PERMISSÕES DE USUÁRIO

Sérgio Mergen

Sumário

- Comandos básicos
- Permissões sobre triggers
- Permissões sobre functions e procedures

Comandos Básicos

- O acesso à tabelas pode ser restrito através de permissões de usuário
- Para isso, SGBDs possuem comandos DCL
  - Data Control Language
- Cabe ao administrador de banco de dados usar comandos da DCL para definir permissões de acesso a cada usuário.
- Os próximos slides mostram exemplos de uso desses comandos

Comandos Básicos

- Criação de um usuário
  - CREATE USER usuario@host;
  - Onde
    - Usuario = nome do usuário
    - Host = a partir de onde o usuário está se conectando
      - O percent (%) é um caractere curinga usado para indicar qualquer endereço

- Remoção de um usuário
  - DROP USER usuario@host;

Comandos Básicos

- Exemplos
  - CREATE USER **joao@localhost**;
    - O usuário joão pode se conectar a partir da própria máquina onde está rodando o MySQL.
  - CREATE USER **joao@'%'**;
    - O usuário joão pode se conectar a partir de qualquer máquina
  - DROP USER **joao@localhost**;
    - O usuário joao não pode mais se conectar a partir da própria máquina onde está rodando o MySQL

Comandos Básicos

- Sintaxe simplificada para atribuição de permissões
  - GRANT privilegio ON objeto TO usuário;

- Exemplos de privilégios
  - ALL: todas permissões concedidas
  - SELECT: permissão para consultas de registros
  - INSERT: permissão para inserção de registros
  - DELETE: permissão para remoção de registros
  - CREATE ROUTINE: permissão para criação de funções/procedimentos
  - ...

Comandos Básicos

- Sintaxe simplificada para atribuição de permissões
  - GRANT privilegio ON objeto TO usuário;

- Exemplos de objetos
  - *.* todos objetos de todos esquemas (databases)
  - bd1.* todos objetos do banco bd1
  - bd1.tab1 tabela tab1 do banco bd1

Comandos Básicos

- Sintaxe simplificada para atribuição de permissões
  - GRANT privilegio ON objeto TO usuário;

- Exemplos
  - GRANT ALL ON ***.*** TO **joao@localhost**
    - **todas** permissões em **todos** bancos concedidas ao **joao**
  - GRANT SELECT ON **bd1.*** TO **joao@localhost**
    - Permissão de **select** em **todos** objetos do banco **bd1** concedida ao **joao**
  - GRANT SELECT ON **bd1.tab1** TO **joao@localhost**
    - Permissão de **select** na tabela **tab1** do banco **bd1** concedida ao **joao**
  - GRANT CREATE ROUTINE ON * TO joao@'%‘
    - Permissão de criar funções/procedimentos em **todos bancos** concedida ao **joao** a partir de **qualquer máquina**

Comandos Básicos

- Sintaxe simplificada para remoção de permissões
  - REVOKE privilegio ON objeto FROM usuário;

