Distribution Management System

(IT252 Database Management System)

Submitted By:-

Yash Gupta (181IT153)
Kartik Reddy Kotla (181IT123)
Jaidev Chittoria (181IT119)

Under the guidance of

Dr. Melwyn D'Souza

In partial fulfillment for the award of the degree of

Bachelor of Technology
in
Information Technology

at

Department of Information Technology
National Institute of Technology Karnataka,
Surathkal

Abstract

The Distribution Management System is a one-stop solution to manage the distribution model of a Business. By using the DMS web application , users can manage their large stock units with streamline payment records and measure performance to enhance the efficiency.

The core idea of this project is to create a web application which connects all the hierarchies. In a typical business there is a Wholesaler and retailers/customers. This system will allow wholesalers ,retailers and customers to manage their inventory efficiently along with placing orders as well as help in measuring performance metrics . It's an easy way to manage large scale management systems and helps in digitizing the whole system. Keeping a track of orders, transactions and stock has been simplified and this app is user friendly enabling absolutely anyone to use it.

Table of Contents

TOPIC **PAGE**

1. Project Overview

2. Database Design

3. Implementation

1.Project Overview

1.1 Introduction

In this 21st century ; the age of technology, in most of the developing countries such as ours, where there is so much advancement in technology and with the reach of internet to around 650+ million people in the country even then the wholesale businesses are still managed and maintained on papers.This is an issue that can not be ignored because this can sometimes lead to inefficient handling of day-to-day business matters and can also lead to security threats to the business due to the presence of physical records.

This project ; Distribution Management System, aims to develop a web application to manage and maintain a business more efficiently and improve the security of the business. The lightweight nature of the web application makes it ideal for small businesses which cannot afford the expensive alternates present in the market.

The Distributional Management System (DMS) has an administrator login for wholesalers with built-in features to keep a check on the sales and the inventory so as to efficiently run the sales division. The customers and the suppliers can also be registered so as to ensure the smooth functioning of the business. There is also a customer login option which helps the customer in monitoring his purchases from the wholesaler. These features of DMS greatly simplifies the daily functioning of the business.

