Montando a Database e Suas Tabelas:

Inicia-se criando uma database chamada PIZZARIA e as tabelas pertencentes à esta database. Assim...:

```
create database pizzaria;

CREATE TABLE TipoProduto (
  idTipoProduto INT(2) NOT NULL,
  DescTipo VARCHAR(40) NULL,
  PRIMARY KEY(idTipoProduto)
);

CREATE TABLE Pizzaria (
  idPizzaria INTEGER NOT NULL,
  NomePizzaria VARCHAR(40) NULL,
  PRIMARY KEY(idPizzaria)
);

CREATE TABLE Entregador (
  idEntregador INTEGER NOT NULL,
  Nome VARCHAR(40) NULL,
  Salario REAL(8,2) NULL,
  PRIMARY KEY(idEntregador)
);

CREATE TABLE Cliente (
  idCliente INT(5) NOT NULL,
  idPizzaria INTEGER NOT NULL,
  Nome VARCHAR(40) NULL,
  Telefone INT(8) NULL,
  PRIMARY KEY(idCliente),
  INDEX Cliente_FKIndex1(idPizzaria),
  FOREIGN KEY(idPizzaria)
    REFERENCES Pizzaria(idPizzaria)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);

CREATE TABLE Produto (
  idProduto INTEGER NOT NULL,
  idTipoProduto INT(2) NOT NULL,
  Nome VARCHAR(40) NULL,
  preco REAL(8,2) NULL,
  PRIMARY KEY(idProduto),
  INDEX Produto_FKIndex1(idTipoProduto),
  FOREIGN KEY(idTipoProduto)
    REFERENCES TipoProduto(idTipoProduto)
      ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION
);

CREATE TABLE Pedido (
  idPedido INT(5) NOT NULL,
  idEntregador INTEGER NOT NULL,
  idCliente INT(5) NOT NULL,
  PrecoTotal REAL(8,2) NULL,
  PRIMARY KEY(idPedido),
  INDEX Pedido_FKIndex1(idCliente),
  INDEX Pedido_FKIndex2(idEntregador),
  FOREIGN KEY(idCliente)
    REFERENCES Cliente(idCliente)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  FOREIGN KEY(idEntregador)
    REFERENCES Entregador(idEntregador)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);

CREATE TABLE PedidoItem (
  idPedido INT(5) NOT NULL,
  idProduto INTEGER NOT NULL,
  Qtde INT(3) NULL,
  Preco REAL(8,2) NULL,
  PRIMARY KEY(idPedido, idProduto),
  INDEX PedidoItem_FKIndex1(idPedido),
  INDEX PedidoItem_FKIndex2(idProduto),
  FOREIGN KEY(idPedido)
    REFERENCES Pedido(idPedido)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  FOREIGN KEY(idProduto)
    REFERENCES Produto(idProduto)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);
```

Depois de criadas as tabelas iniciaremos a criação dos dados (ou População das tabelas)

Funcionamento:

Populando a tabela com informações Iniciais...

```
Insert into pizzaria values (1,'Pizzaria Bela Massa');

Insert into entregador values (1,'Pedro',300);
Insert into entregador values (2,'Joao',300);

Insert into tipoproduto values (1,'Pizzas');
Insert into tipoproduto values (2,'Bebidas');

Insert into produto values (1,1,'Mussarela',10.00);
Insert into produto values (2,1,'Portuguesa',14.00);
Insert into produto values (3,1,'Calabresa',12.00);
Insert into produto values (4,1,'Quatro Queijos',13.00);
Insert into produto values (5,2,'Coca Cola',3.50);
Insert into produto values (6,2,'Guarana Antarctica',2.00);
Insert into produto values (7,2,'Fanta Laranja',2.50);

Select * from pizzaria;
Select * from entregador;
Select * from tipoprodto;
Select * from produto;
```

As informações iniciais servem para colocar em funcionamento o sistema. Ou seja, até este instante, a pizzaria está organizando suas informações para iniciar seu funcionamento. Depois de geradas estas informações, os clientes ligarão solicitando os produtos para a pizzaria. Assim, será realizado o cadastro dos mesmos...:

```
Insert into cliente values (1,1,'Maria',36452431);
Insert into cliente values (2,1,'Jose',37863422);
Insert into cliente values (3,1,'Ana',39872346);

Select * from cliente;
```

Criando uma Trigger:

A finalidade da trigger é que ela atualize uma determinada tabela automaticamente. Ou seja, quando o cliente ligar e solicitar seu pedido, os detalhes deste pedido (ou os produtos adquiridos) serão registrados na tabela PEDIDOITEM.

