Aula 6

Banco de Dados

Prof. Lucas Rafael Filipak

Conversa Inicial

Banco de dados

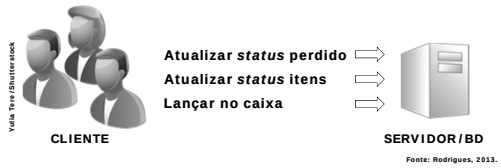
- *Stored procedure*
- *Functions*
- *Triggers*
- Estruturas de programação

Stored procedure

- *Stored procedure* ou um procedimento de armazenagem foi implementado com base na versão 5.0 do MySQL
- Parte do processamento (programação) para o banco de dados

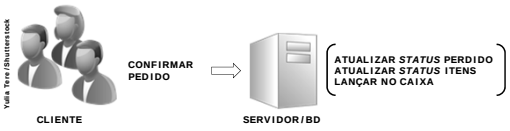
- Validação de dados
- Executar instruções SQLs
- Controle de acesso
- Receber parâmetros
- Retornar valores

Sem *stored procedure*



Fonte: Rodrigues, 2013.

Com *stored procedure*



Fonte: Rodrigues, 2013.

Pontos negativos

- Segundo Rodrigues (2013):
  - Necessidade de maior conhecimento da sintaxe do banco de dados para escrita de rotinas em SQL
  - As rotinas ficam mais facilmente acessíveis. Alguém que tenha acesso ao banco poderá visualizar e alterar o código

Pontos positivos

- Simplificação da execução de instruções SQL pela aplicação
- Transferência de parte da responsabilidade de processamento para o servidor
- Facilidade na manutenção, reduzindo a quantidade de alterações na aplicação

Delimiter

- Padrão
- É utilizado para trocar o caractere de finalização

```
DELIMITER $$
—Seu objetivo 1 aqui
$$
—Seu objetivo 2 aqui
$$
```

Criando *Procedures*

```
CREATE PROCEDURE nome_procedimento (parâmetros)
Declarações;
```

Chamando *Procedures*

```
CALL nome_procedimento (parâmetros)
```

Exemplo *Procedures*

```
CREATE PROCEDURE ExibeNúmeroDePedidos
SELECT C.id-cliente, C.nome-cliente, COUNT (*)
        FROM        Clientes C, Pedidos P
        WHERE       C.id-cliente = P.id-cliente
        GROUP BY    C.id-cliente, C.nome-cliente
```

Fonte: Ramarkrishnan, 2011, p. 174.

```
CALL ExibeNúmeroDePedidos
```

Exemplo *Procedures*

```
CREATE PRODUCEDURE valorPedido (varPedido smallint)
SELECT CONCAT (Valor_final, ' é o total do pedido', varPedido) AS Valor_total
FROM Pedidos
WHERE id_pedido = varPedido;
```

```
CALL valorPedido(3);
```

Parâmetros

- ***IN*** → **é utilizado apenas para recebimento de dados, não é utilizado para *feedback***
- ***OUT*** → **é um parâmetro de saída, não sendo informado um valor fixo (direto), apenas uma variável para o retorno**
- ***INOUT*** → **esse modo de parâmetro pode ser utilizado como entrada ou saída, não podendo ser informado um valor fixo**

Exemplo com parâmetros

```
CREATE PROCEDURE AcrescInventário (
                IN Livro_isbn CHAR (10),
                IN qtidadeAcresc INTEGER)
UPDATE Livros
        SET     qtidade_em_estoque – qtidade_em_estoque – qtidadeAcresc
        WHERE   livro_isbn = isbn
```

Fonte: Ramarkrishnan, 2011, p. 174.

Apagando *procedure*

```
DROP PROCEDURE nome_procedure;
```

Function

- Segundo Puga (2013, p. 280), "as *functions* são muito semelhantes as *procedures*, o que os difere, do ponto de vista estrutural, é a inclusão da cláusula *RETURN*"

Criando uma *Function*

```
CREATE FUNCTION nome_função (parâmetros)
RETURNS tipo_dados
código_da_função
```

Chamando uma *Function*

```
SELECT nome_função (parâmetros);
```

Exemplo *Function*

```
CREATE FUNCTION fn_teste (a DECIMAL (10,2), b INT)
RETURNS INT
RETURN a * b;
```

Fonte: Reis, 2014.

```
SELECT fn_teste(2.5, 4) AS Resultado;
```

Fonte: Reis, 2014.

Exemplo *Function*

```
CREATE FUNCTION verPreço (a SMALINT)
RETURNS VARCHAR (60)
RETURN
(SELECT CONCAT ("O preço do livro", Nome_Livro
'é', Preço_livro)
FROM Livros
WHERE id_livro=a;
```

Fonte: Adaptado de Reis, 2014.

```
SELECT fn_verPreço(9);
```

Fonte: Reis, 2014.

Apagando *Function*

```
DROP FUNCTION nome_função;
```

Triggers

Trigger (gatilho)

- Um *trigger* é um objeto do banco de dados que sempre está associado a uma tabela e é disparado automaticamente antes ou depois de um evento DML

Pontos positivos

- Segundo Rodrigues (2016):
  - 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

Pontos negativos

- 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 internamente

Sintaxe

- nome → nome da *trigger*
- momento → quando a *trigger* vai ser executada: *BEFORE* (antes) ou *AFTER* (depois)
- evento → qual o comando que vai fazer a *trigger* disparar: *INSERT*, *UPDATE*, *DELETE* e *REPLACE*
- tabela → é a tabela a que o *trigger* está associado

Exemplo Sintaxe

```
CREATE TRIGGER nome momento evento
ON tabela
FOR EACH ROW
BEGIN
/*corpo do código*/
END
```

Fonte: Rodrigues, 2016.

