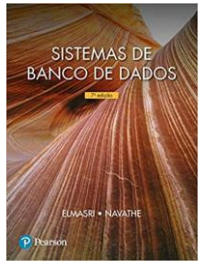
Banco de Dados

SEGURANÇA DE BANCO DE DADOS

André L. Maravilha (https://andremaravilha.github.io/)

Bibliografia



ELMASRI, R.; NAVATHE, S. B. ***Sistemas de banco de dados.*** 7ª edição. São Paulo: Pearson, 2018.

- *Capítulo 30: Segurança de banco de dados*

Introdução

- A segurança do banco de dados é uma área extensa, que tenta resolver muitos problemas, como:
  - Questões legais e éticas com relação ao direito de acessar certas informações. Algumas informações podem ser consideradas particulares e não podem ser acessadas legalmente por organizações ou pessoas não autorizadas.
    - Nos Estados Unidos, existem várias leis que controlam a privacidade da informação. No Brasil, recentemente foi aprovada a lei geral de proteção de dados pessoais (LGPD).
      - http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709.htm
  - Questões políticas em nível governamental, institucional ou corporativo quanto aos tipos de informações que não devem ser tornar públicas – por exemplo, classificação de crédito e registros médicos pessoais.
  - Questões relacionadas ao sistema, como os **níveis de sistema** em que várias funções de segurança devem ser impostas – por exemplo, se uma função de segurança deve ser tratada no nível de hardware, no nível do sistema operacional ou no nível do SGBD.
  - A necessidade, em algumas organizações, de identificar vários **níveis de segurança** e categorizar os dados e usuários com base nessas classificações – por exemplo, altamente secreta, secreta, confidencial e não classificada.

Introdução

Ameaças aos bancos de dados

- As ameaças aos bancos de dados podem resultar na perda ou degradação de alguns ou de todos os objetivos de segurança comumente aceitos:
  - **Perda de integridade:** A integridade do banco de dados refere-se ao requisito de que a informação seja protegida contra modificação imprópria. A integridade é perdida se mudanças não autorizadas forem feitas nos dados por atos intencionais ou acidentais. Se a perda da integridade do sistema ou dos dados não for corrigida, o seu uso continuado pode resultar em decisões imprecisas, fraudulentas ou errôneas.
  - **Perda de disponibilidade:** A disponibilidade do banco de dados refere-se a tornar os objetos disponíveis a um usuário humano ou a um programa ao qual eles têm um direito legítimo. A perda de disponibilidade ocorre quando o usuário ou o programa não consegue acessar esses objetos.
  - **Perda de confidencialidade:** A confidencialidade do banco de dados refere-se à proteção dos dados contra exposição não autorizada. O impacto da exposição não autorizada de informações confidenciais pode variar desde a violação da lei geral de proteção de dados pessoais (LGPD) até o comprometimento da segurança nacional. A exposição não autorizada, não antecipada ou não intencional poderia resultar em perda de confiança pública, constrangimento ou ação legal contra a organização.

Introdução

Segurança de banco de dados

- Ao considerar as ameaças enfrentadas pelos bancos de dados, é importante lembrar que o sistema de gerenciamento de banco de dados não pode ser responsável por manter a confidencialidade, a integridade e a disponibilidade dos dados.
  - Em vez disso, o banco de dados funciona como uma parte de uma rede de serviços, incluindo aplicativos, servidores web, firewalls, terminadores SSL e sistemas de monitoramento de segurança.

"A segurança de um sistema como um todo é tão forte quanto seu elo mais fraco, um banco de dados pode ser comprometido, mesmo que esteja perfeitamente seguro por seus próprios méritos."

Introdução

Segurança de banco de dados

- Em um sistema de banco de dados multiusuário, o SGBD precisa oferecer técnicas para permitir que certos usuários ou grupos de usuários acessem partes selecionadas de um banco de dados sem que obtenham acesso ao restante dele.
  - Isso é particularmente importante quando um grande banco de dados integrado precisa ser usado por diversos usuários diferentes dentro da mesma organização.

Por exemplo, informações confidenciais, como salários de funcionários ou análises de desempenho, devem ser mantidas como confidenciais para a maioria dos usuários do sistema de banco de dados.

Introdução

Segurança de banco de dados

- Um SGBD normalmente inclui um **subsistema de segurança e autorização do banco de dados** que é responsável por garantir a segurança de partes de um banco de dados contra acesso não autorizado.

  - É comum referir-se a dois tipos de mecanismos de segurança de banco de dados:

    - **Mecanismos de segurança discricionário:** Usados para conceder privilégios aos usuários, incluindo a capacidade de acessar arquivos de dados, registros ou campos específicos em um modo especificado (leitura, inserção, exclusão ou atualização).

