Banco de Dados

Pós-Graduação em Ciência da Computação

Prof. Dr. Ronaldo Celso Messias Correia
ronaldo.correia@unesp.br

Stored Procedure e Triggers MySQL

Stored Procedure

- São funções escritas usando SQL e ficam armazenadas no servidor
- Podem receber parâmetros e executar funções complexas, retornando ou não as informações para o usuário
- Vantagens:
  - Minimizam o tráfego da rede
  - São mais rápidos, aproveitam a capacidade do servidor e da otimização do SGBD
  - Facilitam a manutenção. Pode ser acessada por diversos programas, se houver alguma modificação a ser feita, basta fazê-la no BD
  - A resposta curta é, sempre que você puder. Não existem desvantagens em se usar stored procedures.
- Limitações:
  - Passar qualquer informação variável para a stored procedure como parâmetros ou colocá-las em uma tabela que a stored procedure possa acessar.
  - A linguagem de escrita de stored procedures e triggers pode ser muito limitada para operações mais complexas.

Stored Procedures - Sintaxe

```
CREATE PROCEDURE sp_name ([parameter[,...]])
[characteristic ...]
BEGIN
<corpo da rotina>
END
[parameter:
  [ IN | OUT | INOUT ] nome_parametro tipo
tipo:
  Qualquer tipo de dados válidos no MySQL

characteristic:
LANGUAGE SQL
[NOT] DETERMINISTIC
SQL SECURITY {DEFINER | INVOKER}
COMMENT string

Corpo:
Declarações de procedimento em SQL válida
```

Select * from INFORMATION_SCHEMA.ROUTINES

- <nome> - é o nome da store procedure
- Param1, Param2 – parâmetros

  VendaDatas(Data1 Date, Data2 Date)
- Cada parâmetro é um parâmetro IN por default
  - IN – Parâmetro de entrada
  - OUT – Parâmetro de saída (será assumido NULL como valor de entrada)
  - INOUT – Entrada e Saída

Stored Procedure - Exemplo 1

```
DELIMITER $$
DROP PROCEDURE IF EXISTS simpleproc $$
CREATE PROCEDURE simpleproc()
BEGIN
    SELECT 'OLA';
END $$

DELIMITER ;
```

Para executar o procedimento

- CALL simpleproc()

O exemplo usa o comando delimiter para alterar o delimitador de instrução para antes da definição do procedure. Isto permite que o delimitador ';' (padrão) usado no corpo de procedure seja passado para o servidor em vez de ser interpretado pelo MySQL

Stored Procedure - Exemplo 2

```
DELIMITER //
CREATE PROCEDURE insere_cliente(v_nome VARCHAR(60), v_endereco VARCHAR(20))
BEGIN
    IF ((v_nome != '') AND (v_endereco != '')) THEN
    INSERT INTO cliente (nome, endereco) VALUES (v_nome, v_endereco);
ELSE
    SELECT 'NOME e ENDEREÇO devem ser fornecidos para o cadastro!' AS Msg;
END IF;
END;
```

Para executar o procedimento

- CALL insere_cliente ("jose da silva", "Rua das flores");

```
CREATE TABLE cliente (
  id int auto_increment primary key,
  nome varchar(60) not null,
  endereco varchar(40) not null,
  genero varchar(1)
)
```

Stored Procedure - Exemplo 3

```
DELIMITER //
CREATE PROCEDURE updatecliente(v_id int, v_nome VARCHAR(60), v_endereco VARCHAR(20))
BEGIN
    IF ((v_id > 0) AND (v_id !='' ) AND (v_nome != null) AND (v_endereco != '')) THEN
    UPDATE cliente SET nome = v_nome, endereco=v_endereco WHERE id = v_id;
ELSE
    SELECT 'NOME e ENDEREÇO devem ser fornecidos para o cadastro!' AS Msg;
END IF;
END;
```

Para executar o procedimento

- CALL updatecliente (10,'jose da silva xavier', 'Rua do Amor Perfeito');

Stored Procedure - Exemplo 4

```
DELIMITER //
CREATE PROCEDURE deletecliente(v_id int)
BEGIN
        IF ((v_id > 0) AND (v_id !='')) THEN
        DELETE FROM cliente WHERE id = v_id;
ELSE
        SELECT 'ID não informado" AS Msg;
END IF;
END;
```

Para executar o procedimento

- CALL deletecliente (10);

Stored Procedure - Exemplo 5

Exemplo de utilização de parâmetro tipo IN

```
CREATE PROCEDURE sp_in (p VARCHAR(10))
  SET @X = P;
```

Exemplo de utilização de parâmetro tipo OUT

```
CREATE PROCEDURE sp_out (OUT p VARCHAR(10))
  SET P = 'ola';
```

Exemplo de utilização de parâmetro tipo INOUT

```
CREATE PROCEDURE sp_inout (INOUT p int)
BEGIN
  SET @X = P * 2;
 SET  P = @X;
END;
```

