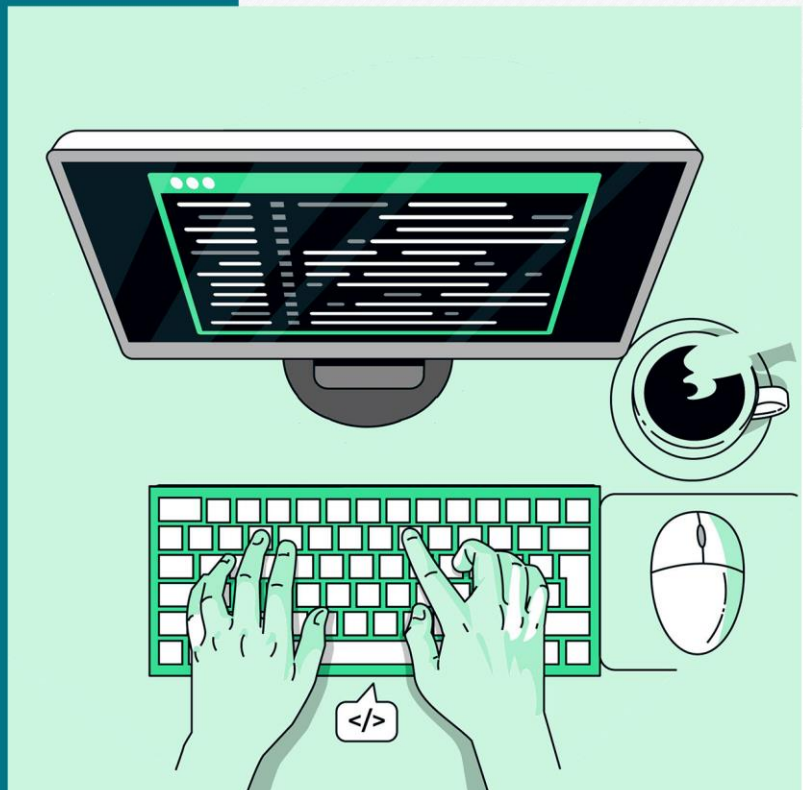

AGENDA 3

GATILHOS



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...



É muito comum, em aplicações que utilizam **bancos de dados**, que ações sejam disparadas em resposta ou como consequência de outras, realizando operações de cálculo, validações e, em geral, surtindo alterações na base de dados.

Em muitos casos, os programadores optam por executarem tais ações a partir da própria aplicação, executando várias instruções SQL em sequência para obter o resultado esperado. De fato, essa é uma solução que pode até ser tida como mais segura, por certos pontos de vista, mas tende a tornar ainda mais “pesada” a execução de certas tarefas, requisitando mais recursos da máquina cliente. A “solução” (ou pelo menos uma forma alternativa) a essa está na utilização de **TRIGGERS** no banco de dados, automatizando certas ações com base em eventos ocorridos.

Retirado de “MySQL Básico: Triggers”, disponível em <https://www.devmedia.com.br/mysql-basico-triggers/37462>.

Acessado em 23/10/2019.

O que é um TRIGGER?

Um **TRIGGER** ou gatilho é um objeto de banco de dados, associado a uma tabela, definido para ser disparado, respondendo a um evento em particular. Tais eventos são os comandos da **DML** (Data Manipulation Language, Linguagem de Manipulação de Dados): **INSERT**, **DELETE** ou **UPDATE**.

Podemos definir inúmeros **TRIGGERS** em uma base de dados baseados diretamente em qual dos comandos acima irá dispará-lo, sendo que, para cada um, podemos definir vários **TRIGGERS**. Os **TRIGGERS** poderão ser disparados para trabalharem antes ou depois do evento. Veremos como definir o momento de atuação do **TRIGGER** mais à frente.

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

1 • describe compra_produto;

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	Field	Type	Null	Key	Default	Extra
	id compra produto	int(11)	NO	PRI	NULL	auto increment
	codiao compra	int(11)	NO	MUL	NULL	
	codiao produto	int(5)	NO	MUL	NULL	
	quantidade	double(10,1)	NO		1.0	
	preco	double(10,2)	NO		NULL	

Baseado na tabela `compra_produto`, podemos definir os **TRIGGERS** para serem disparados, por exemplo, antes (**BEFORE**) ou depois (**AFTER**) de um **INSERT**. Agora sabemos então que para cada momento **BEFORE** ou **AFTER**, podemos ter um **TRIGGER** a ser disparado para defender alguma lógica.

A sintaxe geral de definição de um **TRIGGER** é a seguinte:

```
create
    [definer = { user | current_user }]
    trigger trg_name trg_time trg_event
    on tbl_name for each row trg_stmt
```

Onde:

- **definer**: Quando o **TRIGGER** for disparado, esta opção será checada para verificar com quais privilégios este será disparado. Utilizará os privilégios do usuário informado em **user** (`'root'@'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()**;
- **trg_name**: define o nome do **TRIGGER**, por exemplo, **trg_test**;
- **trg_time**: define se o **TRIGGER** será ativado antes (**BEFORE**) ou depois (**AFTER**) do comando que o disparou;
- **trg_event**: aqui se define qual será o evento, **INSERT**, **DELETE** ou **UPDATE**;
- **tbl_name**: nome da tabela onde o **TRIGGER** ficará “pendurado” aguardando o **trg_event**;
- **trg_stmt**: as definições do que o **TRIGGER** deverá fazer quando for disparado.

Definir dados de antes (OLD) e depois (NEW)

Em meio aos **TRIGGERS** temos dois operadores importantíssimos que nos possibilitam acessar as colunas da tabela alvo do comando **DML**, ou seja, podemos acessar os valores que serão enviados para a tabela `compra_produto` antes (**BEFORE**) ou depois (**AFTER**) de um **UPDATE**, por exemplo.

Tais operadores nos permitirão então, ter dois momentos, o antes e o depois, podendo examinar os valores para que sejam ou não inseridos, atualizados ou excluídos da tabela.

Antes mesmo de analisarmos os operadores, temos que analisar as seguintes diretrizes:

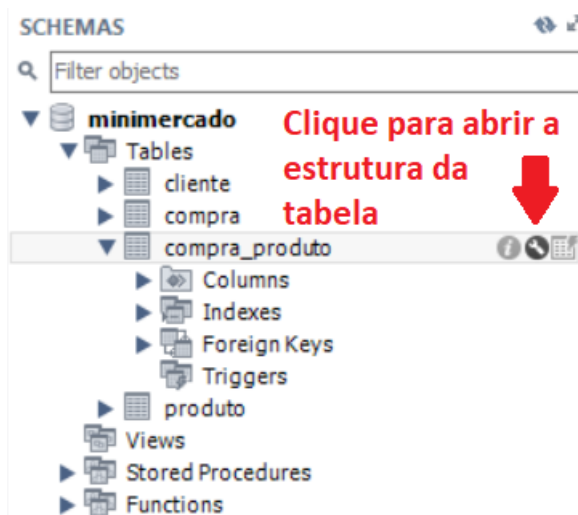
- **INSERT**: o operador `NEW.nome_coluna`, nos permite verificar o valor enviado para ser inserido em uma coluna de uma tabela. `OLD.nome_coluna` não está disponível.
- **DELETE**: o operador `OLD.nome_coluna` nos permite verificar o valor excluído ou a ser excluído. `NEW.nome_coluna` não está disponível.
- **UPDATE**: tanto `OLD.nome_coluna` quanto `NEW.nome_coluna` estão disponíveis, antes (**BEFORE**) ou depois (**AFTER**) da atualização de uma linha.