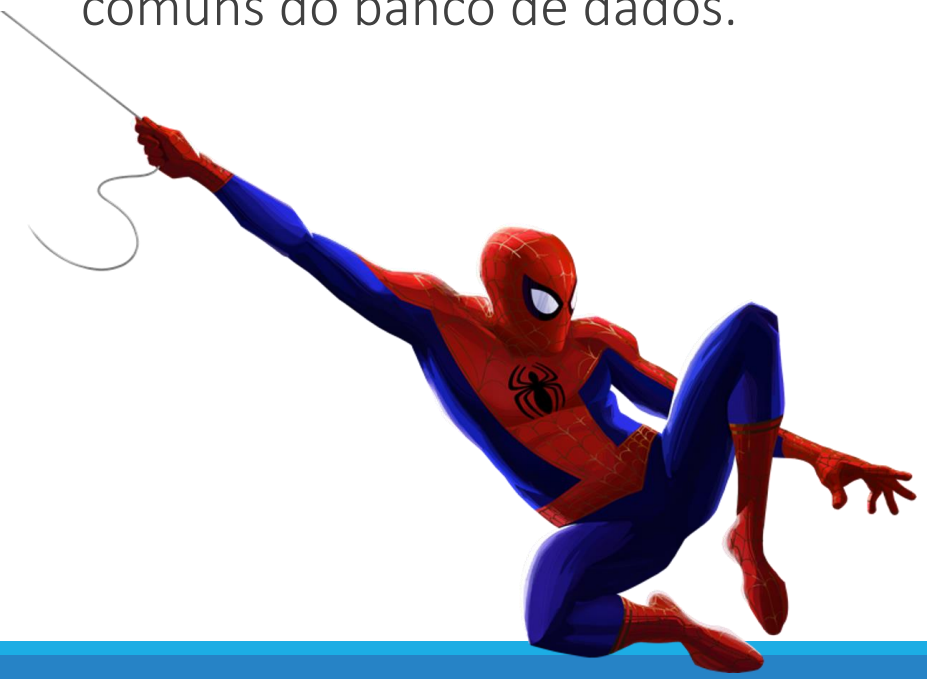
    - **Mecanismos de segurança obrigatórios:** Usados para impor a segurança multinível pela classificação de dados e usuários em várias classes (ou níveis) de segurança e depois, pela implementação da política de segurança apropriada da organização. Por exemplo, uma política de segurança típica é permitir que os usuários em certo nível de classificação (ou liberação) vejam apenas os itens de dados classificados no próprio nível de classificação do usuário (ou em nível inferior).

Medidas de controle

- Para fornecer segurança em bancos de dados contra as possíveis formas de ameaça, quatro medidas de controle principais são usadas:
  - **Controle de acesso:** É o mecanismo de segurança de um SGBD para restringir o acesso de pessoas não autorizadas ao sistema de banco de dados como um todo. É implementado através de contas de usuários e senhas para controlar o processo de *login* pelo SGBD.
  - **Controle de inferência:** Uso de bancos de dados estatísticos para fornecer informações estatísticas ou resumos dos valores com base em diversos critérios. Por exemplo, um banco de dados para estatísticas de população pode oferecer estatísticas com base em faixas etárias, níveis de renda e outros critérios. Os usuários dos bancos de dados estatísticos tem permissão para acessar e recuperar informações estatísticas sobre uma população, mas não podem acessar informações confidenciais detalhadas sobre indivíduos específicos. Medidas de controle correspondentes a esses bancos de dados são chamadas de medidas de controle de inferência.
  - **Controle de fluxo:** É impedir que informações fluam de modo que cheguem até usuários não autorizados. Percursos para as informações fluírem implicitamente de maneiras que violam a política de segurança de uma organização são denominados canais secretos.
  - **Criptografia:** É utilizada para proteger dados confidenciais (como números de cartões de crédito) que são transmitidos por meio de algum tipo de rede de comunicação. A criptografia também pode ser usada para oferecer proteção adicional para partes confidenciais de um banco de dados. Os dados são codificados com o uso de algum algoritmo de codificação. Usuários não autorizados que acessarem dados codificados terão dificuldade para decifrá-los, mas os usuários autorizados recebem o algoritmo de codificação e decodificação (ou chaves) para decifrá-los.

Segurança de bancos de dados e o DBA

- O administrador do banco de dados (DBA) é a autoridade central para gerenciar um sistema de banco de dados. As responsabilidades do DBA incluem conceder privilégios aos usuários que precisam usar o sistema e classificar os usuários e dados de acordo com a política da organização.
- O DBA tem uma **conta de DBA** no SGBD, também conhecida como **conta do sistema** ou **conta de superusuário**, que oferece capacidades poderosas que não estão disponíveis às contas e usuários comuns do banco de dados.



"Com grandes poderes vêm grandes responsabilidades."

Segurança de bancos de dados e o DBA

- Os comandos exclusivos da conta de DBA incluem aqueles necessários para realizar as seguintes ações:
  1. Criação de conta para um usuário ou grupo de usuários para acesso ao SGBD.
  2. Concessão de privilégios a determinadas contas.
  3. Revogação (anulação) de privilégios anteriormente concedidos a certas contas.
  4. Atribuição de níveis de segurança a contas de usuários ao nível de liberação apropriado.

- O DBA é o responsável pela segurança geral do sistema de banco de dados.
  - A ação **1** na lista acima é usada para controlar o acesso ao SGBD como um todo.
  - As ações **2** e **3** são utilizadas para controlar a autorização ***discricionária*** ao banco de dados.
  - A ação **4** é utilizada para controlar a autorização ***obrigatória***.

Auditoria em sistemas de bancos de dados

- O sistema precisa registrar todas as operações no banco de dados que são aplicadas por um certo usuário em cada sessão desde acesso até o momento o seu encerramento.
  - Os programas de aplicação também podem ser considerados usuários.

- Essas operações consistem na sequência de interações entre o usuário e o banco de dados:
  - Quando o usuário inicia uma sessão *(login),* o SGBD pode registrar o número de conta do usuário e associá-lo ao computador ou dispositivo do qual o usuário realizou a conexão.
  - Todas as operações aplicadas desse computador ou dispositivo são atribuídas à conta do usuário até que a sessão seja encerrada (*logoff*).
  - É particularmente importante registrar as operações de atualização aplicadas ao banco de dados de modo que, se for adulterado, o DBA possa determinar qual usuário mexeu nele.