Ou seja, quando o cliente ligar e especificar os produtos que deseja comprar, cada produto ficará armazenado na tabela PEDIDOITEM. Porém o valor (ou preço) dos produtos contidos nesta tabela será atualizado conforme o preço especificado na tabela PRODUTO (Ou seja: Preço = Qtde * Preço Unitário). Assim...:

Utilizando o Software – MySQL Front

```
create trigger CalculaPreco
        before insert on PedidoItem
        for each row
        begin
                set new.Preco=new.Qtde*(select preco from Produto where  idProduto=new.idProduto);
        end;
```

Utilizando o Software – HEIDI SQL

```
delimiter //
create trigger CalculaPreco
        before insert on PedidoItem
        for each row
        begin
                set new.Preco=new.Qtde*(select preco from Produto where
        idProduto=new.idProduto);
        end;
//


show triggers;
```

Iniciando o processo de atendimento ao cliente...:

Quando o cliente ligar e fizer seu pedido, o preço dos produtos contidos no pedido (realizado por este cliente) serão atualizados automaticamente pelo trigger. Ou seja, primeiramente cria-se um pedido para o Cliente, **não informando o valor total deste pedido**. Ou seja, da seguinte forma:

insert into pedido (idPedido, idEntregador, idCliente) values(1,1,2);

Pedido	Entregador	Cliente	PreçoTotal
1	Pedro	Maria	(Será calculado pela Stored Procedure)

select * from pedido;

Depois de criado um pedido (sem o valor total), o cliente especifica neste pedido (Pedido 1) os produtos que deseja adquirir. São eles:

Pedido	Produto	Qtde	Preço
1	Mussarela	2	(Será calculado pelo trigger)
1	Portuguesa	3	(Será calculado pelo trigger)
1	Guarana Antarctica	2	(Será calculado pelo trigger)

Assim, quando forem cadastrados os produtos, os valores (O Preço de cada item será atualizado conforme a fórmula acima mencionada) de cada produto serão atualizados pelo trigger...:

insert into pedidoItem (idPedido,idProduto,Qtde) values(1,1,2);
insert into pedidoItem (idPedido,idProduto,Qtde) values(1,2,3);
insert into pedidoItem (idPedido,idProduto,Qtde) values(1,6,2);

Select * from pedidoitem;

Criando uma Stored Procedure:

A finalidade da Stored Procedure (SP) é que ela atualize uma determinada tabela quando acionada. Assim, a intenção desta atividade é que a SP feche o pedido feito pelo cliente alterando o valor total do pedido conforme a soma dos preços de cada produto feito pelo cliente. Ou seja, quando o cliente fechar seu pedido, o valor total deste pedido (conforme o preço dos produtos) será registrado na tabela PEDIDO.

Assim, quando o cliente encerrar seu pedido o valor Total ficará atualizado na tabela PEDIDO (inclusive para não haver divergência entre seus valores: PEDIDO e PEDIDOITEM). Porém o valor (ou preço) dos produtos contidos nesta tabela será atualizado conforme o preço especificado na tabela PEDIDOITEM (Ou seja: PreçoTotal = Soma dos Valores dos produtos conforme o Pedido). Assim...:

Utilizando o Software – MySQL Front

```
create procedure SomaPedido(out valortot real(8,2))
begin
        select sum(Preco) into valortot from pedidoitem;
        update pedido set precototal= (select @a);
end;
```

Utilizando o Software – HEIDI SQL

```
delimiter //
create procedure SomaPedido(out valortot real(8,2))
begin
        select sum(Preco) into valortot from pedidoitem;
        update pedido set precototal= (select @a);
end;
//
```

Para fechar o pedido do cliente, a Stored Procedure deve ser executada... ou seja:

```
call SomaPedido(@a);
```

Exercícios:

O exemplo acima foi realizado com um único pedido dentro da base de dados... Qual seria a funcionalidade desta Stored Procedure se houvessem mais pedidos?

Resposta: Provavelmente a Stored Procedure somaria TODOS os valores contidos na tabela PEDIDOITEM (select sum(Preco) into valortot from pedidoitem;) e atribuiria também a TODOS os pedidos contidos na tabela PEDIDO;

Portanto, isso poderia ser resolvido utilizando-se de Triggers e/ou Stored Procedures? Como?