Atividade Prática - CRUD Tabela Cliente

```
CREATE TABLE cliente (
   cod_cliente INTEGER(3) PRIMARY KEY NOT NULL,
   nome_cliente VARCHAR (40) NOT NULL,
   endereco_cliente VARCHAR(40) NOT NULL,
   cidade_cliente VARCHAR(40) NOT NULL,
   cidade_cep VARCHAR(30) NOT NULL,
   uf_cliente VARCHAR(2) NOT NULL,
   cgc_cliente VARCHAR(30) NOT NULL,
   ie_cliente VARCHAR(4) NULL
);
```

script banco de dados pedido - Documentos Google

Atividade Prática - CRUD Tabela Cliente

CREATE PROCEDURE insereCliente(.....)

CREATE PROCEDURE updateCliente(v_id int, ...)

CREATE PROCEDURE deleteCliente(v_id int)

Atividade Prática - CRUD Tabela Vendedor

```
CREATE TABLE vendedor (
   cod_vendedor INTEGER(3) PRIMARY KEY NOT NULL,
   nome_vendedor VARCHAR(40) NOT NULL,
   salario_vendedor INTEGER NOT NULL,
   comissao_vendedor CHAR NOT NULL
);
```

script banco de dados pedido - Documentos Google

Atividade Prática - CRUD Tabela Vendedor

CREATE PROCEDURE inserevendedor(.....)

CREATE PROCEDURE updatevendedor(v_id int, ...)

CREATE PROCEDURE deletevendedor(v_id int)

Atividade Prática - CRUD Tabela Produto

```
CREATE TABLE produto (
        cod_produto INTEGER(3) PRIMARY KEY NOT NULL,
        unidade_produto VARCHAR(3) NOT NULL,
        desc_produto VARCHAR(40) NOT NULL,
        valor_unitario INTEGER(8) NOT NULL
);
```

script banco de dados pedido - Documentos Google

Atividade Prática - CRUD Tabela Produto

CREATE PROCEDURE insereproduto(.....)

CREATE PROCEDURE updateproduto(v_id int, ...)

CREATE PROCEDURE deleteproduto(v_id int)

Variáveis de Usuário

- O MySQL suporta variáveis específicas da conexão com a sintaxe @nomevariável
- As variáveis não precisam ser inicializadas.
- Elas contém NULL por padrão e podem armazenar um valor inteiro, real ou uma string.
- Todas as variáveis de uma thread são automaticamente liberadas quando uma thread termina.
- Variáveis de usuários devem ser utilizadas em expressões onde são permitidas.
- O tipo padrão de uma variável é baseada no tipo da variável no início da instrução. (Assume-se que uma variável não atribuída possui o valor NULL e é do tipo STRING).

```
SET @variável= { expressão inteira | expressão real | expressão string }
[,@variável= ...]
```

Variáveis de Usuário

- Para atribuir um valor a uma variável em outras instruções diferentes de SET utilizar o operador de atribuição := em vez de =, porque = é reservado para comparações em instruções diferentes de SET:

```
SET @t1=0, @t2=0, @t3=0;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
+------------------------------+-------+-------+-------+
| @t1:=(@t2:=1)+@t3:=4 | @t1  | @t2  | @t3  |
+------------------------------+-------+-------+-------+
|                            5 |     5 |     1 |     4 |
+------------------------------+-------+-------+-------+

mysql> SET @X='oi';
mysql> SELECT @X
```

Variáveis Locais

- Uma variável local somente será válida durante a execução de um procedimento armazenado, seja ele uma Stored Procedure, uma Trigger ou Stored Function, sendo que, após o término da execução de tais procedimentos, esta variável é destruída da memória, juntamente com seu respectivo valor.
- Para ser declarada, precisa estar entre os chamados compound statements, ou comandos aninhados dentro de um procedimento qualquer, que por sua vez também são chamados de Stored Routines.
- Para se declarar uma variável local, é necessário estarmos posicionados entre BEGIN ... END

```
DELIMITER //
CREATE PROCEDURE SP_TEST(IN num INT)
BEGIN
        DECLARE x INT DEFAULT 0;
        SET x = num;
END;
```

