizadas consigam obter acesso às informações. Perda da confidencialidade: A confidencialidade do banco de dados refere-se à proteção dos dados contra a exposição não autorizada.

1.2 Mecanismos de segurança

Geralmente, os bancos de dados possuem dois tipos de mecanismos de segurança.



Mecanismos de segurança discricionários: utilizados para conceder privilégios a usuários (leitura, inclusão, exclusão e atualização).

Mecanismos de segurança obrigatórios: implementados para estabelecer níveis de acesso classificando os dados e os usuários. Baseiam-se no conceito de papéis e nas políticas de segurança da empresa.

1.3 Segurança de banco de dados e DBA

O administrador de banco de dados (DBA) é a autoridade máxima. Suas funções são: incluir concessão de privilégios e classificar usuários e dados de acordo com as políticas de acesso da empresa.

Possuindo uma conta no SGBD conhecida como “conta de superusuário” (no banco de dados MySQL, esta conta é similar à do usuário “*root*”), o DBA concede e revoga privilégios a usuários específicos ou a grupos de usuários por meio de comandos privilegiados, podendo executar as operações a seguir.

Criação de contas: cria contas para novos usuários ou grupos para assim habilitar o acesso ao banco.

Concessão de privilégios: é a ação em que se concedem novos privilégios a determinadas contas.

Revogação de privilégios: consiste em cancelar certos privilégios antes concedidos a algumas contas.

Atribuição de nível de segurança: é a atribuição de contas de usuário a níveis de liberação de segurança apropriados.

Dessa forma, o DBA é o responsável por garantir que os dados estejam seguros de qualquer ameaça externa.

2. Controle de acesso discricionário

Este controle é baseado no conceito de direitos de acesso ou privilégios e na maneira de conceder os privilégios aos usuários. Um privilégio permite que um usuário acesse um dado de formas diferentes.

Como forma de visualização de privilégios para usuários de banco de dados, observe a figura 1, onde são mostrados os privilégios do usuário "*root*" no MySQL por meio do menu de Utilizadores / Edição de privilégios do usuário "*root*", da ferramenta **phpMyAdmin**. Note que a figura 1 demonstra que o usuário "*root*" possui todos os privilégios em nível de alteração de dados, estrutura e administração do SGBD.