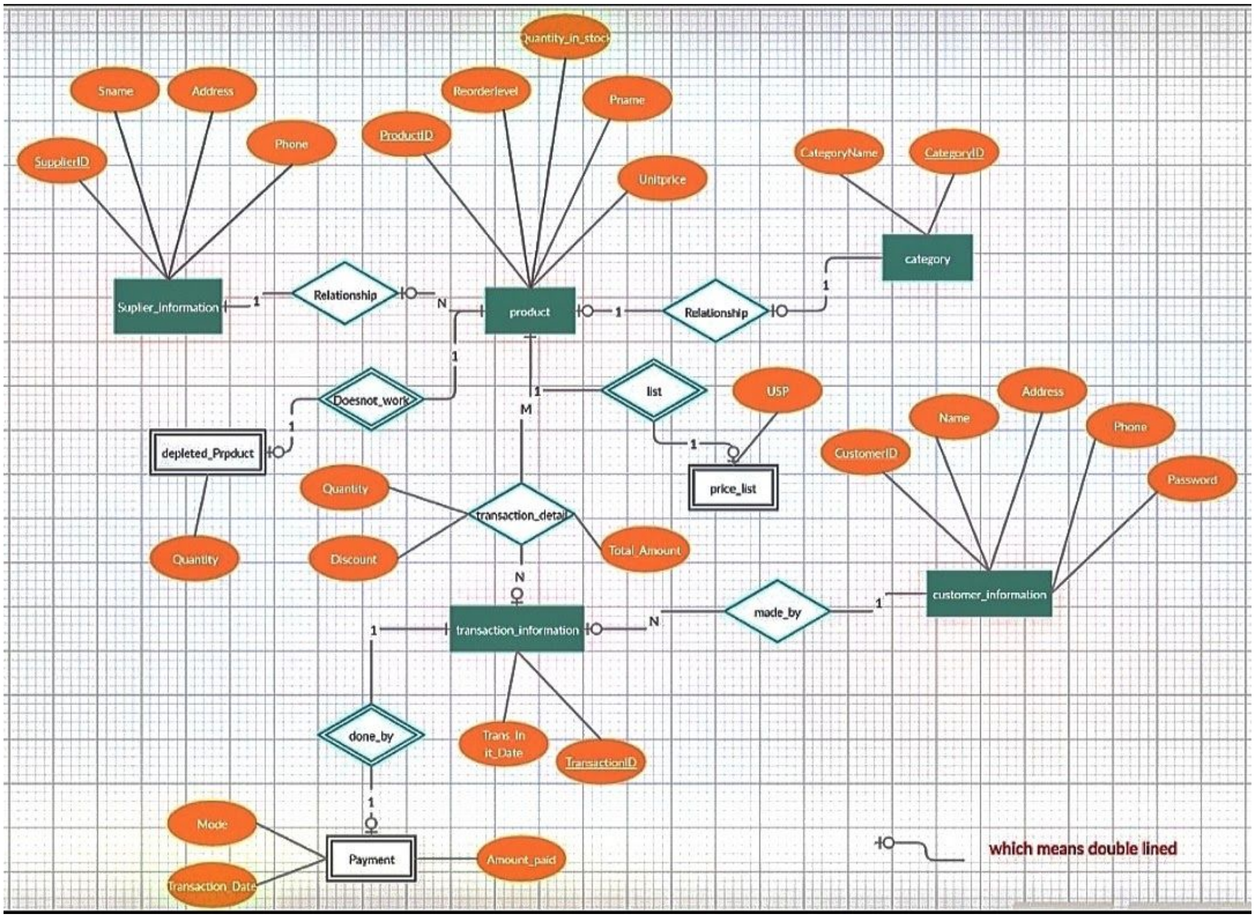
Since, many other developed countries already use such applications to speed up daily business activities ; a lot of information is available in the public domain regarding the desirable structure of such applications. We have used this information to design and model this project.