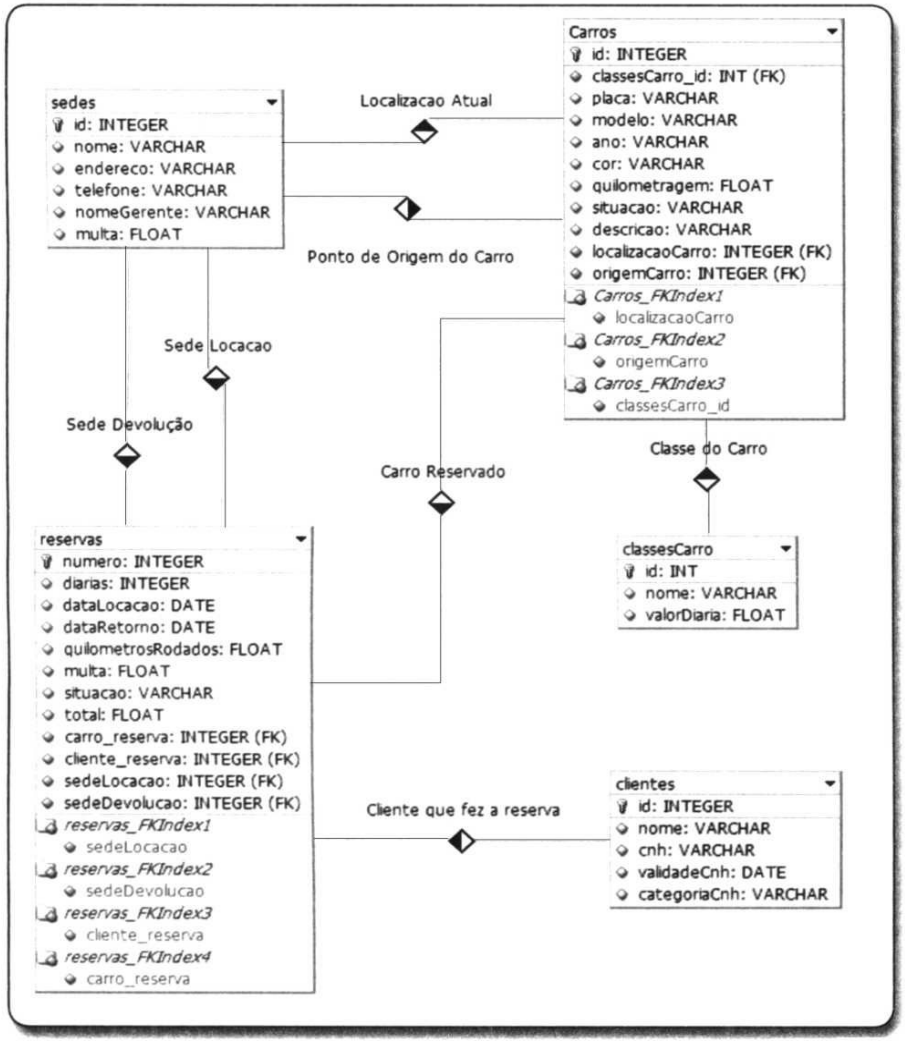
MySQL :: Manual de Referência do MySQL 8.4 :: 15.6.4.2 Escopo e Resolução de Variáveis Locais

Variáveis Locais

- Variáveis locais não são case sensitives, ou seja, VAR, var, VaR e vAr são a mesma coisa. O que devemos nos atentar é quanto ao seu escopo.
- O escopo das variáveis locais no MySQL é definido pela sua posição, aninhada em blocos de um procedimento armazenado no servidor de bancos de dados MySQL.

```
CREATE PROCEDURE SP_ESCOPO()
BEGIN
        DECLARE x INT DEFAULT 0;
        BEGIN
                DECLARE x INT DEFAULT 0;
                BEGIN
                        DECLARE x INT;
                END;
                SET x = NULL;
        END;
        SELECT x  AS Var;
END;
```

l de Carros


Carros
id: INTEGER
classesCarro_id: INT (FK)
placa: VARCHAR
modelo: VARCHAR
ano: VARCHAR
cor: VARCHAR
quilometragem: FLOAT
situacao: VARCHAR
descricao: VARCHAR
localizacaoCarro: INTEGER (FK)
origemCarro: INTEGER (FK)
Carros_FKIndex1
localizacaoCarro
Carros_FKIndex2
origemCarro
Carros_FKIndex3
classesCarro_id
sedes
id: INTEGER
nome: VARCHAR
endereco: VARCHAR
telefone: VARCHAR
nomeGerente: VARCHAR
multa: FLOAT
Localizacao Atual
Ponto de Origem do Carro
Sede Locacao
Sede Devolução
Carro Reservado
Classe do Carro
reservas
numero: INTEGER
diarias: INTEGER
dataLocacao: DATE
dataRetorno: DATE
quilometrosRodados: FLOAT
multa: FLOAT
situacao: VARCHAR
total: FLOAT
carro_reserva: INTEGER (FK)
cliente_reserva: INTEGER (FK)
sedeLocacao: INTEGER (FK)
sedeDevolucao: INTEGER (FK)
reservas_FKIndex1
sedeLocacao
reservas_FKIndex2
sedeDevolucao
reservas_FKIndex3
cliente_reserva
reservas_FKIndex4
carro_reserva
classesCarro
id: INT
nome: VARCHAR
valorDiaria: FLOAT
Cliente que fez a reserva
clientes
id: INTEGER
nome: VARCHAR
cnh: VARCHAR
validadeCnh: DATE
categoriaCnh: VARCHAR

