





Rotinas Armazenadas

- 
- ▶ Uma rotina armazenada é um conjunto de instruções SQL que podem ser armazenadas no servidor.
 - ▶ Uma vez que isso tenha sido feito, os clientes não precisam reenviar as instruções individuais, mas pode solicitar a execução da rotina armazenada.
 - ▶ Uma rotina armazenada é um trecho de código SQL que pode ser executada através de gatilhos (próxima matéria), por outras rotinas armazenadas ou aplicações desenvolvidas em Java, C#, PHP, etc.

Vantagens

- ▶ Normalmente rotinas armazenadas ajudam a aumentar o desempenho das aplicações. Uma vez criadas, são compiladas e armazenados no banco de dados.
- ▶ No entanto MySQL implementa de forma ligeiramente diferente, sendo compiladas por demanda. Depois de compilar, o MySQL coloca em uma cache. O MySQL mantém seu próprio cache de rotinas armazenadas para cada conexão. Se um aplicativo usa uma rotina armazenada várias vezes em uma única conexão, a versão compilada é usada, caso contrário, a rotina armazenada é executada como uma consulta.

- 
- ▶ Ajuda a reduzir o tráfego entre a aplicação e o servidor de banco de dados, porque em vez de enviar várias instruções SQL longas, a aplicação tem de enviar apenas o nome e os parâmetros da rotina armazenada.
 - ▶ São reutilizáveis e transparentes para todas as aplicações. As rotinas armazenadas expõe a interface de banco de dados para todas as aplicações para que os desenvolvedores não tenham que desenvolver funções que já estão definidas nas rotinas armazenadas.
 - ▶ São seguras. Administrador de banco de dados pode conceder permissões adequadas para aplicativos que acessam rotinas armazenadas no banco de dados, sem dar qualquer permissão nas tabelas.

Desvantagens

- ▶ Se usar um monte de rotinas armazenadas, o uso de memória de cada conexão aumentará substancialmente. Além disso, se usar um grande número de operações lógicas, o uso da CPU também vai aumentar porque o servidor de banco de dados não está bem concebido para operações lógicas.
- ▶ É difícil depurar rotinas armazenadas. Apenas alguns sistemas de gerenciamento de banco de dados permitem depurar rotinas armazenadas. Infelizmente, o MySQL não oferece facilidades para a depuração de rotinas armazenadas.
- ▶ Não é fácil para desenvolver e manter rotinas armazenadas. O desenvolvimento e manutenção exige um conjunto de habilidades especializadas que nem todos os desenvolvedores de aplicações possuem. Isso pode levar a problemas tanto de desenvolvimento quanto de manutenção de aplicações.

Tipos

- ▶ Tipos de rotinas armazenadas
 - ▶ Procedimento
 - ▶ Sem retorno
 - ▶ Função
 - ▶ Com retorno
 - ▶ Ex: count()

Parâmetros

- ▶ **IN** => Parâmetro de entrada. É passado um valor que vai ser utilizado no corpo da procedure;
- ▶ **OUT** => Parâmetro de saída. Retorna um valor que pode ficar armazenado na memória do servidor;
- ▶ **INOUT** => Possui a funcionalidade de entrada e saída ao mesmo tempo.

Parâmetro IN - Exemplo

```
DELIMITER $$  
CREATE PROCEDURE GetOfficeByCountry(  
    IN countryName VARCHAR(255))  
BEGIN  
    SELECT *  
    FROM offices  
    WHERE country = countryName;  
END $$  
DELIMITER ;
```


Parâmetro IN - Exemplo

- ▶ Para executar
 - ▶ CALL GetOfficeByCountry('USA')

	officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
▶	1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
	2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
	3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA

Parâmetro OUT - Exemplo

```
DELIMITER $$  
CREATE PROCEDURE CountOrderByStatus(  
    IN orderStatus VARCHAR(25),  
    OUT total INT)  
BEGIN  
    SELECT count(orderNumber)  
    INTO total  
    FROM orders  
    WHERE status = orderStatus;  
END$$  
DELIMITER ;
```