Auditoria em sistemas de bancos de dados

- Para manter um registro de todas as atualizações, pode-se modificar o **log do sistema**.
  - O log do sistema inclui uma entrada para cada operação aplicada ao banco de dados que pode ser exigida para a recuperação de uma falha de transação ou falha do sistema. Podemos expandir as entradas de log de modo que também incluam o número de conta do usuário e o computador on-line ou ID do dispositivo que aplicou cada operação registrada no log.
- Se houver suspeita de qualquer adulteração, é realizada uma **auditoria do banco de dados**, que consiste em rever o log para examinar todos os acessos e operações aplicadas durante certo período.
  - Quando uma operação ilegal ou não autorizada é encontrada, o DBA pode determinar o número de conta usado para realizar a operação.
- As auditorias são particularmente importantes para bancos de dados confidenciais, atualizados por muitas transações e usuários, como nos sistemas bancários que os atualizam por meio de seus diversos caixas.
  - Um log de banco de dados, utilizado principalmente para fins de segurança, às vezes é chamado de **trilha de auditoria**.

Segurança da informação e privacidade da informação

- O avanço rápido do uso da tecnologia da informação (TI) na indústria, no governo e no meio acadêmico gera questões e problemas desafiadores com relação à proteção e ao uso de informações pessoais.
  - Questões como ***quem*** e ***quais direitos*** à informação sobre indivíduos para ***quais finalidades*** tornam-se mais importantes à medida que seguimos para um mundo em que é tecnicamente possível conhecer quase tudo sobre qualquer um.

- Existe uma sobreposição considerável entre questões relacionadas ao acesso a recursos (segurança) e questões relacionadas uso apropriado da informação (privacidade).

  - **Segurança** na tecnologia da informação diz respeito a muitos aspectos da proteção de um sistema contra uso não autorizado, incluindo autenticação de usuários, criptografia de informação, controle de acesso, políticas de *firewall* e detecção de intrusão.

  - O conceito de **privacidade** vai além da segurança. Privacidade examina como o uso da informação pessoal que um sistema adquire sobre um usuário está de acordo com suposições explicitas ou implícitas relativas a esse uso. Em outras palavras, privacidade é a capacidade de os indivíduos controlarem os termos sob os quais sua informação pessoal é adquirida e usada.

Contas de usuário

Criar uma nova conta de usuário

- A linguagem SQL não especifica comandos para criação de contas de usuário. Esse mecanismo é implementado por cada SGBD.

- Geralmente, a sintaxe utilizada para criar um novo usuários é similar a apresentada abaixo.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
CREATE USER [IF NOT EXISTS] usuario IDENTIFIED BY [PASSWORD] senha
```

  - Exemplo 1:
    - *Criar um usuário de nome **andre** com senha **123456**, permitindo o acesso ao SGBD apenas a partir de conexões iniciadas na própria máquina em que o SGBD é executado.*

```
CREATE USER 'andre'@'localhost' IDENTIFIED BY '123456';
```

Contas de usuário

Criar uma nova conta de usuário

- A linguagem SQL não especifica comandos para criação de contas de usuário. Esse mecanismo é implementado por cada SGBD.

- Geralmente, a sintaxe utilizada para criar um novo usuários é similar a apresentada abaixo.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
CREATE USER [IF NOT EXISTS] usuario IDENTIFIED BY [PASSWORD] senha
```

  - Exemplo 1:
    - *Criar um usuário de nome **andre** com senha **123456**, permitindo o acesso ao SGBD apenas a partir de conexões iniciadas na própria máquina em que o SGBD é executado.*

```
CREATE USER 'andre'@'localhost' IDENTIFIED BY '123456';
```

Antes de ser armazenada no SGBD, a senha é transformada por uma função de *hash*. Assim, a senha fica armazenada de forma segura. No MariaDB, é utilizada a função PASSWORD para gerar o *hash* de uma senha.

Contas de usuário

Criar uma nova conta de usuário

- A linguagem SQL não especifica comandos para criação de contas de usuário. Esse mecanismo é implementado por cada SGBD.

- Geralmente, a sintaxe utilizada para criar um novo usuários é similar a apresentada abaixo.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
CREATE USER [IF NOT EXISTS] usuario IDENTIFIED BY [PASSWORD] senha
```

  - Exemplo 2:
    - *Criar um usuário de nome **andre** com senha **123456**, permitindo o acesso ao SGBD apenas a partir de conexões iniciadas na própria máquina em que o SGBD é executado.*

```
CREATE USER 'andre'@'localhost' IDENTIFIED BY PASSWORD '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

Aqui a senha é, também, **123456**. No entanto, é informado diretamente o valor de *hash* obtido com a função **PASSWORD** para o texto **'123456'**.

Contas de usuário

Criar uma nova conta de usuário

- A linguagem SQL não especifica comandos para criação de contas de usuário. Esse mecanismo é implementado por cada SGBD.

- Geralmente, a sintaxe utilizada para criar um novo usuários é similar a apresentada abaixo.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
CREATE USER [IF NOT EXISTS] usuario IDENTIFIED BY [PASSWORD] senha
```

  - Exemplo 2:
    - *Criar um usuário de nome **andre** com senha **123456**, permitindo o acesso ao SGBD apenas a partir de conexões iniciadas na própria máquina em que o SGBD é executado.*

```
CREATE USER 'andre'@'localhost' IDENTIFIED BY PASSWORD '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

Observe que para isso devemos especificar **IDENTIFIED BY PASSWORD** ao invés de apenas **IDENTIFIED BY**.

Contas de usuário

Criar uma nova conta de usuário

- A linguagem SQL não especifica comandos para criação de contas de usuário. Esse mecanismo é implementado por cada SGBD.