Stored Procedure - Exemplo 6

Sistema para aluguel de carros - LOCAÇÃO

```
DELIMITER //
CREATE PROCEDURE registrarlocacao(v_diarias int, v_datalocacao DATE, v_carro int, v_sede int)
BEGIN
// Inserindo registro de nova reserva
INSERT INTO reservas (diarias, datalocacao, carro_reserva, sedelocacao) VALUES (v_diarias, v_datalocacao, v_carro, v_sede)

// Atualizando a situação do carro
UPDATE carros SET situacao = “alugado” WHERE id = v_carro
END;
```

Stored Procedure - Exemplo 7

Sistema para aluguel de carros - DEVOLUÇÃO

```
DELIMITER //
CREATE PROCEDURE registrardevolucao(v_reserva int, v_dataretorno DATE, v_quilometrosrodados float(8,2), v_multa float(10,2), v_total float(10,2), v_sedelocacao int, v_sededevolucao int))
BEGIN
// Atualizando registro de nova reserva
UPDATE reservas SET dataretorno = v_dataretorno, quilometrosrodados = v_quilometrosrodados, multa = v_multa, total = v_total, sededevolucao = v_sededevolucao WHERE id = v_reserva;
// Atualizando a situação do carro
IF (v_sedelocacao != v_sededevolucao) THEN
   UPDATE carros SET situacao = "fora do ponto de origem" WHERE id = v_carro;
ELSE
  UPDATE carros SET situacao = "disponivel" WHERE id = v_carro;
END IF
END;
```

Desvios Condicionais e Iterações

IF THEN ELSE

```
create procedure sp_lista_clientes(in opcao integer)
begin
   if opcao = 0 then
      select * from cliente where genero= 'F';
   else
      if opcao = 1 then
        select * from cliente where genero = 'M';
      else
        select * from cliente;
      end if;
   end if;
end $$
```

```
IF condition THEN
   statement/;s
  ELSE statement/s;
ENDIF
```

Desvios Condicionais e Iterações

- CASE

```
create procedure sp_lista_clientes(in opcao integer)
begin
   CASE opcao
        WHEN 0 THEN select * from cliente where sexo = 'F';
        WHEN 1 THEN  select * from cliente where sexo =
'M';
      else
          select * from cliente;
   END CASE;
END
```

```
CASE variable
  WHEN condição1 statement/s;
  WHEN condição2 statement/s;
ELSE
  statement/s
END CASE
```

Desvios Condicionais e Iterações

- REPEAT UNTIL

```
create procedure sp_repeat(in var1 integer)
begin
  REPEAT
      SELECT var1;
      set var1 = var1 + 1;
      UNTIL var1 > 5
    END REPEAT;
END;
```

```
REPEAT
   statement/s;
UNTIL condição
END REPEAT
```

Desvios Condicionais e Iterações

- WHILE

```
create procedure sp_while(in var1 integer)
begin
    WHILE (var1 < 20) DO
      SELECT var1;
      set var1 = var1 + 1;
    END WHILE;
END;
```

```
WHILE condição DO
   statement/s;
END WHILE
```

Tratamento de Erros

```
CREATE TABLE cliente (
   id int primary key,
   nome varchar(60) not null,
   cpf varchar(11) not null unique,
   email varchar(50) not null unique,
   genero varchar(1)
);
insert into cliente(id, nome, cpf, email) values
(2, 'maria ', '123456', 'maria@gmail.com');
```

Manipuladores e Cursores

- Com o advento das stored procedures, certas condições podem exigir tratamento específico. Estas condições podem ser relacionadas a erros, bem como controle de fluxo geral dentro da rotina.
- Manipuladores permitem executar declarações caso certa condição esteja presente
- Cursores permitem iterar através de um resultset, processando-o linha a linha.
- Cursores: O termo é um acrônimo para CURrent Set Of Records (conjunto de registros corrente)
  - São utilizados para posicionar um ponteiro em uma linha específica e podem permitir atualizações para as linhas com base na posição atual (O MySQL não suporta)

Tratamento de Erros - Condições (Condition)

MySQL :: Manual de Referência do MySQL 4.1 :: 13.1 Erros Retornados (nust.na)

- O tratamento de erros dentro de procedimentos armazenados no MySQL é baseado em **condições** .
- Caso uma determinada condição for atendida, um erro que podemos personalizar, baseado nos tipos de condições existentes, será disparado.
  - Erros: violação de uma chave primária ou índice único, violação de integridade referencial ou mesmo um WARNING em meio ao processamento do procedimento.

```
DECLARE condition_name CONDITION FOR condition_value

condition_value:
    SQLSTATE [VALUE] sqlstate_value  | mysql_error_code
```

- Permite associar um nome simbólico a uma condição de erro (MySQL error code ou SQLSTATE).
- Facilita a compreensão e reutilização em múltiplos handlers.
- Deve ser declarada antes dos cursores e handlers.