Percebemos então que, ao inserir uma nova linha em uma tabela, temos os valores das colunas disponível através do operador `NEW.nome_coluna`, quando excluímos uma linha, temos ainda os valores das colunas da linha excluída através do operador `OLD.nome_coluna` e temos os dois operadores disponíveis no **UPDATE** pois essa declaração seria como um **DELETE** seguido por um **INSERT**.

Criemos então um primeiro **TRIGGER**, bem básico que não fará nada mais que validar se os dados foram passados em uma declaração **INSERT** antes (**BEFORE**) que sejam cadastrados na tabela de exemplo. Validaremos a quantidade comprada igual ou menor que 0 (zero), caso isso aconteça, atribuiremos a quantidade 1 (um), garantindo que toda compra sempre tenha valor positivo para a quantidade.

Para criar o **TRIGGER** siga os passos abaixo de 1 a 5 ou assista ao vídeo clicando [aqui](#).

1º Passo:



2º Passo:

Será apresentada a Estrutura da Tabela `compra_produto`:

Table Name: `compra_produto` Schema: `minimercado`

Collation: `utf8 - default collation` Engine: `InnoDB`

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
<code>id_compra_produto</code>	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
<code>codigo_compra</code>	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>codigo_produto</code>	INT(5)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>quantidade</code>	DOUBLE(10,1)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'1.0'
<code>preco</code>	DOUBLE(10,2)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column Name:

Collation:

Comments:

Clique para criar o TRIGGER

Data Type:

Default:

Storage: ☐ Virtual ☐ Stored

☐ Primary Key ☐ Not Null ☐ Unique

☐ Binary ☐ Unsigned ☐ Zero Fill

☐ Auto Increment ☐ Generated

Columns Indexes Foreign Keys Triggers Partitioning Options

3º Passo:

Será apresentada a interface para criação do **TRIGGER** na Tabela `compra_produto`:

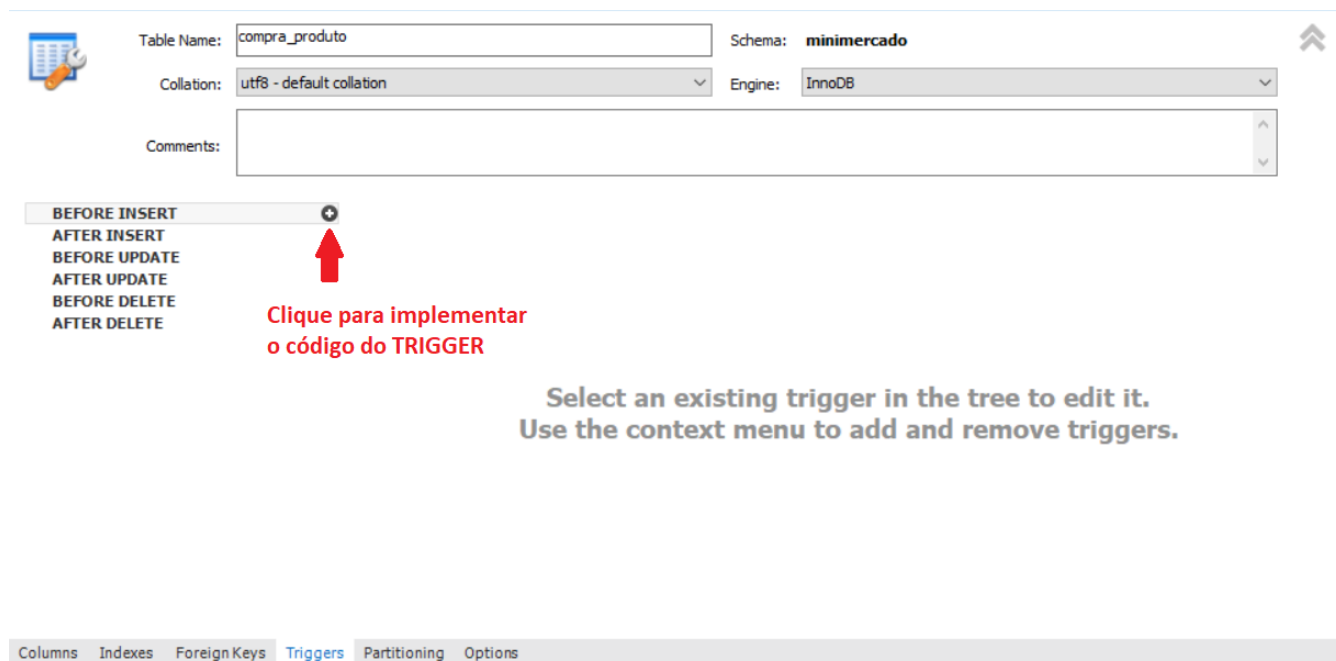


Table Name: Schema: **minimercado**

Collation: Engine:

Comments:

BEFORE INSERT
AFTER INSERT
BEFORE UPDATE
AFTER UPDATE
BEFORE DELETE
AFTER DELETE

Clique para implementar o código do TRIGGER

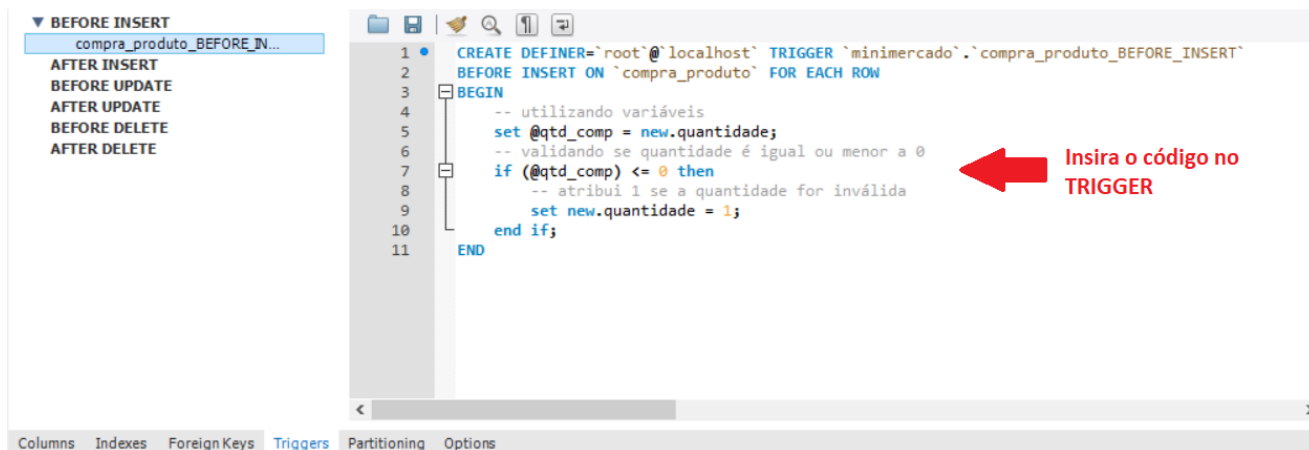
Select an existing trigger in the tree to edit it.
Use the context menu to add and remove triggers.

