Trader Database Management System

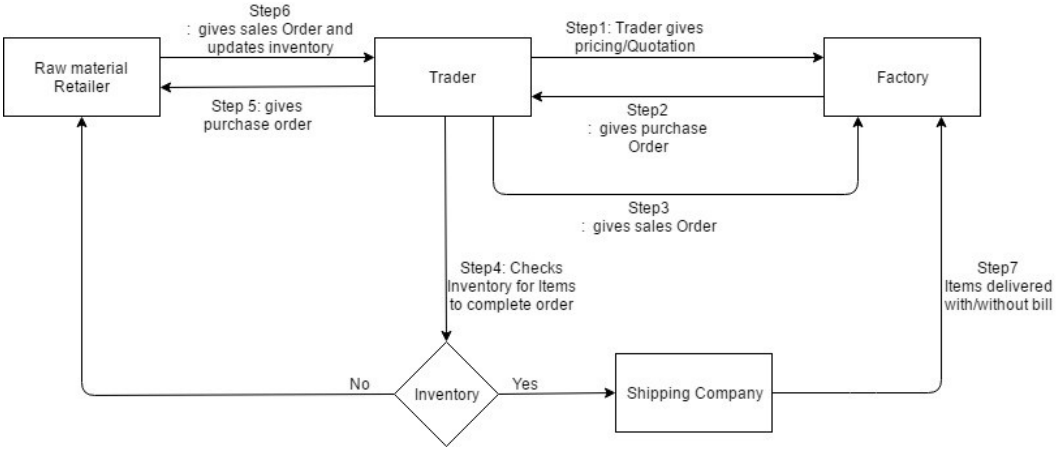
A trading company name M B Enterprise supplying hardware material to pharmaceutical factory, over the years, is handling its business processes manually and keeping records of each sales and purchases in books. However, with increase in sales, CEO is finding difficulties in book keeping and accounting. He is in a need of a system which can handle the accounting and inventory system.

Aim is to create a database system for a M B Enterprise company for tracking Orders and manage billing and inventory systems.

CEO proposes below business process which they are following:

Step 1: Trader gives quotation to Factory (who needs materials like electric hardware or pumps) about the Items/Products and its pricing.
Step 2: On Receiving quotation, Factory gives order to trader which is called as Purchase order(PO).
Step 3: On receiving PO, trading company will issue Sales Order(Challan) which includes contract of material and the price at which trader will give it to factory.
Step 4: Based on Validation of Sales Order; if Sales Order is perfect; Then Trader will look at the inventory for Stock/products that is required to complete order and number of quantities.
Step 5: If stock is not available, he will purchase from Retailer at agreed price and will supply it to factory. Else If stock is available in inventory, he will transport it to factory through Transport company.
Step 6: If stock is not available, Sales order is issued between trader and retailer.
Step 7: Once Sales order are finalized, Bill is generated and sent it with product or may be after a couple of agreed days.
Step 8: Trader needs to check the payment is not due above 90 days with either side; retailer as well as Factory.
Step 9: Trader starts sending alert to Factory for payment dues after 75 days.

Flowchart:



To automate and create above process; below Tables have been designed:

1. Retailer – Details of raw material seller. Name, id and Location.
2. Factory – details of factory- name, location
3. Products - Product id, product name
4. Stocks Inventory – quantity of product.
5. Shipping – details of delivery company.
6. Factory Payment – payment due from factory side details.
8. Purchase Order – order details from factory side.
9. Sales Order – contract details from trader side.