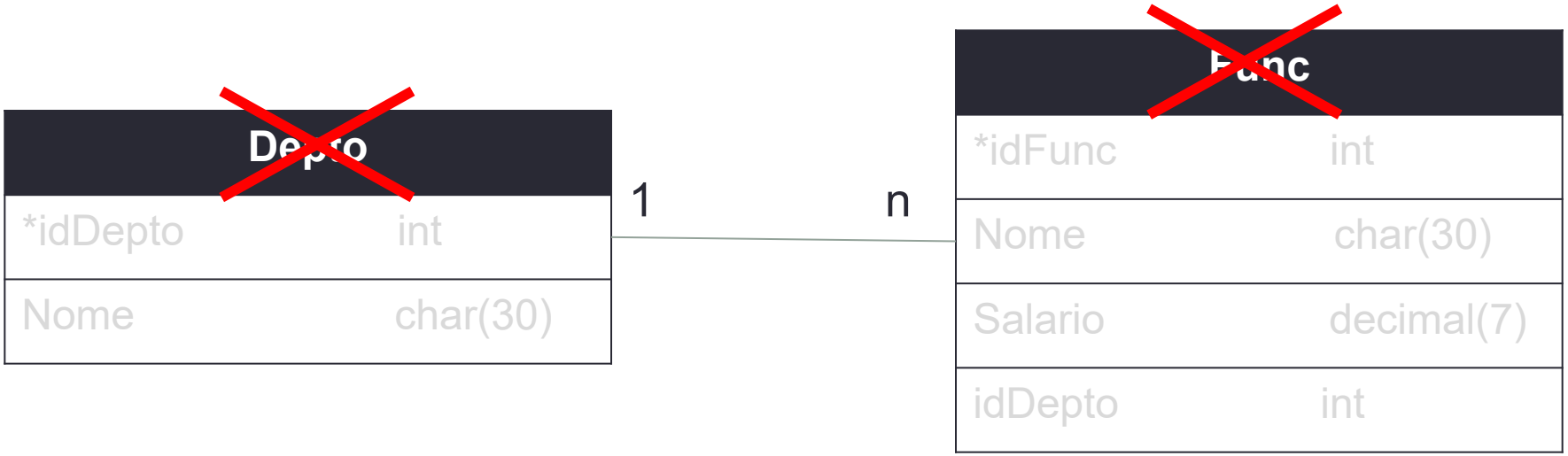
- Exemplos
  - REVOKE ALL ON ***.*** FROM **joao@localhost**
    - **todas** permissões em **todos** objetos removidas do **joao**
  - REVOKE SELECT ON ***.*** FROM **joao@localhost**
    - Permissão de **select** em **todos** objetos removida do **joao**
  - REVOKE SELECT ON **bd1.*** FROM **joao@localhost**
    - Permissão de **select** em **todos** objetos do banco **bd1** removida do **joao**
  - REVOKE SELECT ON **bd1.tab1** FROM **joao@localhost**
    - Permissão de **select** na tabela **tab1** do banco **bd1** removida do **joao**

Exemplo

- Ex.
  - CREATE USER **joao@localhost**;
  - Inicialmente joão não tem acesso a nada. Nenhum esquema de banco poderá ser visualizado

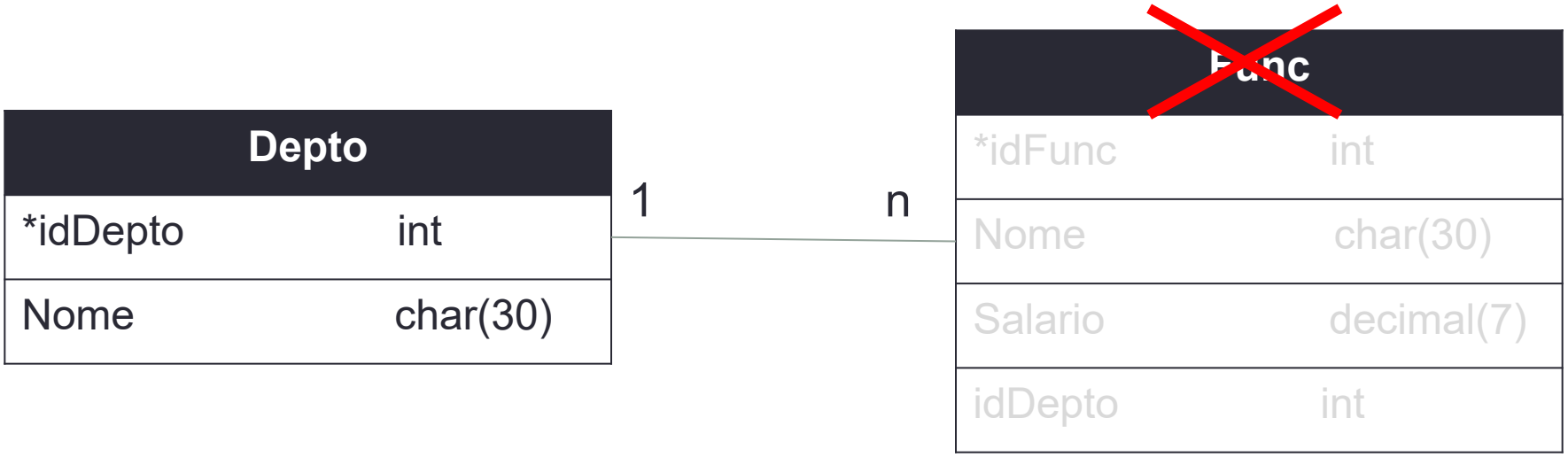
Exemplo

- Ex.
  - GRANT SHOW DATABASES ON ***.*** TO **joao@localhost**;
  - O João pode visualizar o banco de dados bd1
    - mas nenhuma tabela poderá ser acessada