Exemplo *Trigger*

```
CREATE TRIGGER tr_desconto BEFORE INSERT
ON Livros
FOR EACH ROW
SET NEW.preco_desconto=(NEW.preco_produto*0.90);
```

Registros *NEW* e *OLD*

- As palavras *NEW* e *OLD* são utilizadas para acessar os registros antes ou depois da execução da *trigger*
- Pode-se acessar os registros que serão enviados para uma tabela antes (*BEFORE*) ou depois (*AFTER*) de um *UPDATE*

- Bianchi (2008) explica os registros *NEW* e *OLD*:
  - *INSERT*: o operador NEW.nome_coluna nos permite verificar o valor enviado para ser inserido em uma coluna de uma tabela. OLD.nome_coluna não está disponível

- *DELETE*: o operador OLD.nome_coluna nos permite verificar o valor excluído ou a ser excluído. NEW.nome_coluna não está disponível
- *UPDATE*: tanto OLD.nome_coluna quanto NEW.nome_coluna estão disponíveis, antes (BEFORE) ou depois (AFTER) da atualização de uma linha

```
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;
```

Fonte: Rodrigues, 2016.

Limitações

- Segundo Bianchi (2008):
  - Não se pode chamar diretamente um *trigger* com *CALL*, como se faz com um *stored procedures*
  - Não é permitido iniciar ou finalizar transações em meio a *triggers*

- *Triggers* ainda não podem ser implementadas com a intenção de devolver para o usuário ou para uma aplicação mensagens de erros

Apagando *Triggers*

```
DROP TRIGGER nome_da_trigger;
```

Estruturas de programação

- As *stored procedures* podem ter dentro dos seus procedimentos alguns processos, como:
  - Condicionais
  - Laços de repetição
  - Funções

Sintaxe Condicionais

```
IF<condição>THEN
  comandos sql caso verdadeiro
ELSE
  comandos sql caso falso
END IF
```

Condicional

```
DELIMITER//
CREATE PROCEDURE lista_clientes (IN opção integer)

BEGIN
  IF opção = 0 THEN
     SELECT * FROM clientes where sexo = F;
  ELSE
    IF opção = 1 THEN
       SELECT * FROM clientes WHERE sexo = M;
    ELSE
      SELECT*FROM clientes;
    END IF;
  END IF;
END//
```

Chamando *Procedures*

```
IF opção = 0 THEN
   SELECT * FROM clientes where sexo = F;
ELSE
   IF opção = 1 THEN
      SELECT * FROM clientes WHERE sexo = M;
   ELSE
     SELECT * FROM clientes;
   END IF;
END IF;
```

```
CALL lista_clientes (0);

CALL lista_clientes (1);

CALL lista_clientes (2);
```

Laço de Repetição

```
DELIMITER //
CREATE PROCEDURE acumulador (limite TINYINT UNSIGNED)
BEGIN
 DECLARE contador TINYINT UNSIGNED DEFAULT 0;
 DECLARE soma INT DEFAULT 0;
 WHILE contador < limite DO
   SET contador = contador + 1;
   SET soma = soma + contador;
 END WHILE;
 SELECT soma;
END//
```

```
CALL acumulador(10);
```

Finalizando

- ***Stored procedure***
- ***Functions***
  - ***Return***
- ***Triggers***
  - ***NEW* e *OLD***
- **Estruturas de programação**
  - **Condicional, laço, função etc.**

Referências

- **BIANCHI, W. MySQL: triggers. DevMedia, [S.d]. Disponível em: <https://www.devmedia.com.br/mysql-triggers/8088>. Acesso em: 29 ago. 2018.**
- **PUGA, S. Banco de dados: implementação em SQL, PL/SQL, Oracle 11g. São Paulo: Pearson Education do Brasil, 2013.**
- **RAMARKRISHNAN, R. Sistemas de gerenciamento de banco de dados. 3. ed. Porto Alegre: AMGH, 2011.**
- **REIS, F. dos. MySQL: procedimentos armazenados. Bosón Treinamentos, 13 fev. 2014. Disponível em: <http://www.bosontreinamentos.com.br/mysql/mysql-procedimentos-armazenados-stored-procedures-basico-34/>. Acesso em: 29 ago. 2018.**
- **REIS, F. dos. MySQL: rotinas armazenadas – funções (create function). Bosón Treinamentos, 13 fev. 2017. Disponível em: <http://www.bosontreinamentos.com.br/mysql/mysql-rotinas-armazenadas-funcoes-create-function-33/>. Acesso em: 29 ago. 2018.**
- **RODRIGUES, J. MySQL Básico: triggers. DevMedia, [S.d]. Disponível em: <https://www.devmedia.com.br/mysql-basico-triggers/37462>. Acesso em: 29 ago. 2018.**
- **_____. Stored procedures no MySQL. DevMedia, [S.d]. Disponível em: <https://www.devmedia.com.br/stored-procedures-no-mysql/29030>. Acesso em: 29 ago. 2018.**