

TRIGGERS

Sérgio Mergen

Trigger

- Recurso disponível em muitos SGBDs
- Permite executar ações nas tabelas como resposta a um 'gatilho' (trigger).

Trigger

- O evento disparado pelo gatilho pode ser um dos seguintes
 - Inserção de registro
 - Atualização de registro
 - Remoção de registro
- O disparo pode ocorrer
 - Antes do evento
 - Depois do evento

Sintaxe de uma Trigger (mySQL)

CREATE

[DEFINER = { nome_usuario | 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:

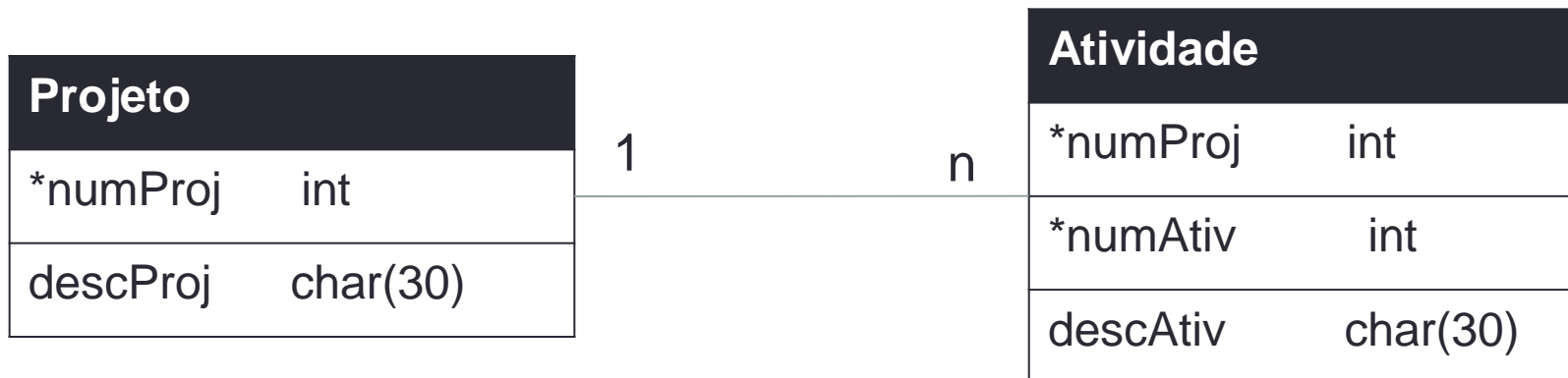
{ INSERT | UPDATE | DELETE }

Aplicações para triggers

- Triggers podem ser usadas para diversas finalidades
- Nessa aula veremos algumas delas
 - Remoção em cascata
 - Check constraint
 - Função de cálculo
 - Controle de especialização
 - Histórico de versões

Triggers e Remoção em Cascata

- A relação de chave estrangeira entre projeto e atividade protege contra vários tipos de ações indevidas
- Ex.
 - Inserção de atividade em projeto inexistente
 - Remoção de projeto que possua atividades vinculadas



Triggers e Remoção em Cascata

- No entanto, o DBA gostaria que o mecanismo de proteção não impedisse a exclusão de projetos
 - A única forma possível de burlar essa proteção é garantir que as atividades do projeto sejam todas removidas antes da remoção do projeto
 - O DBA está estudando maneiras de realizar essa remoção sem que o desenvolvedor precise se preocupar com isso



Triggers e Remoção em Cascata

- No entanto, o DBA gostaria que o mecanismo de proteção não impedisse a exclusão de projetos
 - A única forma possível de burlar essa proteção é garantir que as atividades do projeto sejam todas removidas antes da remoção do projeto
 - Uma das alternativa é usando as ações referenciais (**Referential Actions**) ao criar uma restrição de chave estrangeira



Triggers e Remoção em Cascata

[CONSTRAINT [*nome_restricao*]]

FOREIGN KEY [*nome_indice*] (*col1*, ...)

REFERENCES *tabela_referenciada* (*col1*,...)