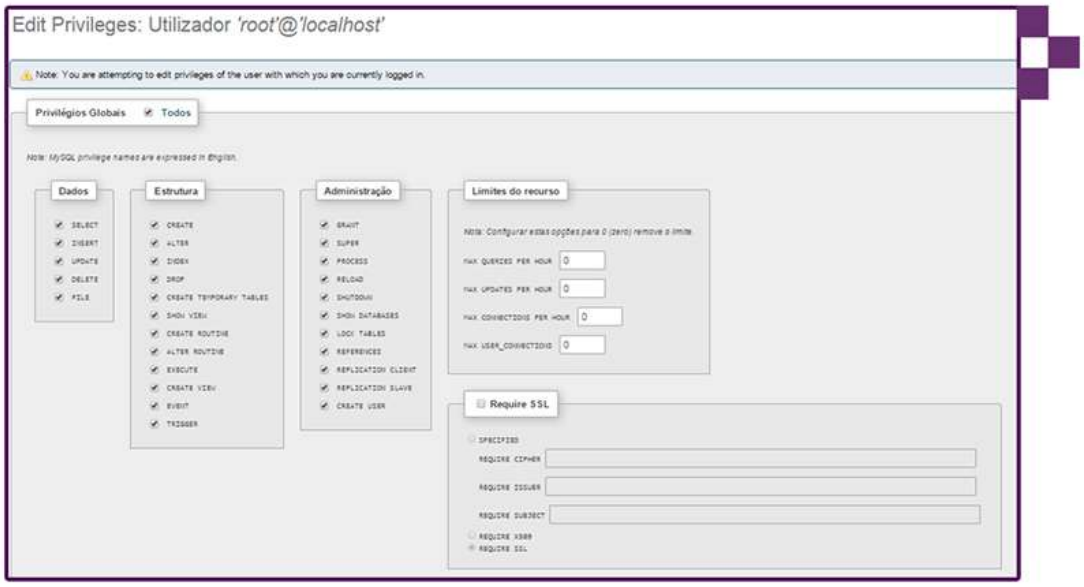


Figura 1 – Privilégios do usuário "root" no banco de dados MySQL

Um usuário que cria um objeto adquire todos os direitos sobre ele automaticamente. A partir de então, o banco de dados guarda todos os privilégios que são concedidos a outros usuários e, dessa forma, garante que apenas os usuários autorizados possam acessar o objeto.

Em diversos bancos de dados, o controle de acesso discricionário é implementado por meio do uso dos comandos **GRANT** e **REVOKE**. O comando **GRANT** concede privilégios sobre os objetos do banco de dados (tabelas, por exemplo) para outros usuários, enquanto o comando **REVOKE** revoga os privilégios concedidos.

Devemos compreender as definições a seguir sobre mecanismos discricionários.

Usuários: são as pessoas que estão representadas por um nome de autorização. Os usuários podem ser classificados em grupos de acordo com um perfil ou nível de autorização. Um usuário que pertence a um grupo, implicitamente, recebe os privilégios relacionados ao grupo a que ele pertence.

Privilégio: define uma permissão individual associada a um nome autorizado, habilitando-o a acessar ou modificar um recurso do banco de dados. Os privilégios também podem ser concedidos a grupos de usuários.

Objetos: os usuários necessitam de privilégios para acessar os objetos guardados no banco de dados. Os privilégios variam de acordo com a natureza do objeto. Por exemplo, uma tabela possui uma lista de privilégios diferente das *triggers*. São exemplos de objetos: tabelas, funções e procedimentos.

2.1 Comando GRANT

Como vimos anteriormente, o comando **GRANT** tem o objetivo de adicionar privilégios. A sintaxe deste comando é a seguinte:

GRANT privilégios **ON** objeto **TO** usuários **[WITH GRANT OPTION]**

Privilégios

- **SELECT**: o direito de leitura sobre as colunas da tabela especificada
- **INSERT**: o direito de inserir linhas
- **UPDATE** (nome-da-coluna, ...): o direito de alterar valores na coluna indicada como objeto
- **DELETE**: o direito de excluir linhas de uma tabela especificada como objeto
- **INDEX**: o direito de criar um novo índice sobre a tabela. Este privilégio aplica-se somente a tabelas
- **TRIGGER**: permite criar, alterar e apagar *triggers*
- Outros.

Objeto: Geralmente uma tabela.

Usuários: Usuário ou grupo de usuários que receberão o privilégio.

Se um usuário recebe o privilégio com a opção **GRANT OPTION** ele pode passar esse privilégio para outro usuário por meio do comando **GRANT**. Um usuário que cria uma tabela herda automaticamente todos os privilégios aplicáveis à tabela juntamente com o direito de conceder os privilégios para outros usuários. O usuário que cria uma visão tem os mesmos privilégios sobre ela.

Veja, abaixo, um exemplo de concessão de privilégio para o usuário “game_user” de leitura, inserção, atualização e deleção de registros na tabela “Tab_Jogador”.

GRANT SELECT, INSERT, UPDATE, DELETE **ON** Tab_Jogador **TO** game_user;

Outro exemplo seria a concessão de privilégio de leitura para o grupo de usuários “PUBLIC” na tabela “Tab_Jogador”, como pode ser visto abaixo.

GRANT SELECT **ON** Tab_Jogador **TO** PUBLIC;

2.2 Comando REVOKE

Para revogar privilégios concedidos, utiliza-se o comando **REVOKE**. A sintaxe do comando **REVOKE** é a seguinte:

REVOKE privilégios **ON** objeto **FROM** usuários **{ RESTRICT | CASCADE }**

Privilégios: **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **INDEX**, **TRIGGER**, entre outros.