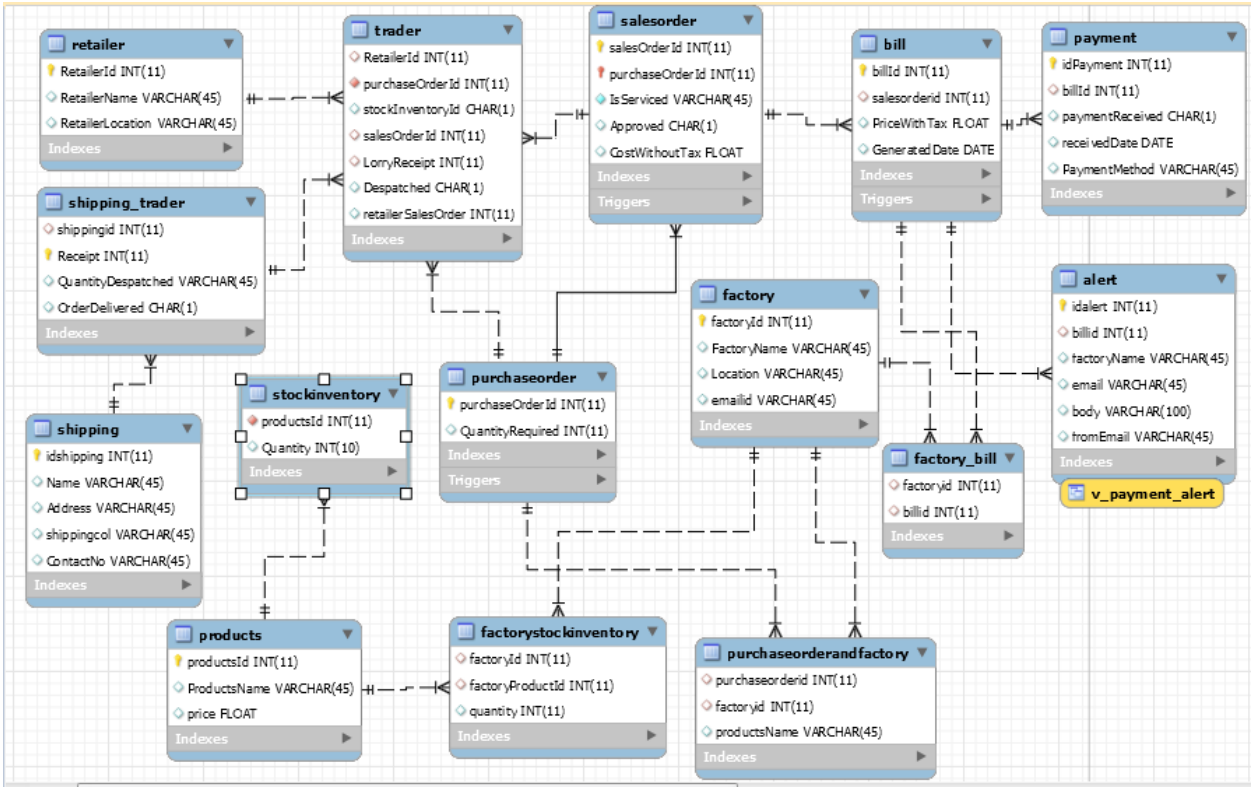
10. Bill – Bill details related to order placed.
11. FactoryStockInventory – quantity of products in factory.
12. Trader – Main table having order details.
13. Payment – Details related to payments.
14. Alert – for Sending alerts for due payment.
15. purchaseorderandfactory – details related to purchaseorder related to factory.

Relationships between Tables:

1. One Trader deals with many factories.
2. Many retailers deal with one trader.
3. One Trader ships through many Shipping companies.
4. One factory gives many purchase orders.
5. Trader gets many purchase orders.
6. One factory gets many sales orders.
7. Sales Order has one or many products.
8. Trader gives many sales order
9. One purchase order will have one sales order.
10. One trader has many orders.
11. Trader deals in many products.
12. Trader has one Stock inventory
13. Stock Inventory has many items.
14. Exactly One Payment is done with respect to one Bill.
15. Alert is sent for delayed bills.