Columns Indexes Foreign Keys **Triggers** Partitioning Options

4º Passo:

Será apresentada a interface para digitação do código do TRIGGER. Você deverá inserir o código a seguir:

```
-- utilizando variáveis
set @qtd_comp = new.quantidade;
-- validando se quantidade é igual ou menor a 0
if (@qtd_comp) <= 0 then
    -- atribui 1 se a quantidade for inválida
    set new.quantidade = 1;
end if;
```



▼ BEFORE INSERT
compra_produto_BEFORE_IN...

AFTER INSERT
BEFORE UPDATE
AFTER UPDATE
BEFORE DELETE
AFTER DELETE

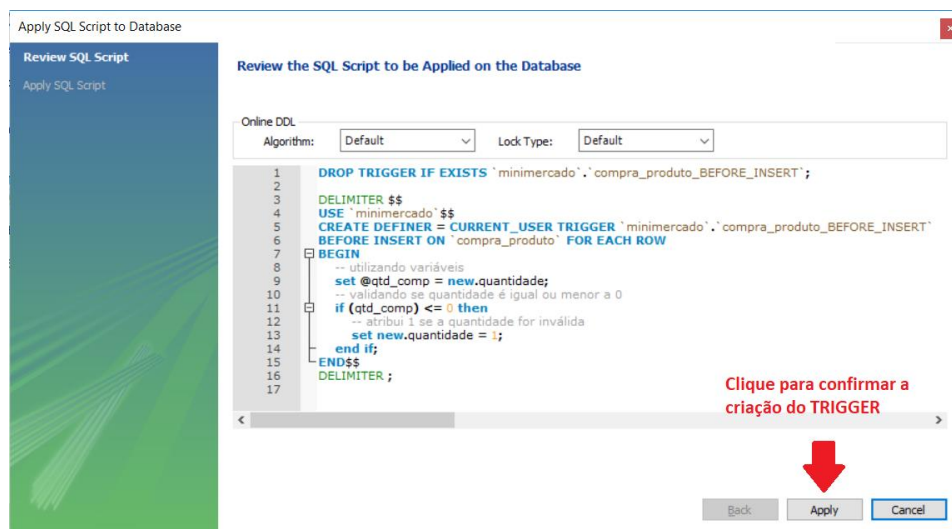
```
1 CREATE DEFINER='root'@'localhost' TRIGGER `minimercado`.`compra_produto_BEFORE_INSERT`
2 BEFORE INSERT ON `compra_produto` FOR EACH ROW
3 BEGIN
4     -- utilizando variáveis
5     set @qtd_comp = new.quantidade;
6     -- validando se quantidade é igual ou menor a 0
7     if (@qtd_comp) <= 0 then
8         -- atribui 1 se a quantidade for inválida
9         set new.quantidade = 1;
10    end if;
11 END
```

Insira o código no TRIGGER

Columns Indexes Foreign Keys **Triggers** Partitioning Options

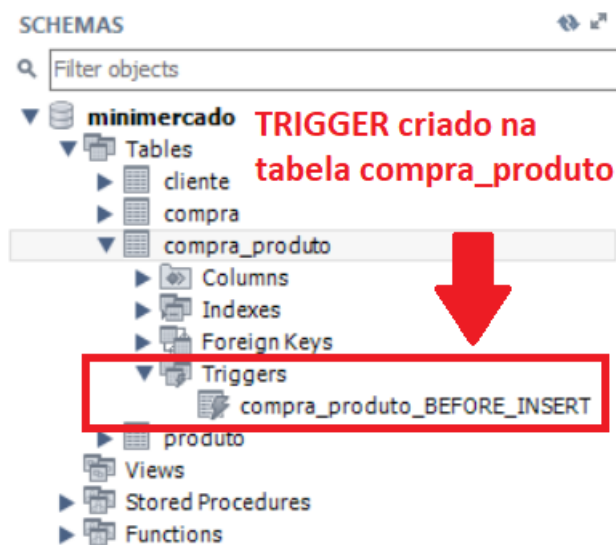
5º Passo:

Clique no botão **Apply** para revisar o código do TRIGGER antes de criá-lo.



6º Passo:

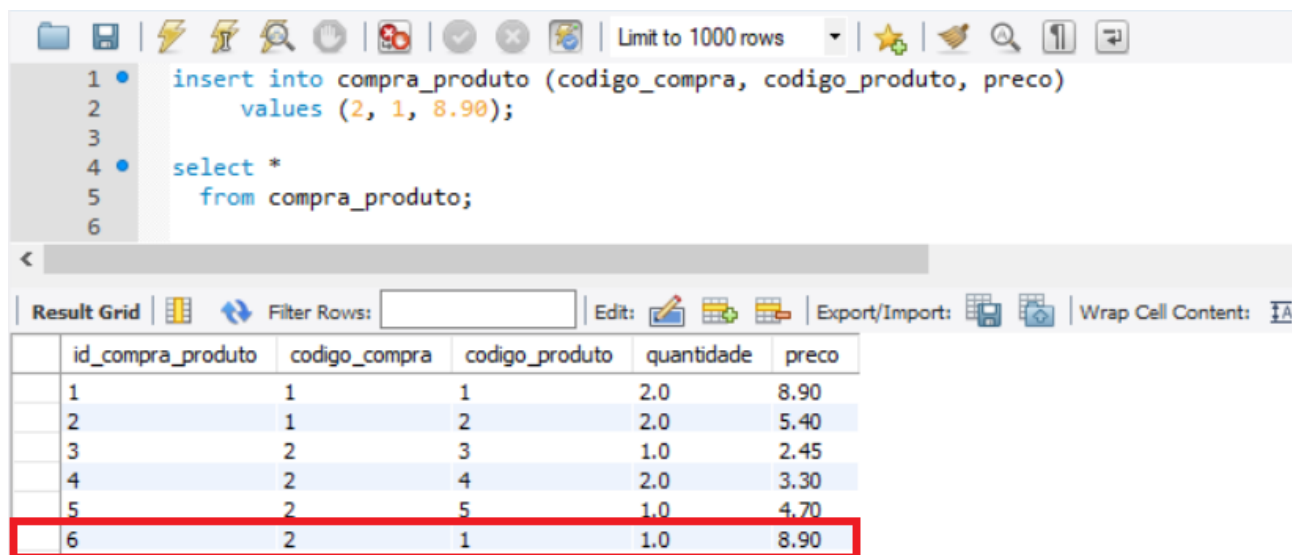
Clique no botão **Finish** finalizar o processo de criação do **TRIGGER**. O **TRIGGER** criado será apresentado em **Triggers** da Tabela **compra_produto**.