- Geralmente, a sintaxe utilizada para criar um novo usuários é similar a apresentada abaixo.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
CREATE USER [IF NOT EXISTS] usuario IDENTIFIED BY [PASSWORD] senha
```

  - **Exemplo 3:**
    - *Criar um usuário de nome **andre** com senha **123456**, permitindo o acesso ao SGBD somente a partir de conexões com o IP de origem igual a 172.68.25.59.*

```
CREATE USER 'andre'@'172.68.25.59' IDENTIFIED BY '123456';
```

Endereço IP (IPv4 ou IPv6) ou nome do *host* de origem da conexão com o SGBD.

Contas de usuário

Criar uma nova conta de usuário

- A linguagem SQL não especifica comandos para criação de contas de usuário. Esse mecanismo é implementado por cada SGBD.

- Geralmente, a sintaxe utilizada para criar um novo usuários é similar a apresentada abaixo.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
CREATE USER [IF NOT EXISTS] usuario IDENTIFIED BY [PASSWORD] senha
```

  - **Exemplo 3:**
    - *Criar um usuário de nome **andre** com senha **123456**, permitindo o acesso ao SGBD a partir de qualquer origem.*

```
CREATE USER 'andre'@'%' IDENTIFIED BY '123456';
```

O operador curinga **%** é utilizado.

Contas de usuário

Criar uma nova conta de usuário

- A linguagem SQL não especifica comandos para criação de contas de usuário. Esse mecanismo é implementado por cada SGBD.

- Geralmente, a sintaxe utilizada para criar um novo usuários é similar a apresentada abaixo.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
CREATE USER [IF NOT EXISTS] usuario IDENTIFIED BY [PASSWORD] senha
```

  - **Exemplo 4:**
    - *Criar um usuário de nome **andre** com senha **123456**, permitindo o acesso ao SGBD a partir de qualquer origem.*

```
CREATE USER 'andre' IDENTIFIED BY '123456';
```

Se o *host* não for especificado, é utilizado o operador curinga **%**. Dessa forma, o usuário poderá se conectar ao SGBD de qualquer origem.

Contas de usuário

Remover uma conta de usuário

- Para excluir um usuário, utilizamos o comando **DROP USER**.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
DROP USER [IF NOT EXISTS] usuario
```

  - **Exemplo 1:**
    - *Apagar a conta de usuário **andre**.*

```
DROP USER 'andre';
```

Como a parte de identificação da origem de conexão não foi especificada, todos as "contas" do usuário **andre** são excluídas.

Contas de usuário

Remover uma conta de usuário

- Para excluir um usuário, utilizamos o comando **DROP USER**.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
DROP USER [IF NOT EXISTS] usuario
```

  - **Exemplo 2:**
    - *Remover do usuário **andre** apenas a autorização de conexão a partir do endereço 172.68.25.59.*

```
DROP USER 'andre'@'172.68.25.59';
```

Contas de usuário

Alterar a senha uma conta de usuário

- Para alterar a senha de acesso de uma conta de usuário, utilizamos o comando **SET PASSWORD**.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
SET PASSWORD [FOR usuario] = hash_senha
```

  - **Exemplo 1:**
    - *Alterar a senha do usuário **andre** para **"qwerty"**.*

```
SET PASSWORD FOR 'andre'@'%' = PASSWORD('qwerty');
```

Contas de usuário

Alterar a senha uma conta de usuário

- Para alterar a senha de acesso de uma conta de usuário, utilizamos o comando **SET PASSWORD**.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
SET PASSWORD [FOR usuario] = hash_senha
```

  - **Exemplo 1:**
    - *Alterar a senha do usuário **andre** para **"qwerty"**.*

```
SET PASSWORD FOR 'andre'@'%' = PASSWORD('qwerty');
```

Deve ser informado o valor *hash* da nova senha. Para isso, utilizamos a funão **PASSWORD**.

Contas de usuário

Alterar a senha uma conta de usuário

- Para alterar a senha de acesso de uma conta de usuário, utilizamos o comando **SET PASSWORD**.
  - Sintaxe utilizada pelo MariaDB e MySQL.

```
SET PASSWORD [FOR usuario] = hash_senha
```

  - **Exemplo 1:**
    - *Alterar a senha do usuário **andre** para **"qwerty"**.*