E-R Diagram:

Based on above flow; E-R diagram is as follows:



Techniques:
Based on above ER diagram and flow chart, the techniques used to complete this system are:

1. Procedures
2. Triggers
3. Transactions
4. View.

Based on problem statement, below database has been designed:

1. Factory will give purchase order for quantity less than 5 which will trigger salesorder table and an entry will be present in purchaseorder table and salesorder table.

```
### Procedure to create purchaseorder based on factory inventory #########
delimiter //
create procedure sp_purchaseOrderInsert( in quant int ,in fid int)
Begin

if exists(select fs.quantity from products, factorystockinventory fs,factory fi
where productsId = factoryProductId
and fs.factoryId = fi.factoryId
and fi.factoryId = fid and fs.quantity < 5)

then
set autocommit =0;
start transaction;
set @factoryid = fid;
set @quantity = quant;
insert into purchaseorder (QuantityRequired)
values(quant);
insert into trader (purchaseorderid) values ((select purchaseOrderid from purchaseOrder
order by purchaseorderid desc limit 1));
commit;
set autocommit = 1;
end if;
end; //
```

2.

```
## Trigger to enter purcahse order into salesorder  table ################

delimiter $$
create trigger afterPOtoSO after
Insert on purchaseorder
for each row
begin
declare quant int;

insert into purchaseorderandfactory(purchaseorderid,factoryid,productsName)
(select (select purchaseorderid from purchaseorder order by 1 desc limit 1 ),
@factoryid,
products.Productsname
from products,
factorystockinventory fs,factory fi
where productsId = factoryProductId
and fs.factoryId = fi.factoryId
and fi.factoryId = @factoryid and fs.quantity < 5);

set quant = @quantity;
set @sumforsales = (select (sum((price*1.20)) * quant) from products where
productsName in (select productsName from purchaseorderandfactory where
purchaseorderid = (select purchaseorderid from purchaseorder order by 1 desc limit 1)));

insert into salesorder(purchaseOrderID,costwithoutTax) values(new.purchaseOrderID,(@sumforsales));
end $$

##############################################################################
```

3. On confirmation of salesorder, it will get entry to traders table for order processing

```
##Function to send sales order and waiting for approval which also  and
####calls procedure to insert updated sales order into trader table###

delimiter $$
create function fn_salesOrderconfirm(c char(1), i int)
returns char(1)
begin
declare s char(1);
if (c = 'y')
then
set s = 'Y';

update salesorder
set Approved = 'Y'
where salesorderid = i;
call sp_insertTotrader(i);
else
set s = 'N';

#return s;
end if;
return s;
end $$

######################################################################
```

4. Trader will check its stock inventory

```
################ Stock Inventory Check ######################

select p.ProductsName,s.Quantity,
pu.QuantityRequired from products p, stockinventory s,purchaseorder pu where
p.productsId = s.productsId
and pu.purchaseorderid =(select purchaseorderid from salesorder where salesorderid = 31)
and p.ProductsName in (select productsName from purchaseorderandfactory ps
where ps.purchaseorderid =
(select purchaseorderid from trader where salesorderID = 31));

##################Retailer call#############################################################
```

5. If stock is present, trader will ship it

```
# Procedure to send shipping from stockinventory and updating inventory quantity and trader for Lorryreceipt

delimiter $$
create procedure sp_Shipping(in salesOrder int,in shippingid int)
begin
set autocommit = 0;
Start transaction;
select @d := QuantityRequired from purchaseorder where purchaseorderid =
(select purchaseorderid from trader where salesorderID = salesorder);
select @d;

update Stockinventory s
set s.quantity = quantity - @d
where s.productsid = (select p.productsid from products p, purchaseorder pu where
p.productsId = s.productsid
and pu.purchaseorderid =(select purchaseorderid from salesorder where salesorderid = salesorder)
and p.ProductsName in (select productsName from purchaseorderandfactory ps
where ps.purchaseorderid =
(select purchaseorderid from trader where salesorderID = salesorder)));

insert into shipping_trader(shippingid,QuantityDespatched)values(shippingid,@d);

update trader set Lorryreceipt = (select Receipt from shipping_trader order by Receipt desc limit 1),
stockInventoryId = 'Y'
where salesorderid = salesorder;
commit;
set autocommit = 1;
end $$
```