Tratamento de Erros

- **SQLSTATE: padrão SQL ANSI/ISO**
  - String de 5 caracteres (ex: '23000', '42S02')
  - Definido pelo padrão SQL (ANSI/ISO)
  - Portável entre diferentes bancos de dados
  - Representa categorias de erro
    - SQLSTATE '23000' -- Violação de restrição (ex: chave primária duplicada)
    - SQLSTATE '42S02' -- Tabela não existe
- **error_code: código específico do MySQL**
  - Número inteiro (ex: 1062, 1051)
  - Definido exclusivamente pelo MySQL
  - Usado nas mensagens de erro exibidas pelo MySQL
  - Não é portável (outros SGBDs não usam os mesmos códigos).
  - Pode ser usado diretamente em DECLARE HANDLER, mas não em SIGNAL.

Tratamento de Erros

- **Usando error_code**
  - DECLARE duplicate_entry CONDITION FOR 1062;

- **Usando SQLSTATE**
  - DECLARE duplicate_entry CONDITION FOR SQLSTATE '23000';

- Um error_code MySQL geralmente tem um SQLSTATE correspondente
  - 1062 (error_code) → '23000' (SQLSTATE)
  - Mensagem: Entrada '%s' duplicada para a chave %d

- Ambos funcionam, mas o SQLSTATE é mais portável

MySQL :: MySQL 8.0 Reference Manual :: 15.6.7.1 DECLARE ... CONDITION Statement

Tratamento de Erros - Manipuladores

- Permite capturar erros, warnings ou condições específicas
- Rotina executada automaticamente para tratar condições específicas (como erros, avisos ou eventos) que ocorrem durante a execução de procedimentos armazenados, funções ou triggers, permitindo um controle personalizado sobre como o sistema deve responder a essas situações.

```
DECLARE handler_type HANDLER FOR condition_value[,...] sp_statement
```

MySQL :: MySQL 8.0 Reference Manual :: 13.6.7.2 DECLARE ... HANDLER Statement

Tratamento de Erros - Manipuladores

- Tipos de Ação: indica o que o MySQL deve fazer depois de executar o handler
  - CONTINUE: permite ao processo continuar depois que as ações do manipulador (handler) foram executados
  - EXIT: encerram imediatamente o bloco atual BEGIN/END

Tipo	Exemplo	Significado
Código de erro	`1051`, `1062`	Erros MySQL específicos.
SQLSTATE	`'23000'`, `'42S02'`	Valores padrão ANSI SQL.
Nome da condição	`no_such_table`	Declarado com `DECLARE CONDITION`.
`SQLWARNING`		Qualquer aviso (`SQLSTATE` inicia com `'01'`).
`NOT FOUND`		`SQLSTATE '02000'` (cursor sem dados).
`SQLEXCEPTION`		Qualquer erro (`SQLSTATE` ≠ `'00'`, `'01'`, `'02'`).

Tratamento de Erros - Manipuladores

- Tipos de Ação: indica o que o MySQL deve fazer depois de executar o handler
  - CONTINUE: permite ao processo continuar depois que as ações do manipulador (handler) foram executados
  - EXIT: encerram imediatamente o bloco atual BEGIN/END

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'
BEGIN
  -- tratamento de erro
END;
```

```
DECLARE chave_duplicada CONDITION FOR SQLSTATE '23000';
DECLARE CONTINUE HANDLER FOR chave_dulpicada
BEGIN
  -- Código de tratamento
END;
```

Tratamento de Erros - Exemplo 1

```
CREATE PROCEDURE manipula1()
BEGIN
  DECLARE coluna_desconhecida CONDITION FOR SQLSTATE '42S22';
  DECLARE EXIT HANDLER FOR coluna_desconhecida
  BEGIN
     SELECT 'erro de coluna desconhecida';
  END;
  SELECT coluna;
  SELECT ‘continua’;
END;
```

- É declarada uma condição chamada coluna_desconhecida, que irá surgir quando for atingido o SQLSTATE 42S22, que ocorre quando uma coluna é desconhecida
- O manipulador EXIT exibe a mensagem de erro.
- No corpo da procedure a declaração SELECT coluna (para ativar o código de erro) e SELECT ‘ continua’, que nunca será executada, pois a procedure será encerrada assim que a condição estiver presente

Tratamento de Erros - Exemplo 1

```
CREATE TABLE cliente (
   id int auto_increment primary key,
   nome varchar(60) not null,
   endereco varchar(40) not null,
   cpf varchar(11) not null unique,
   email varchar(50) not null unique,
   genero varchar(1)
);
```

Tratamento de Erros - Exemplo 1