Parâmetro OUT - Exemplo

- ▶ Para Executar

- ▶ CALL CountOrderByStatus('Shipped',@total);
- ▶ SELECT @total;

	@total
▶	303

- ▶ CALL CountOrderByStatus('in process',@total);
- ▶ SELECT @total AS total_in_process;

	total_in_process
▶	6

Parâmetro INOUT - Exemplo

```
DELIMITER $$
```

```
CREATE PROCEDURE set_counter(INOUT count INT(4),IN inc INT(4))
```

```
BEGIN
```

```
    SET count = count + inc;
```

```
END$$
```

```
DELIMITER ;
```

Parâmetro INOUT - Exemplo

```
DELIMITER $$
```

```
CREATE PROCEDURE set_counter(INOUT count INT(4),IN inc INT(4))
```

```
BEGIN
```

```
    SET count = count + inc;
```

```
END$$
```

```
DELIMITER ;
```

Parâmetro INOUT - Exemplo

- ▶ Dentro do procedimento armazenado, aumentamos o contador com o valor do parâmetro inc.
- ▶ SET @counter = 1;
- ▶ CALL set_counter(@counter,1); -- 2
- ▶ CALL set_counter(@counter,1); -- 3
- ▶ CALL set_counter(@counter,5); -- 8
- ▶ SELECT @counter; -- 8

Delimitador

- ▶ DELIMITER \$\$
 - ▶ Necessário para o MySQL não interpretar os ; do código internos do procedimento armazenado como fim do comando
- ▶ END\$\$
 - ▶ Identifica o final do procedimento armazenado
- ▶ DELIMITER ;
 - ▶ Retorna a interpretar o ; como fim de comando SQL

Variáveis

- ▶ DECLARE nome tipo(tamanho) DEFAULT valor_padrão;
 - ▶ Ex: DECLARE total_sale INT DEFAULT 0
- ▶ Declarar mais de uma variável do mesmo tipo
 - ▶ DECLARE x, y INT DEFAULT 0
- ▶ Alterar o valor da variável
 - ▶ SET total_count = 10;
- ▶ Além do comando SET, pode usar SELECT INTO para atribuir o resultado de uma consulta a uma variável. A consulta deve retornar um valor escalar
 - ▶ Ex:
 - ▶ DECLARE total_products INT DEFAULT 0
 - ▶ SELECT COUNT(*) INTO total_products
 - ▶ FROM products
- ▶ Uma variável que começa com o símbolo @ no início é variável de sessão. Ela está disponível e acessível até a sessão terminar.

