

Banco de Dados 2

PROCEDURES

Procedimentos (Procedures)

Conjunto de comandos SQL que podem ser armazenados no servidor

Procedimentos (Procedures)

Também chamados de procedimentos armazenados (**stored procedures**), representam porções de código SQL e não SQL que ficam armazenados de forma compilada no catálogo do SGBD

Procedimentos (Procedures)

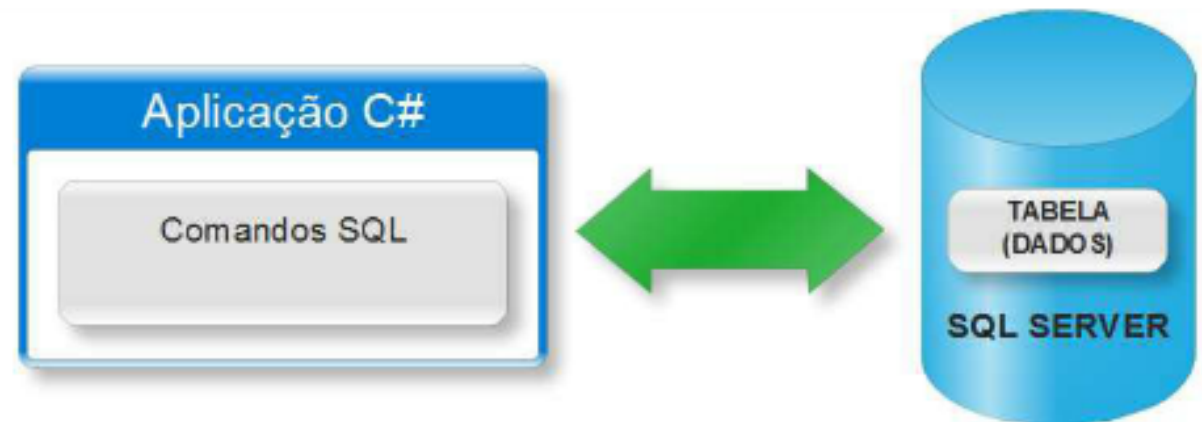
São ativados explicitamente por aplicações, ***triggers*** ou outras rotinas.

Como nas linguagens tradicionais de programação, um procedimento realiza um processamento qualquer e não devolve valor algum em seu final.

Procedimentos (Procedures)

Modelos de acesso

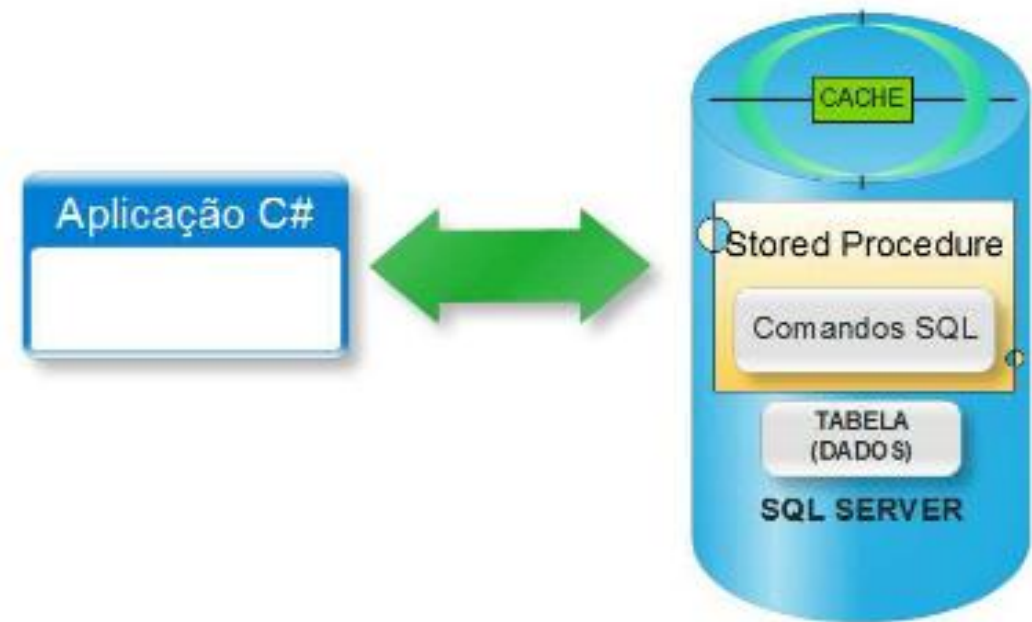
Modelo de Acesso ao Banco de Dados sem utilização de Stored Procedures



