Bases de Dados

SQL: STRUCTURED QUERY LANGUAGE

- STORED PROCEDURES
- STORED FUNCTIONS

Stored procedure

- Sequência de comandos SQL
- Guardado dentro da base de dados
- Pode ser chamado por:
  - Triggers
  - Outros stored procedures
    - *Stored procedure* recursivo: um *stored procedure* que se chama a si próprio. O MySQL não suporta muito bem esta funcionalidade
  - Aplicações

Vantagens e Desvantagens

VANTAGENS

- Aumento de performance de aplicações na mesma sessão. O procedure é recompilado ao início de cada sessão.
- Redução do tráfego entre as aplicações e o servidor de base de dados. Em vez das queries a aplicação apenas precisa de enviar os nomes e parâmetros dos stored procedures.
- Os stored procedures são reutilizáveis e transparentes entre aplicações.
- Aumentam a segurança. Pode dar-se permissões de acesso aos stored procedures sem dar acesso às tabelas.

DESVANTAGENS

- A utilização de muitos stored procedures faz com que haja um aumento da utilização de memória.
- Quando o stored procedure tem muitas operações lógicas há uma maior utilização de CPU.
- É difícil fazer debug.
- Desenvolvimento e manutenção de stored procedures não são triviais

Exemplo

```
DELIMITER //
CREATE PROCEDURE GetAllClients()
  BEGIN
    SELECT * FROM cliente;
  END //
DELIMITER ;
```

```
CALL getAllClients();
```

1. `DELIMITER //`

altera o caracter que determina o final do statement de ; para //

1. `CREATE PROCEDURE GetAllClients()`

mostra que vamos crirar o procedure com este nome, sem parâmetros

3. `BEGIN` (…) `END`

corpo do stored procedure. É onde se coloca as queries.

3. `DELIMITER ;`

muda o delimitador de novo para ;

idCliente	pNome	uNome	telefone	preferencia	maxRenda	prefQuartos
CR56	Adelina	Santos	253867493	Apartamento	350	NULL
CR62	Maria	Pereira	253987345	Apartamento	600	NULL
CR74	Miguel	Silva	253475283	Moradia	750	NULL
CR76	Joao	Alves	223987567	Apartamento	425	NULL

Utilização de variáveis declaração

```
DECLARE variable_name datatype(size) DEFAULT default_value;
```

- variable_name deve seguir as regras para definição de nomes de tabelas e colunas
- datatype: qualquer dos tipos de dados permitidos
- Ao declarar uma variável o seu valor inicial é NULL. Pode inicializar-se com a atribuição de um valor default

- Exemplos:
  - `DECLARE total_salario INT DEFAULT 0;`
  - `DECLARE x, y INT DEFAULT 0;`
  - `DECLARE @nr_clientes INT DEFAULT 0;`
    - Cria uma variável que é acessível em toda a sessão. Sem o @ é acessível apenas dentro do stored procedure em questão.

Utilização de variáveis atribuição

```
SET variable_name = value;
```

- Exemplos:
  - `SET total_salario = 10000;`
  - `SELECT SUM(salario) INTO total_salario FROM funcionario;`

```
DELIMITER //
CREATE PROCEDURE GetSumSalario()
  BEGIN
    DECLARE total_salario INT DEFAULT 0;
    SELECT SUM(salario) INTO total_salario FROM funcionario;
    SELECT total_salario;
  END //
DELIMITER ;

CALL GetSumSalario();
```

total_salario
102000

Utilização de parâmetros

```
MODE param_name param_type(param_size)
```

MODE:

- **IN**: parâmetros de entrada que são passados como argumento ao stored procedure. O seu valor é protegido – mesmo que se altere dentro do stored procedure, o valor inicial mantém-se.
- **OUT**: parâmetros de saída. O stored procedure não consegue aceder ao seu valor inicial. O valor calculado dentro do stored procedure é passado ao programa que o chamou.
- **INOUT**: parâmetro de entrada que pode ser modificado pelo stored procedure e o novo valor é passado ao programa que o chamou