Ao tentarmos inserir um registro de compra de produto onde a quantidade é menor ou igual a 0, o **TRIGGER** ao ser disparado atribuirá o valor enviado para 1 (um) através do operador **NEW.nome_coluna**. Como na Tabela de exemplo a coluna **quantidade** foi configurada com valor **default 1**, ou seja, se não for definido conteúdo o valor **1** será atribuído automaticamente, como demonstrado a seguir:

Inserindo um registro sem a definição da quantidade

Vamos inserir um registro na compra com o código 2 (dois):



The screenshot shows a database interface with a SQL editor and a result grid. The SQL editor contains the following code:

```
1 • insert into compra_produto (codigo_compra, codigo_produto, preco)
2     values (2, 1, 8.90);
3
4 • select *
5     from compra_produto;
6
```

The result grid displays the following data:

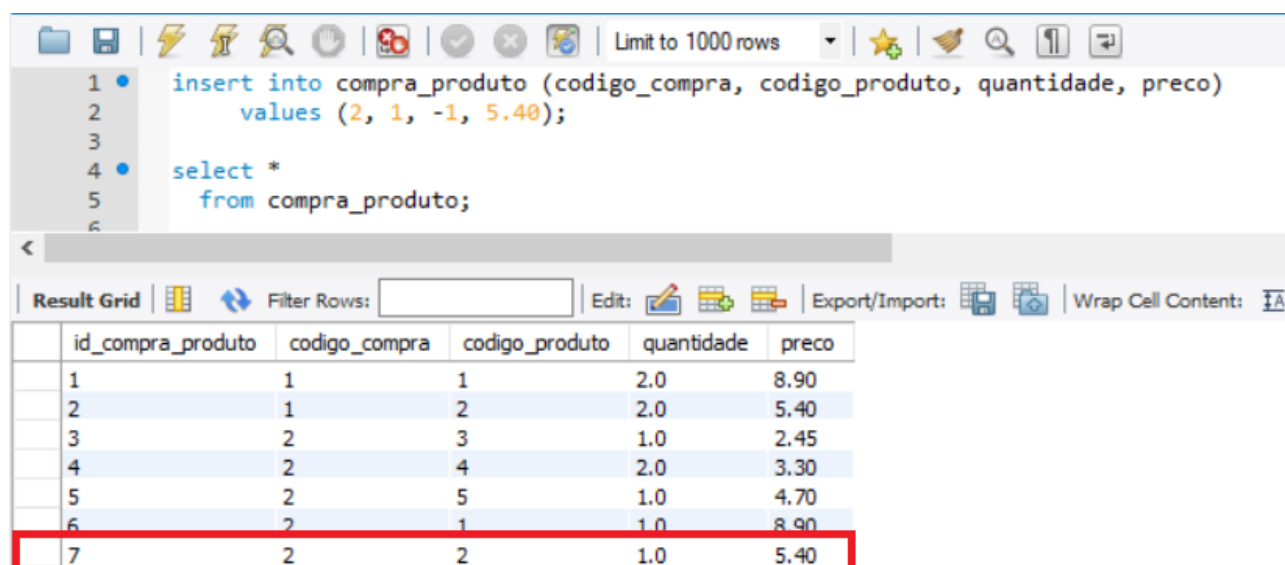
	id_compra_produto	codigo_compra	codigo_produto	quantidade	preco
1	1	1	1	2.0	8.90
2	1	2	2	2.0	5.40
3	2	3	3	1.0	2.45
4	2	4	4	2.0	3.30
5	2	5	5	1.0	4.70
6	2	1	1	1.0	8.90

The row with id_compra_produto 6 is highlighted with a red border.

Obs.: Mesmo não sendo definido um valor para o campo `quantidade`, o valor 1 (um) foi definido.

Inserindo um registro definindo uma quantidade menor ou igual 0

Vamos inserir mais um registro na compra com o código 2 (dois):



The screenshot shows a database interface with a SQL editor and a result grid. The SQL editor contains the following code:

```

1 • insert into compra_produto (codigo_compra, codigo_produto, quantidade, preco)
2   values (2, 1, -1, 5.40);
3
4 • select *
5   from compra_produto;
6

```

The result grid displays the following data:

	id_compra_produto	codigo_compra	codigo_produto	quantidade	preco
1	1	1	1	2.0	8.90
2	1	2	2	2.0	5.40
3	2	3	3	1.0	2.45
4	2	4	4	2.0	3.30
5	2	5	5	1.0	4.70
6	2	1	1	1.0	8.90
7	2	2	2	1.0	5.40

Obs.: Mesmo sendo definido um valor menor que 0 (zero) para o campo `quantidade`, o valor 1 (um) foi definido, por ação do TRIGGER.

Excluindo um TRIGGER

Para excluir um **TRIGGER**, utilize a sintaxe:

```
DROP TRIGGER trigger_name;
```

Onde:

`trigger_name`: é o nome do **TRIGGER**.

Pela interface gráfica do **Workbench**, podemos acessar a área de **TRIGGERS** da estrutura da Tabela, assim como fizemos para criá-la:

Table Name: compra_produto Schema: minimercado

Collation: utf8 - default collation Engine: InnoDB

Comments:

▼ BEFORE INSERT
 compra_produto_BEFORE_INSERT
 AFTER INSERT
 BEFORE UPDATE
 AFTER UPDATE
 BEFORE DELETE
 AFTER DELETE

Clique para excluir o TRIGGER

```

1 CREATE DEFINER='root'@'localhost' TRIGGER `minimercado`.`compra_produto_BEFORE_INSERT`
2 BEFORE INSERT ON `compra_produto` FOR EACH ROW
3 BEGIN
4     -- utilizando variáveis
5     set @qtd_comp = new.quantidade;
6     -- validando se quantidade é igual ou menor a 0
7     if (@qtd_comp) <= 0 then
8         -- atribui 1 se a quantidade for inválida
9         set new.quantidade = 1;
10    end if;
11 END
  
```

Columns Indexes Foreign Keys Triggers Partitioning Options

Podemos aproveitar o contexto do minimercado e criar um **TRIGGER** para atualização automática da quantidade em estoque a partir da compra de um produto pelo cliente utilizando para isso o seguinte código no gatilho de **AFTER INSERT**, ou seja, após a inserção.

update produto

Quantidade do produto atualmente em estoque

Quantidade do produto comprada pelo cliente

```

set quantidade_estoque = quantidade_estoque - new.quantidade
where codigo_produto = new.codigo_produto;
  