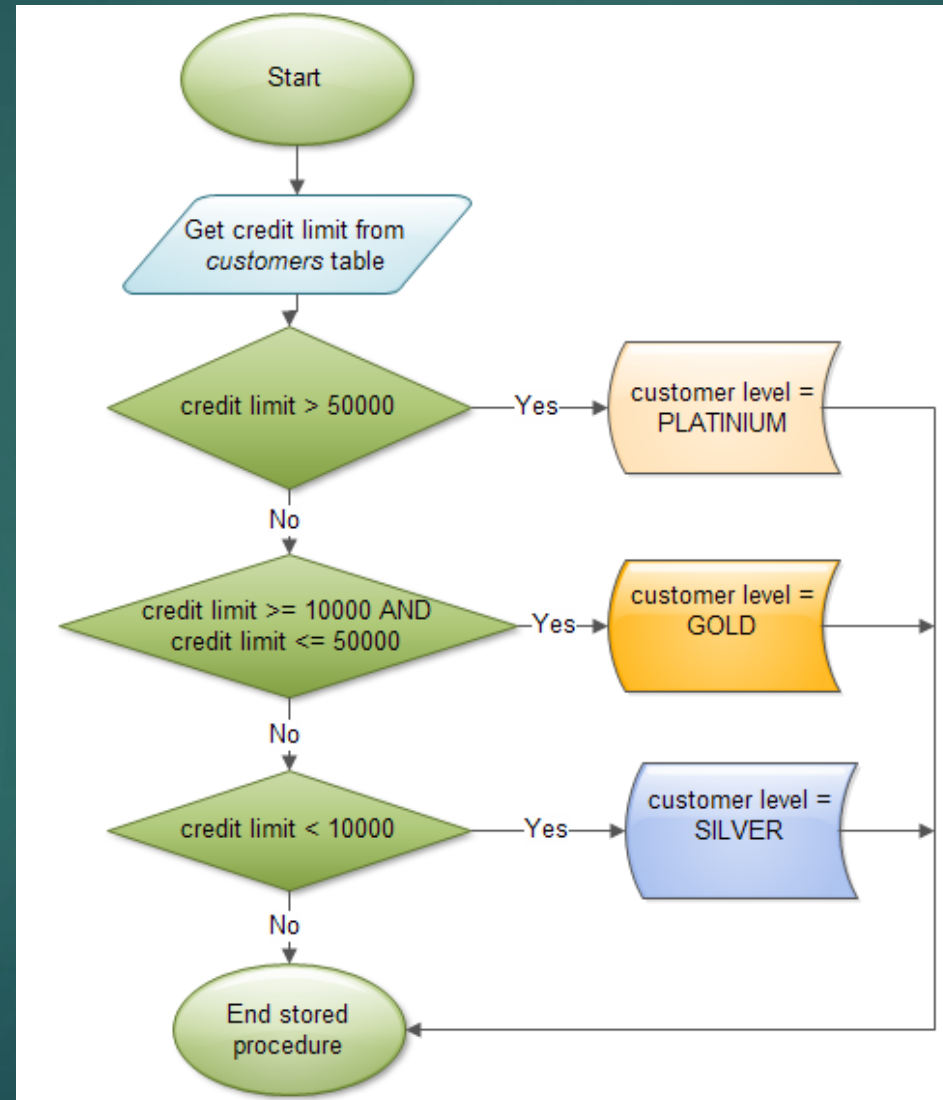
IF

- ▶ IF if_expression THEN commands
[ELSEIF elseif_expression THEN commands]
[ELSE commands]
END IF;
- ▶ Ex:
DELIMITER \$\$
CREATE PROCEDURE GetCustomerLevel(
 in p_customerNumber int(11),
 out p_customerLevel varchar(10))
BEGIN
 DECLARE creditlim double;
 SELECT creditlimit INTO creditlim
 FROM customers
 WHERE customerNumber = p_customerNumber;

IF

```
IF creditlim > 50000 THEN
  SET p_customerLevel = 'PLATINUM';
ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN
  SET p_customerLevel = 'GOLD';
ELSEIF creditlim < 10000 THEN
  SET p_customerLevel = 'SILVER';
END IF;
END$$
```

IF



CASE

▶ CASE case_expression
 WHEN when_expression_1 THEN commands
 WHEN when_expression_2 THEN commands
 ...
 ELSE commands
END CASE;

CASE

► Ex:

```
DELIMITER $$
```

```
CREATE PROCEDURE GetCustomerShipping(
```

```
    in p_customerNumber int(11),
```

```
    out p_shipping varchar(50))
```

```
BEGIN
```

```
    DECLARE customerCountry varchar(50);
```

```
    SELECT country INTO customerCountry
```

```
    FROM customers
```

```
    WHERE customerNumber = p_customerNumber;
```

CASE

► Ex:

```
CASE customerCountry
```

```
  WHEN 'USA' THEN
```

```
    SET p_shipping = '2-day Shipping';
```

```
  WHEN 'Canada' THEN
```

```
    SET p_shipping = '3-day Shipping';
```

