

AGENDA 2

FUNÇÕES

```
static bool init(CURL *conn, char *url) {
    CURLcode code;
    conn = curl_easy_init();
    if (conn == NULL) {
        fprintf(stderr, "Failed to create CURL connection\n");
        exit(EXIT_FAILURE);
    }
    code = curl_easy_setopt(conn, CURLOPT_ERRORBUFFER,
        errorBuffer);
    if (code != CURLE_OK) {
        fprintf(stderr, "Failed to set error buffer [%d]\n",
            code);
        return false;
    }
    code = curl_easy_setopt(conn, CURLOPT_URL, url);
    if (code != CURLE_OK) {
        fprintf(stderr, "Failed to set URL [%s]\n", errorBuffer);
        return false;
    }
    code = curl_easy_setopt(conn, CURLOPT_FOLLOWLOCATION,
        1L);
    if (code != CURLE_OK) {
        fprintf(stderr, "Failed to set redirect option [%s]\n",
            errorBuffer);
        return false;
    }
    code = curl_easy_setopt(conn, CURLOPT_WRITEFUNCTION,
        writer);
    if (code != CURLE_OK) {
        fprintf(stderr, "Failed to set writer [%s]\n",
            errorBuffer);
        return false;
    }
    code = curl_easy_setopt(conn, CURLOPT_WRITEDATA,
        &buffer);
    if (code != CURLE_OK) {
        fprintf(stderr, "Failed to set write data [%s]\n",
            errorBuffer);
        return false;
    }
    return true;
}

static void StartElement(void *voidContext,
    const xmlChar *name,
    const xmlChar **attributes) {
    Context *context = (Context *)voidContext;
    if (COMPARE((char *)name, "TITLE")) {
        context->title = "";
        context->addTitle = true;
    }
    (void) attributes;
}

// Libxml end element callback function
static void EndElement(void *voidContext,
    const xmlChar *name) {
    Context *context = (Context *)voidContext;
    if (COMPARE((char *)name, "TITLE")) {
        context->addTitle = false;
    }
    // Text handling helper function
}

static void handleCharacters(Context *context,
    const xmlChar *chars,
    int length) {
    if (context->addTitle) {
        context->title.append((char *)chars, length);
    }
    // Libxml PCDATA callback function
}

static void Characters(void *voidContext,
    const xmlChar *chars,
    int length) {
    Context *context = (Context *)voidContext;
    handleCharacters(context, chars, length);
}

static void cdata(void *voidContext,
    const xmlChar *chars,
    int length) {
    Context *context = (Context *)voidContext;
    handleCharacters(context, chars, length);
}
```

GEEaD - Grupo de Estudos de Educação a Distância
Centro de Educação Tecnológica Paula Souza

GOVERNO DO ESTADO DE SÃO PAULO
EIXO TECNOLÓGICO DE INFORMAÇÃO E COMUNICAÇÃO
CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS
TECNOLOGIAS DA INFORMAÇÃO III

Expediente

Autor:

José Mendes da Silva Neto

Revisão Técnica:

Eliana Cristina Nogueira Barion

Revisão Gramatical:

Juçara Maria Montenegro Simonsen Santos

Editoração e Diagramação:

Flávio Biazim



MERGULHANDO
NO TEMA...



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.

São dois tipos de rotinas armazenadas, um deles você já conheceu na agenda anterior, os **Procedures**, o outro são as **Funções**, até são um pouco similares, mas possuem aplicações diferentes.

São invocadas de formas diferentes também (call x declaração). 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 ainda permite que criemos nossas próprias funções, conforme demonstrado a seguir:

Sintaxe para criação de uma **Function** no **MySQL**:

```
delimiter $$
create function nome_funcao ([parâmetros])
returns tipo_dados
begin
    /*corpo da função*/
return <expressão ou valor ou conteúdo da variável de retorno>;
end $$
delimiter ;
```

Onde:

nome_funcao: nome que identificará a **função**. Este nome segue as mesmas regras para definição de variáveis, não podendo iniciar com número ou caracteres especiais (exceto o underline “_”).

parâmetros: são opcionais e, caso não sejam necessários, devem permanecer apenas os parênteses vazios na declaração da **function**. Para que uma **function** receba parâmetros, é necessário utilizá-los dentro dos parênteses.

returns: define o tipo de conteúdo que será retornado pela função.

return: define a expressão utilizada para obter o resultado da função, pode também ser um valor ou uma variável. O resultado da expressão, o valor ou a variável, devem possuir conteúdo compatível com o tipo de dado da cláusula **returns**.