6. If stock is not present, trader will buy it from retailer

```
####################### Procedure for reatiler update ##############################

delimiter $$

create procedure sp_retailerUpdateInventory
(in salesorder int, in retailerid int, in retailersalesorder int)
begin
#update trader
update trader
set retailerid = retailerid,retailerSalesOrder = retailerSalesOrder where salesorderid = salesorder;

#update stockinventory
update stockinventory set quantity =
quantity + (select quantityrequired from purchaseorder where purchaseOrderId
= (select purchaseorderid from salesorder where salesorderid = salesorder))
where productsid in (select Productsid from products where ProductsName in
(select Productsname from purchaseorderandfactory where purchaseorderid in
(select purchaseorderid from salesorder where salesorderid = salesorder)));
end $$

###############################################################################################
```

7. Shipping will complete the delivery and update factory stock inventory

```
##############procedure to update factory stock and shipping and trader ##############
delimiter $$

create procedure sp_factoryStockUpdate(in salesorder int)
begin
set autocommit = 0;
Start Transaction;

update factorystockinventory fs set fs.quantity = fs.quantity +
(select quantityDespatched from shipping_TRADER where receipt =
(select Lorryreceipt from trader where salesorderid = salesorder))
where factoryid = (select factoryid from purchaseorderandfactory
where purchaseOrderId = (select purchaseorderid from trader where salesorderid = salesorder)
group by factoryId)
and factoryProductId in (select productsid from products
where productsName in (select productsName from purchaseorderandfactory
where purchaseorderid = (select purchaseorderid from trader where salesorderid = salesorder)));

update trader set despatched = 'Y' where salesorderid = salesorder;
update shipping_trader set orderDelivered = 'Y'
where Receipt = (select LorryReceipt from trader where salesorderid = salesorder);

update salesorder
set IsServiced = 'Y' where salesorderid = salesorder;

commit;
set autocommit = 1;
end $$
```

8. On completion of delivery, trigger will be generated for bill generation and payment generation

```
################### Trigger for bill update ########################################

delimiter $$

create trigger billGeneration
after update on salesorder
for each row
begin
if New.isServiced = 'Y'
then
insert into bill(salesOrderid,pricewithTax,generatedDate)
values(new.salesorderid,
(select (CostWithoutTax*1.12) from salesorder
where salesorderid = new.salesorderid),curdate());
end if;
end $$

#####################################################################################################

################## insertion in payment table#########################################################

delimiter $$
create Trigger PaymentInsert
after insert on Bill
for each row
begin
insert into payment(billid) values(new.billid);
call sp_billinsertfactory(new.billid);
end $$
```

9. Trader will check for payment dues for greater than 75 days;

```
## view for payment alert
CREATE
    VIEW `finalproject`.`v_payment_alert` AS
    SELECT
        `p`.`billId` AS `billid`
    FROM
        (`finalproject`.`payment` `p`
        JOIN `finalproject`.`bill` `b`)
    WHERE
        ((`p`.`paymentReceived` = 'N')
            AND ((TO_DAYS(CURDATE()) - TO_DAYS(CAST(`b`.`GeneratedDate` AS CHAR CHARSET UTF8))) > 75));
```

10.If payment is due for greater than 75 days, alert is send to respective factories.

```
################ Alert for sending email ###################################
delimiter $$
CREATE DEFINER=`bhavik`@`%` PROCEDURE `sp_alert`()
begin
declare done int default false;
declare count int ;

declare cur1 cursor for select p.billid from payment p,bill b where paymentReceived = 'N'
and  datediff(curdate(),cast(b.GeneratedDate as char)) > 75;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
open cur1;
#select @count := count(*) from payment;
l1: loop
fetch cur1 into count;
if done then
select 'Email list aded to Alert list' as '';
leave l1;
else
insert into alert (billid,factoryName,email,body) values((count),
(select factoryName from factory where factoryid =
(select factoryId from factory_bill where billid = count)),
(select emailid from factory where factoryid =
(select factoryId from factory_bill where billid = count)),(
'Payment due by 15 days from now'));
end if;
end loop;
close cur1;
end $$
#############################################################################
```

The above process helped to deliver the requirement in following ways.