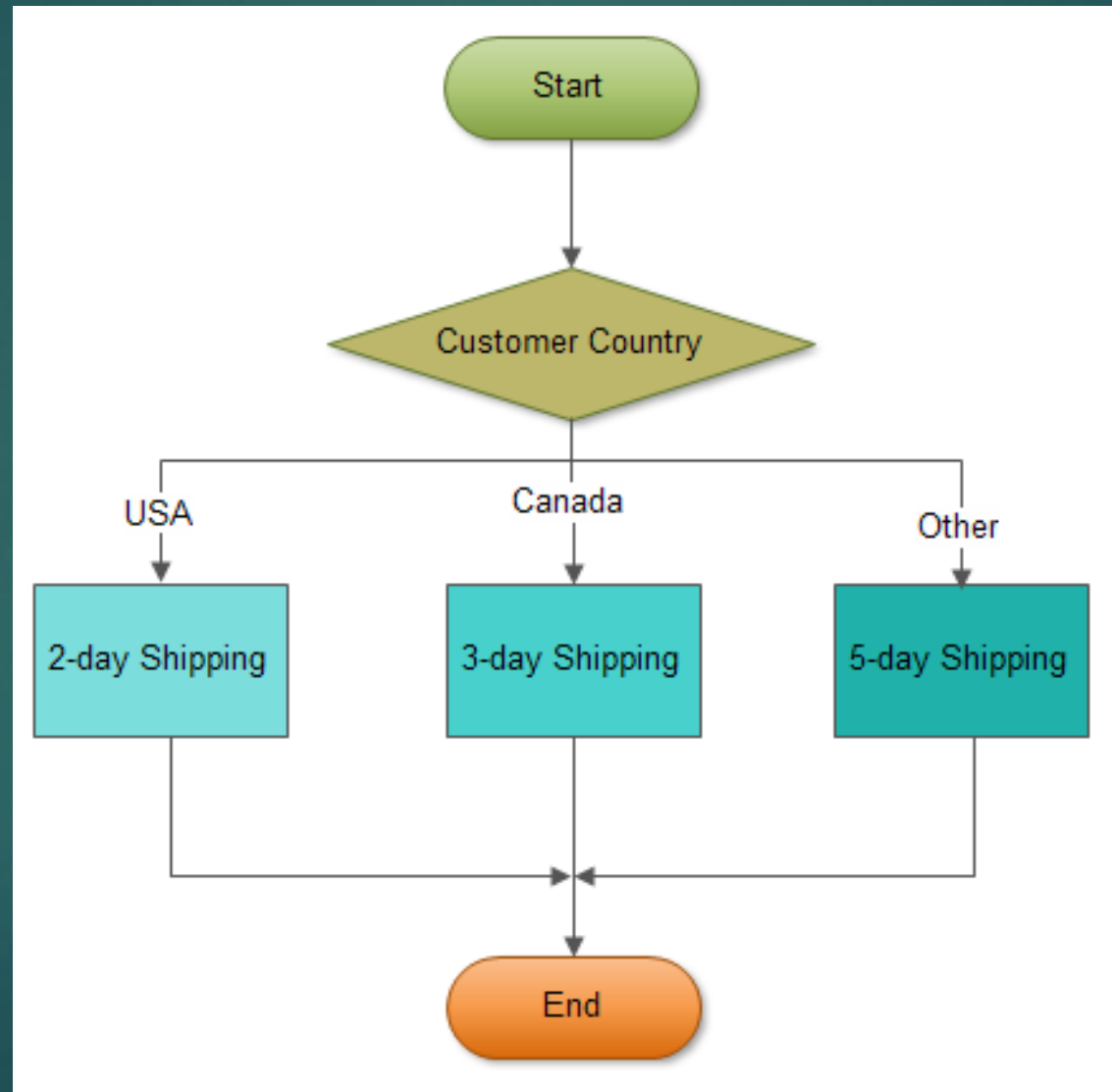
```
  ELSE
```

```
    SET p_shipping = '5-day Shipping';
```

```
END CASE;
```

```
END$$
```


CASE



CASE

► Teste

```
SET @customerNo = 112;
```

```
SELECT country into @country  
FROM customers  
WHERE customernumber = @customerNo;
```

```
CALL GetCustomerShipping(@customerNo,@shipping);
```

```
SELECT @customerNo AS Customer,  
       @country    AS Country,  
       @shipping   AS Shipping;
```

