Views / Funções / Procedures / Triggers

Banco de Dados II

Banco de Dados II

Views / Funções / Procedures / Triggers

VIEWS – Criando Tabelas Virtuais (Visões)

Uma **View** (Exibição / Visão) é uma tabela virtual (estrutura de dados) baseada no conjunto de resultados de uma consulta SQL, criada a partir de um conjunto de tabelas (ou outras views) presentes no banco, que servem com *tabelas-base*.

Contém linhas e colunas como uma tabela real, e pode receber comandos como declarações JOIN, WHERE e funções como uma tabela normal. A view fica armazenada no banco de dados.

Mostra sempre resultados de dados atualizados, pois o motor do banco de dados recria os dados toda vez que um usuário consulta a visão.

Banco de Dados II

Views / Funções / Procedures / Triggers

Aplicações das Views

Usamos views para propósitos diversos:

- Para simplificar o acesso a dados que estão armazenados em múltiplas tabelas relacionadas

- Implementar segurança nos dados de uma tabela, por exemplo criando uma visão que limite os dados que podem ser acessados, por meio de uma cláusula WHERE

- Prover isolamento de uma aplicação da estrutura específica de tabelas do banco acessado.

Banco de Dados II

Views / Funções / Procedures / Triggers

Como criar Views no MySQL

Podemos criar uma visão no MySQL usando a sintaxe abaixo:

Sintaxe:

```
CREATE (OR REPLACE) VIEW Nome_visão
AS SELECT colunas
FROM tabela
WHERE condições;
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Como criar Views no MySQL

Podemos criar uma visão no MySQL usando a sintaxe abaixo:

Sintaxe:

```
CREATE OR REPLACE VIEW Nome_visão
AS SELECT colunas
FROM tabela
WHERE condições;
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Como criar Views no MySQL

Vamos utilizar a view criada, realizando uma consulta nela como se estivéssemos consultando uma tabela real do banco de dados::

```
SELECT Livro, Autor
FROM vw_LivrosAutores
ORDER BY Autor;
```

Livro	Autor
SSH, the Secure Shell	Daniel
Using Samba	Gerald
Fedora and Red Hat Linux	Mark
Linux Command Line and Shell Scripting	Richard
Windows Server 2012 Inside Out	William
Microsoft Exchange Server 2010	William

Veja como ficou mais simples retornar os nomes dos livros e dos autores agora – o código fica muito mais limpo e simples, facilitando por exemplo sua incorporação em uma aplicação que acesse o banco de dados do MySQL.

Banco de Dados II

Views / Funções / Procedures / Triggers

Como alterar uma View

Podemos alterar uma view, para que ela funcione de forma diferente, usando o comando **ALTER VIEW**. Vamos alterar a view vw_LivrosAutores para incluir também os preços dos livros. Veja o código abaixo:

```
ALTER  VIEW vw_LivrosAutores AS
SELECT tbl_Livro.Nome_Livro AS Livro,
tbl_autores.Nome_Autor AS Autor, Preco_Livro AS
Valor
FROM tbl_Livro
INNER JOIN tbl_autores
ON tbl_Livro.ID_Autor = tbl_autores.ID_Autor;
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Como alterar uma View

Podemos testá-la com uma consulta:

```
SELECT * FROM vw_LivrosAutores;
```

Livro	Autor	Valor
Linux Command Line and Shell Scripting	Richard	68.35
SSH, the Secure Shell	Daniel	58.30
Using Samba	Gerald	61.45
Fedora and Red Hat Linux	Mark	62.24
Windows Server 2012 Inside Out	William	66.80
Microsoft Exchange Server 2010	William	45.30

Banco de Dados II

Views / Funções / Procedures / Triggers

Como visualizar as views existentes nos bancos de dados

Podemos consultar as views existentes em um banco de dados com o comando a seguir:

```
SHOW FULL TABLES IN nome_banco
WHERE TABLE_TYPE LIKE 'VIEW';
```

Por exemplo, para ver todas as visões criadas em um banco de nome db_MeusLivros:

```
SHOW FULL TABLES IN db_MeusLivros
WHERE TABLE_TYPE LIKE 'VIEW';
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Como visualizar as views existentes nos bancos de dados

Podemos consultar as views existentes em um banco de dados com o comando a seguir:

```
SHOW FULL TABLES IN nome_banco
WHERE TABLE_TYPE LIKE 'VIEW';
```