[ON DELETE *ação_de_referencia*]

[ON UPDATE *ação_de_referencia*]

- ***Ação_de_referencia (Referential Action):***
 - **NO ACTION:** não faz nada. É o default, caso não seja definido um.
 - **CASCADE:** propaga a modificação para esta tabela
 - **SET NULL:** atribui nulo para a chave estrangeira

Triggers e Remoção em Cascata

Ex.

```
ALTER TABLE atividade  
    ADD FOREIGN KEY (numProj)  
    REFERENCES projeto (numProj)  
    ON DELETE CASCADE;
```

- O comando acima garante que a remoção de um projeto irá antes gerar uma remoção em cascata de todas as suas atividades
- No entanto
 - Alguns SGBDs podem não suportar ações referenciais

Triggers e Remoção em Cascata

- Veremos agora a solução baseada em trigger

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

A trigger **remove** atividades relacionadas ao projeto **antes** que o projeto seja de fato **removido**

Triggers e Remoção em Cascata

- Veremos agora a solução baseada em trigger

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


O símbolo para indicar o final de um comando SQL é trocado temporariamente para \$\$

Triggers e Remoção em Cascata

- Veremos agora a solução baseada em trigger

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

Caso contrário, o SGBD pensaria que o primeiro sinal de ponto e vírgula encontrado indicasse o final da trigger, e geraria erro