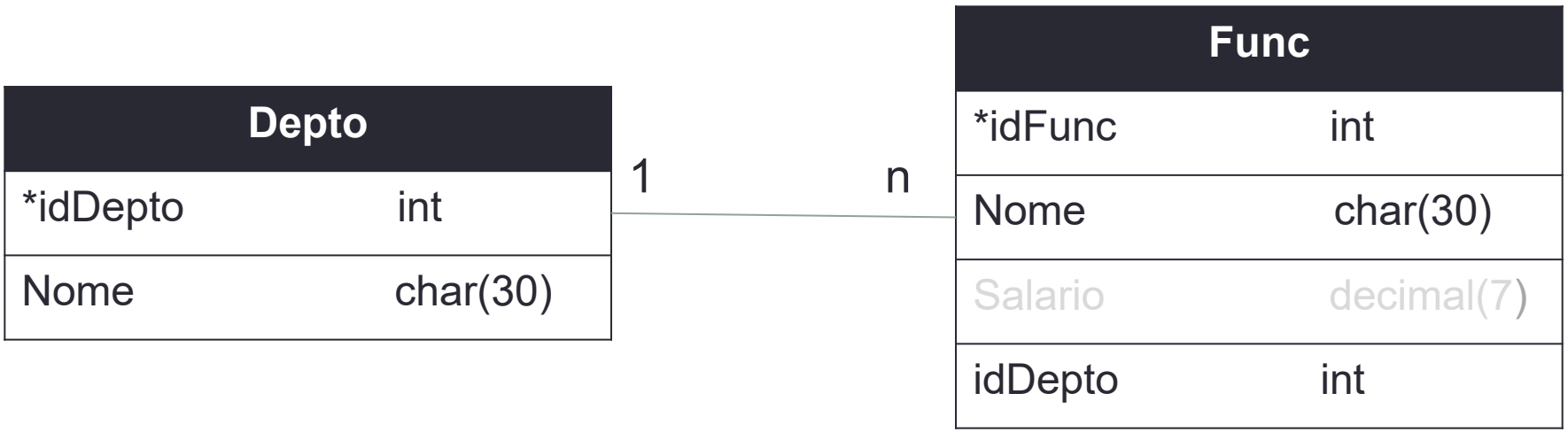
Exemplo

- Ex.
  - GRANT SELECT ON **bd1.depto** TO **joao@localhost**;
  - Agora João pode realizar consultas sobre a tabela depto



Exemplo

- Ex.
  - GRANT SELECT(**idFunc, nome, idDepto**) ON **bd1.Func** TO **joao@localhost**;
  - Agora João pode realizar consultas sobre algumas colunas da tabela Func



Sumário

- Comandos básicos
- Permissões sobre triggers
- Permissões sobre functions e procedures

Permissões sobre triggers

- Para objetos do tipo trigger, existem dois tipos de privilégios que devem ser atendidos:
  - Privilégios para execução
  - Privilégios necessários enquanto o objeto estiver sendo executado

Permissões sobre triggers

- Para que uma trigger seja disparada como resposta a um evento de usuário, esse usuário deve ter permissões especiais

- Ex. GRANT TRIGGER ON ***.*** TO **joao@localhost**;

  - Permissão de **criação e remoção de trigger** concedida ao **joao** em **todas** tabelas

Permissões sobre triggers

- Objetos do tipo trigger possuem um corpo de código
  - que pode ser composto por múltiplos comandos SQL
- A execução é impedida
  - caso se verifique falta de permissão para algum desses comandos
- Esse tipo de controle é definido
  - no ato de criação do objeto

Sintaxe de uma Trigger (mySQL)

CREATE

[DEFINER = { **nome_usuário** | CURRENT_USER }]

TRIGGER **nome_trigger**

quando_disparar **evento_que_dispara** ON **nome_tabela**

FOR EACH ROW

corpo da trigger

As permissões para execução de uma trigger são determinadas pela cláusula DEFINER.

quando_disparar:

{ BEFORE | AFTER }

evento_que_dispara:

{ INSERT | UPDATE | DELETE }

Sintaxe de uma Trigger (mySQL)

CREATE

[DEFINER = { **nome_usuário** | CURRENT_USER }]

TRIGGER **nome_trigger**

quando_disparar **evento_que_dispara** ON **nome_tabela**

FOR EACH ROW

corpo da trigger

nome_usuario: nome de usuário existente.