Por exemplo, para ver todas as visões criadas em um banco de nome db_MeusLivros:

```
SHOW FULL TABLES IN db_MeusLivros
WHERE TABLE_TYPE LIKE 'VIEW';
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Como visualizar as views existentes nos bancos de dados

Caso o banco a ser verificado seja o selecionado atualmente, não é necessário informar seu nome ao consultar as views, como segue:

```
SHOW FULL TABLES
WHERE TABLE_TYPE LIKE 'VIEW';
```

Finalmente, podemos ver todas as views de todos os bancos de dados existentes na instância do MySQL com o comando a seguir:

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM information_schema.TABLES
WHERE TABLE_TYPE LIKE 'VIEW';
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Como excluir views

Se precisarmos excluir uma view do MySQL, basta usar o comando DROP **VIEW** seguido do nome da View. Vamos excluir a view vw_LivrosAutores:

```
DROP VIEW vw_LivrosAutores;
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Rotinas Armazenadas – Funções (CREATE FUNCTION)

Vamos dar início ao estudo de Rotinas Armazenadas em MySQL. Uma Rotina Armazenada é um subprograma que pode ser criado para efetuar tarefas específicas nas tabelas do banco de dados, usando comandos da linguagem SQL e Lógica de Programação.

Banco de Dados II

Views / Funções / Procedures / Triggers

Funções e Procedimentos

São dois tipos de rotinas armazenadas, parte da especificação SQL. As Funções e Procedimentos são um pouco similares, porém possuem aplicações diferentes.

São invocadas de formas diferentes também (CALL x declaração). Agora vamos falar especificamente sobre Funções, e a seguir, os Procedimentos Armazenados (Stored Procedures).

Banco de Dados II

Views / Funções / Procedures / Triggers

Funções

Uma função é usada para gerar um valor que pode ser usado em uma expressão. Esse valor é geralmente baseado em um ou mais parâmetros fornecidos à função. As funções são executadas geralmente como parte de uma expressão.

O MySQL possui diversas funções internas que o desenvolvedor pode utilizar, e também permite que criemos nossas próprias funções, e é isso que mostraremos como fazer agora.

Banco de Dados II

Views / Funções / Procedures / Triggers

Funções

Sintaxe para criação de uma Função

```
CREATE FUNCTION nome_função (parâmetros)
RETURNS tipo_dados
BEGIN
   /* bloco de códigos da função */
   RETURN <retorno>
END
código_da_função;
```

Como chamar uma função

```
SELECT
   nome_função(parâmetros)
FROM
   nome_função(parâmetros)
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Funções

Criando uma função que recebe a data no formato (AAAA-MM-DD) e convertendo para (DD-MM-AAAA)

```
DELIMITER $$
CREATE FUNCTION FUNC_DATA_PTBR(P_DATA DATETIME)
RETURNS CHAR(10)
DETERMINISTIC
/* a função é determinística quando ela não afeta dados do bd, portanto é considerada segura */
COMMENT 'Function serve para retornar datas no formato dd/mm/yyyy'
BEGIN
    RETURN DATE_FORMAT(P_DATA, '%D/%M/%Y');
END
$$
DELIMITER ;
```

Obs.: A instrução DELIMITER altera o delimitador que é a instrução final de cada linha de comando de ; para $$ (recomendável) e no fim da função ele retorna para ;

Banco de Dados II

Views / Funções / Procedures / Triggers

Funções

Chamando a função criada

```
DELIMITER $$
CREATE FUNCTION FUNC_DATA_PTBR(P_DATA DATETIME)
RETURNS CHAR(10)
DETERMINISTIC
/* a função é determinística quando ela não afeta dados do bd, portanto é considerada segura */
COMMENT 'Function serve para retornar datas no formato dd/mm/yyyy'
BEGIN
    RETURN DATE_FORMAT(P_DATA, '%D/%M/%Y');
END
$$
DELIMITER ;
```

Obs.: A instrução DELIMITER altera o delimitador que é a instrução final de cada linha de comando de ";" para "$$" (recomendável) e no fim da função ele retorna para ;
DATE FORMAT – Altera o formato da data

Banco de Dados II

Views / Funções / Procedures / Triggers

Excluindo uma função

Para excluir uma função usamos o comando **DROP FUNCTION**, de acordo com a sintaxe a seguir:

```
DROP FUNCTION nome_função
```

Exemplo

```
DROP FUNCTION FUNC_DATA_PTBR
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Procedures

Um procedimento armazenado (**Stored Procedure**, em inglês) é uma sub-rotina disponível para aplicações que acessam sistemas de bancos de dados relacionais.