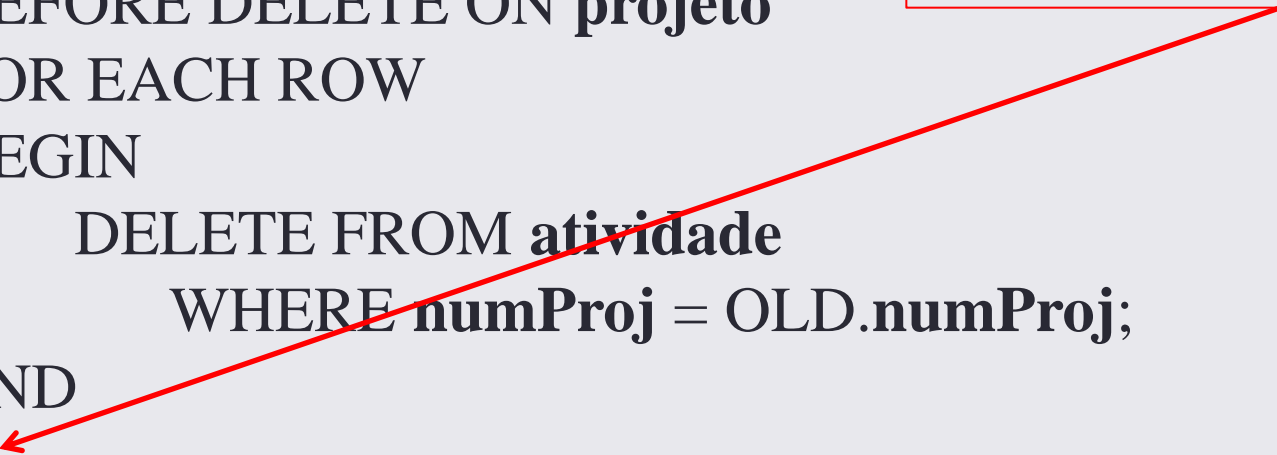


Triggers e Remoção em Cascata

- Veremos agora a solução baseada em trigger

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

Somente aqui o código da trigger é considerado encerrado

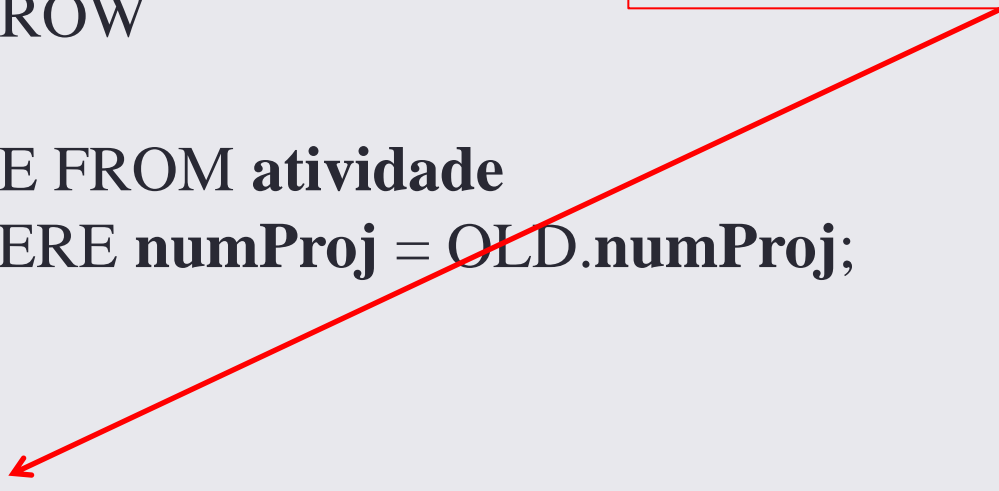


Triggers e Remoção em Cascata

- Veremos agora a solução baseada em trigger

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

Agora pode-se voltar a usar o ponto e vírgula para indicar o término de um comando SQL



Triggers e Remoção em Cascata

- Veremos agora a solução baseada em trigger

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


O **BEGIN-END** pode ser usado quando o corpo da trigger possuir mais do que um comando SQL. No caso em questão, ele não precisaria ser usado

Triggers e Remoção em Cascata

- Veremos agora a solução baseada em trigger

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

A palavra **OLD** é usada acessar os valores antigos do registro que está passando pela mudança.



Aplicações para triggers

- Triggers podem ser usadas para diversas finalidades
- Nessa aula veremos algumas delas
 - Remoção em cascata
 - **Check constraint**
 - Função de cálculo
 - Controle de especialização
 - Histórico de versões

Triggers e Check constraints

- A data final do projeto deve obrigatoriamente ser superior à data de início do projeto
- Não podem ser aceitos registros de projeto que desrespeitem essa regra
- O DBA quer que essa regra seja verificada pelo próprio SGBD

Projeto	
*numProj	int
descProj	char(30)
dataIni	datetime
dataFim	datetime

Triggers e Check constraints

- Check constraints podem ser usadas para essa finalidade

```
ALTER TABLE projeto
```

```
    ADD CONSTRAINT check_data
```

```
    CHECK (dataIni < dataFim );
```

- No entanto, o SGBD usado não suporta esse recurso
- Nesse caso, pode-se recorrer a triggers

Projeto	
*numProj	int
descProj	char(30)
dataIni	datetime
dataFim	datetime

Triggers e Check constraints

- Veremos agora a solução baseada em trigger

```
DELIMITER $$
CREATE TRIGGER insProj
BEFORE INSERT ON projeto
FOR EACH ROW
BEGIN
    IF NEW.dataIni > NEW.dataFim THEN
        SIGNAL SQLSTATE '45000' SET message_text = "Data
            final deve ser maior que a data inicial";
    END IF;
END
$$
DELIMITER ;
```

A trigger impede a execução da **inserção** de um projeto quando a data final for superior à data inicial

Triggers e Check constraints

- Veremos agora a solução baseada em trigger

```
DELIMITER $$  
CREATE TRIGGER insProj  
BEFORE INSERT ON projeto  
FOR EACH ROW  
BEGIN
```

```
    IF NEW.dataIni > NEW.dataFim THEN
```

```
        SIGNAL SQLSTATE '45000' SET message_text = "Data  
        final deve ser maior que a data inicial";
```

```
    END IF;
```

```
END
```

```
$$
```

```
DELIMITER ;
```

Pode-se usar estruturas
condicionais dentro de triggers

Triggers e Check constraints

- Veremos agora a solução baseada em trigger

```
DELIMITER $$  
CREATE TRIGGER insProj  
BEFORE INSERT ON projeto  
FOR EACH ROW  
BEGIN
```

```
    IF NEW.dataIni > NEW.dataFim THEN
```

```
        SIGNAL SQLSTATE '45000' SET message_text = "Data  
        final deve ser maior que a data inicial";
```


```
    END IF;
```

```
END
```

```
$$
```

```
DELIMITER ;
```

A palavra **NEW** é usada para acessar os valores novos do registro que está passando pela mudança.