Procedimentos (Procedures)

Modelos de acesso

Modelo de Acesso ao Banco de Dados utilizando Stored Procedures



Procedures – Criação (sem parâmetros)

```
DELIMITER $$
```

```
CREATE PROCEDURE <nome procedure>()  
BEGIN  
    SELECT * FROM empregado ORDER BY nome DESC;  
    SELECT nome FROM produtos;  
END $$
```

```
DELIMITER ;
```

-- Obs.: É importante definir um finalizador diferente de ; em um ***procedure***

Procedures – Criação (com parâmetros)

```
DELIMITER $$
```

```
CREATE PROCEDURE <nome procedure> (IN nomevariavel INT)
AS
BEGIN
    SELECT * FROM empregado ORDER BY nome DESC;
    SELECT nome FROM produtos;
END $$
```

```
DELIMITER ;
```

Procedimentos - Parâmetros

No MySQL, um parâmetro adquire um dos seguintes modos:

IN, OUT e INOUT

IN - pode ser passado, mas qualquer alteração dentro do stored procedure não altera o parâmetro.

OUT - o stored procedure pode alterar o parâmetro e devolve-lo ao programa de chamada.

INOUT - é a combinação do modo **IN** e **OUT**, você pode passar parâmetros para o stored procedure e recuperá-los com um novo valor a partir do programa de chamada.

Procedimentos - Execução

Execução **sem** parametros

```
CALL <nome procedure>;
```

Execução **com** parametros

```
CALL <nome procedure>(<par 1>, <par 2>);
```

Procedimentos - Alteração

```
ALTER PROCEDURE <nome procedure>  
AS  
BEGIN  
    SELECT * FROM empregado;  
    ORDER BY cpf ASC;  
END
```

Procedimentos - Exclusão

```
DROP PROCEDURE <schema>.<nome proceudre>;
```

Algumas rotinas precisam ser executadas
antes ou **depois** de um determinado
evento no banco de dados

Regras de negócios complexas precisam ser **verificadas** a cada operação no banco de dados

TRIGGERS

Triggers – O que são?

Conjunto de operações executadas **automaticamente** quando uma **alteração** é feita em uma tabela

Triggers – O que são?

Objeto associado a uma tabela

Vários *triggers* podem ser associados a um mesmo BD

Obs.: É importante definir um finalizador diferente de ; em um *trigger*

Triggers – Execução

É disparada (pelo SGBD) **antes** ou **depois** de um **INSERT**, **UPDATE** ou **DELETE**

Pode haver até **seis** triggers por tabela

Triggers – Utilização

Atender regras de negócios

Verificar a **integridade** dos dados, pois é possível fazer uma verificação antes da inserção do registro

Contornar **erros** na regra de negócio do sistema no banco de dados

Auditar as **mudanças** nas tabelas.

Comando CREATE TRIGGER

Um trigger típico tem três componentes:

Evento(s)

Condição

Ação

Triggers em SQL

Comando CREATE TRIGGER

Um trigger típico tem três partes:

Evento(s)

Condição

Ação

INSERT

UPDATE

DELETE

Triggers em SQL

Comando CREATE TRIGGER

Um trigger típico tem três partes:

Evento(s)

Condição

Ação

NÃO

disponível no

MySQL

Triggers – Acesso a outras tabelas

TRIGGER é sempre associado a uma tabela, porém os seus comandos **podem acessar dados de outras tabelas**

Triggers – Eventos em cascata

Pode-se usar TRIGGERS para exclusão e atualização **em cascata**

Se um comando executado **violar** uma **CONSTRAINT**, a TRIGGER **não irá disparar**

Triggers – Restrições

Não são permitidos os seguintes comandos:

ALTER DATABASE, CREATE DATABASE, DROP
DATABASE, LOAD DATABASE, LOAD LOG,
RESTORE DATABASE, RESTORE LOG,
RECONFIGURE

Triggers – Palavras chave OLD e NEW

Triggers tem acesso a duas tabelas em memória referenciadas pelas palavra chave **OLD** e **NEW**

Triggers – Palavras chave OLD e NEW

:OLD e **:NEW** (No MySQL se omite os dois pontos)

Quando a operação for de **INSERT** temos acesso apenas a tabela **:NEW**

Quando a operação for de **UPDATE** temos acesso as tabela **:OLD**, **:NEW**

Quando a operação for de **DELETE** temos acesso apenas a tabela **:OLD**

Triggers – no MySQL

Sugestão de padrão de nomenclatura

“trg”+ nome da tabela + id do evento

AI: **A**fter **I**nsert (Após a Inserção)

AU: **A**fter **U**pdate (Após a Atualização)

AD: **A**fter **D**elete (Após a Exclusão)

BI: **B**efore **I**nsert (Antes da Inserção)

BU: **B**efore **U**pdate (Antes da Atualização)

BD: **B**efore **D**elete (Antes da Exclusão)

Triggers – Habilitando e Desabilitando

Para desabilitar temporariamente um trigger:

```
ALTER TABLE Nome_da_Tabela  
DISABLE TRIGGER Nome_da_Trigger
```

Para habilitar novamente uma trigger:

```
ALTER TABLE Nome_da_Tabela  
ENABLE TRIGGER Nome_da_Trigger
```

Controlar estoque no MySQL usando triggers

Controle de estoque de produtos é uma funcionalidade básica em sistemas desenvolvidos para o comércio e empresas em geral.

Controlar estoque no MySQL usando triggers

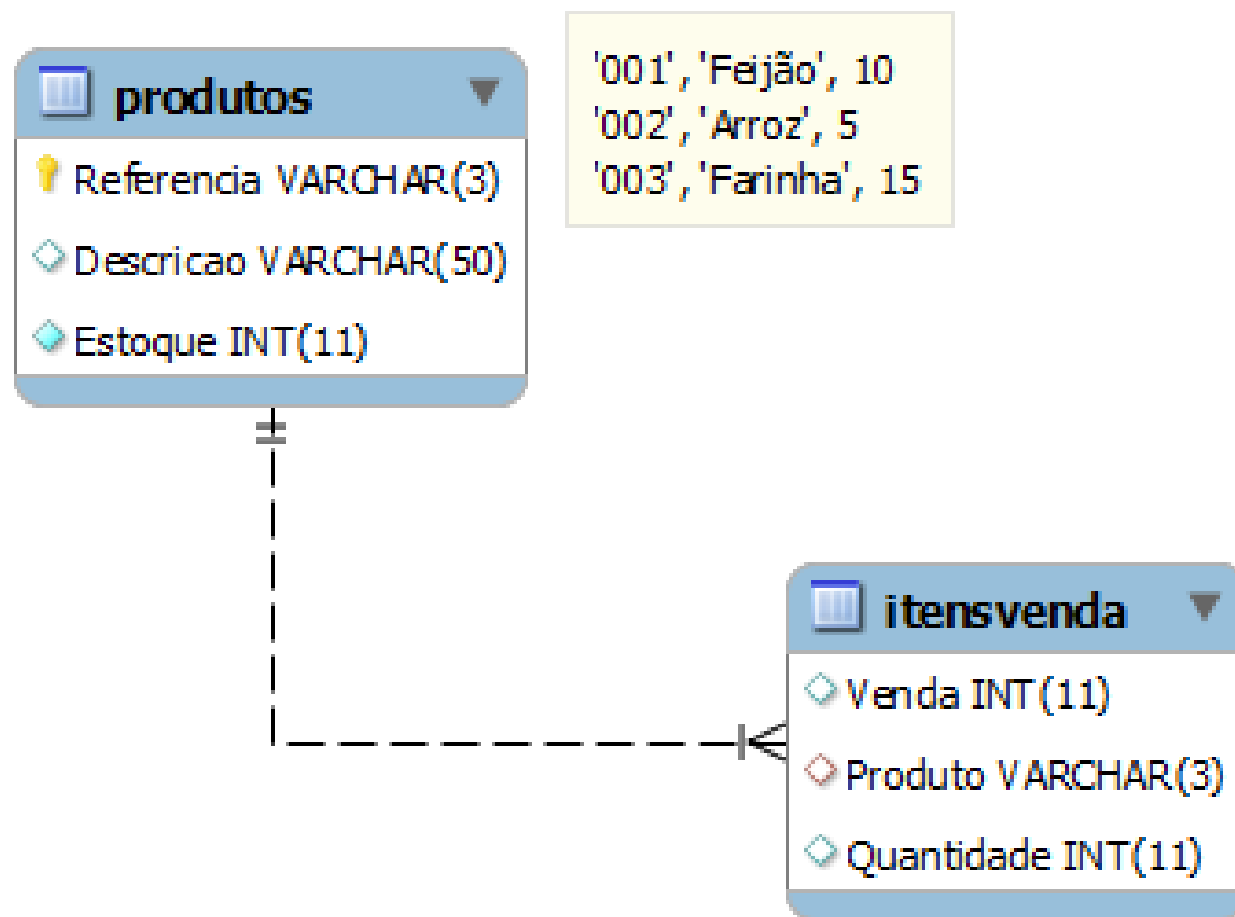
Permitir aos usuários consultar, em tempo real, a disponibilidade de um determinado produto.