```

Table Name: compra_produto Schema: minimercado

Collation: utf8 - default collation Engine: InnoDB

Comments:

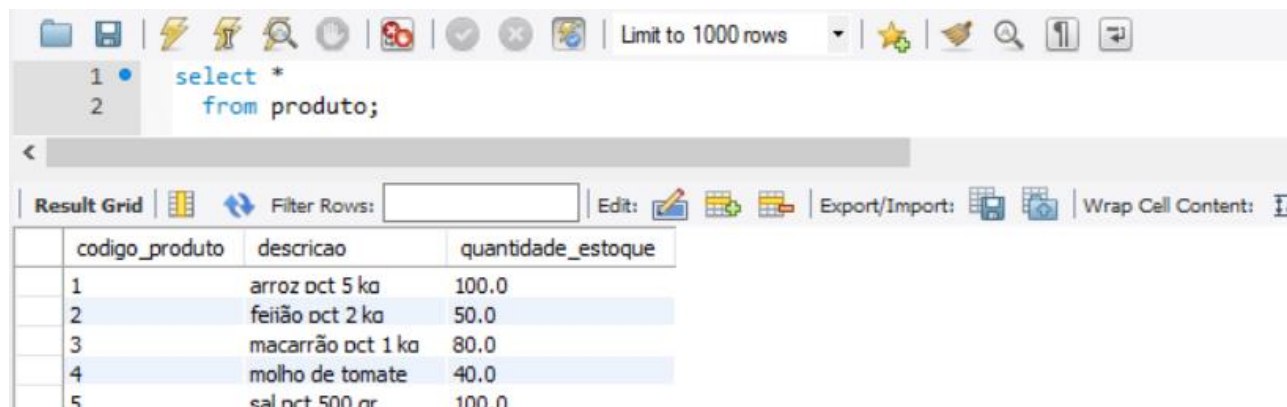
▼ BEFORE INSERT
 compra_produto_BEFORE_INSERT
 ▼ AFTER INSERT
 compra_produto_AFTER_INSERT
 BEFORE UPDATE
 AFTER UPDATE
 BEFORE DELETE
 AFTER DELETE

```

1 CREATE DEFINER='root'@'localhost' TRIGGER `minimercado`.`compra_produto_AFTER_INSERT`
2 AFTER INSERT ON `compra_produto` FOR EACH ROW
3 BEGIN
4     update produto
5     set quantidade_estoque = quantidade_estoque - new.quantidade
6     where codigo_produto = new.codigo_produto;
7 END
  
```

Columns Indexes Foreign Keys Triggers Partitioning Options

Para evidenciar a execução do **TRIGGER** vamos verificar o saldo atual de todos os produtos cadastrados no estoque.



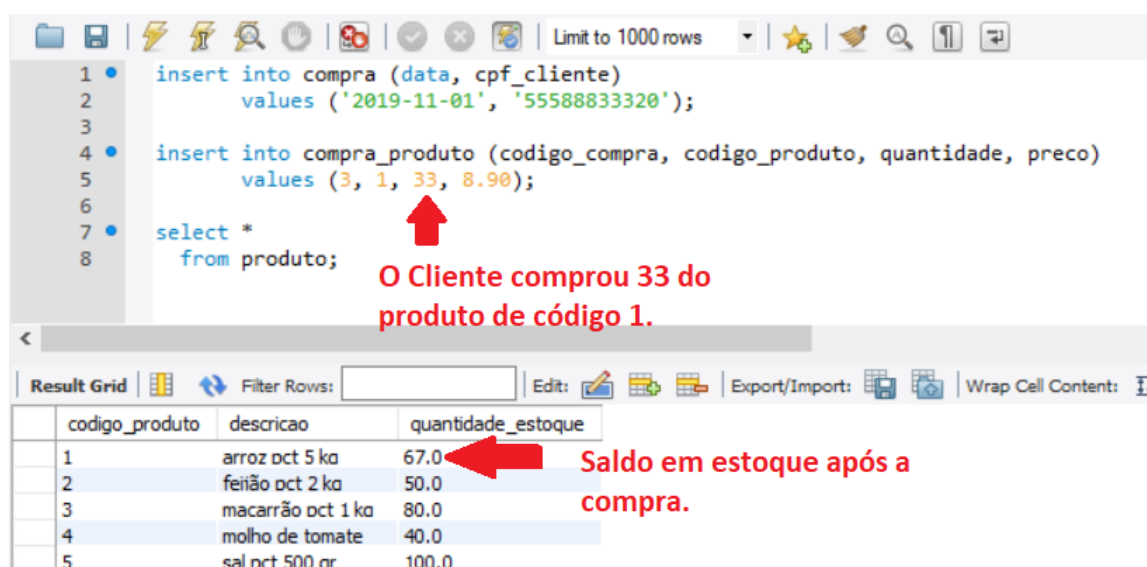
The screenshot shows a SQL IDE interface. At the top, there's a toolbar with various icons. Below it, a query editor shows the following SQL code:

```
1 select *
2 from produto;
```

Below the query editor, there's a 'Result Grid' section. It contains a table with the following data:

	codigo_produto	descricao	quantidade_estoque
1	1	arroz pct 5 ka	100.0
2	2	feijão pct 2 ka	50.0
3	3	macarrão pct 1 ka	80.0
4	4	molho de tomate	40.0
5	5	sal pct 500 gr	100.0

Vamos simular a venda de um dos produtos e depois verificar como ficou o saldo.