Scenario 1:

1. Factory with id 1 having stock as

```
59 •  select * from factorystockinventory;
60
```

Result Grid | Filter Rows: | Export:

factoryId	factoryProductId	quantity
1	1	14
1	2	4
1	3	14
1	4	14
1	5	4
1	6	14
1	7	4
1	8	14
1	9	14
1	10	14

```
17  ######### purchase order insert also triggering insert to sales order#########################
18 •  call sp_purchaseOrderInsert(3,1);
19 •  select * from purchaseorder;
20 •  select * from trader;
21 •  select * from salesorder;
```

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

salesOrderId	purchaseOrderId	IsServiced	Approved	CostWithoutTax
1	1	N	NULL	1152
NULL	NULL	NULL	NULL	NULL

2. On approval, salesorder will get entry in traders table

```
26  ###############Sales order confirm function#################################################
27
28  select fn_salesOrderconfirm('y',1);
29  select * from trader;
30
```

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

RetailerId	purchaseOrderId	stockInventoryId	salesOrderId	LorryReceipt	Despatched	retailerSalesOrder
NULL	1	NULL	1	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

3. Stock Inventory Check

```
33  ################ Stock Inventory Check ######################
34
35  select p.ProductsName,s.Quantity,
36  pu.QuantityRequired from products p, stockinventory s,purchaseorder pu where
37  p.productsId = s.productsId
38  and pu.purchaseorderid =(select purchaseorderid from salesorder where salesorderid = 1)
39  and p.ProductsName in (select productsName from purchaseorderandfactory ps
40  where ps.purchaseorderid =
41  (select purchaseorderid from trader where salesorderID = 1));
```

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

ProductsName	Quantity	QuantityRequired
MOtors	4	3
Safetv Stciker	7	3
Gears	7	3

4. Product is in stock, shipping will be called which will generate Lorry receipt

```
48  #############Shipping procedure if stock present in stockinventory ######
49
50  call sp_shipping(1,1);
51  select * from shipping_trader;
52  select * from trader;
53
54  ##############################################################################
55
```

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

shippingid	Receipt	QuantityDespatched	OrderDelivered
1	1	3	NULL
NULL	NULL	NULL	NULL

5. After shipping, factory will update factory stock inventory

```
call sp_factoryStockUpdate(1);
select * from factorystockinventory;
```

Result Grid | Filter Rows: | Export:

factoryId	factoryProductId	quantity
1	1	14
1	2	7
1	3	14
1	4	14
1	5	7
1	6	14
1	7	7
1	8	14
1	9	14
1	10	14

6. This will generate triggers for bill and payment

```
select * from bill;
```

Result Grid | Filter Rows: | Edit:

billId	salesorderid	PriceWithTax	GeneratedDate
1	1	1290.24	2017-12-13
NULL	NULL	NULL	NULL

```
select * from payment;
```

Result Grid | Filter Rows: | Edit:

idPayment	billId	paymentReceived	receivedDate	PaymentMethod
1	1	N	NULL	NULL
NULL	NULL	NULL	NULL	NULL

7. If factory pays, it will get updated in payment method

```
##################### Payment Confirmation ###################
update payment set paymentReceived = 'Y',ReceivedDate = curdate(),
PaymentMethod = 'Cheque Transaction'
where billid = 1;
select * from payment;
```

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

idPayment	billId	paymentReceived	receivedDate	PaymentMethod
1	1	Y	2017-12-13	Cheque Transaction
NULL	NULL	NULL	NULL	NULL