1.2 Functional requirements

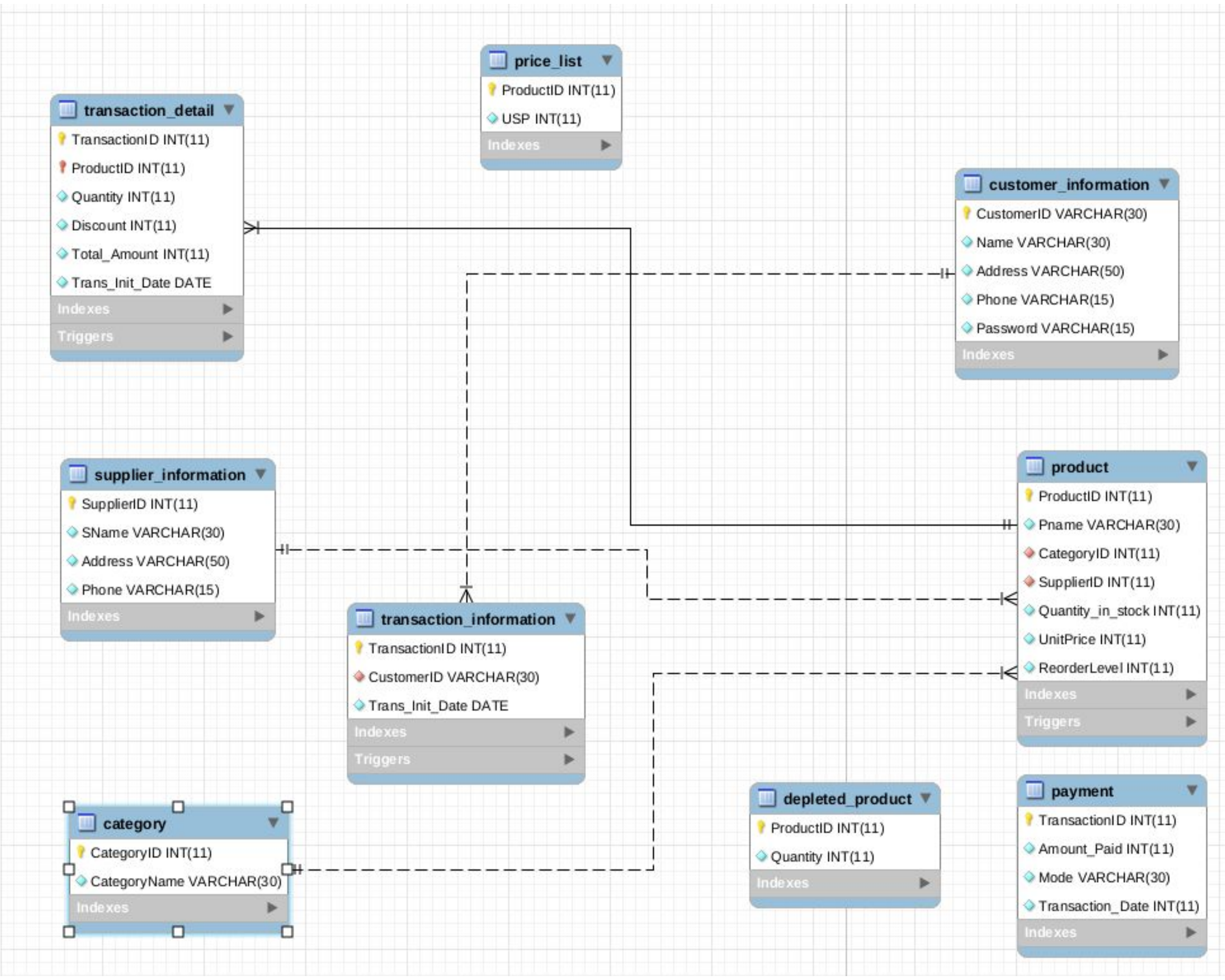
- Add/update product details by accessing them category-wise
- Add/update supplier and customer details.
- Stock maintenance.
- Adding transactions and storing the payment details accordingly.
- Transactions lookup in a specific period of time.
- Generating bill for any past transaction using its unique TransactionID if required.
- Search through the available product category-wise