Sintaxe de declaração de parâmetros em **Functions**:

```
(nome tipo, nome tipo, nome tipo)
```

Onde:

nome: nome do parâmetro, também segue as mesmas regras de definição de variáveis.

tipo: nada mais é do que o tipo de dado do parâmetro (**int**, **varchar**, **decimal**, etc).

Como invocar (chamar) uma Função:

```
select nome_funcao([parâmetros]);
```

IMPORTANTE: para aplicar os exemplos a seguir utilize o banco de dados minimercado.

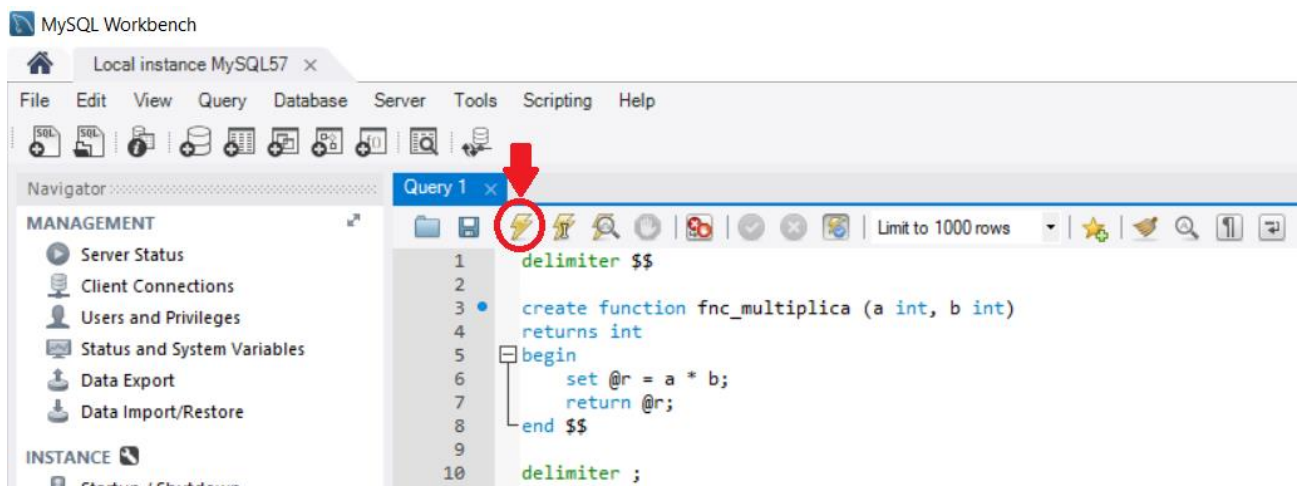
A seguir temos alguns exemplos do uso das **Funções**:

Exemplo 1:

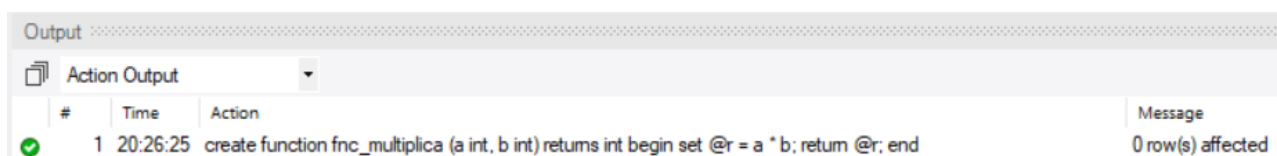
Vamos criar uma função que receba dois valores inteiros e retorne o resultado da multiplicação entre eles:

```
delimiter $$
create function fnc_multiplica (a int, b int)
returns int
begin
    set @r = a * b;
    return @r;
end $$
delimiter ;
```

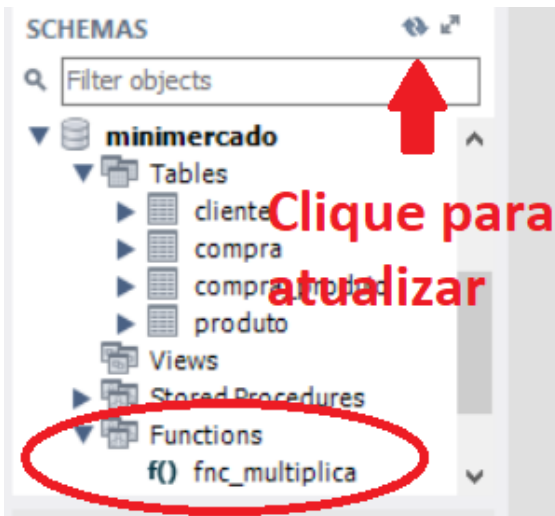
Para criar a **function** pela janela SQL:



Function criada com sucesso!!!