Triggers e Check constraints

- Veremos agora a solução baseada em trigger

```
DELIMITER $$
CREATE TRIGGER insProj
BEFORE INSERT ON projeto
FOR EACH ROW
BEGIN
    IF NEW.dataIni > NEW.dataFim THEN
        SIGNAL SQLSTATE '45000' SET message_text = "Data
            final deve ser maior que a data inicial";
    END IF;
END
$$
DELIMITER ;
```


Sinalização de que um erro ocorreu e a execução do comando deve ser encerrada

Triggers e Check constraints

- Veremos agora a solução baseada em trigger

```
DELIMITER $$  
CREATE TRIGGER insProj  
BEFORE INSERT ON projeto  
FOR EACH ROW  
BEGIN  
    IF NEW.dataIni > NEW.dataFim THEN  
        SIGNAL SQLSTATE '45000' SET message_text = "Data  
        final deve ser maior que a data inicial";  
    END IF;  
END  
$$  
DELIMITER ;
```

Mensagem de erro a ser
retornada



Aplicações para triggers

- Triggers podem ser usadas para diversas finalidades
- Nessa aula veremos algumas delas
 - Remoção em cascata
 - Check constraint
 - **Função de cálculo**
 - Controle de especialização
 - Histórico de versões

Triggers e Funções de Cálculo

- A tabela Projeto possui uma coluna redundante chamada totalHoras.
- Essa coluna armazena o total de horas gastas em atividades do projeto

Projeto	
*numProj	int
descProj	char(30)
totalHoras	int

1

n

Atividade	
*numProj	int
*numAtiv	int
descAtiv	char(30)
horas	int

Triggers e Funções de Cálculo

- O DBA não quer que as aplicações que atualizem registros de atividade precisem se preocupar com a atualização respectiva da tabela projeto
- Ou seja, as atualizações em projetos deverão ser feitas pelo próprio SGBD.
- Triggers podem ser usadas para esse fim

Projeto	
*numProj	int
descProj	char(30)
totalHoras	int

1

n

Atividade	
*numProj	int
*numAtiv	int
descAtiv	char(30)
horas	int

Triggers e Funções de Cálculo

- Veremos agora a solução baseada em trigger

```
DELIMITER $$
CREATE TRIGGER insAtividade
AFTER INSERT ON atividade
FOR EACH ROW
BEGIN
    UPDATE projeto
        SET totalHoras = totalHoras + NEW.horas
        WHERE projeto.numProj = NEW.numProj;
END
$$
DELIMITER ;
```

A trigger **atualiza** o total de horas de um projeto **depois** da **inserção** de atividades

Triggers e Funções de Cálculo

- Veremos agora a solução baseada em trigger

```
DELIMITER $$
CREATE TRIGGER insAtividade
AFTER INSERT ON atividade
FOR EACH ROW
BEGIN
    UPDATE projeto
        SET totalHoras = totalHoras + NEW.horas
        WHERE projeto.numProj = NEW.numProj;
END
$$
DELIMITER ;
```

Outras duas triggers são necessárias para o controle de redundâncias adequado.
Quais seriam elas?

Aplicações para triggers

- Triggers podem ser usadas para diversas finalidades
- Nessa aula veremos algumas delas
 - Remoção em cascata
 - Check constraint
 - Função de cálculo
 - Controle de especialização
 - Histórico de versões

Triggers e Controle de Especialização

- O modelo abaixo é uma das abstrações usadas para representar especialização em bancos de dados relacionais
 - No entanto, as restrições de chave estrangeira por si só não realizam a consistência de forma completa



Triggers e Controle de Especialização

- Por exemplo, não há forma de impedir via chave estrangeira que uma sala virtual tenha o mesmo oid de uma sala física
- No entanto, triggers podem ajudar



Triggers e Controle de Especialização

- Duas triggers são necessárias
 - Uma para salaVirtual
 - Uma para salaFísica
- Em cada uma delas deve-se, antes da inserção
 - Verificar se um oid já está sendo usado
 - E gerar erro ao tentar usá-lo como oid de outra sala
 - Gerar um novo oid
 - Caso ele não esteja sendo usado ainda