Exemplo

Um mercado que, ao realizar vendas, precisa que o estoque dos produtos seja automaticamente reduzido.

A devolução do estoque deve também ser automática no caso de remoção de produtos da venda.

Exemplo



Exemplo

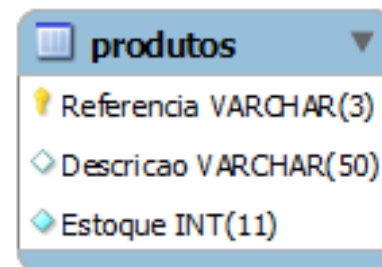
```
DROP SCHEMA IF EXISTS aula04;  
CREATE SCHEMA IF NOT EXISTS aula04;  
USE aula04;
```

```
-- produto
```

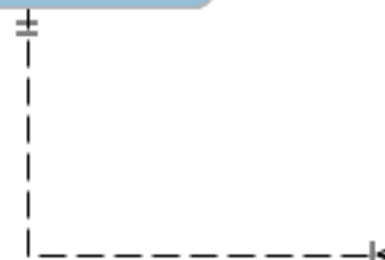
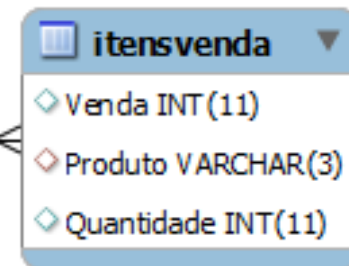
```
DROP TABLE IF EXISTS produtos;
```

```
CREATE TABLE IF NOT EXISTS produtos (  
  referencia VARCHAR(3) NOT NULL,  
  descricao VARCHAR(50) NULL DEFAULT NULL,  
  estoque INT(11) NOT NULL DEFAULT 0,  
  PRIMARY KEY (referencia)  
);
```

```
INSERT INTO produtos (referencia, descricao, estoque) VALUES ('001', 'Feijão', 10);  
INSERT INTO produtos (referencia, descricao, estoque) VALUES ('002', 'Arroz', 5);  
INSERT INTO produtos (referencia, descricao, estoque) VALUES ('003', 'Farinha', 15);
```



'001', 'Feijão', 10
'002', 'Arroz', 5
'003', 'Farinha', 15



Exemplo

```
-- itensVenda
```

```
DROP TABLE IF EXISTS itensVenda;
```

```
CREATE TABLE IF NOT EXISTS itensVenda (
```

```
  Venda      INT(11) NULL DEFAULT NULL,
```

```
  produto    VARCHAR(3) NULL DEFAULT NULL,
```

```
  quantidade INT(11) NULL DEFAULT NULL,
```

```
  CONSTRAINT fk_itensVenda_produto
```