Verifique como ficou a estrutura:



IMPORTANTE: Caso tenha alguma dúvida, clique [aqui](#) para assistir a um vídeo que mostrará como criar a function pela janela SQL.

O objetivo dessa função é obter o resultado da multiplicação entre dois números inteiros. Assim, caso desejássemos multiplicar dois valores, poderíamos usar a **function** como mostra o código a seguir:

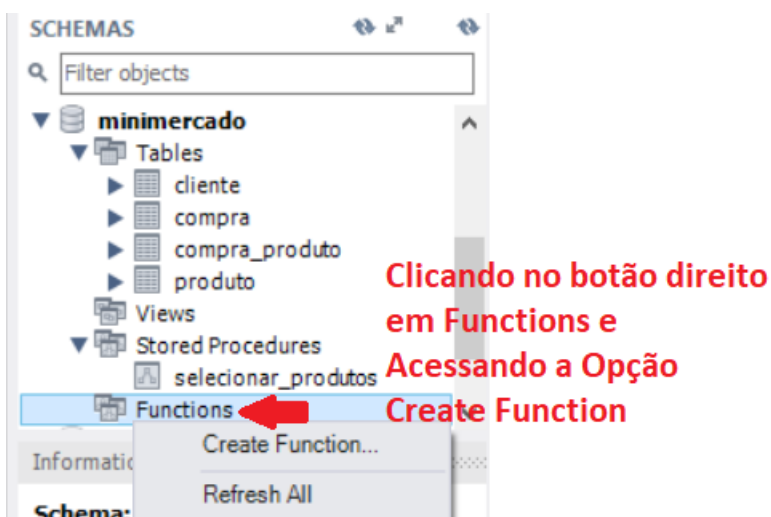
Chamando a função:

```
select fnc_multiplica (8, 10) resultado;
```

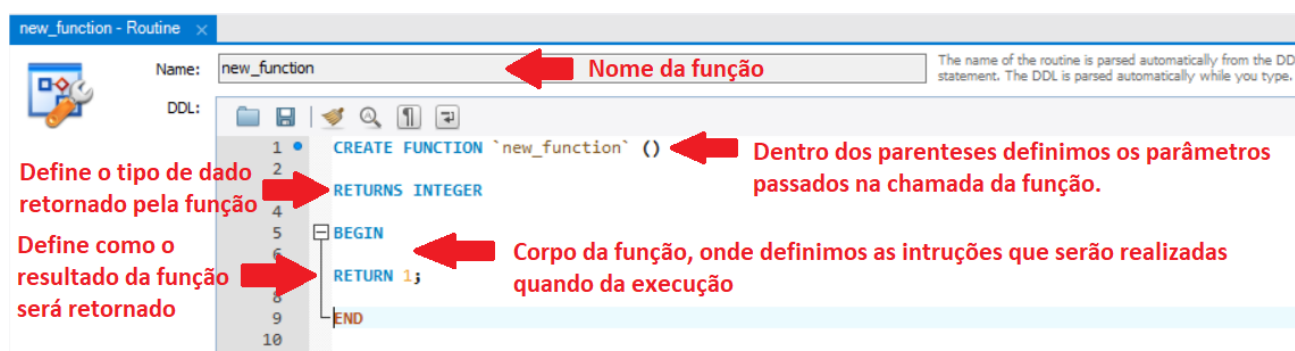
clique no botão  para executar



Você também pode criar uma **function** utilizando a interface gráfica do Workbench:



É apresentada a interface para criação de **functions**, onde você deve definir o nome, os parâmetros e as instruções que deverão ser executadas.



Quando temos uma aplicação que faz um **select** por diversas vezes durante um código, ou seja, se você escreve esse **select** mais de uma vez, fica muito mais simples fazer uma futura mudança apenas na **procedure** ou **function** porque durante o código você só a chamou. Caso contrário, você teria uma amarração por variável em seu código, não acho essa solução elegante, ou teria que percorrer todo ele para fazer as trocas.