Parâmetros Exemplo IN

```
DELIMITER //
CREATE PROCEDURE GetStoreByCity(IN city VARCHAR(30))
  BEGIN
    SELECT * FROM loja WHERE cidade = city;
  END //
DELIMITER ;
```

```
CALL GetStoreByCity('Porto');
```

idLoja	rua	cidade	codPostal
B007	Av. Aliados 2345	Porto	4321

```
CALL GetStoreByCity('Felgueiras');
```

idLoja	rua	cidade	codPostal
B002	R. Curral 23	Felgueiras	1234
B005	R. Bombeiros 12	Felgueiras	1234

Parâmetros Exemplo OUT

```
DELIMITER $$
CREATE PROCEDURE CountStaffByGender(
  IN gender CHAR(1),
  OUT total INT)
  BEGIN
    SELECT count(*) INTO total FROM funcionario WHERE genero = gender;
  END $$
DELIMITER ;
```

```
CALL CountStaffByGender('M',@totalM);
SELECT @totalM;
```

@totalM
2

```
CALL CountStaffByGender('F',@totalF);
SELECT @totalF;
```

@totalF
4

Parâmetros Exemplo INOUT

```
DELIMITER $$
CREATE PROCEDURE set_counter(INOUT count INT(4),IN inc INT(4))
  BEGIN
    SET count = count + inc;
  END$$
DELIMITER ;
```

```
SET @counter = 1;
CALL set_counter(@counter,1);
CALL set_counter(@counter,1);
CALL set_counter(@counter,5);
SELECT @counter;
```

@counter
8

IF

IF

```
IF expression THEN
   statements;
END IF
```

IF … ELSE

```
IF expression THEN
   statements;
ELSE
   else-statements;
END IF;
```

IF … ELSEIF … ELSE

```
IF expression THEN
   statements;
ELSEIF elseif-expression THEN
   elseif-statements;
...
ELSE
   else-statements;
END IF;
```

Exemplo

IF

```
DELIMITER $$
CREATE PROCEDURE GetStaffLevel(
  IN p_staffNR VARCHAR(4),
  OUT p_staffLevel VARCHAR(10))
  BEGIN
    DECLARE salarioLim double;
    SELECT salario INTO salarioLim
      FROM funcionario
      WHERE idFuncionario = p_staffNR;
    IF salarioLim >= 30000 THEN
      SET p_staffLevel = 'Tres';
    ELSEIF (salarioLim < 30000
    AND salarioLim >= 10000) THEN
      SET p_staffLevel = 'Dois';
    ELSEIF salarioLim < 10000 THEN
      SET p_staffLevel = 'Um';
    END IF;
  END$$
DELIMITER ;
```

```
CALL GetStaffLevel('SA9',@levelSA9);
SELECT @levelSA9;
```

@levelSA9
Um

```
CALL GetStaffLevel('SG37',@levelSG37);
SELECT @levelSG37;
```

@levelSG37
Dois

```
CALL GetStaffLevel('SL21',@levelSL21);
SELECT @levelSL21;
```

@levelSL21
Tres

CASE

SIMPLES

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

SEARCHED (SUBSTITUI O IF)

```
CASE
    WHEN condition_1 THEN commands
    WHEN condition_2 THEN commands
    ...
    ELSE commands
END CASE;
```

Exemplo

CASE (1)

```
DELIMITER $$
CREATE PROCEDURE GetDistrito(
  IN  p_propID CHAR(5),
  OUT p_distrito VARCHAR(50))
  BEGIN    DECLARE city VARCHAR(30);
    SELECT cidade INTO city
      FROM propriedade
      WHERE idPropriedade = p_propID;
    CASE city
      WHEN 'Felgueiras' THEN
        SET p_distrito = 'Porto';
      WHEN 'Porto' THEN
        SET p_distrito= 'Porto';
      ELSE
        SET p_distrito= 'Braga';
    END CASE;
  END$$
DELIMITER ;
```

```
SET @propID = 'PG36';

SELECT cidade into @city
FROM propriedade
WHERE idPropriedade = @propID;

CALL GetDistrito(@propID,@distrito);

SELECT @propID, @city, @distrito;
```

