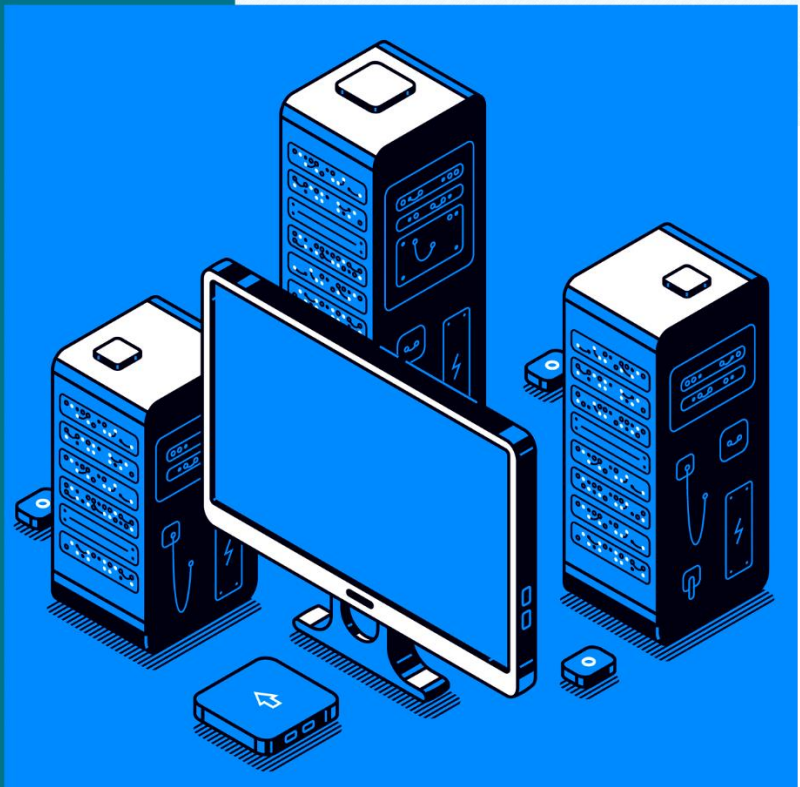


---

# AGENDA 6

---

BLOCOS  
DE LINGUAGEM  
DE CONSULTA  
ESTRUTURADA



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  
PROGRAMAÇÃO MOBILE I

**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*



O Banco Dados é a alma da empresa, seja ela uma microempresa ou uma empresa gigantesca como o Google, sem os dados as pessoas não têm informações para a tomada de decisão. Por esse motivo você profissional de tecnologia da informação deve sim dominar esse assunto que sempre estará presente no seu dia a dia.

Retirado de “O que é um Banco de Dados e Qual a sua importância para um Empresa”, disponível em <https://www.linkedin.com/pulse/o-que-é-um-banco-de-dados-e-qual-sua-importância-para-william-miranda>.

Acessado em 07/08/2019.

Agora imagine uma instituição financeira, quantos processos o sistema dos bancos do Brasil, Santander, Itaú entre outros, executam por segundo, enquanto e quando as pessoas fazem um saque, um depósito ou qualquer outra transação. Eles acontecem a todo momento de acordo com as regras definidas para cada operação. A transformação do dado em informação, em todo negócio, com certeza passa por esses processos.



Os comandos da linguagem SQL são muito poderosos, mas normalmente consegue-se melhorar o desempenho das aplicações através da programação do banco de dados. Ao desenvolver módulos que sejam executados diretamente no servidor diminui-se o tráfego de informações na rede, esconde-se boa parte das estruturas das tabelas e agiliza-se o processamento e retorno das mensagens. Internamente o banco de dados possui mecanismos integrados que permitem unir as estruturas tradicionais de programação com os comandos SQL.

Retirado de “SQL e Programação de Banco de Dados”,  
disponível em <https://www.devmedia.com.br/sql-e-programacao-de-banco-de-dados/3139>.

Acessado em 07/08/2019.



Durante nossos estudos vamos acompanhar o desenvolvimento de uma integração entre dois sistemas. O MySQL possui vários recursos que nos auxiliarão nesse projeto, a programação realizada diretamente no **banco de dados**, agrupando instruções de acordo com as necessidades do negócio com certeza é um deles, espero que aproveite a experiência. Vamos lá!!!!

Ana Lúcia tem um consultório médico, para aproveitar melhor o espaço físico e agregar mais serviços em sua clínica, ela adquiriu dois aparelhos para realizar exames de imagem, um de RX e outro de ULTRASSOM.

Durante o processo de aquisição desses aparelhos ela recebeu diversos representantes, e um fator que foi levado muito em consideração na sua escolha foi a possibilidade de integrar o sistema que gerencia o seu consultório com o que chamamos de worklist (lista de trabalho) desses aparelhos, onde o principal objetivo é agilizar o atendimento dos pacientes que realizarão esses exames.

Para desenvolver essa integração ela contratou a empresa TecDesenvSis, que definiu o Carlos, um dos membros da sua equipe de desenvolvimento, como Analista responsável por esse projeto. Com certeza ele terá um grande desafio!!!