```
DELIMITER $$
CREATE PROCEDURE insereCliente(v_nome VARCHAR(60), v_endereco VARCHAR(20),
v_cpf VARCHAR(11), v_email VARCHAR(50))
BEGIN
   DECLARE EXIT HANDLER FOR SQLSTATE '23000'
       BEGIN
              SELECT 'Violação de chave duplicada!' AS Msg;
       END;
       IF ((v_nome != '') AND (v_endereco != '')) THEN
            INSERT INTO cliente (nome, endereco, cpf,email) VALUES (v_nome, v_endereco, v_cpf,
v_email);
       ELSE
          SELECT 'NOME e ENDEREÇO devem ser fornecidos para o cadastro!' AS Msg;
       END IF;
END $$
```

Tratamento de Erros - Exemplo 2

```
CREATE PROCEDURE violacao_chave(IN v_id INT)
BEGIN
        DECLARE EXIT HANDLER FOR SQLSTATE '23000'
        BEGIN
                SELECT 'Violação de chave primária!' AS Msg;
        END;
        INSERT INTO cliente  SET id =v_id;
END;
```

- O SQLSTATE aparecerá sempre que um erro for enviado a um usuário quando este executa uma operação ilegal ou que viole a integridade dos dados.
- Quando tentar inserir uma informação duplicada em uma coluna que é chave primária de uma tabela, o erro 1062 com o SQLSTATE 23000 é retornado

Tratamento de Erros - Exemplo 3

```
CREATE PROCEDURE test.sp_2(IN num CHAR(1))
BEGIN
        DECLARE EXIT HANDLER FOR SQLWARNING
        BEGIN
                SELECT 'O dado foi truncado!' AS Msg;
        END;
        INSERT INTO test.tbl_1 SET id =num;
END;
```

- Ao enviar um dado do tipo CHAR para ser inserido em uma coluna do tipo INT, causará um WARNING e uma mensagem de truncamento dos dados, condição declarada no HANDLER com a condição EXIT que encerrará o procedure.

Cursores em MySQL

- O MySQL não suporta todas as recursos dos cursores
- Os Cursores no MySQL são não sensitivos (não devemos atualizar uma tabela enquanto estamos usando um cursor); são somente leitura (não podemos fazer atualizações usando a posição do cursor); e não rolantes (só podemos avançar para o próximo registro e não para traz e/ou para frente)
- São usados para acessar um resultset que possa recuperar uma ou mais linhas. São usados para posicionar um ponteiro em uma linha específica

```
DECLARE cursor_name CURSOR FOR sql_statement
```

- Vários cursores podem ser declarados num mesmo procedure

Cursores em MySQL - Exemplo 1

- OPEN sp_cursos1: ativa o cursor previamente declarado
- FETCH: retorna a próxima linha do resultset atual. Os resultados devem ser armazenados em algum lugar
  - As variáveis x e y armazenam as duas colunas retornadas pelo SELECT id, nome FROM cliente que compõe o cursor
- CLOSE sp_cursos1: fecha o cursor
- No exemplo acima apenas a primeira linha do resultset é retornada

```
CREATE PROCEDURE exemplo_cursor1 (OUT rid INT, OUT rnome INT)
        BEGIN
    DECLARE x,y INT;
    DECLARE sp_cursor1 CURSOR
                        FOR SELECT id, nome FROM cliente;
                OPEN sp_cursor1;
                        FETCH sp_cursor1 INTO x, y;
                CLOSE sp_cursor1;
                SET rid = x;
                SET rnome = y;
        END $
```

```
CREATE TABLE cliente1209 (
   id int auto_increment primary key,
   nome varchar(60) not null,
   endereco varchar(40) not null,
   cpf varchar(11) not null unique,
   email varchar(50) not null unique,
   genero varchar(1)
);
```

Cursores em MySQL - Exemplo 2

- clientes (id_cliente, nome)
- pedidos (id_pedido, id_cliente, data_pedido)
- itens_pedido (id_item, id_pedido, produto, quantidade, preco_unitario)

- Para cada cliente que teve pedidos no último mês, calcular o total gasto e armazenar isso em uma nova tabela chamada relatorio_gastos.

Cursores em MySQL - Exemplo 2

```
CREATE PROCEDURE exemplo_cursor2 (OUT rid INT, OUT rnome
VARCHAR(60))
        BEGIN
                DECLARE x, z INT;
                DECLARE y VARCHAR(60);
                DECLARE sp1_cursor CURSOR
                        FOR SELECT id, nome FROM cliente;
                DECLARE CONTINUE HANDLER FOR NOT FOUND SET z = 1;
                OPEN sp1_cursor;
                REPEAT
                        FETCH sp1_cursor INTO x, y;
                        UNTIL (z=1)
                END REPEAT;
                CLOSE sp1_cursor;
                SET rid = x;
                SET rnome = y;
        END
```

Cursores em MySQL - Exemplo 2 - Funcionando