The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
1 insert into compra (data, cpf_cliente)
2 values ('2019-11-01', '55588833320');
3
4 insert into compra_produto (codigo_compra, codigo_produto, quantidade, preco)
5 values (3, 1, 33, 8.90);
6
7 select *
8 from produto;
```

Below the query editor, there's a 'Result Grid' section. It contains a table with the following data:

	codigo_produto	descricao	quantidade_estoque
1	1	arroz pct 5 ka	67.0
2	2	feijão pct 2 ka	50.0
3	3	macarrão pct 1 ka	80.0
4	4	molho de tomate	40.0
5	5	sal pct 500 gr	100.0

Annotations in red text and arrows are present:

- An arrow points to the value '33' in the SQL query, with the text: "O Cliente comprou 33 do produto de código 1."
- An arrow points to the value '67.0' in the 'quantidade_estoque' column of the first row, with the text: "Saldo em estoque após a compra."

Com a ação do **TRIGGER** o saldo do produto **Arroz pct 5 kg** que era de 100 passou a ser de 67, após o lançamento da compra de 33 unidades.

Restrições em relação à TRIGGERS

A implementação deste recurso atualmente no **MySQL** tem várias limitações, a conferir as principais:

- Não se pode chamar diretamente um **TRIGGER** com **CALL**, como se faz com um **Stored Procedures**;

- Não é permitido iniciar ou finalizar transações em meio à **TRIGGERS**;

- Não se pode criar um **TRIGGER** para uma tabela temporária – **TEMPORARY TABLE**;

TRIGGERS ainda não podem ser implementadas com a intenção de devolver para o usuário ou para uma aplicação mensagens de erros.

Retirado e adaptado de “MySQL – TRIGGERS”, disponível em <https://www.devmedia.com.br/mysql-triggers/8088>.

Acessado em 06/11/2019.

Agora é com você!!!



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 já sabe como incluir o registro na **worklist** dos aparelhos e o que fazer quando um exame é realizado, para isso ele criou os **PROCEDURES** `incluir_worklist` e `confirmar_realizacao_exame` respectivamente. O desafio agora é tentar responder duas questões fundamentais para seguir com o projeto:

- 1 - Quando devo incluir o registro na **worklist** do aparelho?
- 2 - Quando devo confirmar a realização do exame?

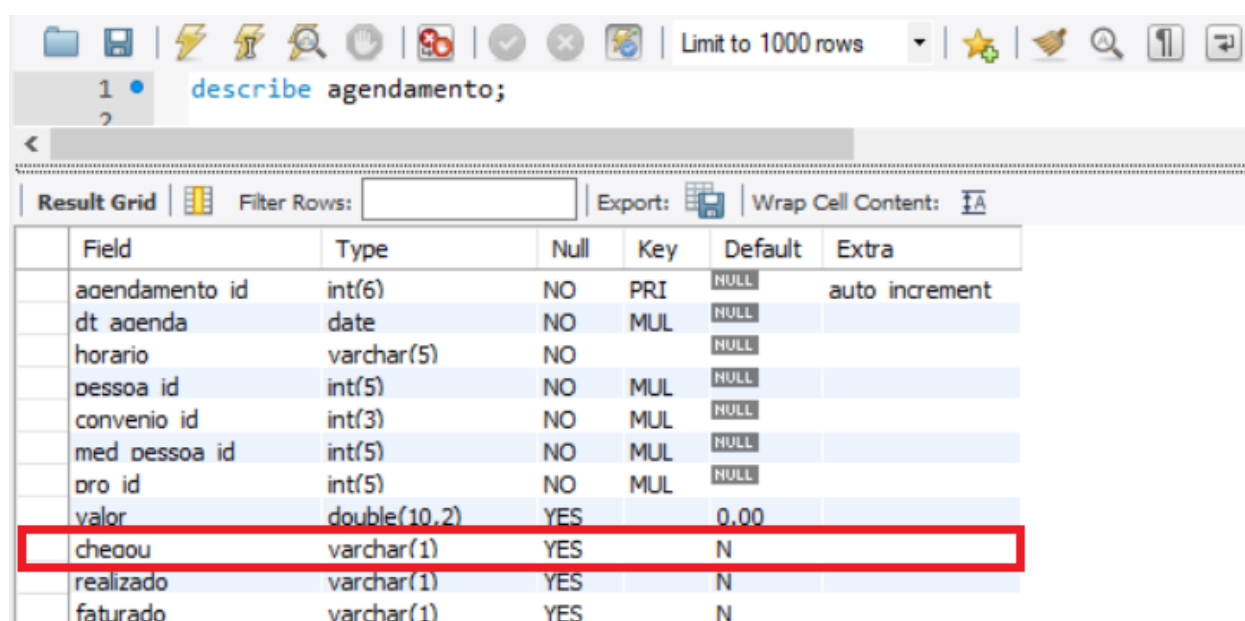
Será que você consegue auxiliar o Carlos a responder essas questões e propor uma solução? Vamos ver!!! Como foi? Conseguiu chegar em alguma solução? Foi difícil? Tenho certeza que não!!!!

O principal objetivo da fase de análise de um projeto é a coleta de requisitos. Esses requisitos devem ficar sempre à disposição da equipe desenvolvedora para que todas as soluções sejam definidas com base nas necessidades e regras de negócio do cliente, sem desvios de foco ou alteração de

escopo. No caso do Consultório da Dra. Ana Lúcia, temos duas informações que nos ajudarão a auxiliar o Analista Carlos nesse novo desafio. São elas:

A) “Somente serão integrados os agendamentos recepcionados para realização de procedimentos de imagem.”

Isso quer dizer que o registro deverá ser incluído na **worklist** do aparelho quando o **agendamento for recepcionado**. Mas o que caracteriza um registro de **agendamento recepcionado**?



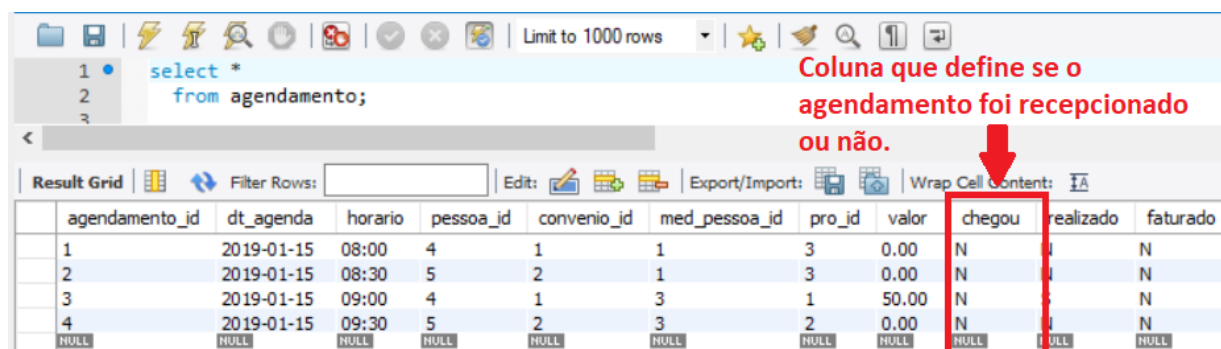
1 • describe agendamento;

2

Result Grid | Filter Rows: | Export: | Wrap Cell Content: IA

Field	Type	Null	Key	Default	Extra
agendamento id	int(6)	NO	PRI	NULL	auto increment
dt agenda	date	NO	MUL	NULL	
horario	varchar(5)	NO		NULL	
pessoa id	int(5)	NO	MUL	NULL	
convenio id	int(3)	NO	MUL	NULL	
med pessoa id	int(5)	NO	MUL	NULL	
pro id	int(5)	NO	MUL	NULL	
valor	double(10,2)	YES		0.00	
chegou	varchar(1)	YES		N	
realizado	varchar(1)	YES		N	
faturado	varchar(1)	YES		N	

Estrutura da Tabela Agendamento



1 • select *

2 from agendamento;

3

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: IA

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