Podem ser usados para validação de dados, controle de acesso, execução de declarações SQL complexas e muitas outras situações.

Banco de Dados II

Views / Funções / **Procedures** / Triggers

Criando uma Procedure

Sintaxe para criação de stored procedures no MySQL

```
DELIMITER $$
CREATE PROCEDURE nome_procedimento (parâmetros)
BEGIN
   /*CORPO DO PROCEDIMENTO*/
END $$
DELIMITER ;
```

Onde consta “nome_procedimento”, deve-se informar o nome que identificará o procedimento armazenado. Este nome segue as mesmas regras para definição de variáveis, não podendo iniciar com número ou caracteres especiais (exceto o underline “_”).

Banco de Dados II

Views / Funções / Procedures / Triggers

Criando uma Procedure

Os “parâmetros” são opcionais e, caso não sejam necessários, devem permanecer apenas os parênteses vazios na declaração do procedure. Para que um procedimento receba parâmetros, é necessário seguir certa sintaxe (dentro dos parênteses), apresentada abaixo (Listagem 2).

```
(MODO nome TIPO, MODO nome TIPO, MODO nome TIPO
```

O “nome” dos parâmetros também segue as mesmas regras de definição de variáveis.
O “TIPO” nada mais é que do tipo de dado do parâmetro (int, varchar, decimal, etc).
O “MODO” indica a forma como o parâmetro será tratado no procedimento, se será apenas um dado de entrada, apenas de saída ou se terá ambas as funções. Os valores possíveis para o modo são:

Banco de Dados II

Views / Funções / Procedures / Triggers

Criando uma Procedure

Os “parâmetros” são opcionais e, caso não sejam necessários, devem permanecer apenas os parênteses vazios na declaração do procedure. Para que um procedimento receba parâmetros, é necessário seguir certa sintaxe (dentro dos parênteses), apresentada abaixo:

Sintaxe de declaração de parâmetros em stored procedures

```
(MODO nome TIPO, MODO nome TIPO, MODO nome TIPO
```

O “nome” dos parâmetros também segue as mesmas regras de definição de variáveis.
O “TIPO” nada mais é que do tipo de dado do parâmetro (int, varchar, decimal, etc).

Banco de Dados II

Views / Funções / Procedures / Triggers

Criando uma Procedure

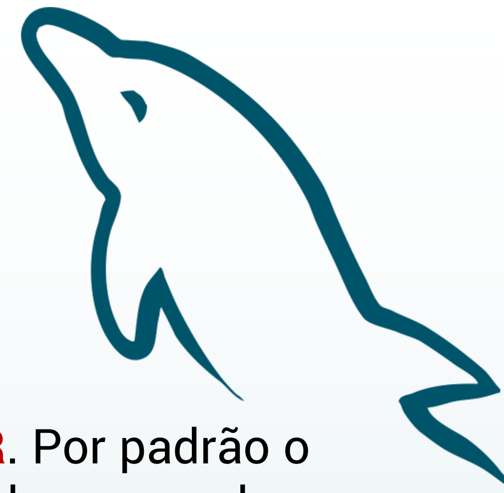
O "MODO" indica a forma como o parâmetro será tratado no procedimento, se será apenas um dado de entrada, apenas de saída ou se terá ambas as funções. Os valores possíveis para o modo são:

- **IN**: indica que o parâmetro é apenas para entrada/recebimento de dados, não podendo ser usado para retorno;

- **OUT**: usado para parâmetros de saída. Para esse tipo não pode ser informado um valor direto (como 'teste', 1 ou 2.3), deve ser passada uma variável "por referência";

- **INOUT**: como é possível imaginar, este tipo de parâmetro pode ser usado para os dois fins (entrada e saída de dados). Nesse caso também deve ser informada uma variável e não um valor direto.

Banco de Dados II



Views / Funções / Procedures / Triggers

Criando uma Procedure

Outro ponto que merece destaque é o uso do comando **DELIMITER**. Por padrão o MySQL utiliza o sinal de ponto e vírgula como delimitador de comandos, separando as instruções a serem executadas. No entanto, dentro do corpo do stored procedure será necessário separar algumas instruções internamente utilizando esse mesmo sinal, por isso é preciso inicialmente alterar o delimitador padrão do MySQL (neste caso, para $$) e ao fim da criação do procedimento, restaurar seu valor padrão.