8. Trader will get the details of order complete as:

```
20  select * from trader;
```

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

RetailerId	purchaseOrderId	stockInventoryId	salesOrderId	LorryReceipt	Despatched	retailerSalesOrder
NULL	1	Y	1	1	Y	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Scenario 2:

1. if Product is not available with trader inventory

```
33  ################ Stock Inventory Check ######################
34
35  select p.ProductsName,s.Quantity,
36  pu.QuantityRequired from products p, stockinventory s,purchaseorder pu where
37  p.productsId = s.productsId
38  and pu.purchaseorderid =(select purchaseorderid from salesorder where salesorderid = 2)
39  and p.ProductsName in (select productsName from purchaseorderandfactory ps
40  where ps.purchaseorderid =
41  (select purchaseorderid from trader where salesorderID = 2));
42
```

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

ProductsName	Quantity	QuantityRequired
MOtors	1	3
Wires	10	3
Switched	10	3

2. Trader will give order to retailer

```
42
43  ##################Retailer call#############################################################
44  #(in salesorder int, in retailerid int, in retailersalesorder int)
45
46  call sp_retailerUpdateInventory(2,1,1);
47  select * from trader;
```

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

RetailerId	purchaseOrderId	stockInventoryId	salesOrderId	LorryReceipt	Despatched	retailerSalesOrder
NULL	1	Y	1	1	Y	NULL
1	2	NULL	2	NULL	NULL	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

3. Payment confirmation and sales-order complete:

```
66  ######################## Payment Confirmation ####################
67  update payment set paymentReceived = 'Y',ReceivedDate = curdate(),
68  PaymentMethod = 'Cheque Transaction'
69  where billid = 2;
70  select * from payment;
71
```

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

idPayment	billId	paymentReceived	receivedDate	PaymentMethod
1	1	Y	2017-12-13	Cheque Transaction
2	2	Y	2017-12-13	Cheque Transaction
NULL	NULL	NULL	NULL	NULL

4. Factory Stock Update:

```
14  select * from factorystockinventory;
15  select * from stockinventory;
```

Result Grid | Filter Rows: | Export:

factoryId	factoryProductId	quantity
2	1	14
2	2	7
2	3	14
2	4	14
2	5	14
2	6	7
2	7	14
2	8	7
2	9	14
2	10	14

Scenario 3:

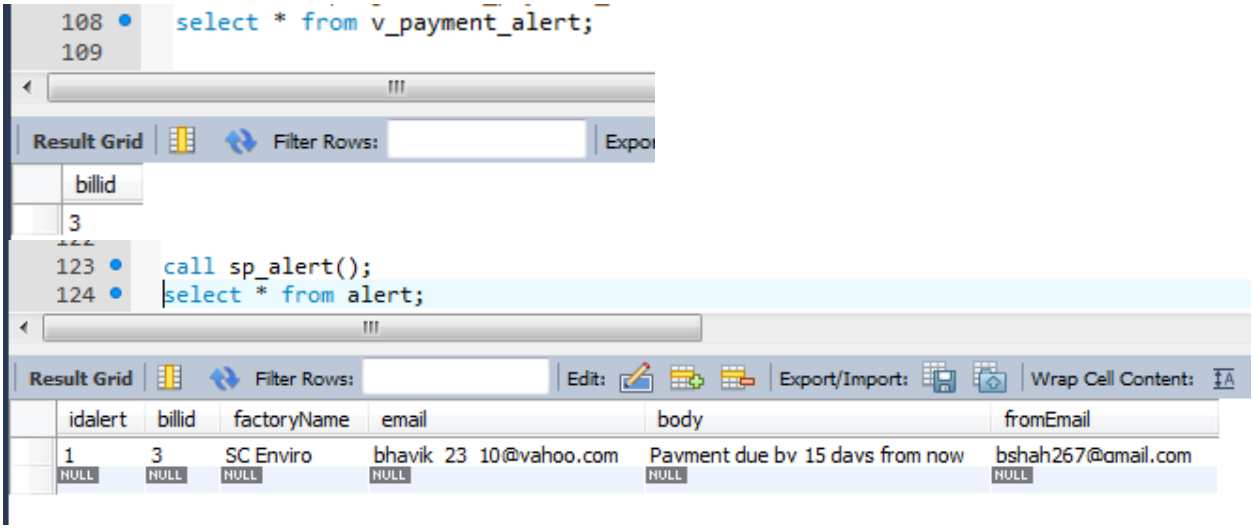
1. if payment is due by more than 75 days:

```
62
63  select * from bill;
```