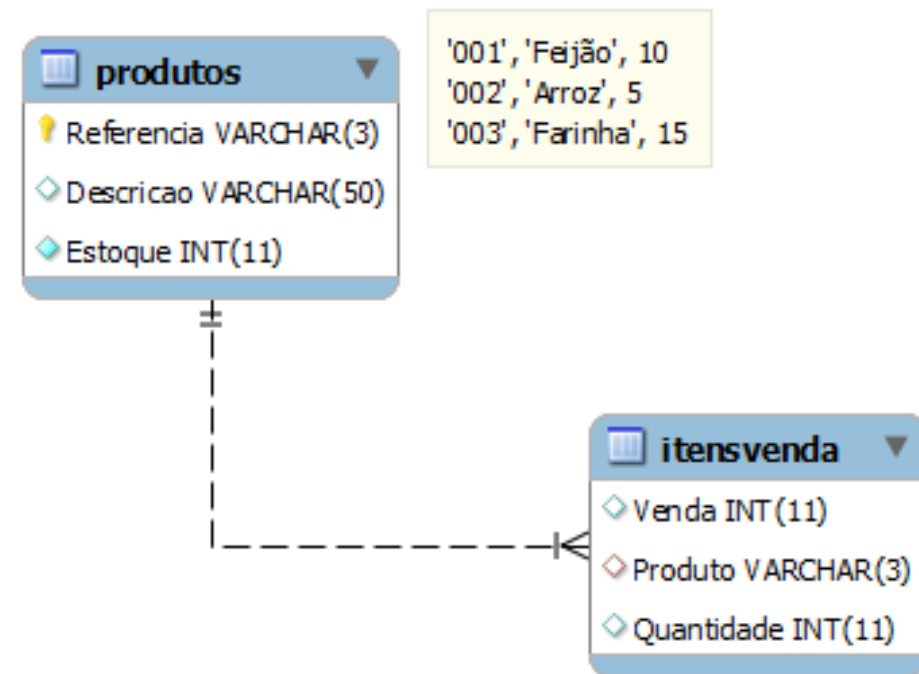
```
    FOREIGN KEY (produto) REFERENCES produtos (referencia) ON DELETE NO ACTION ON UPDATE NO  
ACTION  
);
```

produtos
Referencia VARCHAR(3)
Descricao VARCHAR(50)
Estoque INT(11)

'001', 'Feijão', 10
'002', 'Arroz', 5
'003', 'Farinha', 15

itensvenda
Venda INT(11)
Produto VARCHAR(3)
Quantidade INT(11)

Exemplo



Ao inserir e remover registro da tabela **itensVenda**, o estoque do produto referenciado deve ser alterado na tabela produto.

Para isso, serão criados dois triggers: um AFTER INSERT para dar baixa no estoque e um AFTER DELETE para fazer a devolução da quantidade do produto.

Exemplo

Observação:

Como usaremos instruções que requerem ponto e vírgula no final, alteraremos o delimitador de instruções para \$\$ e depois de criar os triggers, voltaremos para o padrão.


Essa alteração não está diretamente ligada aos triggers.

Apenas para registrar e conferir, a imagem a seguir mostra um SELECT feito sobre a tabela produto após a inserção dos registros de exemplo.

Referencia	Descricao	Estoque
001	Feijão	10
002	Arroz	5
003	Farinha	15

Exemplo

Criação dos gatilhos(triggers) para executar as ações já discutidas.



```
-- itensVenda
DELIMITER $$

CREATE TRIGGER trg_itensvenda_AI AFTER INSERT
ON itensVenda
FOR EACH ROW
BEGIN
    UPDATE produtos SET estoque = estoque - NEW.quantidade WHERE referencia = NEW.produto;
END$$

DELIMITER ;
```

Foi utilizado o registro NEW para obter as informações da linha que está sendo inserida na tabela.