	Customer	Country	Shipping
►	112	USA	2-day Shipping

CASE

- ▶ A declaração simples CASE só permite verificar o valor de uma expressão contra um conjunto de valores distintos.
- ▶ A fim de executar expressões mais complexas, tais como faixas de valores, usa-se a instrução CASE sem variável. A instrução CASE fica equivalente a instrução IF, no entanto a sua construção é muito mais legível.
- ▶ CASE
 - ▶ WHEN condition_1 THEN commands
 - ▶ WHEN condition_2 THEN commands
 - ▶ ...
 - ▶ ELSE commands
- ▶ END CASE;

CASE

- ▶ O MySQL avalia cada condição de teste na cláusula WHEN até encontrar uma condição de teste que resulte em um valor TRUE, executando os comandos correspondentes ao THEN.
- ▶ Se nenhuma condição for TRUE, o comando relativo ao ELSE será executado

CASE

```
DELIMITER $$  
CREATE PROCEDURE GetCustomerLevel(  
    in p_customerNumber int(11),  
    out p_customerLevel varchar(10))  
BEGIN  
    DECLARE creditlim double;  
    SELECT creditlimit INTO creditlim  
    FROM customers  
    WHERE customerNumber = p_customerNumber;
```

CASE

```
CASE
  WHEN creditlim > 50000 THEN
    SET p_customerLevel = 'PLATINUM';
  WHEN (creditlim <= 50000 AND creditlim >= 10000) THEN
    SET p_customerLevel = 'GOLD';
  WHEN creditlim < 10000 THEN
    SET p_customerLevel = 'SILVER';
END CASE;
END$$
```

CASE

- ▶ Teste
- ▶ CALL GetCustomerLevel(112,@level);
- ▶ SELECT @level AS 'Customer Level';

	Customer Level
▶	PLATINUM

WHILE

- ▶ WHILE expression DO
- ▶ Statements
- ▶ END WHILE

WHILE

```
DELIMITER $$  
DROP PROCEDURE IF EXISTS WhileLoopProc$$  
CREATE PROCEDURE WhileLoopProc()  
BEGIN  
    DECLARE x INT;  
    DECLARE str VARCHAR(255);  
    SET x = 1;  
    SET str = '';
```

```
    WHILE x <= 5 DO  
        SET str = CONCAT(str,x,',');  
        SET x = x + 1;  
    END WHILE;  
    SELECT str;  
END$$  
DELIMITER ;
```

REPEAT

- ▶ REPEAT
- ▶ Statements;
- ▶ UNTIL expression
- ▶ END REPEAT

REPEAT

```
DELIMITER $$  
  
DROP PROCEDURE IF EXISTS  
RepeatLoopProc$$  
  
CREATE PROCEDURE RepeatLoopProc()  
BEGIN  
    DECLARE x INT;  
    DECLARE str VARCHAR(255);  
    SET x = 1;  
    SET str = '';
```

```
REPEAT  
    SET str = CONCAT(str,x,',');  
    SET x = x + 1;  
  
UNTIL x > 5  
END REPEAT;  
  
SELECT str;  
  
END$$  
  
DELIMITER ;
```

LEAVE, ITERATE e LOOP

- ▶ LEAVE permite parar o laço de repetição
 - ▶ Semelhante ao break em java
- ▶ ITERATE permite parar a execução de uma iteração da repetição
- ▶ LOOP permite criar um laço de repetição infinito

LEAVE, ITERATE e LOOP

```
DELIMITER $$
DROP PROCEDURE IF EXISTS LOOPLoopProc$$
CREATE PROCEDURE LOOPLoopProc()
BEGIN
    DECLARE x INT;
    DECLARE str VARCHAR(255);
    SET x = 1;
    SET str = '';
```

```
    loop_label: LOOP
        IF x > 10 THEN
            LEAVE loop_label;
        END IF;
        SET x = x + 1;
        IF (x mod 2) THEN
            ITERATE loop_label;
        ELSE
            SET str = CONCAT(str,x,',');
        END IF;
    END LOOP;
    SELECT str;
END$$
DELIMITER ;
```