Tendo criado o procedure, chamá-lo é bastante simples. Para isso fazemos uso da palavra reservada **CALL**, como mostra o código abaixo.

Sintaxe para chamar um stored procedure

```
CALL nome_procedimento(parâmetros);
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Criando uma Procedure

Exemplos

Usando parâmetro de entrada

```
DELIMITER $$

CREATE PROCEDURE Selecionar_Produtos(IN
quantidade INT)
BEGIN
    SELECT * FROM PRODUTOS
    LIMIT quantidade$$
END $$
DELIMITER;
```

Esse procedimento tem por função fazer um select na tabela PRODUTOS, limitando a quantidade de registros pela quantidade recebida como parâmetro.

Banco de Dados II

Views / Funções / Procedures / Triggers

Criando uma Procedure

Exemplos

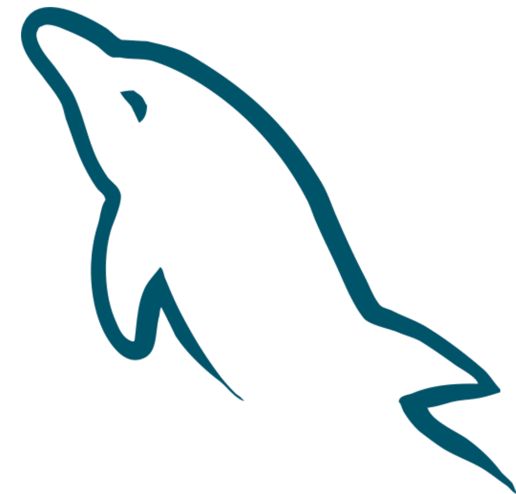
Assim, caso desejássemos selecionar dois registros dessa tabela, poderíamos usar o procedure como mostra o comando abaixo:

Chamando procedure com parâmetro de entrada

```
CALL Selecionar_Produtos(2);
```

O próximo código mostra um exemplo de recebimento e retorno de parâmetro de saída.

Banco de Dados II



Views / Funções / Procedures / Triggers

Criando uma Procedure

Exemplos

Usando parâmetro de saída

```
DELIMITER $$
CREATE PROCEDURE Verificar_Quantidade_Produtos
(OUT quantidade INT)
BEGIN
    SELECT COUNT(*)
    INTO quantidade FROM PRODUTOS;
END $$
DELIMITER ;
```

A função desse procedimento é **retornar a quantidade de registros da tabela PRODUTOS**, passando esse valor para a variável de saída “quantidade”. Para isso foi utilizada a palavra reservada INTO.

Banco de Dados II

Views / Funções / Procedures / Triggers

Criando uma Procedure

Exemplos

Para chamá-lo, usamos um símbolo de arroba (@) seguido do nome da variável que receberá o valor de saída. Feito isso, a variável poderá ser usada posteriormente, como vemos na Listagem 7.

Chamando procedure com parâmetro de saída

```
CALL Verificar_Quantidade_Produtos(@total);
SELECT @total;
```

Ao executar a segunda linha, teremos como retorno o valor da variável @total, que será preenchida no procedure.

Banco de Dados II

Views / Funções / **Procedures** / Triggers

Criando uma Procedure

Exemplos

O terceiro exemplo mostra um stored procedure chamado Elevar_Ao_Quadrado, que recebe uma variável e a altera, definindo-a como o seu próprio valor elevado à segunda potência.

Usando parâmetro de entra e saída

```
DELIMITER $$

CREATE PROCEDURE Elevar_Ao_Quadrado(INOUT numero INT)
BEGIN
    SET numero = numero * numero;
END $$
DELIMITER ;
```

Banco de Dados II

Views / Funções / Procedures / Triggers

Usando variáveis no corpo do procedimento

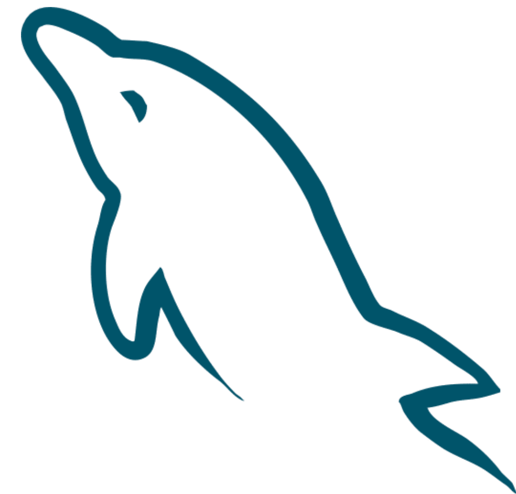
É possível declarar variáveis no corpo dos stored procedures, para isso basta utilizar a seguinte sintaxe:

Sintaxe de declaração de variáveis