Coluna que define se o agendamento foi recepcionado ou não.

Registros da Tabela Agendamento

Ao analisar a estrutura e os registros da Tabela **agendamento**, podemos verificar que o campo **chegou** define o que podemos chamar de **agendamento recepcionado**, quando preenchido com a letra “S”.

IMPORTANTE: Não se esqueça que a integração será utilizada somente para procedimento de imagem. Mas fique tranquilo, criamos a Function `fnc_tipo_proc()` para resolver isso.

Por padrão, um agendamento é criado como **não recepcionado**, ou seja, campo **chegou** igual a letra “N”, significando que para que ele seja considerado recepcionado, é necessário aplicar uma alteração no registro (**UPDATE**), passando seu conteúdo para “S”. Você se lembra dos **TRIGGERS** ou gatilhos???? Com certeza sim!!!! Podemos implementar gatilhos para executar processos antes (**BEFORE**) ou depois (**AFTER**) de operações de inserção (**INSERT**), alteração (**UPDATE**) ou exclusão (**DELETE**).

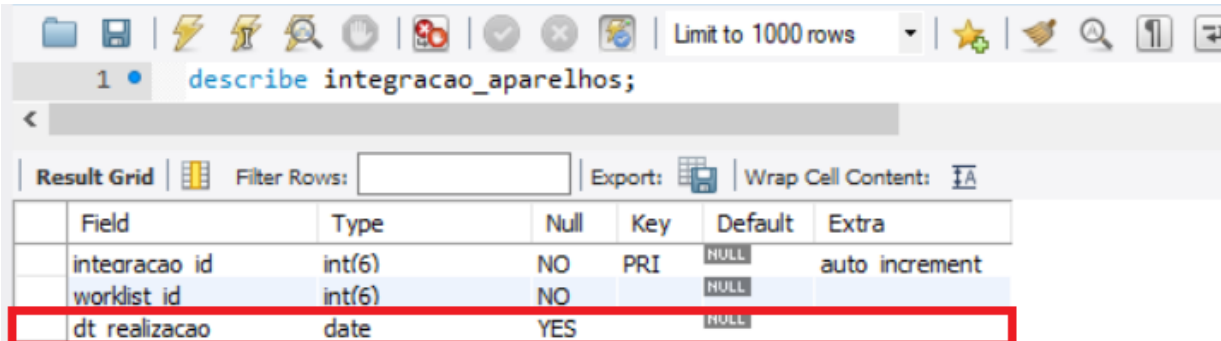
Com isso, você conseguiu auxiliar o Carlos a responder a primeira questão!!!!

Quando devo incluir o registro na worklist dos aparelhos?

*“Devo incluir o registro na worklist dos aparelhos após a alteração do conteúdo do campo **chegou** do registro de agendamento para a letra “S”.*

B) “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 à realização do exame.”

Isso quer dizer que o processo de confirmação de realização do exame deverá ser executado quando o aparelho retornar à execução do exame por meio da Tabela `integracao_aparelhos`. Mas o que caracteriza um registro integrado que tenha sido **retornado sua realização do aparelho?**



Field	Type	Null	Key	Default	Extra
integracao id	int(6)	NO	PRI	NULL	auto increment
worklist id	int(6)	NO		NULL	
dt realizacao	date	YES		NULL	

Estrutura da Tabela Integracao_Aparelhos

Ao analisar a estrutura e os registros da Tabela `integracao_aparelhos`, podemos verificar a existência de 3 (três) colunas:

- `integracao_id`: correspondente ao identificador da integração.
- `worklist_id`: correspondente ao identificador da worklist do agendamento que foi integrado.
- `dt_realizacao`: correspondente a data de realização do exame.

Logo, podemos concluir que a realização de um exame é definida quando o campo `dt_realizacao` for preenchido (**UPDATE**), ou seja, enquanto ele não for preenchido significa apenas que a integração do exame foi confirmada e que está aguardando sua realização, situação essa definida na inclusão (**INSERT**) do registro.

IMPORTANTE: O processo de integração do registro de agendamento do exame com os aparelhos e o retorno de sua realização é um processo que deverá ser realizado pela Equipe Técnica da empresa fornecedora dos mesmos, tendo seu fluxo e execução descritos na documentação de integração, entregue no início do projeto. Não se preocupe como isso irá acontecer, na próxima e última agenda iremos falar mais sobre isso.

Agora você conseguiu auxiliar o Carlos a responder a segunda questão!!!!

Quando devo confirmar a realização do exame?

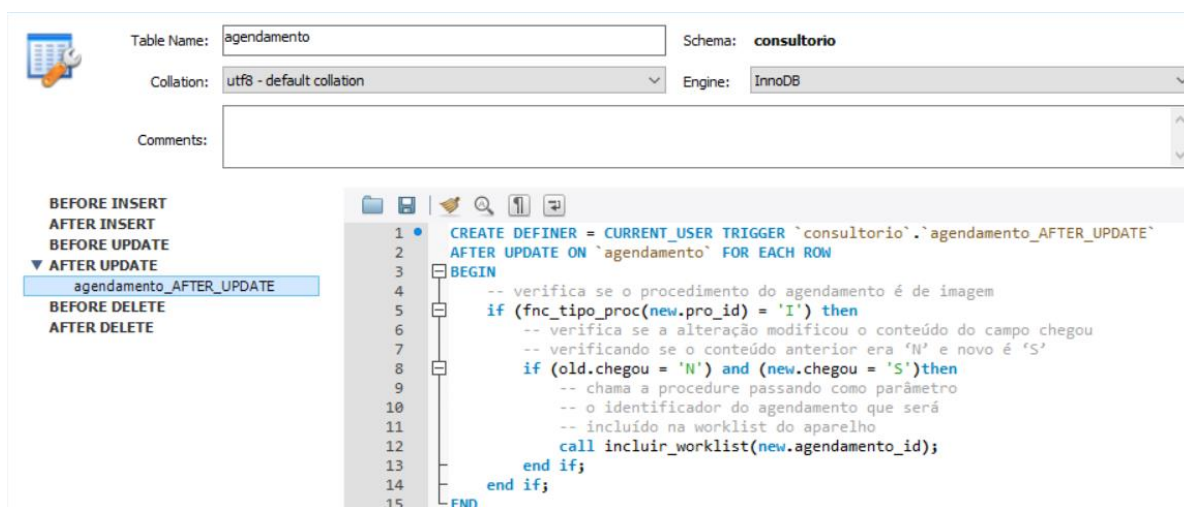
“Devo confirmar a realização do exame após o preenchimento do conteúdo do campo `dt_realizacao` após a alteração do registro.

Nos dois casos, os conteúdos dos campos são definidos após uma operação de alteração realizada nos registros, sendo assim, e com base nos conhecimentos adquiridos nesta agenda, concluímos que Carlos conseguirá cumprir mais essa etapa do projeto, sem a necessidade de solicitar qualquer tipo de alteração em nenhum dos sistemas envolvidos, apenas com a implementação no banco de

dados de dois **TRIGGERS** (gatilhos) de **AFTER UPDATE** (após uma alteração no registro) nas Tabelas de **agendamento** e **integracao_aparelhos**, utilizando a seguinte codificação:

- Na Tabela **agendamento**