```
SET PASSWORD FOR 'andre'@'%' = PASSWORD('qwerty');
```

Se não for utilizada a cláusula **FOR** para especificar o usuário, é feita a modificação de senha do usuário atualmente conectado.

Controle de acesso discricionário

- **Mecanismos de segurança discricionário** é usados para conceder privilégios aos usuários, incluindo a capacidade de acessar arquivos de dados, registros ou campos específicos em um modo especificado (leitura, inserção, exclusão ou atualização).

- O método típico para impor o **controle de acesso discricionário** em um sistema de banco de dados é baseado na **concessão e revogação de privilégios**.

- Vamos considerar os privilégios no contexto de SGBDs relacionais, em particular, no sistema de privilégios desenvolvido originalmente para a linguagem SQL.
  - Muitos SGBDs relacionais atuais utilizam alguma variação dessa técnica. A ideia principal é incluir declarações na linguagem de consulta que permita ao DBA e usuários selecionados concederem e revogarem privilégios.

- O SGBD precisa fornecer acesso seletivo a cada tabela no banco de dados com base em contas específicas.
  - As operações autorizadas em uma tabela também podem ser controladas.
  - Dessa forma, o simples fato de ter uma conta para acesso ao SGBD não necessariamente capacita seu mantenedor a todas as funcionalidades oferecidas.

Controle de acesso discricionário

Tipos de privilégios discricionários

- Existem dois níveis para atribuição de privilégios na utilização do sistema de banco de dados:
  - **Nível de conta:**
    - O DBA especifica os privilégios particulares que cada conta mantém independentemente das tabelas no banco de dados.
    - Os privilégios no nível de conta se aplicam às capacidades fornecidas à própria conta e podem incluir os privilégios relacionados à criação de esquemas e tabelas, alteração de tabelas como a inclusão ou remoção de atributos, exclusão de tabelas, atualização de registros e realização de consultas.
    - Os **privilégios em nível de conta não são definidos como parte da SQL**.
      - Eles são implementados por cada SGBDs de forma específica.
  - **Nível de tabela (ou relação):**
    - Neste nível, o DBA pode controlar o privilégio para acessar cada tabela ou *view* individual no banco de dados.
    - Comandos SQL oferecem privilégios apenas no **nível de tabela (relação) e coluna (atributo)**.
      - Os privilégios no nível de tabela especificam para cada usuário as tabelas individuais sobre as quais cada tipo de comando pode ser aplicado.
      - Alguns privilégios podem, também, ser especificados a colunas individuais das tabelas.

Controle de acesso discricionário

Tipos de privilégios discricionários

- Os privilégios normalmente correspondem ao direito de usar certos comandos SQL para acessar determinadas tabelas.

- Em SQL, os seguintes tipos de privilégios podem ser concedidos a usuários em cada tabela individual:
  - **Privilégio de recuperação (ou leitura):** Dá o privilégio de recuperação de informações. Em SQL, isso dá à conta o privilégio de usar a instrução SELECT para recuperar registros de uma tabela.
  - **Privilégio de modificação:** Dá à conta a capacidade de modificar os registros de uma tabela. Em SQL, isso inclui três privilégios: UPDATE, DELETE e INSERT. Estes correspondem aos três comandos SQL para modificar as linhas de uma tabela. Além disso, tanto os privilégios INSERT quanto UPDATE podem especificar que apenas certas colunas de uma tabela podem ser modificadas pela conta.
  - **Privilégio de referência:** Dá à conta a capacidade de referenciar (ou referir-se a) uma tabela ao especificar restrições de integridade. Esse privilégio também pode ser restrito a colunas específicas de uma tabela.

Controle de acesso discricionário

Tipos de privilégios discricionários

- Considerando o SGBD relacional MariaDB (e MySQL), podemos listar os seguintes privilégios.

PRIVILÉGIO	DESCRIÇÃO	NÍVEIS
`ALL PRIVILEGES`	Concede todos os acessos de um nível específico, exceto GRANT OPTION e PROXY.	Global, Esquema, Tabela
`CREATE`	Habilita a criação de bancos de dados e/ou tabelas.	Global, Esquema, Tabela
`ALTER`	Habilita o uso do comando ALTER TABLE.	Global, Esquema, Tabela
`DROP`	Habilita a exclusão de bancos de dados, tabelas e views.	Global, Esquema, Tabela
`SELECT`	Habilita o uso do comando SELECT.	Global, Esquema, Tabela, Coluna
`INSERT`	Habilita o uso do comando INSERT.	Global, Esquema, Tabela, Coluna
`UPDATE`	Habilita o uso do comando UPDATE.	Global, Esquema, Tabela, Coluna
`DELETE`	Habilita o uso do comando DELETE.	Global, Esquema, Tabela
`REFERENCES`	Habilita a criação de chaves estrangeiras.	Global, Esquema, Tabela, Coluna
`USAGE`	Sinônimo de “nenhum privilégio”. Permite apenas que o usuário se conecte ao prompt de comando. Quando um usuário é criado, é atribuído apenas esse “privilégio” a ele.	N/A
`INDEX`	Habilita a criação e exclusão de índices.	Global, Esquema, Tabela
`GRANT OPTION`	Permite que o usuário conceda ou revogue o privilégio a outras contas de usuário.	Global, Esquema, Tabela, Rotina, Proxy

- A lista de privilégios não está limitada à tabela acima.
  Para uma lista completa, acesse https://mariadb.com/kb/en/grant/#database-privileges

Controle de acesso discricionário

Tipos de privilégios discricionários

- A SQL possui construções da linguagem para especificar a **concessão** e **revogação** de privilégios aos usuários. O DBA pode utilizá-las para conceder e/ou revogar privilégios às contas de usuário.

  - **`GRANT`**
    Comando utilizado para concessão de privilégios a contas de usuário.

  - **`REVOKE`**
    Comando utilizado para revogar privilégios de contas de usuário.

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.
  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`
  - **Exemplo 1:** *Conceder **todos** privilégios para o usuário **andre** quando o acesso for realizado a partir da máquina em que o está executando o SGBD (localhost).*

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

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.
  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`
  - **Exemplo 1:** *Conceder **todos** privilégios para o usuário **andre** quando o acesso for realizado a partir da máquina em que o está executando o SGBD (localhost).*

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

O objeto é definido como **<esquema>.<tabela>**. O caractere curinga ***** (asterisco) indica qualquer coisa (esquema ou tabela). Dessa forma, quando utilizado ***.***, estamos atribuindo um privilégio em nível global, ou seja, qualquer tabela de qualquer banco de dados.

Privilégios globais são privilégios administrativos ou aplicados a todos os bancos de dados do servidor.

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.
  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`
  - **Exemplo 1:** *Conceder **todos** privilégios para o usuário **andre** quando o acesso for realizado a partir da máquina em que o está executando o SGBD (localhost).*

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

É necessário que o usuário **andre** com *host* **localhost** tenha sido criado anteriormente. Por exemplo:

```
CREATE USER 'andre'@'localhost' IDENTIFIED BY 'senha';
```

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.
  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`
  - **Exemplo 2:** *Conceder **todos** privilégios sobre o banco de dados **bd_exemplo** ao usuário **andre** independentemente da origem do acesso ao SGBD.*

```
GRANT ALL PRIVILEGES ON bd_exemplo.* TO 'andre'@'%';
```

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.
  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`
  - **Exemplo 2:** *Conceder **todos** privilégios sobre o banco de dados **bd_exemplo** ao usuário **andre** independentemente da origem do acesso ao SGBD.*

```
GRANT ALL PRIVILEGES ON bd_exemplo.* TO 'andre'@'%';
```

Para especificarmos um um privilégio a nível de esquema (banco de dados), indicamos o nome do banco de dados seguido por **.***, que indica qualquer tabela do banco de dados.

Privilégios a nível de esquema (banco de dados) são aplicados a todos os objetos de um banco de dados.

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.
  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`
  - **Exemplo 2:** *Conceder **todos** privilégios sobre o banco de dados **bd_exemplo** ao usuário **andre** independentemente da origem do acesso ao SGBD.*