Exemplo

Criação dos gatilhos(triggers) para executar as ações já discutidas.



```
DELIMITER $$

CREATE TRIGGER trg_itensvenda_AD AFTER DELETE
ON itensVenda
FOR EACH ROW
BEGIN
    UPDATE produtos SET estoque = estoque + OLD.quantidade WHERE referencia = OLD.produto;
END$$


DELIMITER ;
```

Neste gatilho, se obtém os dados que estão sendo apagados da tabela através do registro OLD.

Exemplo

Tendo criado os triggers, podemos testá-los inserindo dados na tabela **itensVenda**.

Nesse caso, vamos simular uma venda de número 1 que contém três unidades do produto 001, uma unidade do produto 002 e cinco unidades do produto 003.



```
INSERT INTO itensVenda VALUES (1, '001',3);  
INSERT INTO itensVenda VALUES (1, '002',1);  
INSERT INTO itensVenda VALUES (1, '003',5);
```

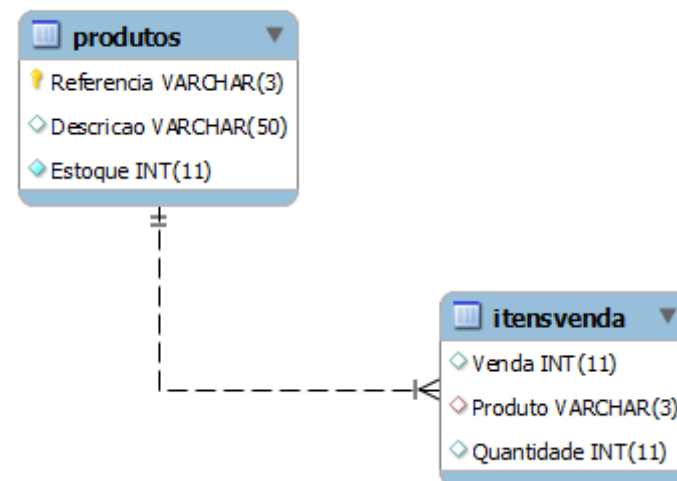
Exemplo

Nota-se que o estoque dos produtos foi corretamente reduzido, de acordo com as quantidades “vendidas”.

Referencia	Descricao	Estoque
001	Feijão	10
002	Aroz	5
003	Farinha	15



Referencia	Descricao	Estoque
001	Feijão	7
002	Aroz	4
003	Farinha	10



Exemplo

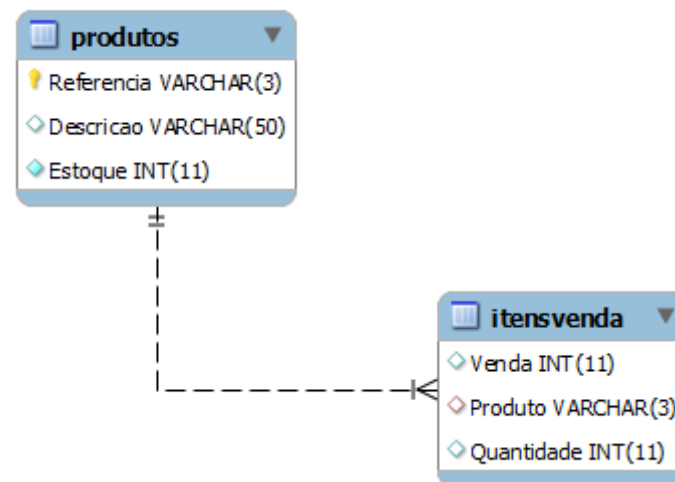
Agora para testar o trigger da exclusão, removeremos o produto 001 dos itens vendidos. Com isso, o seu estoque deve ser alterado para o valor inicial, ou seja, 10.

```
DELETE FROM itensVenda WHERE venda = 1 AND produto = '001';
```

Referencia	Descricao	Estoque
001	Feijão	7
002	Arroz	4
003	Farinha	10



Referencia	Descricao	Estoque
001	Feijão	10
002	Arroz	4
003	Farinha	10



Controlar estoque no MySQL usando triggers

O estoque pode ser controlado através de várias técnicas.
(escolha a sua)

MySQLWorkbench

No MySQLWorkbench é possível visualizar os gatilhos relacionados a uma tabela através do Object browser, como mostra a figura a seguir