```
-- verifica se o procedimento do agendamento é de imagem
if (fnc_tipo_proc(new.pro_id) = 'I') then
-- verifica se a alteração modificou o conteúdo do campo chegou
-- verificando se o conteúdo anterior era 'N' e novo é 'S'
if (old.chegou = 'N') and (new.chegou = 'S') then
-- chama a procedure passando como parâmetro
-- o identificador do agendamento que será
-- incluído na worklist do aparelho
call incluir_worklist(new.agendamento_id);
end if;
end if;
```



- Na Tabela `integracao_aparelhos`

```
-- verifica se a alteração modificou o conteúdo do campo dt_realizacao
-- verificando se o conteúdo anterior era nulo e novo não é
if (old.dt_realizacao is null) and (new.dt_realizacao is not null)
then
-- chama a procedure passando como parâmetro
-- o identificador da integração do agendamento
-- que terá sua realização confirmada
    call confirmar_realizacao_exame(new.worklist_id);
end if;
```

Table Name: `integracao_aparelhos` Schema: `consultorio` Collation: `utf8 - default collation` Engine: `InnoDB`

Comments:

BEFORE INSERT
AFTER INSERT
BEFORE UPDATE
▼ AFTER UPDATE
 `integracao_aparelhos_AFTER...`
BEFORE DELETE
AFTER DELETE

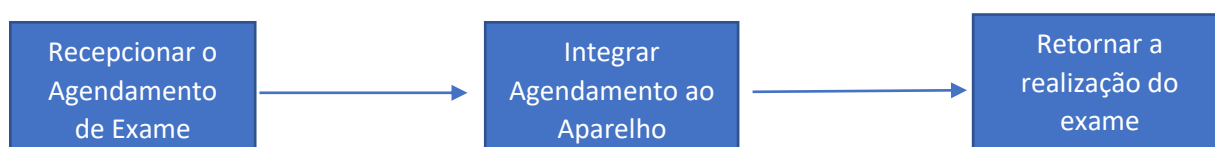
```
1 CREATE DEFINER = CURRENT_USER TRIGGER `consultorio`.`integracao_aparelhos_AFTER_UPDATE`
2 AFTER UPDATE ON `integracao_aparelhos` FOR EACH ROW
3 BEGIN
4     -- verifica se a alteração modificou o conteúdo do campo dt_realizacao
5     -- verificando se o conteúdo anterior era nulo e novo não é
6     if (old.dt_realizacao is null) and (new.dt_realizacao is not null) then
7         -- chama a procedure passando como parâmetro
8         -- o identificador da integração do agendamento
9         -- que terá sua realização confirmada
10        call confirmar_realizacao_exame(new.worklist_id);
11    end if;
12 END
13
```

Para verificar se o processo está sendo executado corretamente, escolha um registro de agendamento de exame que ainda não tenha sido recepcionado:

```
1 select *
2 from agendamento;
```

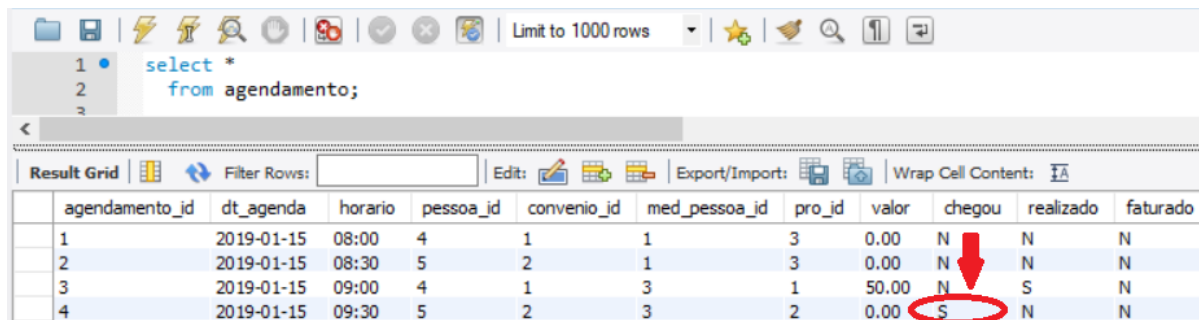
	agendamento_id	dt_agenda	horario	pessoa_id	convenio_id	med_pessoa_id	pro_id	valor	chegou	realizado	faturado
1	1	2019-01-15	08:00	4	1	1	3	0.00	N	N	N
2	2	2019-01-15	08:30	5	2	1	3	0.00	N	N	N
3	3	2019-01-15	09:00	4	1	3	1	50.00	N	S	N
4	4	2019-01-15	09:30	5	2	3	2	0.00	N	N	N

Vamos simular o fluxo a seguir recepcionando o agendamento que possui identificador 4 (quatro):



Recepcionar o agendamento de exame (*passo que será realizado no Sistema do Consultório*)

```
update agendamento a
  set a.chegou = 'S'
  where a.agendamento_id = 4;
```



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

Evidência da recepção do agendamento

```

1 select *
2   from worklist_aparelhos;

```

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	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	RY	N	
2	4	2019-01-15	José de Souza	1991-03-17	AMIL	ULTRASSONOGRAFIA DE ABDOMEN TOTAL	US	N	

Evidência da execução do gatilho da Tabela agendamento com a inclusão do registro na worklist dos aparelhos

2 - Integrar o agendamento ao aparelho (*passo que será realizado no Sistema dos Aparelhos*)

```
insert into integracao_aparelhos (worklist_id)
  values (1);

update worklist_aparelhos w
  set w.integrado = 'S'
  where w.agendamento_id = 4;
```

Você pode estar pensando, esse acesso ao banco de dados por Sistema terceiro não deveria ser controlado? Com certeza sim, mas vamos ver isso na próxima e última agenda, por enquanto estamos somente simulando essa operação para que possamos disparar os gatilhos criados, verificando se tudo funcionando corretamente.

1 • `select *`
2 `from integracao_aparelhos;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

integracao_id	worklist_id	dt_realizacao
1	2	NULL

1 • `select *`
2 `from worklist_aparelhos;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

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
2	4	2019-01-15	José de Souza	1991-03-17	AMIL	ULTRASSONOGRAFIA DE ABDOMEN TOTAL	US	S

Evidências da integração do agendamento

3 - Retornar a realização do exame *(passo que será realizado no Sistema dos Aparelhos)*

```
update integracao_aparelhos i
set i.dt_realizacao = '2019-1-15'
where i.integracao_id = 1;
```

1 • `select *`
2 `from integracao_aparelhos;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

integracao_id	worklist_id	dt_realizacao
1	2	2019-01-15

1 • `select *`
2 `from agendamento;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

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	62.60	S	S	N

Evidências do retorno do Aparelho quanto a realização do exame

Sucesso!!!! Os **TRIGGERS** implementados foram disparados corretamente quando da recepção do agendamento e no retorno do Aparelho quando da realização do exame. Executando tudo o que havia sido previsto e implementado nas Procedures e Functions.

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

Vamos para as atividades online!