- Ex. Joao@localhost

quando_disparar:

{ BEFORE | AFTER }

evento_que_dispara:

{ INSERT | UPDATE | DELETE }

Sintaxe de uma Trigger (mySQL)

CREATE

[DEFINER = { **nome_usuário** | CURRENT_USER }]

TRIGGER **nome_trigger**

quando_disparar **evento_que_dispara** ON **nome_tabela**

FOR EACH ROW

corpo da trigger

quando_disparar:

{ BEFORE | AFTER }

evento_que_dispara:

{ INSERT | UPDATE | DELETE }

CURRENT_USER: o usuário que está conectado executando esse comando de criação de trigger.

Sintaxe de uma Trigger (mySQL)

CREATE

[DEFINER = { **nome_usuário** | CURRENT_USER }]

TRIGGER **nome_trigger**

quando_disparar **evento_que_dispara** ON **nome_tabela**

FOR EACH ROW

corpo da trigger

Por padrão, o DEFINER é o CURRENT_USER

quando_disparar:

{ BEFORE | AFTER }

evento_que_dispara:

{ INSERT | UPDATE | DELETE }

Exemplo

- A trigger abaixo remove em cascata as atividades de um projeto

```
DELIMITER $$
CREATE TRIGGER  remProj
    BEFORE DELETE ON projeto
    FOR EACH ROW
            DELETE FROM atividade WHERE numProj = OLD.numProj;
$$
DELIMITER ;
```

Exemplo

- Suponha que o joao executou o comando abaixo
- Como a trigger não tem DEFINER explícito, o CURRENT_USER (joao) passa a ser o ‘dono’ da trigger
  - Quando a trigger for disparada, são as suas permissões que devem ser verificadas

```
DELIMITER $$
CREATE TRIGGER  remProj
   BEFORE DELETE ON projeto
   FOR EACH ROW
         DELETE FROM atividade WHERE numProj = OLD.numProj;
$$
DELIMITER ;
```

Exemplo

- Suponha que seja executado um delete em projeto pelo joao (dono da trigger)
- Para que a trigger funcione, o seu dono (o joao) deve ter privilégio de
  - Remoção de projeto
  - Remoção de atividade

```
DELIMITER $$
CREATE TRIGGER  remProj
   BEFORE DELETE ON projeto
   FOR EACH ROW
          DELETE FROM atividade WHERE numProj = OLD.numProj;
$$
DELIMITER ;
```

Exemplo

- Nesse outro exemplo, o script foi complementado com a informação de quem é o 'dono' da trigger
  - Não necessariamente será o usuário conectado que executou o comando de criação
  - Ex. o usuário root pode estar criando triggers em nome de outro usuário

```
DELIMITER $$
CREATE DEFINER 'joao@localhost'
TRIGGER  remProj
   BEFORE DELETE ON projeto
   FOR EACH ROW
          DELETE FROM atividade WHERE numProj = OLD.numProj;
$$
DELIMITER ;
```

Exercício

- Maria executou um
  - DELETE FROM **proj** WHERE **idProj=5.**
- Quais são as permissões necessárias para funcionar?
  - Da Maria:
  - Do Joao:

```
DELIMITER $$
CREATE DEFINER ‘joao@localhost’
TRIGGER  remProj
   BEFORE DELETE ON projeto
   FOR EACH ROW
          DELETE FROM atividade WHERE numProj = OLD.numProj;
$$
DELIMITER ;
```

Exercício

- Maria executou um
  - DELETE FROM **proj** WHERE **idProj=5.**
- Quais são as permissões necessárias para funcionar?
  - Da Maria: remoção em projeto
  - Do Joao: remoção em atividade

```
DELIMITER $$
CREATE DEFINER ‘joao@localhost’
TRIGGER  remProj
   BEFORE DELETE ON projeto
   FOR EACH ROW
         DELETE FROM atividade WHERE numProj = OLD.numProj;
$$
DELIMITER ;
```