Listar procedures

- ▶ `SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE expr];`
- ▶ Para listar todos os procedimentos armazenados dos bancos de dados que você tem o privilégio de acesso
 - ▶ `SHOW PROCEDURE STATUS;`
- ▶ Se quer mostrar apenas os de um banco de dados específico
 - ▶ `SHOW PROCEDURE STATUS WHERE db = 'classicmodels';`
- ▶ Se você quer os que tem um determinado padrão. Ex: O nome da procedure contém a palavra product
 - ▶ `SHOW PROCEDURE STATUS WHERE name LIKE '%product%'`

Listar procedures

- ▶ Para mostrar o código-fonte de uma procedure
- ▶ `SHOW CREATE PROCEDURE nome_da_stored_procedure`
 - ▶ Ex:
 - ▶ `SHOW CREATE PROCEDURE GetAllProducts`

Deletar uma procedure

- ▶ DROP PROCEDURE | FUNCTION [IF EXISTS] nome

Stored Functions

- ▶ CREATE **FUNCTION** function_name(param1,param2,...)
 - ▶ RETURNS datatype
 - ▶ [**NOT**] DETERMINISTIC
 - ▶ Statements
-
- ▶ DETERMINISTIC
 - ▶ Sempre retorna a mesma coisa
 - ▶ Ex: Data de nascimento da pessoa
 - ▶ NOT DETERMINISTIC
 - ▶ Depende do momento da execução
 - ▶ Ex: Soma das horas extras dos funcionários

Stored Functions

```
DELIMITER $$
CREATE FUNCTION CustomerLevel(p_creditLimit double) RETURNS VARCHAR(10)
    DETERMINISTIC
BEGIN
    DECLARE lvl varchar(10);
    IF p_creditLimit > 50000 THEN
        SET lvl = 'PLATINUM';
    ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >= 10000) THEN
        SET lvl = 'GOLD';
    ELSEIF p_creditLimit < 10000 THEN
        SET lvl = 'SILVER';
    END IF;
    RETURN (lvl);
END$$
DELIMITER ;
```

Stored Functions

- ▶ A execução pode ser em um select simples
- ▶ SELECT customerName,
- ▶ CustomerLevel(creditLimit)
- ▶ FROM customers;

Stored Functions

- Reescrevendo a procedure GetCustomLevel que fizemos anteriormente

DELIMITER \$\$

```
CREATE PROCEDURE GetCustomerLevel(
```

```
    IN p_customerNumber INT(11),
```

```
    OUT p_customerLevel varchar(10))
```

```
BEGIN
```

```
    DECLARE creditlim DOUBLE;
```

```
    SELECT creditlimit INTO creditlim
```

```
    FROM customers
```

```
    WHERE customerNumber = p_customerNumber;
```

```
    SELECT CUSTOMERLEVEL(creditlim)
```

```
    INTO p_customerLevel;
```

```
END$$
```

```
DELIMITER ;
```

Stored Functions

```
DELIMITER $$
CREATE FUNCTION f_age (in_dob datetime) RETURNS TINYINT
BEGIN
    DECLARE l_age INT;
    IF DATE_FORMAT(NOW( ),'00-%%m-%%d') >= DATE_FORMAT(in_dob,'00-%%m-%%d')THEN
        -- Já fez aniversário esse ano
        SET l_age=DATE_FORMAT(NOW( ),'%%Y')-DATE_FORMAT(in_dob,'%%Y');
    ELSE -- Ainda fará aniversário esse ano
        SET l_age=DATE_FORMAT(NOW( ),'%%Y')-DATE_FORMAT(in_dob,'%%Y')-1;
    END IF;
    RETURN(l_age);
END$$
DELIMITER ;
```