```
DECLARE nome_variável TIPO DEFAULT valor_padrao;
```

A palavra reservada **DECLARE** é obrigatória e é a responsável por indicar que uma variável será declarada com o nome "nome_variavel" (que segue as mesmas regras de nomeação de variáveis). O TIPO é o tipo de dados da variável (int, decimal, varchar, etc). A palavra reservada DEFAULT é opcional e deve ser usada quando se deseja definir um valor inicial (valor_padrao) para a variável.

Banco de Dados II



Views / Funções / Procedures / Triggers

Usando variáveis no corpo do procedimento

Sintaxe de declaração de variáveis

```
DECLARE nome_variável TIPO DEFAULT valor_padrao;
```

Para definir um valor para uma variável, usamos as palavras reservadas **SET** ou **INTO**.

Outro ponto importante de se citar é o ESCOPO das variáveis, que define em que pontos elas são reconhecidas. Uma variável definida dentro de um bloco **BEGIN**/**END** é válida somente dentro dele, ou seja, após o END ela já não é mais reconhecida. Assim, é possível definir várias variáveis com o mesmo nome, mas dentro de blocos BEGIN/END distintos.

Por sua vez, variáveis cujo nome inicia com arroba (@), são chamadas variáveis de sessão, e são válidas enquanto durar a sessão.

Banco de Dados II

Views / Funções / **Procedures** / Triggers

Excluindo procedures

Para excluir uma procedure no MySQL basta utilizar o comando abaixo:

```
DROP PROCEDURE Listar_Funcionarios;
```

Há também a alternativa a seguir:

```
DROP PROCEDURE IF EXISTS Listar_Funcionarios;
```

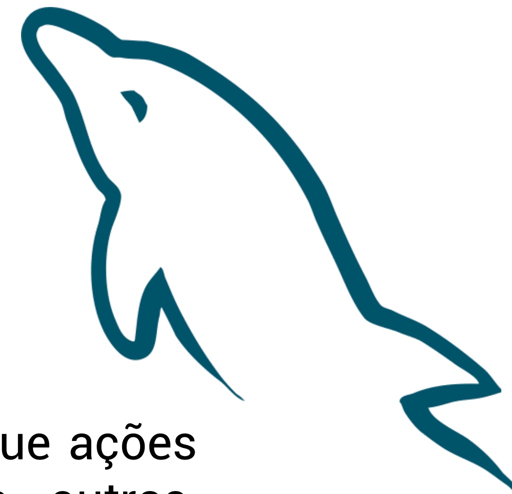
Exibindo procedures

No MySQL temos um comando que exibe todas as stored procedures criadas:

```
SHOW PROCEDURE STATUS;
```

Banco de Dados II

Views / Funções / Procedures / Triggers



O que são Triggers

É muito comum, em aplicações que utilizam bancos de dados, que ações sejam disparadas em resposta ou como consequência de outras, realizando operações de cálculo, validações e, em geral, surtindo alterações na base de dados.

Em muitos casos, os programadores optam por executarem tais ações a partir da própria aplicação, executando várias instruções SQL em sequência para obter o resultado esperado. De fato essa é uma solução que pode até ser tida como mais segura, por certos pontos de vista, mas tende a tornar ainda mais "pesada" a execução de certas tarefas, requisitando mais recursos da máquina cliente.

Banco de Dados II

Views / Funções / Procedures / Triggers

O que são Triggers

A "solução" (ou pelo menos uma forma alternativa) a essa está na utilização de TRIGGERS no banco de dados, automatizando certas ações com base em eventos ocorridos.

Triggers ("gatilhos" em português) são objetos do banco de dados que, relacionados a certa tabela, permitem a realização de processamentos em consequência de uma determinada ação como, por exemplo, a inserção de um registro.

Os triggers podem ser executados ANTES ou DEPOIS das operações de INSERT, UPDATE e DELETE de registros.

Observação: o suporte a triggers foi incluído na versão 5.0.2 do MySQL.

Banco de Dados II

Views / Funções / Procedures / Triggers

Prós e Contras das Triggers

Os principais **pontos positivos** sobre os triggers são:

- Parte do processamento que seria executado na aplicação passa para o banco, poupando recursos da máquina cliente.
- Facilita a manutenção, sem que seja necessário alterar o código fonte da aplicação.

Já **contra sua utilização** existem as seguintes considerações:

- Alguém que tenha acesso não autorizado ao banco de dados poderá visualizar e alterar o processamento realizado pelos gatilhos.
- Requer maior conhecimento de manipulação do banco de dados (SQL) para realizar as operações

Banco de Dados II

Views / Funções / Procedures / Triggers

Sintaxe das Triggers

Onde se tem os seguintes parâmetros:

- nome: nome do gatilho, segue as mesmas regras de nomeação dos demais objetos do banco.

- momento: quando o gatilho será executado. Os valores válidos são BEFORE (antes) e AFTER (depois).

- evento: evento que vai disparar o gatilho. Os valores possíveis são INSERT, UPDATE e DELETE. Vale salientar que os comandos LOAD DATA e REPLACE também disparam os eventos de inserção e exclusão de registros, com isso, os gatilhos também são executados.

- tabela: nome da tabela a qual o gatilho está associado.

Banco de Dados II

Views / Funções / Procedures / Triggers

Sintaxe das Triggers

Não é possível criar mais de um trigger para o mesmo evento e momento de execução na mesma tabela. Por exemplo, não se pode criar dois gatilhos AFTER INSERT na mesma tabela.

Banco de Dados II

Views / Funções / Procedures / Triggers

Os registros NEW e OLD

Como os triggers, são executados em conjunto com operações de inclusão e exclusão, é necessário poder acessar os registros que estão sendo incluídos ou removidos. Isso pode ser feito através das palavras **NEW** e **OLD**.

Em gatilhos executados após a inserção de registros, a palavra reservada **NEW** dá acesso ao novo registro. Pode-se acessar as colunas da tabela como atributo do registro **NEW**, como veremos nos exemplos.

O operador **OLD** funciona de forma semelhante, porém em gatilhos que são executados com a exclusão de dados, o OLD dá acesso ao registro que está sendo removido.

Banco de Dados II

Views / Funções / Procedures / Triggers

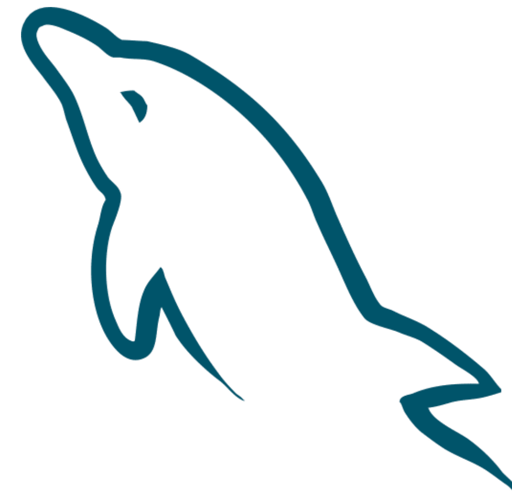
Utilização do trigger

Para exemplificar e tornar mais clara a utilização de gatilhos, simularemos a seguinte situação: 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.

Como se trata de um ambiente hipotético, teremos apenas duas tabelas de estrutura simples, cujo script de criação é mostrado na listagem a seguir.

Banco de Dados II

Views / Funções / Procedures / Triggers



Utilização do trigger

```
CREATE TABLE Produtos(
    Referencia  VARCHAR(3) PRIMARY KEY,
    Descricao   VARCHAR(50) UNIQUE,
    Estoque  INT NOT NULL DEFAULT 0
);
INSERT INTO Produtos VALUES ("001", "Feijão", 10);
INSERT INTO Produtos VALUES ("002", "Arroz", 5);
INSERT INTO Produtos VALUES ("003", "Farinha", 15);