O **banco de dados** possui mecanismos próprios que podem ser utilizados em favor do desenvolvedor. Cada **banco de dados** possui um conjunto específico de comandos que definem a linguagem de programação do **banco de dados**. No caso do **Oracle**, a linguagem é o **PL/SQL**, o **SQL Server** possui o **Transact-SQL**, o **DB2** possui sua própria linguagem de programação, o **PostgreSQL** possui diversas extensões que podem ser utilizadas como linguagem de programação e no **MySQL** não é diferente. Cada **banco de dados** é único sob este aspecto, mas todos trabalham sobre os mesmos conceitos. É possível criar módulos programáveis, como **funções**, **procedimentos**, **objetos**, **pacotes**, **gatilhos** etc. Em todos os casos, há um **engine** responsável pela integração e execução dos módulos no **servidor de banco de dados**.

Adaptado de <https://www.devmedia.com.br/sql-e-programacao-de-banco-de-dados/3139>. Acessado em 07/08/2019

Antes de mergulharmos em módulos programáveis, assim como nas linguagens de programação, o **MySQL** possui um recurso que auxilia muito os desenvolvedores na implementação de instruções **SQL**, que são as variáveis.

O **MySQL** suporta variáveis específicas com a sintaxe **@nomevariável**. Um nome de variável pode conter letras, números e os caracteres especiais **'\_'** (underline), **'\$'** (cifrão) e **'.'** (ponto).



As variáveis não precisam ser inicializadas. Elas contêm **NULL** por padrão e podem armazenar um valor inteiro, real ou uma **string** (conteúdo texto).

Você pode configurar uma variável com a sintaxe a seguir:

```
set @variável= { expressao inteira | expressao real | expressao string }  
[,@variável= ...].
```

Você também pode atribuir um valor a uma variável em outras instruções diferentes de **set**. No entanto, neste caso o operador de atribuição é: **=** (dois pontos e igual) em vez de **=** (igual), porque **=** é reservado para comparações em instruções diferentes de **set**:

```
set @t1=0, @t2=0, @t3=0;
select @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
@t1:=(@t2:=1)+@t3:=4	@t1	@t2	@t3
5	5	1	4

Retirado e adaptado de <http://ftp.nchu.edu.tw/MySQL/doc/refman/4.1/pt/system-variables.html>. Acessado em 25/08/2019

Voltando aos blocos de comandos, dependendo da rotina a ser executada, isso pode requerer várias consultas e atualizações na base, o que acarreta um maior consumo de recursos pela aplicação. No caso de aplicações web, isso se torna ainda mais visível, devido a maior quantidade de informações que precisam trafegar pela rede e de requisições ao **servidor**.

Considerando que os servidores têm configurações melhores, utilizar o **banco de dados** para fazer parte do processamento pode ser uma saída para minimizar o consumo desses recursos.

Mas como executar várias ações no **banco de dados** a partir de uma única instrução? A resposta para essa pergunta é: **Stored Procedures** (ou **Procedimentos Armazenados**, em português).

**Stored Procedures** são rotinas definidas no banco de dados, identificadas por um nome pelo qual podem ser chamadas. Um **procedimento** desses pode executar uma série de instruções, receber parâmetros e retornar valores.

## Usar ou Não usar Stored Procedures

Para exemplificar o funcionamento dos **Stored Procedures** e comparar a execução de uma rotina utilizando e não utilizando essa técnica, utilizaremos como base o seguinte contexto de uma aplicação comercial.

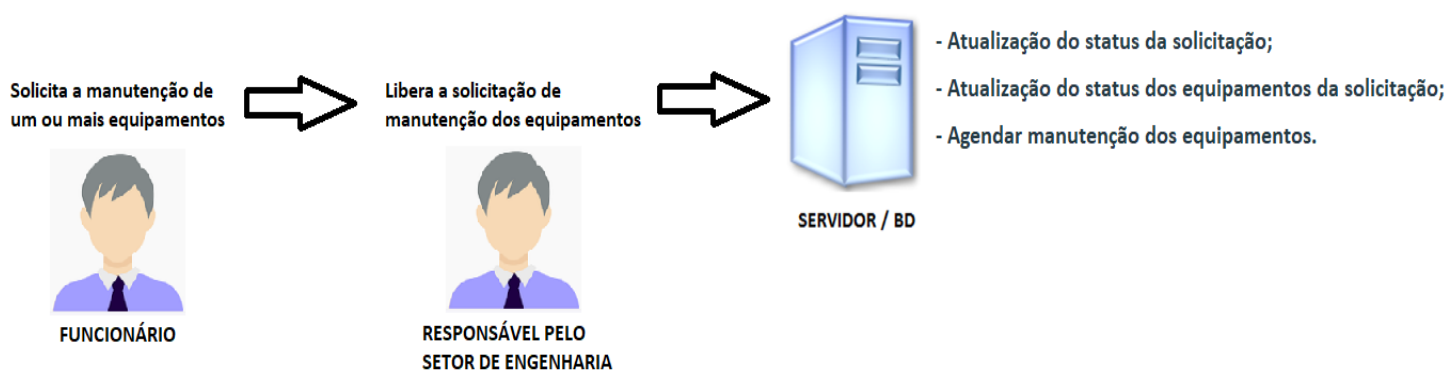


- O funcionário faz uma solicitação de manutenção de um ou vários equipamentos;
- A solicitação permanece com status “SOLICITADA” até ser liberada;
- O responsável pelo setor de engenharia libera a solicitação, agendando a manutenção.

Neste caso, uma saída seria a criação de um processo de liberação de solicitação, pois nenhum agendamento de manutenção de equipamento é realizado enquanto a solicitação não for liberada. Esse processo poderia ter os seguintes passos:

- Atualização do status da solicitação;
- Atualização do status dos equipamentos da solicitação;
- Agendar manutenção dos equipamentos.

Temos então pelo menos três instruções de atualização e/ou inserção. Poderíamos agrupar essas três instruções no corpo de um **procedimento** e chamá-lo a partir da aplicação uma única vez. As ações de update/insert/delete, a partir daí, ficariam por conta do **servidor**. Que poderia ser representada graficamente conforme **Imagem 1**.



A seguir, as principais vantagens e desvantagens do uso dos **procedimentos armazenados**:

#### 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.

**Pontos negativos:**

- 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.

**Criando e invocando Stored Procedures no MySQL**

A seguir vamos entender como trabalhar com **procedures** no **banco de dados MySQL**, iniciando pela sintaxe utilizada para criação desse tipo de objeto.

Sintaxe para criação de **Stored Procedures** no **MySQL**:

```
delimiter $$
create procedure nome_procedimento ([parâmetros])
begin
    /*corpo do procedimento*/
end $$
delimiter ;
```

**Onde:**

**nome\_procedimento**: nome que identificará o **procedimento armazenado**. 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 do **procedure**. Para que um **procedimento** receba parâmetros, é necessário utilizar a seguinte sintaxe (dentro dos parênteses):

Sintaxe de declaração de parâmetros em **Stored Procedures**:



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

### Onde:

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

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

**modo:** indica a forma como o parâmetro será tratado no **procedimento**, se será apenas um dado de entrada, apenas de saída ou se terá ambas as funções. Os valores possíveis para o modo são:

- **in:** indica que o parâmetro é apenas para entrada/recebimento de dados, não podendo ser usado para retorno;
- **out:** usado para parâmetros de saída. Para esse tipo não pode ser informado um valor direto (como 'teste', 1 ou 2.3), deve ser passada uma variável "por referência";
- **inout:** como é possível imaginar, este tipo de parâmetro pode ser usado para os dois fins (entrada e saída de dados). Nesse caso também deve ser informada uma variável e não um valor direto.

Outro ponto que merece destaque é o uso do comando `delimiter`. Por padrão o **MySQL** utiliza o sinal de ponto e vírgula como delimitador de comandos, separando as instruções a serem executadas. No entanto, dentro do corpo do **Stored Procedure** será necessário separar algumas instruções internamente utilizando esse mesmo sinal, por isso é preciso inicialmente alterar o delimitador padrão do **MySQL** (neste caso, para `$$`) e ao fim da criação do procedimento, restaurar seu valor padrão.

Tendo criado o **procedure**, chamá-lo é bastante simples. Para isso fazemos uso da palavra reservada `call`, como mostra o código a seguir:

Sintaxe para chamar um **Stored Procedure**:

```
call nome_procedimento([parâmetros]);
```

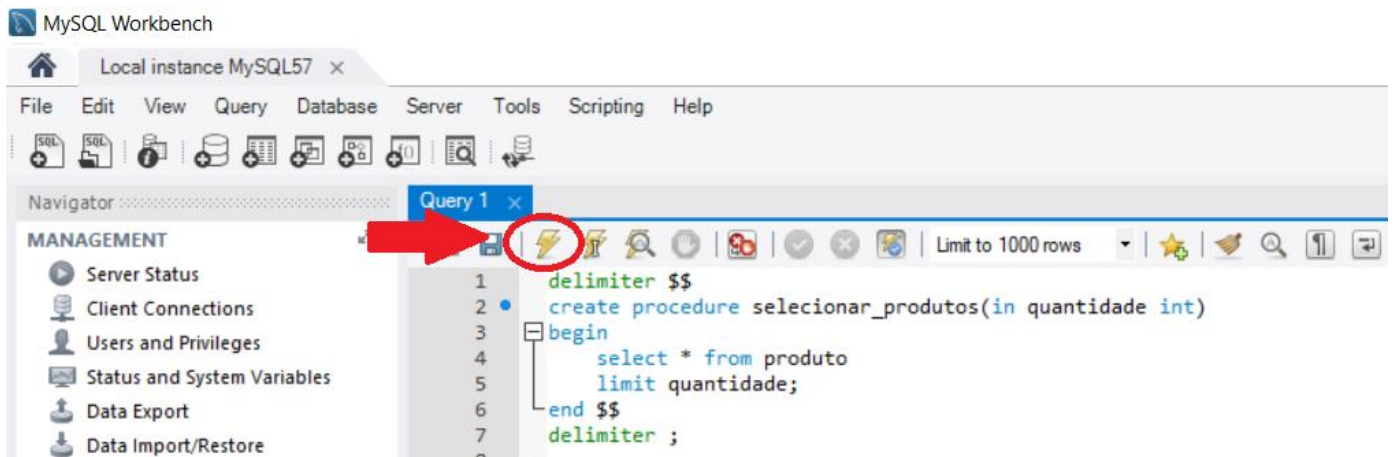
**IMPORTANTE:** para aplicar os exemplos a seguir utilize o banco de dados minimercado. Clique [aqui](#) para baixar o script de criação do banco de dados, depois execute em uma janela SQL do Workbench. Caso tenha alguma dúvida, clique [aqui](#)

A seguir temos um exemplo de uso de cada tipo de parâmetro.

### Exemplo 1: Usando parâmetro de entrada:

```
delimiter $$
create procedure selecionar_produtos(in quantidade int)
begin
    select * from produto
    limit quantidade;
end $$
delimiter ;
```

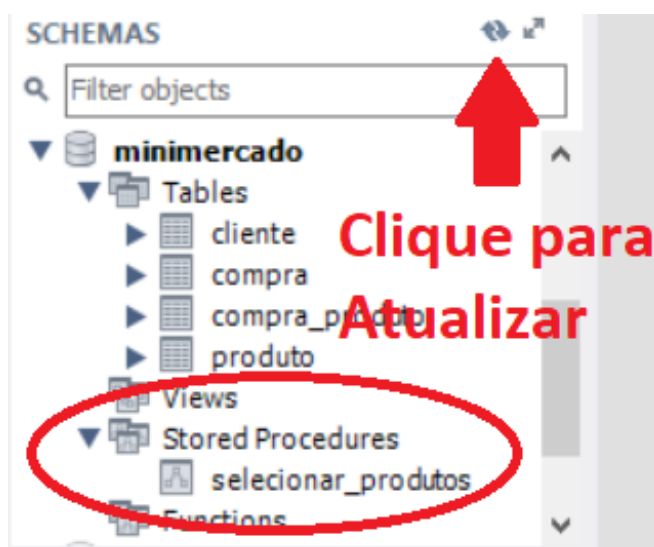
Para criar o procedure pela janela SQL:



Procedure criada com sucesso!!!

Output				
Action Output				
#	Time	Action		Message
✓ 1	20:31:57	create procedure selecionar_produtos(in quantidade int) begin	select * from produto limit quantidade; end	0 row(s) affected

Verifique como ficou a estrutura:



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

Esse **procedimento** tem por função fazer um **select** na tabela **PRODUTO**, limitando a quantidade de registros pela quantidade recebida como parâmetro. Assim, caso desejássemos selecionar dois registros dessa tabela, poderíamos usar o **procedure** como mostra o código a seguir:

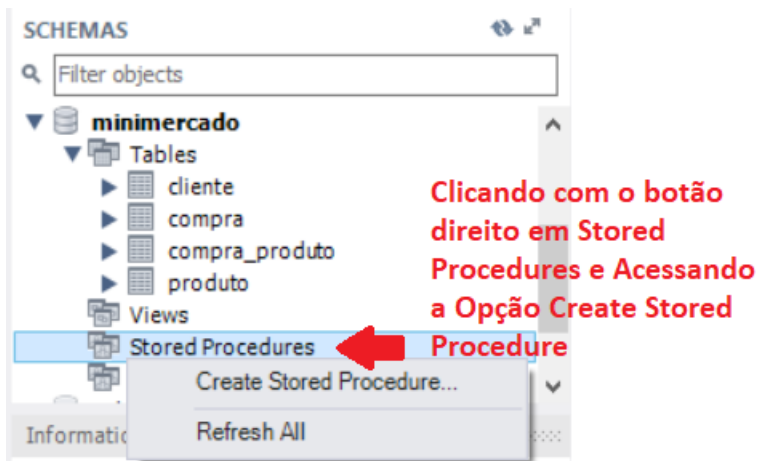
Chamando **procedure** com parâmetro de entrada:

```
call selecionar_produtos(2);
```

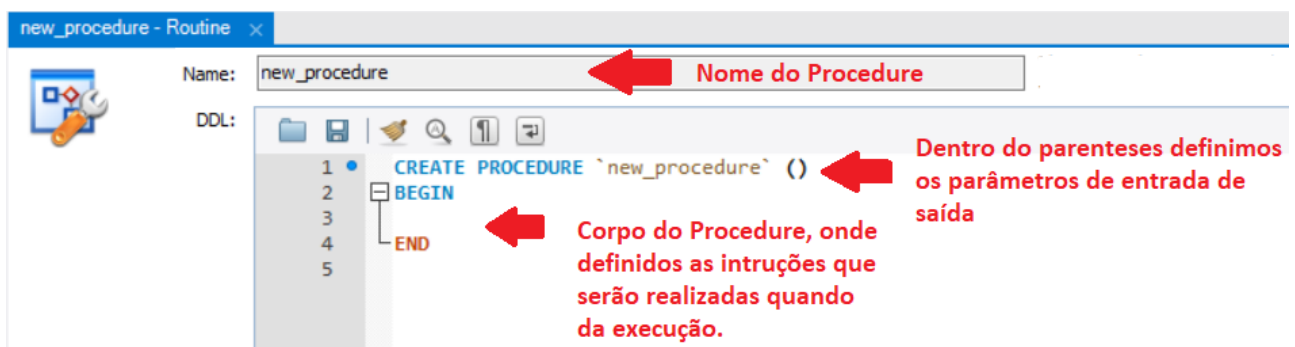
clique no botão  para executar

	codigo_produto	descricao	quantidade_estoque
1		arroz oct 5 kg	100.0
2		feijão oct 2 kg	50.0

Você também pode criar um **procedure** utilizando a interface gráfica do Workbench:

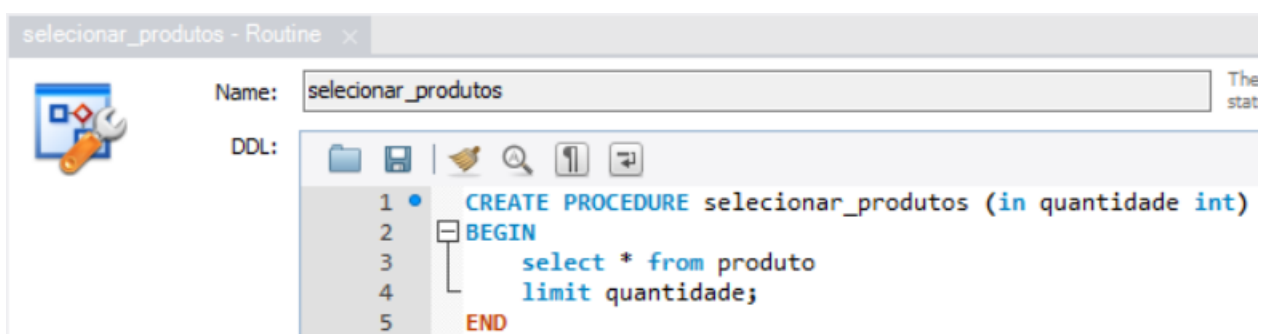



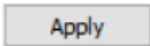
É apresentada a interface para criação de **procedures**, onde você deve definir o nome, os parâmetros e as instruções que deverão ser executadas.



Obs: Para definir o nome, altere no código ``new_procedure`` para `selecionar_produtos`. O **Name** do procedure será alterar automaticamente após sua geração.

Neste caso, você não precisa redefinir o delimitador, o Workbench fará isso automaticamente.



Clique no botão  para gerar o **script** de criação do **procedure**. Será apresentada uma janela para confirmação da criação do **procedure**, clique novamente no botão .

Apply SQL Script to Database



Review the SQL Script to be Applied on the Database

Online DDL

Algorithm:

Lock Type:

```

1  USE `minimercado`;
2  DROP procedure IF EXISTS `selecionar_produtos`;
3
4  DELIMITER $$
5  USE `minimercado` $$
6  CREATE PROCEDURE selecionar_produtos (in quantidade int)
7  BEGIN
8    select * from produto
9    limit quantidade;
10 END$$
11
12 DELIMITER ;
13
14

```

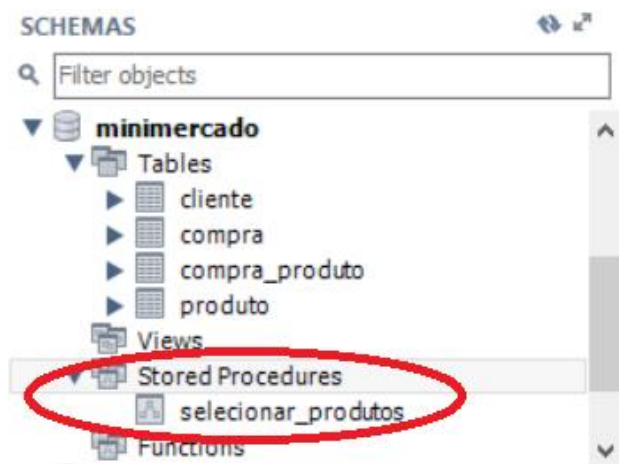
Clique aqui para  
confirmar a geração  
do procedure

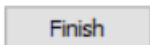



Back

Apply

Cancel



Na janela de confirmação da criação do **procedure** clique no botão . O **procedure** criado será apresentado no quadro **SCHEMA**. Caso o procedure não for apresentado nesse quadro após sua criação, será necessário atualizar o quadro clicando no botão .

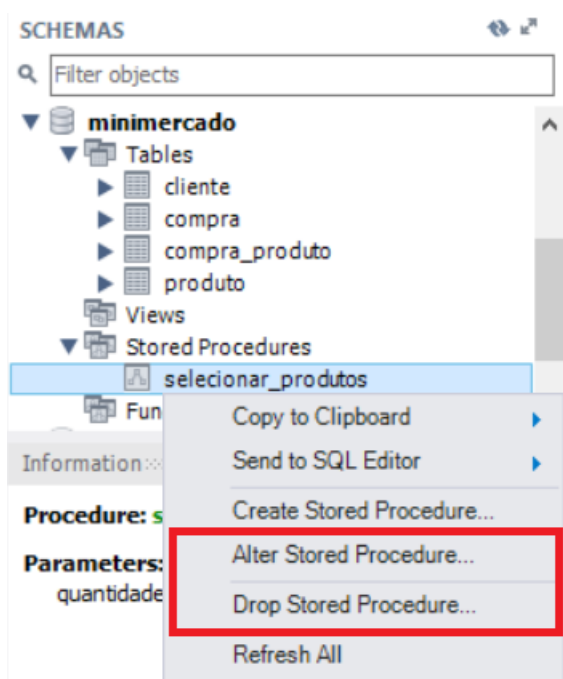


Performance Schema Setup

SCHEMAS



Para excluir ou alterar um procedure utilize as opções **Alter Stored Procedure** ou **Drop Stored Procedure**, do quadro **SCHEMA**.

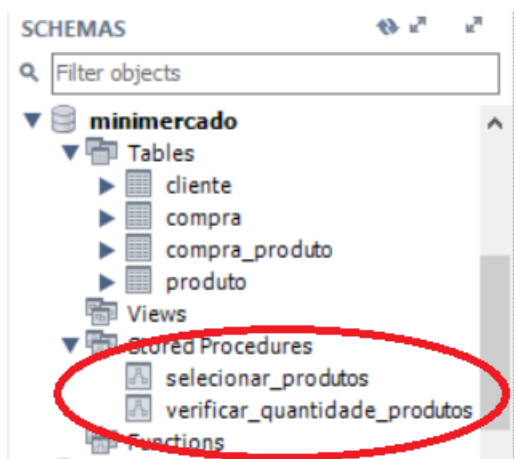


**IMPORTANTE:** Caso tenha alguma dúvida, clique [aqui](#) para assistir a um vídeo que mostrará como criar, alterar e excluir um procedure pela opções gráficas do WorkBench.

O próximo código mostra um exemplo de recebimento e retorno de parâmetro de saída.

## Exemplo 2: Usando parâmetro de saída:

```
delimiter $$
create procedure verificar_quantidade_produtos(out quantidade int)
begin
    select count(*) into quantidade from produto;
end $$
delimiter ;
```



A função desse **procedimento** é retornar à quantidade de registros da tabela **PRODUTO**, passando esse valor para a variável de saída “**quantidade**”. Para isso foi utilizada a palavra reservada **into**.

Para chamá-lo, usamos um símbolo de arroba (@) seguido do nome da variável que receberá o valor de saída. Feito isso, a variável poderá ser usada posteriormente, como demonstrado a seguir:



Chamando **procedure** com parâmetro de saída:

```
call verificar_quantidade_produtos(@total);
select @total;
```

Ao executar a segunda linha, teremos como retorno o valor da variável `@total`, que será preenchida no **procedure**.

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
@total			
5			

O **Exemplo 3** mostra um **Stored Procedure** chamado `eleva_ao_quadrado`, que recebe uma variável e a altera, definindo-a como o seu próprio valor elevado à segunda potência.

### Exemplo 3: Usando parâmetro de entrada e saída

```
delimiter $$
create procedure eleva_ao_quadrado(inout numero int)
begin
    set numero = numero * numero;
end $$
delimiter ;
```

Nesse caso, a mesma variável é usada como entrada e saída, como demonstrado a seguir:

Chamando **procedure** com parâmetro de entrada e saída:

```
set @valor = 5;
call eleva_ao_quadrado(@valor);
select @valor;
```

Result Grid			Filter Rows: <input type="text"/>	Export:	Wrap Cell Content:
	@valor				
	25				

### Exemplo 4: Usando variáveis no corpo do procedimento

É possível declarar variáveis no corpo dos **Stored Procedures**, para isso basta utilizar a seguinte sintaxe de declaração de variáveis:

```
declare nome_variável tipo default valor_padrao;
```

#### Onde:

**declare:** obrigatória, é a responsável por indicar que uma variável será declarada com o nome “nome\_variavel” (que segue as mesmas regras de nomeação de variáveis).

**tipo:** é o tipo de dados da variável (**int**, **decimal**, **varchar**, etc). A palavra reservada **default** é opcional e deve ser usada quando se deseja definir um valor inicial (**valor\_padrao**) para a variável.

A declaração das variáveis deve ser feita logo no início do corpo do **procedure**, para aquelas que serão utilizadas em todo o **procedimento**, ou dentro de um bloco **begin-end** específico que limite seu escopo.

Para definir um valor para uma variável, usamos as palavras reservadas **set** (no caso de passagem direta de valor, como no **Exemplo 2**) ou **into** (no caso de associação de valores dentro de consultas, como no **Exemplo 3**).

Outro ponto importante de se citar é o **ESCOPO** das variáveis, que define em que pontos elas são reconhecidas. Uma variável definida dentro de um bloco **begin/end** é válida somente dentro dele, ou seja, após o **end** ela já não é mais reconhecida. Assim, é possível definir várias variáveis com o mesmo nome, mas dentro de blocos **begin/end** distintos.

Por sua vez, variáveis cujo nome inicia com arroba (@), são chamadas variáveis de sessão, e são válidas enquanto durar a sessão (demonstrado em **Chamando procedure com parâmetro de entrada e saída**).

Retirado e adaptado de <https://www.devmedia.com.br/stored-procedures-no-mysql/29030>.

Acessado em 21/08/2019

## Agora é com você!



Para dar início ao processo de integração do Sistema que gerencia o consultório de Ana Lúcia com os aparelhos adquiridos, foi enviado a ela um manual de integração, onde constava todos os passos para envio e recebimento de informações dos aparelhos.

**IMPORTANTE:** Clique [aqui](#) para baixar o arquivo do banco de dados do consultório, depois execute em uma janela SQL do Workbench.

Após entender como era o processo de atendimento de pacientes no consultório, Carlos analisou o manual e identificou as primeiras atividades que terá que realizar para conseguir implementar essa integração:

**1** - Implementar uma estrutura no banco de dados do consultório para receber os dados do paciente e do exame que ele irá realizar conforme a especificação a seguir:

Nome da Tabela: **WORKLIST\_APARELHOS**

CAMPO	TIPO (Tamanho)	Obrigatório	Padrão	PK	Observação
<b>worklist_id</b>	<b>int(6)</b>	<b>sim</b>		<b>sim</b>	<b>auto incremento</b>
<b>agendamento_id</b>	<b>int(6)</b>	<b>sim</b>		<b>não</b>	
<b>dt_exame</b>	<b>date</b>	<b>sim</b>		<b>não</b>	
<b>paciente</b>	<b>varchar(80)</b>	<b>sim</b>		<b>não</b>	
<b>dt_nascimento</b>	<b>date</b>	<b>sim</b>		<b>não</b>	
<b>convenio</b>	<b>varchar(80)</b>	<b>sim</b>		<b>não</b>	
<b>procedimento</b>	<b>varchar(100)</b>	<b>sim</b>		<b>não</b>	
<b>aparelho</b>	<b>varchar(2)</b>	<b>sim</b>		<b>não</b>	<b>RX ou US</b>
<b>integrado</b>	<b>varchar(1)</b>	<b>sim</b>	<b>N</b>	<b>não</b>	

**2** - Implementar uma estrutura no banco de dados do consultório que será utilizada pelo aparelho para confirmação da integração dos dados e que posterior será utilizada para que o aparelho retorne a realização do exame conforme a especificação a seguir:

Nome da Tabela: **INTEGRACAO\_APARELHOS**

CAMPO	TIPO (Tamanho)	Obrigatório	Padrão	PK	Observação
<b>integracao_id</b>	<b>int(6)</b>	<b>sim</b>		<b>sim</b>	<b>auto incremento</b>
<b>worklist_id</b>	<b>int(6)</b>	<b>sim</b>		<b>não</b>	
<b>dt_realizacao</b>	<b>date</b>	<b>não</b>		<b>não</b>	

**3** - Desenvolver um **procedure** que inclua um registro na Tabela **WORKLIST\_APARELHOS** a partir de um agendamento.

**4** - Desenvolver um **procedure** para confirmar a realização do exame e valorizá-lo de acordo com o convênio no Sistema que gerencia o consultório atribuindo '**s**' ao campo **realizado**, da Tabela **AGENDAMENTO**, a partir do número de identificação do worklist dos aparelhos, campo **worklist\_id**.

Será que você consegue auxiliar o Carlos a realizar essas atividades???? Tenha certeza que sim!!!!

Como foi? Muito fácil? De qualquer forma tenho certeza de que conseguiu.

Neste momento não se preocupe em saber como os aparelhos irão retornar à realização do exame, o mais importante é deixar um processo pronto para receber o identificador do registro do worklist realizado e dar baixa no agendamento vinculado a ele.

Segue abaixo algumas instruções SQL que podem ser utilizadas para implementar as estruturas e desenvolver os **procedures** solicitados:

#### WORKLIST\_APARELHOS

```
create table worklist_aparelhos (
    worklist_id int(6) not null auto_increment,
    agendamento_id int(6) not null,
    dt_exame date not null,
    paciente varchar(80) not null,
    dt_nascimento date default null,
    convenio varchar(80) not null,
    procedimento varchar(100) not null,
    aparelho varchar(2) not null,
    integrado varchar(1) default 'N',
    primary key (worklist_id)
);
```

#### INTEGRACAO\_APARELHOS

```
create table integracao_aparelhos (
    integracao_id int(6) not null auto_increment,
    worklist_id int(6) not null,
    dt_realizacao date,
    primary key (integracao_id)
);
```

#### PROCEDURE INCLUIR\_WORKLIST