Retirado de <https://www.gigasystems.com.br>

Exemplo 2:

Na estrutura da **tabela compra_produto**, temos os campos **quantidade** e **preco**. Vamos criar uma função que calcule o preço total a partir do conteúdo desses dois campos.

COMPRA_PRODUTO							
nome	tipo de dados	tamanho	obrigatório	único	chave primária	chave estrangeira	valor default
id_compra_produto	INT	11	sim	sim	sim	não	não
codigo_compra	INT	11	sim	não	não	sim	não
codigo_produto	INT	5	sim	não	não	sim	não
quantidade	DOUBLE	10,1	sim	não	não	não	1
preco	DOUBLE	10,2	sim	não	não	não	não

Fique à vontade para criar a função do jeito que achar mais fácil, pela janela SQL ou pela interface gráfica, no quadro **SCHEMAS**, utilizando o seguinte código:


```

delimiter $$
create function fnc_preco_total (vl_unitario double, quantidade double)
returns double
begin
    set @r = vl_unitario * quantidade;
    return @r;
end $$
delimiter ;

```

Obs: @r variável utilizada para receber o resultado da multiplicação e ser retornada como resultado da função.

Clique no botão  para criar a function.

Output		
Action Output		
#	Time	Action
1	19:53:21	create function fnc_preco_total (vl_unitario double, quantidade double) returns double begin set @r = vl_unitario * quantidade; return @r; end

Após a criação da **function** vamos utilizá-la apresentando todos os produtos que foram vendidos com o preço total de cada venda, calculado por meio da **function** `fnc_preco_total`.

```

select cp.*
       , fnc_preco_total(cp.preco, cp.quantidade) preco_total
from compra_produto cp

```

Clique no botão  para executar a select.

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	id_compra_produto	codigo_compra	codigo_produto	quantidade	preco	preco_total
1	1	1	2.0	8.90	17.8	
2	1	2	2.0	5.40	10.8	
3	2	3	1.0	2.45	2.45	
4	2	4	2.0	3.30	6.6	
5	2	5	1.0	4.70	4.7	

Veja que utilizamos a **function** passando somente os campos como parâmetros, conforme definido em sua criação:

```
create function fnc_preco_total (vl_unitario double, quantidade double)
```

A cada registro que é apresentado, o MySQL passa os valores para a **function** apresentando o resultado, aqui representado pela coluna `preco_total`, definida como **alias** (apelido) do retorno.

Result Grid Filter Rows: Export: Wrap Cell Content:						
	id_compra_produto	codigo_compra	codigo_produto	quantidade	preco	preco_total
	1	1	1	2.0	8.90	17.8
	2	1	2	2.0	5.40	10.8
	3	2	3	1.0	2.45	2.45
	4	2	4	2.0	3.30	6.6
	5	2	5	1.0	4.70	4.7

Os campos `quantidade` e `preco` são multiplicados um pelo outro por meio da **function** para obter o conteúdo da coluna `preco_total`.

Exemplo 3:

Podemos utilizar as **functions** para obter resultados a partir de consultas executadas dentro delas. Vamos criar uma **function** para obter o valor total de um compra.

```
delimiter $$
create function fnc_valor_compra (cod_compra int)
returns double
begin
    declare total_compra double;
    set total_compra = (select sum(fnc_preco_total(cp.quantidade, cp.preco))
                        from compra_produto cp
                        where cp.codigo_compra = cod_compra);
    return total_compra;
end $$
delimiter ;
```

Veja que utilizamos uma outra forma de declaração de variável:

```
declare total_compra double;
```

E para obter o total de uma compra, achamos o preço total de cada produto vendido, por meio da **function** `fnc_preco_total`, antes de acumular o valor a partir da **function** `SUM`, do **SQL**.

```
select fnc_valor_compra(1) total_compra;
```

Obs.: O número **1 (um)** é o parâmetro correspondente ao identificador da compra que se deseja obter o total, o que significa que somente os registros dessa compra serão considerados na execução da função:

	id_compra_produto	codigo_compra	codigo_produto	quantidade	preco
	1	1	1	2.0	8.90
	2	1	2	2.0	5.40
	3	2	3	1.0	2.45
	4	2	4	2.0	3.30
	5	2	5	1.0	4.70

Clique no botão  para executar a `select`.