@propID	@city	@distrito
PG36	Braga	Braga

Exemplo

CASE (2)

```
DELIMITER $$
CREATE PROCEDURE GetStaffLevelCase(
    IN  p_staffNR VARCHAR(4),
    OUT p_staffLevel  VARCHAR(10))
    BEGIN
      DECLARE salarioLim double;

      SELECT salario INTO salarioLim
      FROM funcionario
      WHERE idFuncionario = p_staffNR;

      CASE
        WHEN salarioLim >= 30000 THEN
          SET p_staffLevel = 'Tres';
        WHEN (salarioLim < 30000
        AND salarioLim >= 10000) THEN
          SET p_staffLevel = 'Dois';
        WHEN salarioLim < 10000 THEN
          SET p_staffLevel = 'Um';
      END CASE;
END$$
DELIMITER ;
```

```
CALL GetStaffLevelCase('SA9',@levelSA9Case);
SELECT @levelSA9Case;
```

@levelSA9Case
Um

```
CALL GetStaffLevelCase ('SG37',@levelSG37Case);
SELECT @levelSG37Case;
```

@levelSG37Case
Dois

```
CALL GetStaffLevelCase ('SL21',@levelSL21Case);
SELECT @levelSL21Case;
```

@levelSL21Case
Tres

Ciclo

WHILE

```
DELIMITER $$
CREATE PROCEDURE test_mysql_while_loop()
    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 ;
```

```
WHILE expression DO
   statements
END WHILE
```

```
CALL test_mysql_while_loop();
```

str
1,2,3,4,5,

Ciclo

REPEAT

```
DELIMITER $$
 CREATE PROCEDURE mysql_test_repeat_loop()
   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 ;
```

```
REPEAT
 statements;
UNTIL expression
END REPEAT
```

```
CALL mysql_test_repeat_loop();
```

str
1,2,3,4,5,

Ciclo LOOP, LEAVE, ITERATE

```
DELIMITER $$
CREATE PROCEDURE test_mysql_loop()
  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 ;
```

Loop: cria um ciclo
Leave: sai do ciclo
Iterate: "salta" o restante código do ciclo e passa à próxima iteração

```
CALL test_mysql_loop();
```

str
2,4,6,8,10,

Stored Function

```
CREATE FUNCTION function_name(param1,param2,…)
  RETURNS datatype
  [NOT] DETERMINISTIC
  statements
```

- Não se pode especificar parâmetros IN, OUT ou INOUT. São todos IN.
- Pode retornar qualquer tipo de dados.
- Para os mesmos parâmetros de entrada, se a função retorna os mesmos resultados, é **determinística**. Caso contrário, é **não determinística**.
- No corpo da função é obrigatório definir pelo menos um return. A função termina assim que se atinja um return.

Exemplo

```
DELIMITER $$
CREATE FUNCTION StaffLevel(p_salario double)
  RETURNS VARCHAR(10)
  DETERMINISTIC
  BEGIN
    DECLARE lvl varchar(10);

    IF p_salario >= 30000 THEN
      SET lvl = 'Tres';
    ELSEIF (p_salario< 30000 AND p_salario >= 10000) THEN
      SET lvl = 'Dois';
    ELSEIF p_salario < 10000 THEN
        SET lvl = 'Um';
    END IF;

  RETURN (lvl);
END$$
DELIMITER ;
```

```
SELECT pNome, uNome, StaffLevel(salario)
FROM funcionario;
```

pNome	uNome	StaffLevel(salario)
Maria	Marques	Um
David	Ferreira	Dois
Ana	Santos	Dois
Susana	Silva	Dois
Joao	Alves	Tres
Julia	Borges	Um