```
delimiter $$
create procedure incluir_worklist (in agend_id int)
begin
    /*1. Passo: busca os dados do agendamento utilizando variáveis*/
    select a.dt_agenda
```

```

        , p.nome
        , p.dt_nascimento
        , c.razao_social
        , pr.descricao
        , pr.aparelho
    into @dt_agenda
        , @nome
        , @dt_nascimento
        , @convenio
        , @descricao
        , @aparelho
    from agendamento a
        inner join pessoa p on p.pessoa_id = a.pessoa_id
        inner join convenio c on c.convenio_id = a.convenio_id
        inner join procedimento pr on pr.pro_id = a.pro_id
    where a.agendamento_id = agend_id;

    /*2. Passo: inserir os dados na worklist dos aparelhos */
    insert into worklist_aparelhos (agendamento_id, dt_exame,
    paciente, dt_nascimento, convenio, procedimento, aparelho)
        values (agend_id, @dt_agenda, @nome, @dt_nascimento,
        @convenio, @descricao, @aparelho);

end $$
delimiter ;

```

#### PROCEDURE CONFIRMAR\_REALIZACAO\_EXAME

```

delimiter $$
create procedure confirmar_realizacao_exame (in workl_id int)
begin
    /*1. Passo: busca identificador do agendamento e o valor do
    procedimento*/
    select wa.agendamento_id
        , cpv.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 cpv
on cpv.pro_id = a.pro_id
and cpv.convenio_id = a.convenio_id
where wa.worklist_id = workl_id;

/*2. Passo: confirma a realização do agendamento e valorização o
procedimento*/

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

end $$
delimiter ;

```

Utilize as instruções abaixo para verificar o funcionamento das procedures:

```
call incluir_worklist(3);
```

Obs.: o valor 3 (três) corresponde ao identificador do agendamento passado como parâmetro para inclusão do registro na worklist dos aparelhos.

```
select * from worklist_aparelhos;
```

worklist_id	agendamento_id	dt_exame	paciente	dt_nascimento	convenio	procedimento	aparelho	integrado
1	3	2019-01-15	Gustavo Camilo	1983-08-25	UNIMED	RAIO X DE TORAX	RX	N

```
call confirmar_realizacao_exame(1);
```

Obs.: o valor 1 (um) corresponde ao identificador do registro de worklist dos aparelhos passado como parâmetro para realização e valorização do exame do agendamento vinculado a ele.

```
select * from agendamento;
```

agendamento_id	dt_agenda	horario	pessoa_id	convenio_id	med_pessoa_id	pro_id	valor	chegou	realizado	faturado
1	2019-01-15	08:00	4	1	1	3	0.00	N	N	N
2	2019-01-15	08:30	5	2	1	3	0.00	N	N	N
3	2019-01-15	09:00	4	1	3	1	50.00	N	S	N
4	2019-01-15	09:30	5	2	3	2	0.00	N	N	N

Obs.: neste processo são atualizados os conteúdos dos campos valor e realizado do agendamento vinculado ao registro da worklist dos aparelhos.

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



Este exercício deve ser entregue de forma on-line como atividade da agenda.

### Exercício

Utilize o banco de dados imobiliária para resolver os exercícios.

Clique [aqui](#) para baixar o script de criação do banco de dados.

1 - Desenvolva um procedure que atualize o valor de todos os apartamentos de acordo com o identificador do condomínio e o percentual de aumento passados como parâmetro. O processo deverá alterar o valor dos apartamentos somente do condomínio e percentual definido por um valor inteiro (10% = 10), passados como parâmetros.

2 - Desenvolva um procedure de efetivação de compra de apartamento de acordo com o identificador do proprietário comprador e o número do apartamento passados como parâmetro. O processo deverá atribuir ao apartamento o identificador do novo proprietário.

**IMPORTANTE:** quando o parâmetro for do tipo `varchar` o tamanho deverá ser definido. Exemplo:

```
procedure exemplo (in nome varchar(80)) ;
```

Obs.: nos dois exercícios evidencie a alteração realizada apresentando os registros antes e depois da execução dos procedures.



Você acaba de conhecer um pouco mais de banco de dados, agora utilizando Stored Procedures ou procedimentos armazenados pela interface gráfica WorkBench com o SGBD MySQL e a linguagem SQL. O MySQL possui vários outros recursos que você pode utilizar, aproveite para aprofundar-se nesse assunto.

Seguem algumas dicas de vídeo e artigo que se relacionam com o conteúdo estudado. Estas dicas são muito importantes para você!

### Vídeos

MySQL - Procedimentos Armazenados (Stored Procedures) Básico – 34.

Disponível em <https://www.youtube.com/watch?v=w3F4JQ8ndJ0>. Acessado em 31/08/2019.



### Artigos

Stored Procedures e Functions no MySQL com PhpMyAdmin.

Disponível em <https://www.devmedia.com.br/stored-procedures-e-functions-no-mysql-com-phpmyadmin/30837>. Acessado em 31/08/2019.