Result Grid		 Filter Rows: <input type="text"/>	Export: 	Wrap Cell Content: 
	total_compra			
	28.6			

Para que você entenda melhor como chegamos nesse valor total da compra, é importante que você compreenda que esse valor é obtido a partir da execução da **function** `fnc_preco_total` em cada um dos registros.

	id_compra_produto	codigo_compra	codigo_produto	quantidade	preco
	1	1	1	2.0	8.90
	2	1	2	2.0	5.40

Somando o valor obtido de cada registro a partir da execução da **function** `SUM`:

	id_compra_produto	codigo_compra	codigo_produto	quantidade	preco	preco_total
	1	1	1	2.0	8.90	17.8
	2	1	2	2.0	5.40	10.8

Chegando assim ao valor total da compra de R\$ 28,60, tudo isso executado pela **function** `fnc_valor_compra` apresentando somente o resultado, deixando dentro dela as regras para obtenção desse valor. Com isso, qualquer alteração na regra para obtermos o valor total de uma compra, será necessário somente fazê-la na **function**, assim todas as chamadas serão automaticamente afetadas, o que faz parte das melhores práticas de programação.

Agora é com você!!!



Durante a implementação de um projeto com certeza iremos identificar validações, verificações e cálculos que são executados mais de uma vez ou que possuem regras muito específicas referente ao negócio onde o Sistema está ou será implantado.

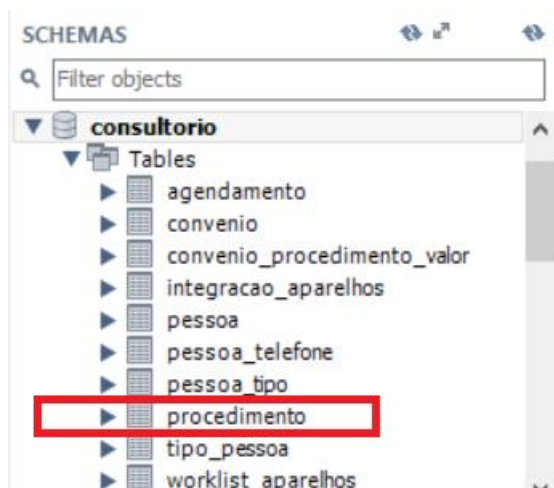
IMPORTANTE: Utilize o banco de dados do consultório para desenvolver os exemplos.

Carlos, analista responsável pelo projeto de integração dos Sistemas no Consultório da Dra. Ana Lúcia criou as estruturas e os processos para inclusão dos atendimentos na **worklist** dos aparelhos, além da confirmação de realização dos exames. Por se tratar de uma integração realizada sem a alteração de nenhum dos Sistemas, somente com a implementação de objetos no Banco de Dados, o próximo passo de Carlos é identificar em que ponto do processo a criação de **functions** pode otimizar o seu trabalho.

Para iniciar sua análise ele relacionou alguns requisitos que deverão ser implementados na integração:

- 1 - Somente serão integrados os agendamentos recepcionados para realização de procedimentos de imagem.
- 2 - Valorizar o procedimento após a confirmação de sua realização de acordo com o convênio.

Você consegue auxiliar o Carlos a utilizar as **functions** na melhoria desses processos? Vamos ver!!!



Com relação ao primeiro requisito, o objetivo é identificar se o procedimento recepcionado é um exame de imagem, porque somente esses serão integrados aos aparelhos. Como esse dado é relacionado ao procedimento, precisamos analisar as estruturas do Banco de Dados e descobrir qual delas possui a informação necessária para conseguirmos identificar se um procedimento é de imagem.

Temos uma tabela **procedimento**, já é uma dica por onde devemos começar.

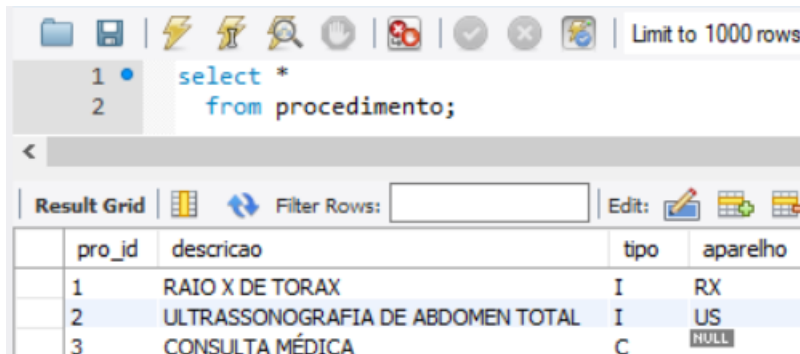
Neste caso o melhor a fazer é obter uma descrição da estrutura da tabela, assim teremos uma visão geral de seus atributos, caso necessário, liste também os registros armazenados, assim terá uma ideia melhor do que é preenchido em cada um deles.