Triggers e Controle de Especialização

```
CREATE TRIGGER insSalaVirtual
BEFORE INSERT ON salaVirtual
FOR EACH ROW
BEGIN
    DECLARE oid INT;
    SELECT oidSala INTO oid FROM sala
    WHERE oidSala = NEW.oidSala;
    IF oid IS NOT NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = "Old de sala já está sendo usado";
    END IF;
    INSERT INTO sala VALUES (NEW.oidSala);
END
```

Essa é a trigger para a sala virtual

Triggers e Controle de Especialização

```
CREATE TRIGGER insSalaVirtual  
BEFORE INSERT ON salaVirtual  
FOR EACH ROW  
BEGIN
```

Declarada uma variável do tipo
INT

```
    DECLARE oid INT;  
    SELECT oidSala INTO oid FROM sala  
    WHERE oidSala = NEW.oidSala;  
    IF oid IS NOT NULL THEN  
        SIGNAL SQLSTATE '45000'  
        SET MESSAGE_TEXT = "Old de sala já está sendo usado";  
    END IF;  
    INSERT INTO sala VALUES (NEW.oidSala);  
END
```

Triggers e Controle de Especialização

```
CREATE TRIGGER insSalaVirtual
BEFORE INSERT ON salaVirtual
FOR EACH ROW
BEGIN
    DECLARE oid INT;
    SELECT oidSala INTO oid FROM sala
    WHERE oidSala = NEW.oidSala;
    IF oid IS NOT NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = "Old de sala já está sendo usado";
    END IF;
    INSERT INTO sala VALUES (NEW.oidSala);
END
```

Variável recebe o retorno de uma consulta usando **SELECT INTO**

Triggers e Controle de Especialização

```
CREATE TRIGGER insSalaVirtual
BEFORE INSERT ON salaVirtual
FOR EACH ROW
BEGIN
    DECLARE oid INT;
    SELECT oidSala INTO oid FROM sala
    WHERE oidSala = NEW.oidSala;
    IF oid IS NOT NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = "Old de sala já está sendo usado";
    END IF;
    INSERT INTO sala VALUES (NEW.oidSala);
END
```

Se o valor da variável for não nulo, é porque esse oid já é usado, e um erro deve ser gerado

Triggers e Controle de Especialização

```
CREATE TRIGGER insSalaVirtual
BEFORE INSERT ON salaVirtual
FOR EACH ROW
BEGIN
    DECLARE oid INT;
    SELECT oidSala INTO oid FROM sala
    WHERE oidSala = NEW.oidSala;
    IF oid IS NOT NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = "Old de sala já está sendo usado";
    END IF;
    INSERT INTO sala VALUES (NEW.oidSala);
END
```

Caso não seja gerado erro, o oid deve ser guardado na tabela sala

Triggers e Controle de Especialização

```
CREATE TRIGGER insSalaVirtual
BEFORE INSERT ON salaFisica
FOR EACH ROW
BEGIN
    DECLARE oid INT;
    SELECT oidSala INTO oid FROM sala
    WHERE oidSala = NEW.oidSala;
    IF oid IS NOT NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = "Old de sala já está sendo usado";
    END IF;
    INSERT INTO sala VALUES (NEW.oidSala);
END
```

Essa é a trigger para a sala física, com a repetição dos mesmos comandos

Aplicações para triggers

- Triggers podem ser usadas para diversas finalidades
- Nessa aula veremos algumas delas
 - Remoção em cascata
 - Check constraint
 - Função de cálculo
 - Controle de especialização
 - Histórico de versões

Triggers e histórico de versões

- A tabela ProjetoTemporal guarda todas as versões dos registros de projeto.
- Uma nova versão é criada sempre que um registro de projeto for adicionado ou modificado

Projeto	
*numProj	int
descProj	char(30)
Custo	decimal(6)

ProjetoTemporal	
*Id	bigint
Data	datetime
Tipo	char(10)
numProj	int
descProj	char(30)
custo	decimal(6)