Exercício

- Maria executou um
  - UPDATE **func** SET **idDepto = 3** WHERE **idFunc=5.**
- Quais são as permissões necessárias para funcionar?
  - Da Maria:
  - Do Joao:

```
DELIMITER $$
CREATE DEFINER 'joao@localhost'
TRIGGER  remProj
   BEFORE UPDATE ON func
   FOR EACH ROW
          SET NEW.depto =
             SELECT nome FROM depto WHERE idDepto = NEW.idDepto;
$$
DELIMITER ;
```

Exercício

- Maria executou um
  - UPDATE **func** SET **idDepto = 3** WHERE **idFunc=5.**
- Quais são as permissões necessárias para funcionar?
  - Da Maria: update em func
  - Do Joao: select em depto

```
DELIMITER $$
CREATE DEFINER 'joao@localhost'
TRIGGER  remProj
   BEFORE UPDATE ON func
   FOR EACH ROW
          SET NEW.depto =
             SELECT nome FROM depto WHERE idDepto = NEW.idDepto;
$$
DELIMITER ;
```

Cuidados

- O administrador pode controlar quem é o DEFINER para garantir que o código da trigger só rode com privilégios seguros
  - O adm pode criar uma trigger a pedidos de um usuário que possua os privilégios necessários
    - Se no futuro esse usuário perder os privilégios, a trigger para de funcionar

- Usuários maliciosos não devem poder criar triggers com definers poderosos, pois isso seria uma forma de **elevação de privilégio**

Sumário

- Comandos básicos
- Permissões sobre triggers
- Permissões sobre functions e procedures

Permissões sobre functions e procedures

- Para que uma function (ou procedure) possa ser chamada por um usuário, esse usuário deve ter permissões especiais

- Ex. GRANT EXECUTE ON ***.*** TO **joao@localhost**;
  - Permissão de **execução** concedida ao **joao** para qualquer function ou procedure

Permissões sobre functions e procedures

- Assim como triggers, objetos do tipo procedure e function também possuem um corpo de código
  - que pode ser composto por múltiplos comandos SQL
- A execução é impedida
  - caso se verifique falta de permissão em algum desses comandos
- Esse tipo de controle é definido
  - no ato de criação do objeto

Sintaxe de uma Function (MySQL)

CREATE
[DEFINER = usuario]
FUNCTION nome ([parâmetro,...])
RETURNS tipo
[característica ...] corpo_da_função

parâmetro: nome tipo

tipo: Qualquer tipo de dado suportado pelo MySQL

característica :
 [NOT] DETERMINISTIC
 | SQL SECURITY {DEFINER | INVOKER}
 | COMMENT string

DEFINER: indica quem é o dono da função

Por padrão, o dono é quem executou o comando de criação

Sintaxe de uma Function(MySQL)

CREATE
[DEFINER = usuario]
FUNCTION nome ([parâmetro,...])
RETURNS tipo
[característica ...] corpo_da_função

As permissões para execução de uma função são determinadas pela cláusula SQL SECURITY

parâmetro: nome tipo

tipo: Qualquer tipo de dado suportado pelo MySQL

característica :
[NOT] DETERMINISTIC
| SQL SECURITY {DEFINER | INVOKER}
| COMMENT string

Sintaxe de uma Function(MySQL)

CREATE
[DEFINER = usuario]
FUNCTION nome ([parâmetro,...])
RETURNS tipo
[característica ...] corpo_da_função

DEFINER: indica que as permissões serão as do dono da função

parâmetro: nome tipo

tipo: Qualquer tipo de dado suportado pelo MySQL

característica :
 [NOT] DETERMINISTIC
 | SQL SECURITY {DEFINER | INVOKER}
 | COMMENT string

Sintaxe de uma Function(MySQL)

CREATE
[DEFINER = usuario]
FUNCTION nome ([parâmetro,...])
RETURNS tipo
[característica ...] corpo_da_função

INVOKER: indica que as permissões serão as de quem está chamando a função

parâmetro: nome tipo

tipo: Qualquer tipo de dado suportado pelo MySQL

característica :
 [NOT] DETERMINISTIC
 | SQL SECURITY {DEFINER | INVOKER}
 | COMMENT string

Sintaxe de uma Function(MySQL)

CREATE
[DEFINER = usuario]
PROCEDURE nome ([parâmetro,...])
RETURNS tipo
[característica ...] corpo_da_função