CREATE TABLE ItensVenda(
    Venda INT,
    Produto VARCHAR(3),
    Quantidade INT
);
```

Criação das tabelas utilizadas

Banco de Dados II

Views / Funções / Procedures / Triggers

Utilização do trigger

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

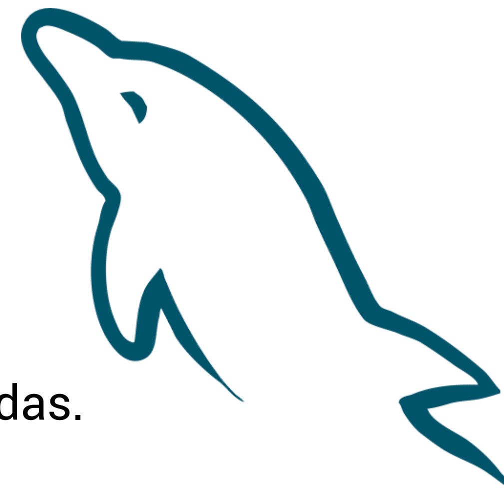
Observação: como usaremos instruções que requerem ponto e vírgula no final, alteraremos o delimitador de instruções para $$ e depois de criar os triggers, voltaremos para o padrão. Essa alteração não está diretamente ligada aos triggers.

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

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

Banco de Dados II

Views / Funções / Procedures / Triggers



Utilização do trigger

Abaixo, serão criados os gatilhos para executar as ações já discutidas.

```
DELIMITER $
CREATE TRIGGER Tgr_ItensVenda_Insert AFTER INSERT ON ItensVenda
FOR EACH ROW
BEGIN
    UPDATE Produtos SET Estoque = Estoque - NEW.Quantidade
WHERE Referencia = NEW.Produto;
END$

CREATE TRIGGER Tgr_ItensVenda_Delete AFTER DELETE ON ItensVenda
FOR EACH ROW
BEGIN
    UPDATE Produtos SET Estoque = Estoque + OLD.Quantidade
WHERE Referencia = OLD.Produto;
END$
DELIMITER ;
```

Criação dos triggers

Banco de Dados II

Views / Funções / Procedures / Triggers

Utilização do trigger

No primeiro gatilho, foi utilizado o registro NEW para obter as informações da linha que está sendo inserida na tabela. O mesmo é feito no segundo gatilho, onde se obtém os dados que estão sendo apagados da tabela através do registro OLD.

Tendo criado os triggers, podemos testá-los inserindo dados na tabela ItensVenda. Nesse caso, vamos simular uma venda de número 1 que ontem 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);
```

Inserindo dados na tabela

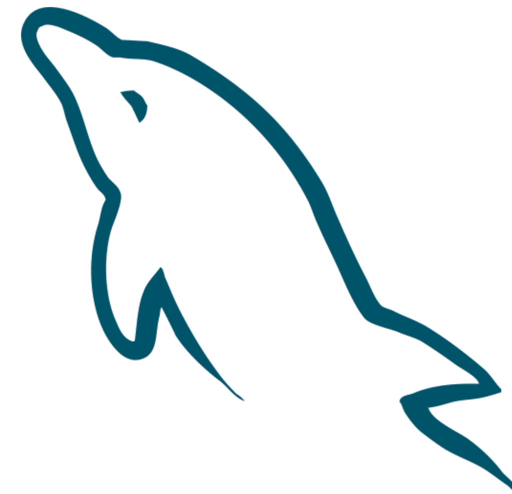
Banco de Dados II

Views / Funções / Procedures / Triggers

Utilização do trigger

```
DELETE FROM ItensVenda
WHERE Venda = 1 AND Produto = "001";
```

Excluindo dados da tabela ItensVenda



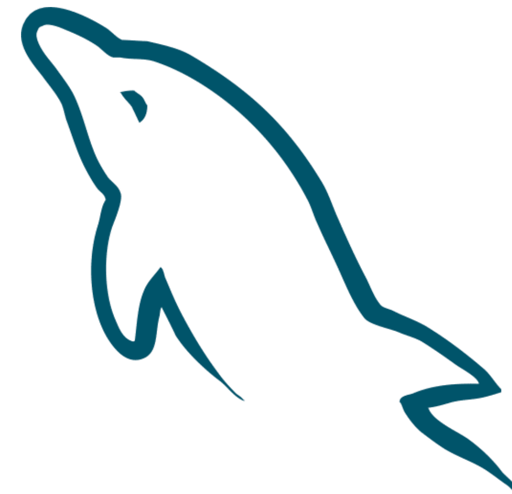
Agora, fazendo uma consulta à tabela Produtos, obtemos o resultado exibido na figura abaixo.

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

Baixa no estoque após a inserção na tabela ItensVenda

Banco de Dados II

Views / Funções / Procedures / Triggers



Utilização do trigger

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

Baixa no estoque após a inserção na tabela ItensVenda

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

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.