```
GRANT ALL PRIVILEGES ON bd_exemplo.* TO 'andre'@'%';
```

É necessário que o usuário **andre** com *host* **%** tenha sido criado anteriormente.
Por exemplo:

```
CREATE USER 'andre'@'%' IDENTIFIED BY 'senha';
```

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.
  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`
  - **Exemplo 3:** *Conceder privilégios **SELECT** e **INSERT** ao usuário **andre** (quando conectado a partir de **localhost**) sobre a tabela **tb_1** do banco de dados **bd_exemplo**.*

```
GRANT SELECT, INSERT ON bd_exemplo.tb_1 TO 'andre'@'localhost';
```

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.

  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`

  - **Exemplo 3:** *Conceder privilégios **SELECT** e **INSERT** ao usuário **andre** (quando conectado a partir de **localhost**) sobre a tabela **tb_1** do banco de dados **bd_exemplo**.*

```
GRANT SELECT, INSERT ON bd_exemplo.tb_1 TO 'andre'@'localhost';
```

Um ou mais privilégios podem ser atribuídos em uma única instrução GRANT. Para isso, basta separar os privilégios por vírgula.

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.
  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`
  - **Exemplo 3:** *Conceder privilégios **SELECT** e **INSERT** ao usuário **andre** (quando conectado a partir de **localhost**) sobre a tabela **tb_1** do banco de dados **bd_exemplo**.*

```
GRANT SELECT, INSERT ON bd_exemplo.tb_1 TO 'andre'@'localhost';
```

Para especificarmos um privilégio a nível de tabela, especificamos o nome do banco de dados (esquema), seguido por um ponto e pelo nome da tabela.

Privilégios a nível de tabela são aplicados a todas as colunas da tabela.

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.
  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`
  - **Exemplo 4:** *Conceder privilégios **SELECT** sobre a coluna **col_1** e **INSERT** sobre as colunas **col_1** e **col_2**, da tabela **tb_1** do banco de dados **bd_exemplo**, ao usuário **andre** (quando conectado a partir de **localhost**)*

```
GRANT SELECT(col_1), INSERT(col_1, col_2) ON bd_exemplo.tb_1 TO 'andre'@'localhost';
```

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **GRANT** para concessão de privilégios a contas de usuários.

  - Sintaxe:
    GRANT `privilegio` **ON** `objeto` **TO** `usuário;`

  - **Exemplo 4:** *Conceder privilégios **SELECT** sobre a coluna **col_1** e **INSERT** sobre as colunas **col_1** e **col_2**, da tabela **tb_1** do banco de dados **bd_exemplo**, ao usuário **andre** (quando conectado a partir de **localhost**)*

```
GRANT SELECT(col_1), INSERT(col_1, col_2) ON bd_exemplo.tb_1 TO 'andre'@'localhost';
```

Para especificarmos um privilégio a nível de coluna, basta indicarmos os nomes das colunas desejadas entre parênteses para cada tipo de privilégio.

Os privilégios que podem ser concedidos a nível de coluna são:
SELECT, INSERT, UPDATE e REFERENCES.

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **REVOKE** para revogação de privilégios de contas de usuários.
  - Sintaxe:
    REVOKE `privilegio` **ON** `objeto` **FROM** `usuário;`
  - **Exemplo 1:** *Inicialmente é concedido privilégios **SELECT**, **INSERT**, **UPDATE** e **DELETE** ao usuário **andre** sobre o banco de dados **bd_exemplo**. Em seguida, o privilégio **DELETE** é revogado.*

```
GRANT SELECT, INSERT, UPDATE, DELETE ON bd_exemplo.* TO 'andre'@'localhost';

REVOKE DELETE ON bd_exemplo.* FROM 'andre'@'localhost';
```

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **REVOKE** para revogação de privilégios de contas de usuários.
  - Sintaxe:
    REVOKE privilegio **ON** objeto **FROM** usuário;
  - **Exemplo 2:** *Conceder todos os privilégios a nível global ao usuário* ***andre****, porém remover qualquer privilégio desse usuário ao banco de dados* ***bd_exemplo****.*

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

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

Controle de acesso discricionário

SQL: Concessão e revogação de privilégios

- Uso do comando **REVOKE** para revogação de privilégios de contas de usuários.
  - Sintaxe:
    REVOKE privilegio **ON** objeto **FROM** usuário;
  - **Exemplo 2:** *Conceder todos os privilégios a nível global ao usuário **andre**, porém remover qualquer privilégio desse usuário ao banco de dados **bd_exemplo**.*

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

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

Não e possível revogar privilégios de forma parcial no MariaDB. Ao tentar executar esse comando, será gerado um erro. Só é possível revogar privilégios exatamente iguais aos concedidos.

No entanto, versões mais recentes do MySQL já suportam revogação parcial de privilégios.

Controle de acesso discricionário

SQL: Exemplos de outros comandos úteis...

- Listar os privilégios existentes:

```
SHOW PRIVILEGES;
```

- Exibir os privilégios concedidos a um usuário:

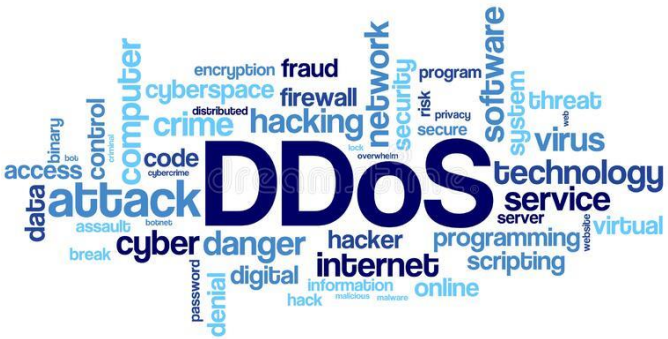
```
SHOW GRANTS FOR andre;
```

- Aplicar as concessões e revogações de privilégios:

```
FLUSH PRIVILEGES;
```

Injeção de SQL

- Injeção de SQL (em inglês, *SQL injection*) é uma das ameaças mais comuns a um sistema de banco de dados. Porém, existem outros ataques a bancos de dados além da injeção de SQL:
  - **Escalada de privilégio não autorizada**
    - Este ataque é caracterizado por um indivíduo que tenta elevar seu privilégio atacando pontos vulneráveis nos sistemas de banco de dados.
  - **Abuso de privilégio**
    - Enquanto o ataque anterior é feito por um usuário não autorizado, este é realizado por um usuário privilegiado. Por exemplo, um administrador que tem permissão para alterar a informação do aluno pode usar esse privilégio para atualizar notas de alunos sem a permissão do professor.
  - **Negação de serviço**
    - Um ataque de negação de serviço (DoS, do inglês *Denial of Service*) é uma tentativa de tornar recursos indisponíveis a seus usuários. Esta é uma categoria de ataque geral em que o acesso a aplicações ou dados da rede é negado aos usuários legítimos pelo estouro do *buffer* ou esgotamento de recursos.
  - **Autenticação fraca**
    - Se o esquema de autenticação do usuário for fraco, um atacante pode personificar a identidade de um usuário legítimo ao obter suas credenciais de *login*.



Injeção de SQL

- Programas e aplicações web que acessam um banco de dados podem enviar comandos e dados a ele, bem como exibir dados recuperados por meio do navegador web.
- Em um **ataque de injeção de SQL**, o atacante injeta uma entrada de cadeia de caracteres pela aplicação, que muda ou manipula a instrução SQL para o proveito do atacante.
- Um ataque de injeção de SQL pode prejudicar o banco de dados de várias maneiras, como na manipulação não autorizada do banco de dados, ou na recuperação de dados confidenciais. Ele também pode ser usado para executar comandos em nível do sistema que podem fazer o sistema negar serviço à aplicação.
- Tipos de ataque de injeção de SQL:
  - Manipulação de SQL.
  - Injeção de código.
  - Injeção de chamada de função.

Injeção de SQL

Manipulação SQL

- Um ataque de manipulação, o tipo mais comum de ataque de injeção, muda um comando SQL na aplicação – por exemplo, acrescentar condições à cláusula WHERE de uma consulta.
  - Outros tipos de ataques de manipulação também são possíveis, como expandir uma consulta com componentes de consulta adicionais, usando operações de união como UNION, INTERSECT ou MINUS.
  - Um ataque de manipulação típico ocorre durante o login de aplicações.

  - **Exemplo:** *Suponha que um procedimento de autenticação ingênuo realize a seguinte consulta e verifique se alguma linha é retornada.*

```
SELECT * FROM usuarios WHERE nome_usuario = 'andre' AND senha = '123456' ;
```

Injeção de SQL

Manipulação SQL

- Um ataque de manipulação, o tipo mais comum de ataque de injeção, muda um comando SQL na aplicação – por exemplo, acrescentar condições à cláusula WHERE de uma consulta.
  - Outros tipos de ataques de manipulação também são possíveis, como expandir uma consulta com componentes de consulta adicionais, usando operações de união como UNION, INTERSECT ou MINUS.
  - Um ataque de manipulação típico ocorre durante o login de aplicações.

  - **Exemplo:** *Suponha que um procedimento de autenticação ingênuo realize a seguinte consulta e verifique se alguma linha é retornada.*

```
SELECT * FROM usuarios WHERE nome_usuario = 'andre' AND senha = '123456' ;
```

O atacante pode tentar alterar (ou manipular) a instrução SQL de forma que o resultado da consulta tenha resultados, mesmo sem conhecer usuário e/ou senha.

Injeção de SQL

Manipulação SQL

- Um ataque de manipulação, o tipo mais comum de ataque de injeção, muda um comando SQL na aplicação – por exemplo, acrescentar condições à cláusula WHERE de uma consulta.
  - Outros tipos de ataques de manipulação também são possíveis, como expandir uma consulta com componentes de consulta adicionais, usando operações de união como UNION, INTERSECT ou MINUS.
  - Um ataque de manipulação típico ocorre durante o login de aplicações.

  - **Exemplo:** *Suponha que um procedimento de autenticação ingênuo realize a seguinte consulta e verifique se alguma linha é retornada.*

```
SELECT * FROM usuarios WHERE nome_usuario = 'andre' AND senha = '123456' ;
```

Suponha que os dados de nome de usuário e senha sejam obtidos através de uma página de *login* de uma aplicação web e, a partir desses dados, a aplicação gera a *query* que será executada para tentar a autenticação do usuário.

Neste caso, foram informados os seguintes dados:
Usuário: **andre**
Senha: **123456**

Injeção de SQL

Manipulação SQL

- Um ataque de manipulação, o tipo mais comum de ataque de injeção, muda um comando SQL na aplicação – por exemplo, acrescentar condições à cláusula WHERE de uma consulta.
  - Outros tipos de ataques de manipulação também são possíveis, como expandir uma consulta com componentes de consulta adicionais, usando operações de união como UNION, INTERSECT ou MINUS.
  - Um ataque de manipulação típico ocorre durante o login de aplicações.

- **Exemplo:** *Suponha que um procedimento de autenticação ingênuo realize a seguinte consulta e verifique se alguma linha é retornada.*

```
SELECT * FROM usuarios WHERE nome_usuario = 'andre' AND senha = '  ?  ' ;
```

Se o atacante informar no campo de senha do formulário o seguinte valor:

```
' OR '' = '
```

Injeção de SQL

Manipulação SQL

- Um ataque de manipulação, o tipo mais comum de ataque de injeção, muda um comando SQL na aplicação – por exemplo, acrescentar condições à cláusula WHERE de uma consulta.
  - Outros tipos de ataques de manipulação também são possíveis, como expandir uma consulta com componentes de consulta adicionais, usando operações de união como UNION, INTERSECT ou MINUS.
  - Um ataque de manipulação típico ocorre durante o login de aplicações.

- **Exemplo:** *Suponha que um procedimento de autenticação ingênuo realize a seguinte consulta e verifique se alguma linha é retornada.*

```
SELECT * FROM usuarios WHERE nome_usuario = 'andre' AND senha = '' OR '' = '' ;
```

Se o atacante informar no campo de senha do formulário o seguinte valor:

```
' OR '' = '
```

Injeção de SQL

Manipulação SQL

- Um ataque de manipulação, o tipo mais comum de ataque de injeção, muda um comando SQL na aplicação – por exemplo, acrescentar condições à cláusula WHERE de uma consulta.
  - Outros tipos de ataques de manipulação também são possíveis, como expandir uma consulta com componentes de consulta adicionais, usando operações de união como UNION, INTERSECT ou MINUS.
  - Um ataque de manipulação típico ocorre durante o login de aplicações.

- **Exemplo:** *Suponha que um procedimento de autenticação ingênuo realize a seguinte consulta e verifique se alguma linha é retornada.*

```
SELECT * FROM usuarios WHERE nome_usuario = 'andre' AND senha = '' OR '' = '' ;
```

Em SQL, o operador **AND** tem precedência maior que **OR**. Dessa forma, independentemente do resultado da condição à esquerda do **OR** a sua condição à direita será verdadeira.

Injeção de SQL

Injeção de código

- **Injeção de código** é um tipo de ataque de injeção de SQL que tenta acrescentar instruções SQL ou comandos adicionais à instrução SQL existente, explorando uma falha do computador, causado pelo processamento de dados inválidos.

- O atacante pode injetar ou introduzir código em um programa de computador para alterar o curso da execução.

- A injeção de código é uma técnica popular para a invasão ou penetração do sistema para obter informações.

Injeção de SQL

Injeção de código

- **Injeção de chamada de função** é um tipo de ataque em que uma função do banco de dados ou uma chamada de função do sistema operacional é inserida em uma instrução SQL vulnerável para manipular os dados ou fazer uma chamada do sistema privilegiada.
  - Por exemplo, é possível explorar uma função que realiza algum aspecto relacionado à comunicação na rede.

- Em particular, consultas SQL criadas dinamicamente podem ser exploradas, visto que são construídas em tempo de execução.

Injeção de SQL

Riscos associados à injeção de SQL

- A injeção de SQL é prejudicial e os riscos associados a ela oferecem motivação para os atacantes.
  - **Impressão digital do banco de dados:** O atacante pode determinar o tipo de banco de dados que está sendo usado no *backend* de modo que possa utilizar ataques específicos ao banco de dados que correspondem a pontos fracos em um SGBD em particular.
  - **Negação de serviço:** O atacante pode inundar o servidor com solicitações, negando assim o serviço a usuários legítimos, ou pode excluir alguns dados.
  - **Contornar a autenticação:** Este é um dos ricos mais comuns, em que o atacante pode obter acesso ao banco de dados como um usuário autorizado e realizar todas as tarefas desejadas.
  - **Identificar parâmetros injetáveis:** Neste tipo de ataque, o atacante reúne informações importantes sobre o tipo e a estrutura do banco de dados de *backend* de uma aplicação web. Esse ataque se torna possível pelo fato de a página de erro padrão retornada pelos servidores de aplicação normalmente ser bastante descritiva.
  - **Executar comandos remotos:** Isso oferece aos atacantes uma ferramenta para executar comandos quaisquer no banco de dados. Por exemplo, um usuário remoto pode executar procedimentos armazenados e funções do banco de dados a partir de uma interface interativa SQL remota.
  - **Realizar escalada de privilégios:** Esse tipo de ataque tira proveito das falhas lógicas dentro do banco de dados para aumentar o nível de acesso.

Injeção de SQL

Técnicas de proteção contra injeção de SQL

- A proteção contra ataques de injeção de SQL pode ser obtida ao se aplicarem certas regras de programação a todos os procedimentos e funções acessíveis pela web.

  - **Variáveis de ligação (usando comandos parametrizados)**
    - O uso de variáveis de ligação (também conhecidas como parâmetros) protege contra ataques de injeção e também melhora o desempenho. Bibliotecas em diversas linguagens de programação para conexão a bancos de dados oferecem esse recurso, como o exemplo usando NodeJS:

```
connection.query("SELECT * FROM usuarios WHERE nome_usuario = ? AND senha = ?",
  [var_usuario, var_senha],
  (error, results) => { /*...*/ });
```

    - O valor de um parâmetro processado de forma que caracteres de escape são manipulados de forma correta.

  - **Filtragem da entrada (validação da entrada)**
    - Esta técnica pode ser usada para remover caracteres de escape das cadeias de caractere de entrada ao utilizar a função **REPLACE** da SQL. Alguns ataques de manipulação de SQL podem ser impedidos com essa técnica, pois os caracteres de escape podem ser utilizados para injetar ataques de manipulação. Porém, como pode haver um grande número de caracteres de escape, esta técnica não é confiável.

  - **Segurança da função**
    - As funções de banco de dados, tanto padrão quanto personalizadas, devem ser restringidas, pois podem ser exploradas nos ataques de injeção de função SQL.

Dúvidas?

André L. Maravilha

andre.maravilha@cefetmg.br

https://andremaravilha.github.io/