Objeto: geralmente uma tabela.

Usuários: usuário ou grupo de usuários que perderão o privilégio.

A opção de **RESTRIC** significa que o comando se recusa a remover a tabela se existirem objetos dependentes. Já a opção **CASCADE** remove automaticamente os objetos que dependem da tabela, ou seja, faz uma remoção em cascata.

Como exemplo do comando **REVOKE** você pode imaginar a remoção do privilégio de leitura para o grupo de usuários “PUBLIC” na tabela “Tab_Jogador”. Para isso, devemos escrever o comando da seguinte forma:

REVOKE SELECT **ON** Tab_Jogador **FROM** PUBLIC;

3. Injeção de SQL

Você já deve ter ouvido falar de problemas relacionados a esta ameaça à segurança da informação, como, por exemplo, empresas que tiveram as informações de banco de dados roubadas em decorrência desta falha. Em 2014, houve um ataque realizado por russos que resultou na obtenção de aproximadamente 1,2 bilhão de *logins* e senhas de internautas.

Você pode ler essa notícia acessando o *link* <https://tecnoblog.net/162786/maior-ataque-hacker-historia-russia/ (https://tecnoblog.net/162786/maior-ataque-hacker-historia-russia/)>.

A injeção de SQL, mais conhecida como *SQL Injection*, é um tipo de ameaça que se aproveita de falhas em sistemas que interagem com bancos de dados utilizando a linguagem SQL.

A intrusão ocorre quando o atacante (conhecido como *hacker*) consegue inserir uma série de instruções SQL dentro de uma consulta SQL por meio da manipulação dos dados inseridos nas entradas de dados das aplicações.

Imaginem que a aplicação de autenticação projetada para o GDD Alien City internamente possui a consulta SQL mostrada na figura 2.

```
SELECT * FROM Tab_Usuarios
WHERE NomeUsuario = 'roberto'
AND   SenhaUsuario = 'senhaderoberto'
```

Figura 2 – Consulta na tabela “Tab_Usuarios” para obtenção de usuário e senha corretos

Caso a entrada de dados de senha não seja válida, o atacante pode alterar a instrução SQL resultante da entrada de senha com a adição de comandos para manipulação de consultas.

Pense que, no momento da autenticação de usuário, foi inserido o nome de um usuário existente na tabela “Tab_Usuarios”. Neste exemplo, foi inserido o nome “roberto” no campo nome de usuário da aplicação de autenticação. Mas, agora, ao invés de uma

senha, o invasor vai tentar inserir comandos para manipular a consulta SQL que será executada internamente no banco de dados. Serão inseridos os seguintes caracteres: **' OR '1'='1**.

Após a inserção desses caracteres, o atacante consegue passar pela autenticação do sistema, porque ele conseguiu inserir mais uma cláusula ao comando **WHERE**, que valida a consulta mesmo sem inserção de senha. A inserção de caracteres para realização do ataque SQL Injection pode ser vista na figura 3, marcada em um retângulo vermelho.

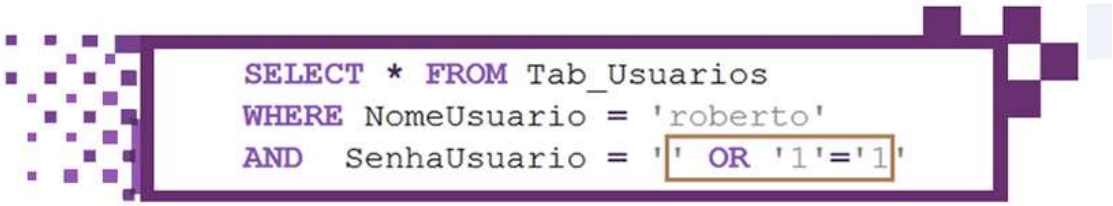


Figura 3 – Consulta na tabela “Tab_Usuarios” com SQL Injection

A proteção contra ataques de SQL Injection pode ser alcançada quando aplicamos algumas regras de programação em todos os procedimentos e em todas as funções que são acessíveis ao usuário. Algumas maneiras de proteção são obtidas pelos meios explicitados a seguir.