As mesmas configurações de permissão (DEFINER e SQL SECURITY) existem quando se está criando **procedures**

parâmetro: [IN | OUT | INOUT] nome tipo

tipo: Qualquer tipo de dado suportado pelo MySQL

característica :
[NOT] DETERMINISTIC
| SQL SECURITY {DEFINER | INVOKER}
| COMMENT string

Exemplo

- A procedure abaixo remove atividades que satisfaçam um filtro passado por parâmetro

- O INVOKER faz com que se verifiquem os privilégios de acesso do usuário que for chamar a procedure

```
DELIMITER $$
CREATE PROCEDURE remocao (IN id INT)
SQL SECURITY INVOKER
        DELETE FROM atividade WHERE numProj > id;
$$
DELIMITER ;
```

Exemplo

- Suponha que a procedure seja executada pelo usuário joão
  - CALL remocao(4);
- A procedure só funcionará caso o joao tenha privilégio de
  - Execução da procedure
  - remoção de atividade

```
DELIMITER $$
CREATE PROCEDURE remocao (IN id INT)
SQL SECURITY INVOKER
        DELETE FROM atividade WHERE numProj > id;
$$
DELIMITER ;
```

Exemplo

- Agora se usa DEFINER em vez de INVOKER.
- Nesse caso, verificam-se os privilégios de acesso do usuário que criou a procedure

```
DELIMITER $$
CREATE PROCEDURE remocao (IN id INT)
SQL SECURITY DEFINER
        DELETE FROM atividade WHERE numProj > id;
$$
DELIMITER ;
```

Exemplo

- Suponha que a procedure seja executada
  - CALL remocao(4);
- Nesse caso, não importa o usuário que fez a chamada.
  - A verificação de privilégio de remoção é feita para o usuário criador

```
DELIMITER $$
CREATE PROCEDURE remocao (IN id INT)
SQL SECURITY DEFINER
        DELETE FROM atividade WHERE numProj > id;
$$
DELIMITER ;
```

Exemplo

- Nesse outro exemplo, o script foi complementado com a informação de quem é o 'dono' da procedure
  - Não necessariamente será o usuário conectado que executou o comando de criação
  - Ex. O usuário administrador pode estar criando procedures em nome de outro usuário

```
DELIMITER $$
CREATE DEFINER 'joao@localhost'
PROCEDURE remocao (IN id INT)
SQL SECURITY DEFINER
        DELETE FROM atividade WHERE numProj > id;
$$
DELIMITER ;
```

Exercício

- Maria executou um
  - CALL remocao(5);
- Quais são as permissões necessárias para funcionar?
  - Da Maria:
  - Do Joao:

```
DELIMITER $$
CREATE DEFINER 'joao@localhost'
PROCEDURE remocao (IN id INT)
SQL SECURITY DEFINER
        DELETE FROM atividade WHERE numProj > id;
$$
DELIMITER ;
```

Exercício

- Maria executou um
  - CALL remocao(5);
- Quais são as permissões necessárias para funcionar?
  - Da Maria: execução da procedure
  - Do Joao: remoção de atividade

```
DELIMITER $$
CREATE DEFINER 'joao@localhost'
PROCEDURE remocao (IN id INT)
SQL SECURITY DEFINER
        DELETE FROM atividade WHERE numProj > id;
$$
DELIMITER ;
```

Exercício

- Maria executou um
  - CALL remocao(5);
- Quais são as permissões necessárias para funcionar?
  - Da Maria:
  - Do Joao:

```
DELIMITER $$
CREATE DEFINER 'joao@localhost'
PROCEDURE remocao (IN id INT)
SQL SECURITY INVOKER
        DELETE FROM atividade WHERE numProj > id;
$$
DELIMITER ;
```

Exercício

- Maria executou um
  - CALL remocao(5);
- Quais são as permissões necessárias para funcionar?
  - Da Maria: execução da procedure, remoção de atividade
  - Do Joao: nenhuma

```
DELIMITER $$
CREATE DEFINER 'joao@localhost'
PROCEDURE remocao (IN id INT)
SQL SECURITY INVOKER
        DELETE FROM atividade WHERE numProj > id;
$$
DELIMITER ;
```

Orientações Gerais

- Não use um super usuário para acessar as aplicações
- Crie diferentes usuários para aplicações diferentes
- Conceda apenas os privilégios necessários para os usuários criados