Triggers e histórico de versões

- Além das colunas da tabela projeto, a tabela temporal possui
 - **Id**: identificador auto incrementável
 - **Data**: quando a modificação ocorreu
 - **Tipo**: se foi ATUALIZAÇÃO ou INSERÇÃO

Projeto	
*numProj	int
descProj	char(30)
Custo	decimal(6)

ProjetoTemporal	
* Id	bigint
Data	datetime
Tipo	char(10)
numProj	int
descProj	char(30)
custo	decimal(6)

Triggers e histórico de versões

- Com a tabela temporal, consultas interessantes podem ser respondidas:
 - Ex. Qual projeto teve o custo modificado mais vezes
 - Ex. Qual a tendência de variação no custo ao longo do tempo

Projeto	
*numProj	int
descProj	char(30)
Custo	decimal(6)

ProjetoTemporal	
*Id	bigint
Data	datetime
Tipo	char(10)
numProj	int
descProj	char(30)
custo	decimal(6)

Triggers e histórico de versões

- O DBA quer que a tabela temporal seja atualizada pelo próprio SGBD
- Para isso, triggers podem ajudar

Projeto	
*numProj	int
descProj	char(30)
Custo	decimal(6)

ProjetoTemporal	
*Id	bigint
Data	datetime
Tipo	char(10)
numProj	int
descProj	char(30)
custo	decimal(6)

Triggers e histórico de versões

```
DELIMITER $$  
CREATE TRIGGER insProjeto  
AFTER INSERT ON projeto  
FOR EACH ROW  
BEGIN  
    INSERT INTO projetoTemporal  
        (data, tipo, numProj, descProj, custo) VALUES  
        (now(), 'insert', NEW.numProj, NEW.descProj, NEW.custo);  
END  
$$
```

Nova versão temporal de projeto
é criada quando um projeto for
inserido

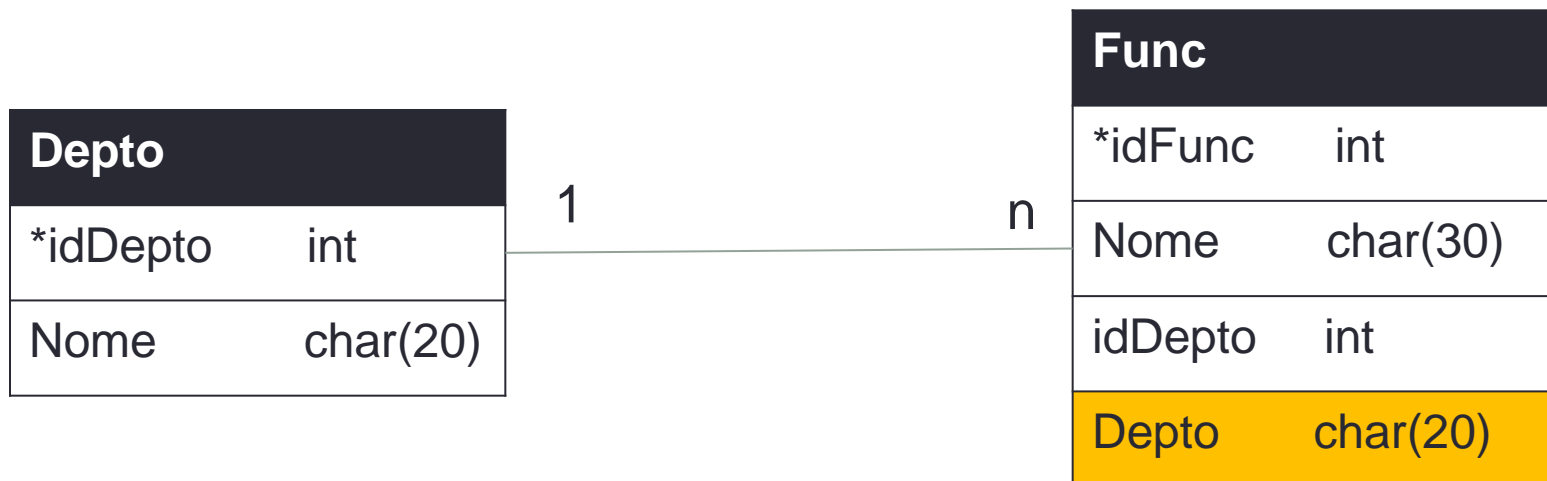
Triggers e histórico de versões

```
DELIMITER $$  
CREATE TRIGGER uptProjeto  
AFTER UPDATE ON projeto  
FOR EACH ROW  
BEGIN  
    INSERT INTO projetoTemporal  
        (data, tipo, numProj, descProj, custo) VALUES  
        (now(), 'update', NEW.numProj, NEW.descProj, NEW.custo);  
END  
$$
```