Result Grid | Filter Rows:

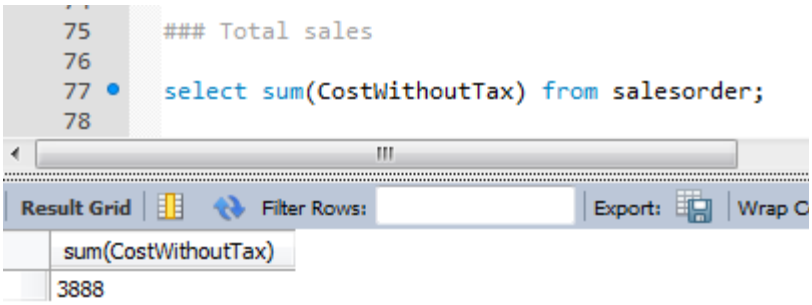
billId	salesorderid	PriceWithTax	GeneratedDate
1	1	1290.24	2017-12-13
2	2	1008	2017-12-13
3	3	2056.32	2017-08-28
NULL	NULL	NULL	NULL

2. Alert will be sent

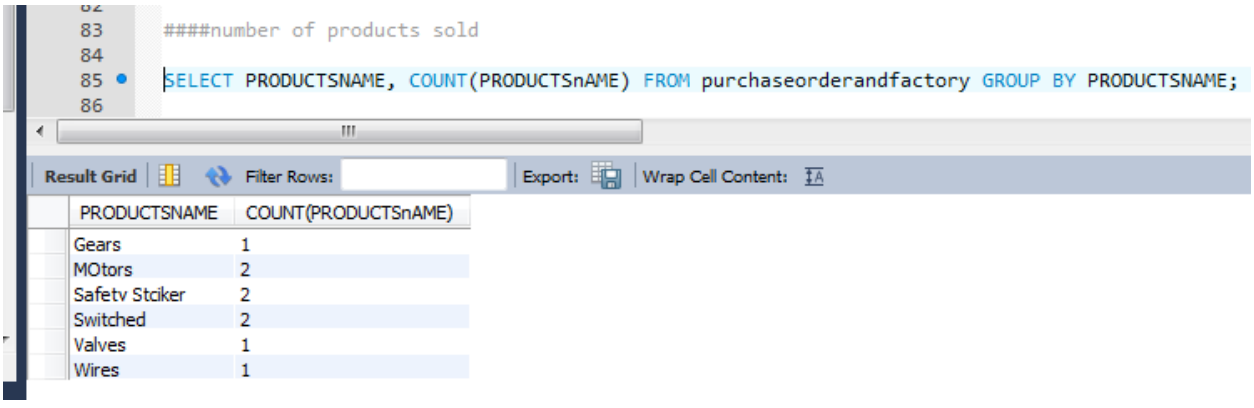


Analytics that can be performed by Trader CEO

1. Total Sales done:



2. Number of products Sold:



3. Delayed cheques

```
103 •  select p.billid from payment p,bill b where paymentReceived = 'N'
104    and  datediff(curdate(),cast(b.GeneratedDate as char)) > 90;
```

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

billid
3

The whole database is being backedup daily at 1 am where load is minimal using .dat file in the events of crash.

backup.bat

Conclusion:

This approach is getting up all the data and requirement detailed by trader and this product can now be deployed to any small scale trader business.