```
CREATE PROCEDURE exemplo_cursor1 (OUT rid INT, OUT rnome VARCHAR(50))
BEGIN
    DECLARE x INT;
    DECLARE y VARCHAR(50);
    DECLARE FIM INT DEFAULT 0;
    DECLARE sp_cursor1 CURSOR
                FOR SELECT cod_cliente, nome_cliente FROM cliente;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET FIM = 1;
    OPEN sp_cursor1;
    REPEAT
        FETCH sp_cursor1 INTO x, y;
        select x,y;
    UNTIL (FIM = 1)
    END REPEAT;
    CLOSE sp_cursor1;
    SET rid = x;
    SET rnome = y;
END $$
```

Cursores em MySQL - Exemplo 2

- Para iterar pelo resultset inteiro e retornar os resultados utilizar um REPEAT UNTIL
- No exemplo 2 um manipulador é declarado com a condição NOT FOUND e atribui 1 à variável Z, sendo justamente z=1 a condição testada pelo REPEAT UNTIL
- A condição NOT FOUND inclui todos os erros com SQLSTATE que começam com 02, um dos quais é o erro NO DATA TO FETCH
- Outros tipos de condições também podem ser abordadas na criação de um HANDLER ou CONDITION:
  - Declarar explicitamente um código de SQLSTATE para tratar o erro;
  - Declarar o SQLWARNING e todos os SQLSTATES iniciados com 01 serão tratados;
  - Declarar NOT FOUND, mais comum em Cursors e Stored Functions, para tratamento de SQLSTATES iniciados com 02;
  - Declarar o SQLEXCEPTION que tratará erros de SQLWARNING ou NOT FOUND.

Cursores em MySQL - Exemplo 3

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE a CHAR(16);
  DECLARE b,c INT;
  OPEN cur1;
  OPEN cur2;
  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF NOT done THEN
      IF b < c THEN
        INSERT INTO test.t3 VALUES (a,b);
      ELSE
        INSERT INTO test.t3 VALUES (a,c);
      END IF;
    END IF;
  UNTIL done END REPEAT;
  CLOSE cur1;
  CLOSE cur2;
END
```

Triggers - Gatilhos

- São procedimentos especiais ativados quando ocorre uma inserção, atualização ou exclusão em uma tabela ou em uma view
- Diferem da stored procedures pelo fato de não serem chamados diretamente pelo usuário: quando ocorre um determinado evento na tabela, eles são executados
- Possibilitam:
  - Manter a integridade dos dados: atualizar tabelas associadas
  - Cria logs do sistema: a cada inclusão
  - Notificações de alterações no banco aos usuários
  - Validação de restrições de integridade mais complexas que as suportadas diretamente pelo SGBD

Triggers - Sintaxe

```
CREATE
[DEFINER = {user | CURRENT_USER}
TRIGGER nome_trigger tempo_trigger evento_trigger
ON nome_tabela FOR EACH ROW
BEGIN
    declaração_trigger
END
```

Triggers - Sintaxe

- DEFINER: Quando o TRIGGER for disparado, esta opção será checada para checar com quais privilégios este será disparado. Utilizará os privilégios do usuário informado em user (´ronaldo´@´localhost´) ou os privilégios do usuário atual (CURRENT_USER). Caso essa sentença seja omitida da criação do TRIGGER, o valor padrão desta opção é CURRENT_USER();
- Nome_trigger: define o nome do procedimento, por exemplo, trg_test;
- Tempo_trigger: define se o TRIGGER será ativado antes (BEFORE) ou depois (AFTER) do comando que o disparou;
- Evento_trigger: aqui se define qual será o evento, INSERT, REPLACE, DELETE ou UPDATE;
- nome_tabela: nome da tabela onde o TRIGGER ficará "pendurado" aguardando o trigger_event;
- declarações_trigger: as definições do que o TRIGGER deverá fazer quando for disparado.

Trigger - Exemplo 1

- Trigger que atualiza o campo VALOR da tabela VENDAS cada vez que se alterar a tabela de itens

Vendas (cod_pedido, codcliente, data, valor_total_venda)

Itens (cod_pedido, codproduto, qtde, preco_un, ValorTotal)

```
delimiter //
CREATE TRIGGER InsItem AFTER INSERT ON Itens
FOR EACH ROW
BEGIN
   UPDATE Vendas set valor_total_venda = valor_total_venda + New.ValorTotal
    WHERE cod_pedido = New.cod_pedido;
END //
```

O alias NEW indica o registro que está sendo inserido

Trigger - Exemplo 2

Sistema para aluguel de carros - Atualização da quilometragem rodada por um carro após sua devolução

```
DELIMITER //
CREATE TRIGGER tr_kmrodados after update on reservas
FOR EACH ROW
UPDATE carros SET quilometragem = quilometragem + NEW.quilometrosrodados
WHERE NEW.carro_reserva = carros.id;
```