Uso de variáveis de ligação para todas as entradas de usuários: nesta técnica, todas as entradas de dados são tratadas como variáveis, e estas são validadas em busca de instruções para manipulação de caracteres.

Validação da entrada no SQL: a entrada de dados é validada por meio da função **Replace** do SQL, que tem o objetivo de substituir um determinado caractere por outro especificado como parâmetro.

Aplicação da segurança nas funções do banco de dados: é prudente restringir o acesso às funções de banco de dados.

4. Criptografia

A criptografia é a técnica de conversão de dados para um formato, chamado texto cifrado, que não pode ser facilmente entendido por pessoas não autorizadas. Ela tem o objetivo de melhorar a segurança e a privacidade das informações em um banco de dados quando os controles de acesso à informação não são utilizados.

Caso você perca ou roubem as informações que foram armazenadas em seu banco de dados, não será fácil entender o conteúdo delas se estiverem criptografadas.

Algumas definições sobre criptografia precisam ficar claras para você. Veja abaixo quais são elas.

Texto cifrado: são informações criptografadas (codificadas).

Texto limpo: representa as informações armazenadas de forma legível a qualquer usuário.

Criptografia: é o processo de tornar o texto limpo em texto cifrado.

Descriptografia: é o processo de transformar texto cifrado de volta para texto limpo.

Funções hash

Uma função **hash** é um algoritmo que mapeia dados de comprimento variável para dados de comprimento fixo. Os valores retornados por uma função **hash** são chamados **valores hash**.

Este processo é unidirecional e impossibilita descobrir o conteúdo original a partir do **hash**. Por esse motivo, as senhas armazenadas em bancos de dados utilizam essas funções antes de salvar a informação de senha do usuário.
Nos bancos de dados, existem várias funções **hash** que podem ser utilizadas para codificar informações, uma delas é a função **MD5**.

Uma forma de utilizar a função **MD5** no MySQL é especificando como parâmetro a entrada de dados onde deseja aplicar a função. O retorno da função é uma cadeia de caracteres com 32 posições.

5. Dicas de segurança para o MySQL

Devido a problemas de segurança que não se referem a falhas do sistema de banco de dados, mas do próprio administrador (DBA), existem diversas recomendações para deixar o MySQL mais seguro.

Veja, a seguir, algumas dicas retiradas da documentação do MySQL.

- Nunca conceda acesso a alguém à tabela “user” do banco de dados “mysql”. Com esse acesso, é possível obter as senhas de outros usuários do MySQL.
- Não mantenha senhas em texto plano em seus bancos de dados. Ao usar senhas, use funções **hash**, como **MD5()** ou **SHA1()**.
- Aprenda e use o sistema de privilégios do MySQL. Dessa forma, você evitará surpresas desagradáveis, como um *drop table* de um usuário que não deveria ter esse privilégio.
- Se o banco de dados for aberto à internet, use criptografia. Use um protocolo como SSL ou SSH para estabelecer uma conexão segura entre o cliente e o servidor.
- Utilize senha para todos os usuários do MySQL. Um cracker geralmente usa um usuário cadastrado no sistema para fazer seus ataques.

Bibliografia

ELMASRI, R.; NAVATHE, S.B. **Sistemas de banco de dados.** Pearson Brasil, 2011.

HEUSER, Carlos Alberto. **Projeto de Bancos de Dados.** Porto Alegre: Sagra Luzzatto, 2009.