Para obter a descrição da estrutura, utilize o comando **describe**:

The image shows a database query tool interface. The command 'describe procedimento;' is entered in the query editor. Below the query, the 'Result Grid' displays the table structure with columns: Field, Type, Null, Key, Default, and Extra.

Field	Type	Null	Key	Default	Extra
pro id	int(5)	NO	PRI	NULL	auto increment
descricao	varchar(50)	NO		NULL	
tipo	varchar(1)	NO		NULL	
aparelho	varchar(2)	YES		NULL	

Veja que a estrutura possui um campo **tipo**, preenchido com a letra I, indicando que é um procedimento de Imagem ou com a letra C, indicando que é uma Consulta, conforme demonstrado a seguir na relação de registros armazenados, obtida pelo comando `select`:



The screenshot shows a database query tool interface. At the top, there's a toolbar with various icons and a 'Limit to 1000 rows' option. Below the toolbar, a SQL query is entered: `select * from procedimento;`. The results are displayed in a table with the following columns: `pro_id`, `descricao`, `tipo`, and `aparelho`.

pro_id	descricao	tipo	aparelho
1	RAIO X DE TORAX	I	RX
2	ULTRASSONOGRAFIA DE ABDOMEN TOTAL	I	US
3	CONSULTA MÉDICA	C	NULL

Agora que já sabemos como um procedimento é classificado, vamos criar uma **function** que receba o identificador do procedimento e nos retorne o seu tipo.

```
delimiter $$
create function fnc_tipo_proc (cod_proc int)
returns varchar(1)
begin
    declare tipo_proc varchar(1);
    set tipo_proc = (select p.tipo
                    from procedimento p
                    where p.pro_id = cod_proc);
    return tipo_proc;
end $$
delimiter ;
```

Após a criação da **function**, execute-a passando como parâmetro o identificador do procedimento, conforme definido no código da **function**:

```
create function fnc_tipo_proc (cod_proc int)
```

Utilize o seguinte comando para executá-la.

```
select fnc_tipo_proc(1) tipo_procedimento;
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
tipo_procedimento			
I			

Com a criação da **function** `fnc_tipo_proc`, quando necessitarmos verificar se um procedimento é de imagem, precisamos apenas chamá-la, passando como parâmetro o identificador do procedimento.

Já no segundo requisito, podemos considerá-lo como uma melhoria, pois na Agenda anterior de Banco de Dados, o **procedure** `confirmar_realizacao_exame`, já faz a valorização. O que precisamos fazer é apenas organizar as ações, valorizar o procedimento é apenas uma das ações que devem ser realizadas pelo **procedure**, nesse caso o melhor é retirá-lo e criar uma **function** específica de valorização do procedimento, centralizando a regra em um só lugar. Imagine se você precisar desenvolver uma consulta para obter os valores a receber por procedimentos que serão realizados. Chamar o **procedure** `confirmar_realizacao_exame` não o ajudará, pois o que você precisa é somente valorizar o procedimento. Criando a **function** somente para valorizar o procedimento, você poderá utilizá-la tanto no **procedure** `confirmar_realizacao_exame` como na consulta para obter os valores a receber.

Entendeu??? Tenho certeza que sim. Sabemos que o valor do procedimento é definido pelo convênio, baseado nisso, vamos criar uma **function** que receba o identificador do procedimento e do convênio e nos retorne o valor.

```
delimiter $$
create function fnc_valor_proc (cod_proc int, cod_conv int)
returns double
begin
    declare valor_proc double(10,2);
    set valor_proc = (select p.valor
                      from convenio_procedimento_valor p
                      where p.pro_id = cod_proc
                      and p.convenio_id = cod_conv);
    return valor_proc;
end $$
delimiter ;
```

Após a criação da **function**, execute-a passando como parâmetro o identificador do procedimento e do convênio, conforme definido no código da **function**:






```
create function fnc_valor_proc (cod_proc int, cod_conv int)
```

Utilize o seguinte comando para executá-la.