Trigger - Exemplo 3

➢ Exclusão de um item da tabela

Vendas (cod_pedido, codcliente, data, valor_total_venda)

Itens (cod_pedido, codproduto, qtde, preco_un, ValorTotal)

```
delimiter //
CREATE TRIGGER DelItem AFTER DELETE ON Itens FOR EACH ROW
BEGIN
   UPDATE Vendas set valor_total_venda = valor_total_venda - OLD.ValorTotal
    WHERE cod_pedido = OLD.cod_pedido;
END //
```

O alias OLD indica o registro que está sendo apagado

Trigger - Exemplo 4

➢ Trigger para atualização de dados

Vendas (pedido , codcliente, data, valor)

Itens (pedido, codproduto, qtde, preco, ValorTotal)

```
Set term # ;
CREATE TRIGGER AtuItem AFTER UPDATE ON Itens
BEGIN
   IF (New.ValorTotal <> OLD.ValorTotal) THEN
      UPDATE Vendas set valor = valor - OLD.ValorTotal +     New.ValorTotal  WHERE
Pedido = OLD.Pedido;
END //
```

Exercícios

1 – Considerando a tabela auditoria_salario com os seguintes atributos:

func_codigo: código do funcionário

salário_inicial: salário antes de ser alterado

salário_alterado: novo salário do funcionario

data_alteração: data da alteração do salário

nome_usuário: usuário que realizou a alteração do salário do funcionário

funcionario (cod_func, nome_func, data_nascimento, endereco, salario)

Criar um trigger que, ao alterar o salário de um empregado, registrar corretamente na tabela auditoria_salario as atualizações.

Exercícios

2- Criar uma stored procedure chamada alteraSalFunc, que altera o salário de um funcionário de acordo com o número de dependentes que ele possui. Por exemplo, se o funcionário possuir 1 dependente, ele terá um aumento de 10%; se o funcionário possuir 2 dependentes, ele terá um aumento de 20%; e assim por diante. Deve ser passado como parâmetro o código do funcionário, e a função deve retornar a porcentagem de aumento do salário do empregado, além de atualizar o salário do funcionário de forma adequada no banco de dados.

funcionario (idfunc, nome, salario, data_admissao, cpf, rg)

dependentes (idfunc, nome_dependente, parentesco, data_nascimento)

SIGNAL

- Instrução SQL padrão para lançar uma exceção.
- Usada em triggers, procedures e functions.
- Permite interromper uma operação quando uma regra de negócio não é atendida.
- Aborta a instrução que o disparou.
- Reverte todos os efeitos dessa instrução (atomicidade por instrução).
- **Sintaxe Básica**

```
SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Mensagem de erro';
```

SIGNAL - Triggers

➢ Exemplo 1 – Validação antes de inserir - Impedir inserção de cliente menor de 18 anos.

```
DELIMITER $$

CREATE TRIGGER trg_cliente_idade
BEFORE INSERT ON clientes
FOR EACH ROW
BEGIN
  IF NEW.cli_idade < 18 THEN
     SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cliente deve ter no mínimo 18 anos.';
  END IF;
END$$

DELIMITER ;
```

SIGNAL - Triggers

➢ Exemplo 2 – Impedir preço abaixo do mínimo

```
DELIMITER $$

CREATE TRIGGER trg_valida_preco_produto
BEFORE UPDATE ON produtos
FOR EACH ROW
BEGIN
  IF NEW.prd_preco < NEW.prd_preco_minimo THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Preço informado está abaixo do valor mínimo permitido.';
  END IF;
END$$

DELIMITER ;
```

SIGNAL - Procedures

➢ Exemplo 3 - Verificar saldo antes de saque

```
CREATE PROCEDURE sacar(IN p_conta INT, IN p_valor DECIMAL(10,2))
BEGIN
  DECLARE v_saldo DECIMAL(10,2);
  SELECT saldo INTO v_saldo FROM contas WHERE conta_id = p_conta;
  IF v_saldo IS NULL THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Conta inexistente.';
  ELSEIF v_saldo < p_valor THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Saldo insuficiente para saque.';
  ELSE
    UPDATE contas SET saldo = saldo - p_valor WHERE conta_id = p_conta;
  END IF;
END$$
DELIMITER ;
```

SIGNAL - Procedures

➢ Exemplo 4 - Inserir cliente com e-mail único

```
DELIMITER $$

CREATE PROCEDURE inserir_cliente(IN p_nome VARCHAR(100), IN p_email VARCHAR(100))
BEGIN
  IF EXISTS (SELECT 1 FROM clientes WHERE cli_email = p_email) THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'E-mail já cadastrado. Utilize outro.';
  ELSE
    INSERT INTO clientes (cli_nome, cli_email) VALUES (p_nome, p_email);
  END IF;
END$$

DELIMITER ;
```