Nova versão temporal de projeto
é criada quando um projeto for
atualizado

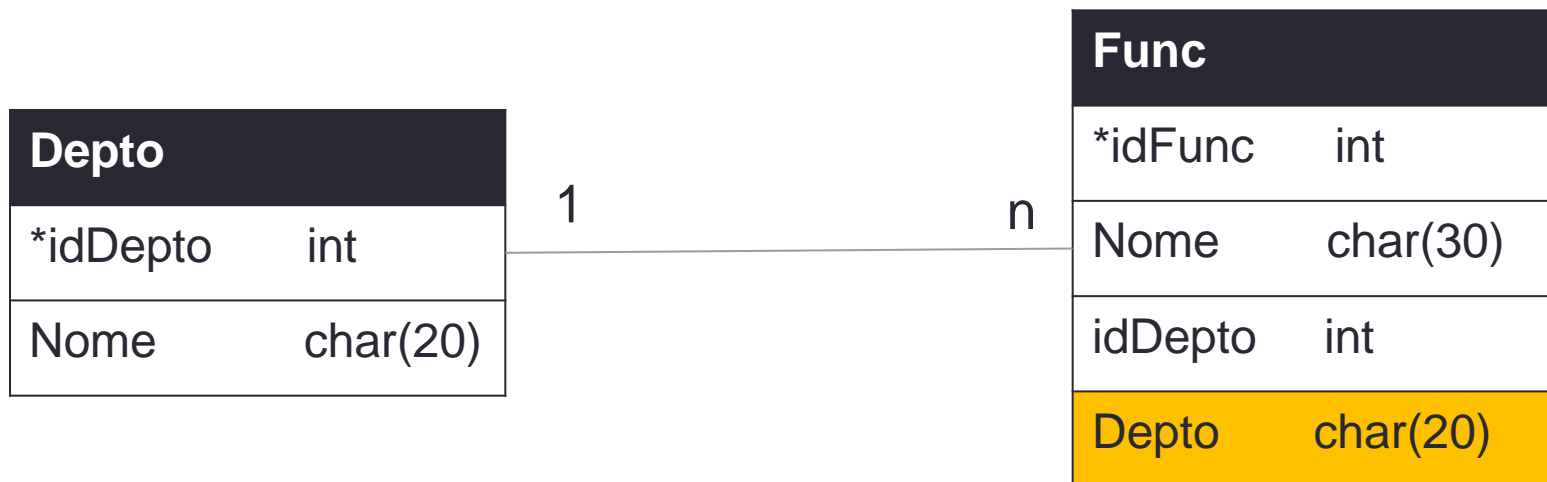
Exercício - 1

- A coluna Depto é redundante
- Seu valor deve ser consistente com o valor da coluna nome existente na tabela depto.
- O DBA quer que essa consistência seja garantida através de triggers que alimentem a coluna redundante



Exercício - 1

- Três triggers são necessárias
 - Uma para quando um novo funcionário for inserido
 - Uma para quando o nome do departamento mudar
 - Uma para quando um funcionário mudar de departamento
- Implemente as duas primeiras triggers



Exercício - 1

- Teste as triggers com os seguintes comandos:

-- insercao em depto
insert into depto values (1,'depto 1');
insert into depto values (2,'depto 2');
insert into depto values (3,'depto 3');

Exercício - 1

- Teste as triggers com os seguintes comandos:

-- insercao em funcionario

```
insert into func(idFunc, nome, idDepto) values (1,'joao', 2);
```

```
insert into func(idFunc, nome, idDepto) values (10,'ana', 3);
```

-- testando trigger de insercao de funcionario

```
select nome, depto from func;
```

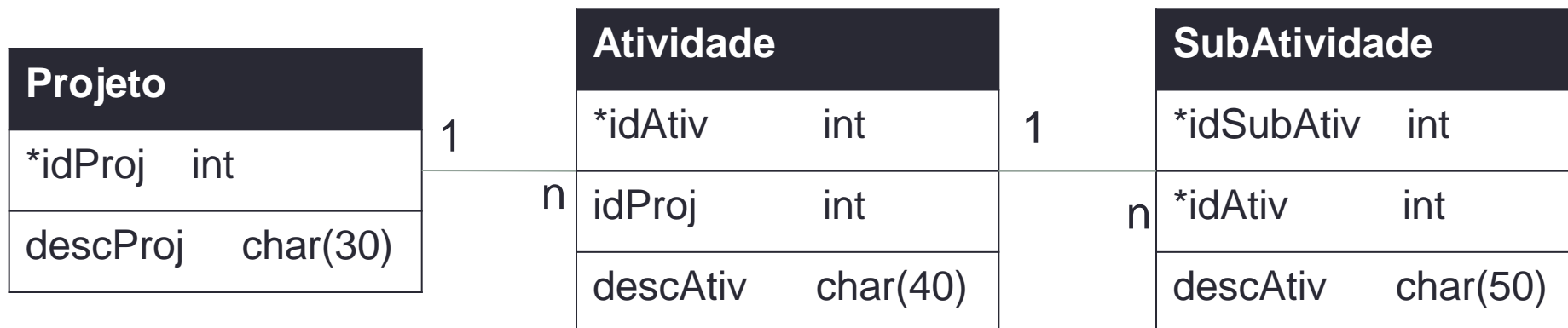
```
update depto set nome = 'departamento 2' where nome = 'depto 2';
```

-- testando trigger de atualizacao de funcionario

```
select nome, depto from func;
```

Exercício - 2

- O DBA deseja que a remoção de um projeto leve à remoção em cascata de atividades e subatividades
- Suponha que o SGBD não suporte o recurso DELETE ON CASCADE
- Dessa forma, é necessário recorrer a triggers
 - Implemente as triggers que resolvem esse problema



Exercício - 3

- O DBA percebeu que a tabela projetoTemporal desperdiça muito espaço
- Muitas vezes uma única coluna tem seu valor mudado entre uma versão de projeto e a versão seguinte
- No entanto, os valores de todos os campos são replicados

Projeto	
*numProj	int
descProj	char(30)
Custo	decimal(6)

ProjetoTemporal	
*Id	bigint
Data	datetime
Tipo	char(10)
numProj	int
descProj	char(30)
custo	decimal(6)

Exercício - 3

- Para minimizar o desperdício, o DBA resolveu ajustar a trigger que gera registros em ProjetoTemporal quando um projeto é atualizado
- Com o ajuste, só serão gravados valores em colunas que sofreram mudanças
- As demais permanecem com valor NULL.
- Implemente esse ajuste

Projeto	
*numProj	int
descProj	char(30)
Custo	decimal(6)

ProjetoTemporal	
*Id	bigint
Data	datetime
Tipo	char(10)
numProj	int
descProj	char(30)
custo	decimal(6)

Exercício - 3

- Teste a trigger executando os seguintes comandos:

-- insercao

```
insert into projeto values (1, 'ACME', 12000);
```

```
insert into projeto values (2, 'ZYX', 1000);
```

-- atualizacao

```
update projeto set custo = 10000 where numProj = 1;
```

```
update projeto set custo = 100000 where numProj = 1;
```

-- consulta

```
select numProj, data, tipo, descProj, custo from projetoTemporal  
order by numProj, data;
```

Atividade Individual

- Um funcionário pode liderar um número máximo de projetos
 - O máximo é específico de cada funcionário (coluna limiteProjetos)
- Crie uma trigger que impeça a inserção de um projeto se isso levar o funcionário líder a ultrapassar o seu limite