Banco de Dados II

Views / Funções / Procedures / Triggers

Utilização do trigger

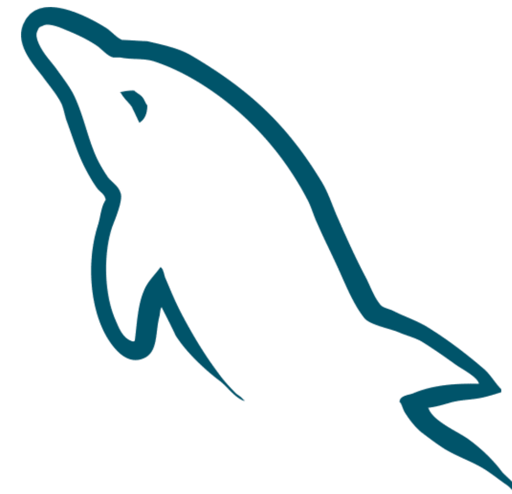
Referencia	Descricao	Estoque
001	Feijão	10
002	Arroz	4
003	Farinha	10
NULL	NULL	NULL

Devolução do estoque após exclusão de registro na tabela ItensVenda

Com isso confirmamos que os gatilhos estão funcionando da forma esperada.

Banco de Dados II

Views / Funções / Procedures / Triggers



Exibindo e excluindo Triggers

```
SHOW TRIGGERS
```

Exibição das triggers criadas

A seguir vemos a exclusão de uma trigger no MySQL:

```
DROP TRIGGER Tgr_ItensVenda_Insert
```

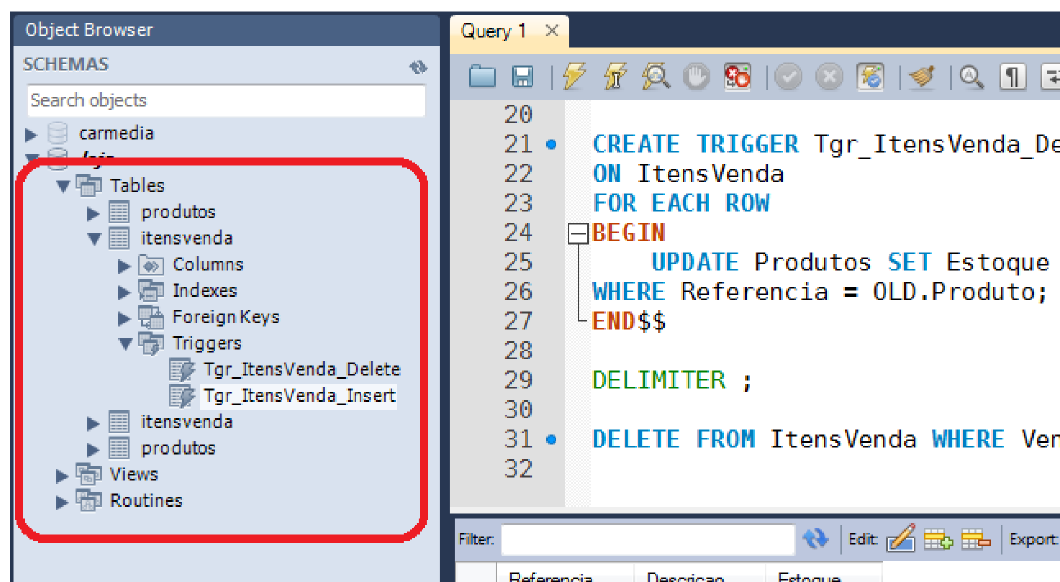
Exclusão de trigger

Banco de Dados II

Views / Funções / Procedures / Triggers

Informações adicionais

Apenas a nível de informação, vale o seguinte comentário: na nova versão do MySQL, no MySQL Workbench é possível visualizar os gatilhos relacionados a uma tabela através do Object browser, como mostra a figura a seguir.



Object Browser no MySQL Workbench

***Nota**: Em ambientes reais, triggers podem ser utilizados para operações mais complexas, por exemplo, antes de vender um item, verificar se há estoque disponível e só então permitir a saída do produto.*

Referências

- **Boson Treinamentos**

  www.bosontreinamentos.com.br/mysql

- **Dev Media**

  www.devmedia.com.br/stored-procedures-no-mysql/29030