```
select fnc_valor_proc(1, 1) valor_procedimento;
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
valor_procedimento			
50			

Os parâmetros informados na execução representam respectivamente o **procedimento**, RAO X DE TORAX e o **convênio** UNIMED, conforme demonstrado a seguir:

Result Grid		 Filter Rows: <input type="text"/>	Edit: 		
pro_id	descricao	tipo	aparelho		
1	RAIO X DE TORAX	I	RX		
2	ULTRASSONOGRAFIA DE ABDOMEN TOTAL	I	US		
3	CONSULTA MÉDICA	C	NULL		

Result Grid	Filter Rows:	Edit:
convenio_id	razao social	cnpj
1	UNIMED	88638266000105
2	AMIL	33202172000105
3	SULAMERICA	87015579000144

Agora que já temos uma **function** própria para valorização de procedimentos podemos melhorar o **procedure** `confirmar_realizacao_exame`, substituindo a consulta para obter o valor do procedimento pela **function**.


```

delimiter $$

create procedure confirmar_realizacao_exame (in workl_id int)
begin
    select wa.agendamento_id
        , epv.valor
    into @agend_id
        , @valor
    from worklist_aparelhos wa
        inner join agendamento a on a.agendamento_id = wa.agendamento_id
        inner join convenio_procedimento_valor epv
        on epv.pro_id = a.pro_id
        and epv.convenio_id = a.convenio_id
    where wa.worklist_id = workl_id;

    update agendamento a
        set a.realizado = 'S'
        , a.valor = @valor
        where a.agendamento_id = @agend_id;
end $$

delimiter ;

```

Substituir a exibição do campo valor pela chamada da function func_valor_proc.

Retirar a junção que era responsável por definir o valor do procedimento

Após as alterações o código do **procedure** confirmar_realizacao_exame ficará da seguinte forma:

```

drop procedure if exists confirmar_realizacao_exame;

delimiter $$

create procedure confirmar_realizacao_exame (in workl_id int)
begin
    select wa.agendamento_id
        , fnc_valor_proc(a.pro_id, a.convenio_id) valor
    into @agend_id
        , @valor
    from worklist_aparelhos wa
        inner join agendamento a on a.agendamento_id = wa.agendamento_id
    where wa.worklist_id = workl_id;

```

Inclua essa instrução antes do código para criação do procedure, ela será utilizada para excluí-lo caso já exista.

```

update agendamento a
  set a.realizado = 'S'
      , a.valor = @valor
  where a.agendamento_id = @agend_id;
end $$
delimiter ;

```

Action Output				
#	Time	Action	Message	
✓ 1	21:19:41	drop procedure if exists confirmar_realizacao_exame	0 row(s) affected	
✓ 2	21:19:41	create procedure confirmar_realizacao_exame (in workl_id int) begin select wa.agendamento...	0 row(s) affected	

Obs.: executando essas instruções em uma Janela SQL, o MySQL irá excluir o procedure e recriá-lo, sem que o erro de procedure já existente seja exibido, se tivesse executado somente o código para criação do procedure.

Action Output				
#	Time	Action	Message	
✗ 1	21:27:20	create procedure confirmar_realizacao_exame (in workl_id int) begin select wa.agendamento...	Error Code: 1304. PROCEDURE confirmar_realizacao_exame already exists	

Lembra do relatório com a previsão dos valores que serão recebidos com a execução dos procedimentos, agora que nós temos a **function** é só utilizá-la para obtê-lo com a instrução:

```

select a.*
      , fnc_valor_proc(a.pro_id, a.convenio_id) valor_receber
  from agendamento a
 where a.realizado = 'N';

```

Result Grid												
	agendamento_id	dt_agenda	horario	pessoa_id	convenio_id	med_pessoa_id	pro_id	valor	chegou	realizado	faturado	valor_receber
1	1	2019-01-15	08:00	4	1	1	3	0.00	N	N	N	90
2	2	2019-01-15	08:30	5	2	1	3	0.00	N	N	N	91.2
4	4	2019-01-15	09:30	5	2	3	2	0.00	N	N	N	62.6

Obs.: utilizamos a instrução de uma maneira bem simples, com os dados brutos da estrutura agendamento, mas poderíamos ter exibido os textos correspondentes aos identificadores da pessoa, convênio e médico utilizando a cláusula **join**.

É isso aí!!! Vamos finalizar a agenda colocando a mão na massa.