2.Database Design

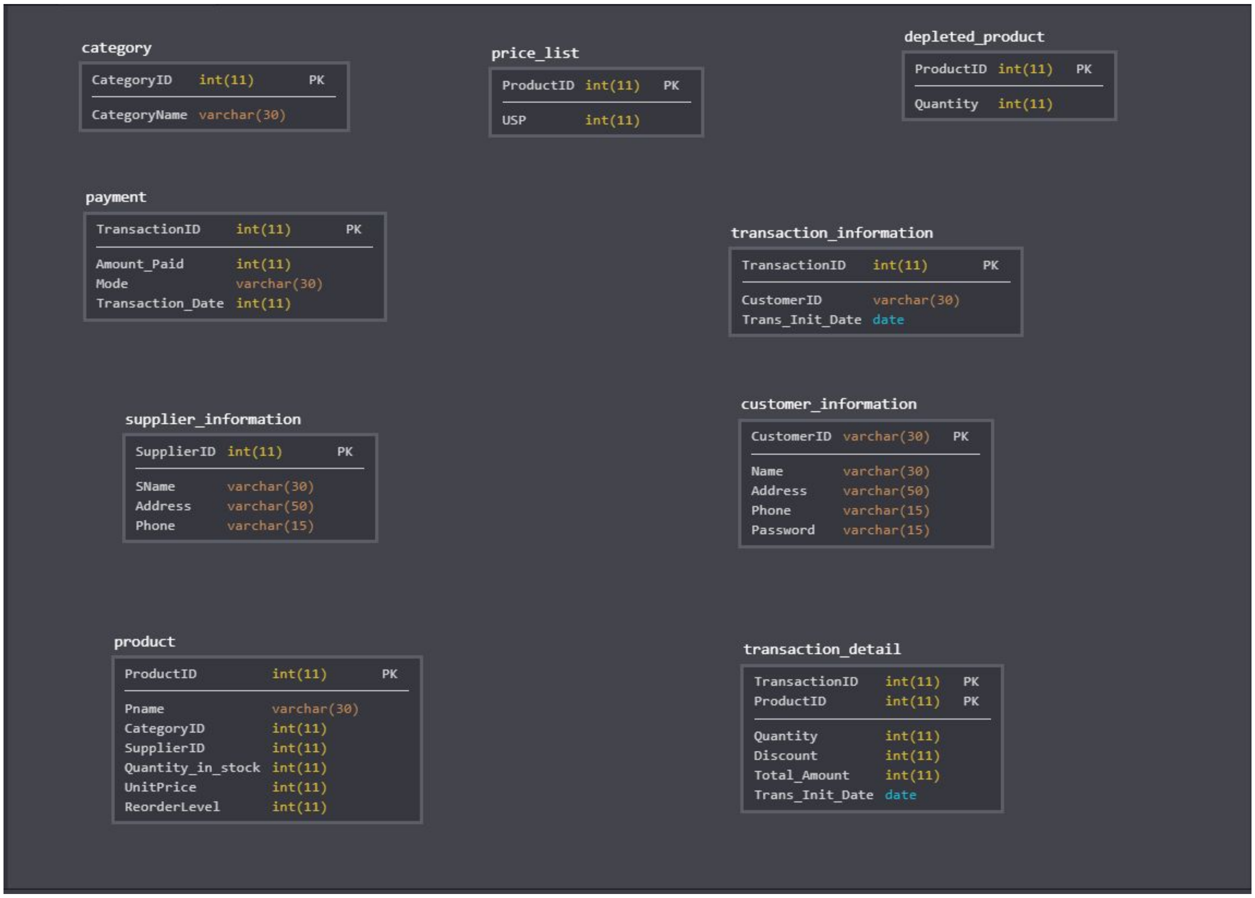
2.1 ER - Diagram



2.2 EER - Diagram



2.3 Tables



category

CategoryID	int(11)	PK
CategoryName	varchar(30)	

price_list

ProductID	int(11)	PK
USP	int(11)	

depleted_product

ProductID	int(11)	PK
Quantity	int(11)	

payment

TransactionID	int(11)	PK
Amount_Paid	int(11)	
Mode	varchar(30)	
Transaction_Date	int(11)	

transaction_information

TransactionID	int(11)	PK
CustomerID	varchar(30)	
Trans_Init_Date	date	

supplier_information

SupplierID	int(11)	PK
SName	varchar(30)	
Address	varchar(50)	
Phone	varchar(15)	

customer_information

CustomerID	varchar(30)	PK
Name	varchar(30)	
Address	varchar(50)	
Phone	varchar(15)	
Password	varchar(15)	

product

ProductID	int(11)	PK
Pname	varchar(30)	
CategoryID	int(11)	
SupplierID	int(11)	
Quantity_in_stock	int(11)	
UnitPrice	int(11)	
ReorderLevel	int(11)	

transaction_detail

TransactionID	int(11)	PK
ProductID	int(11)	PK
Quantity	int(11)	
Discount	int(11)	
Total_Amount	int(11)	
Trans_Init_Date	date	

2.4 Schema

category

CategoryID	CategoryName

supplier_information

supplierID	SName	Address	Phone

Customer_Information

CustomerID	Name	Address	Phone	Password

Transaction_detail

TransactionId	ProductID	Quantity	Discount	Total_Amount	Trans_Init_Date

Transaction_information

TransactionID	CustomerID	Trans_Init_Date

Depleted_product

ProductID	Quantity

Product

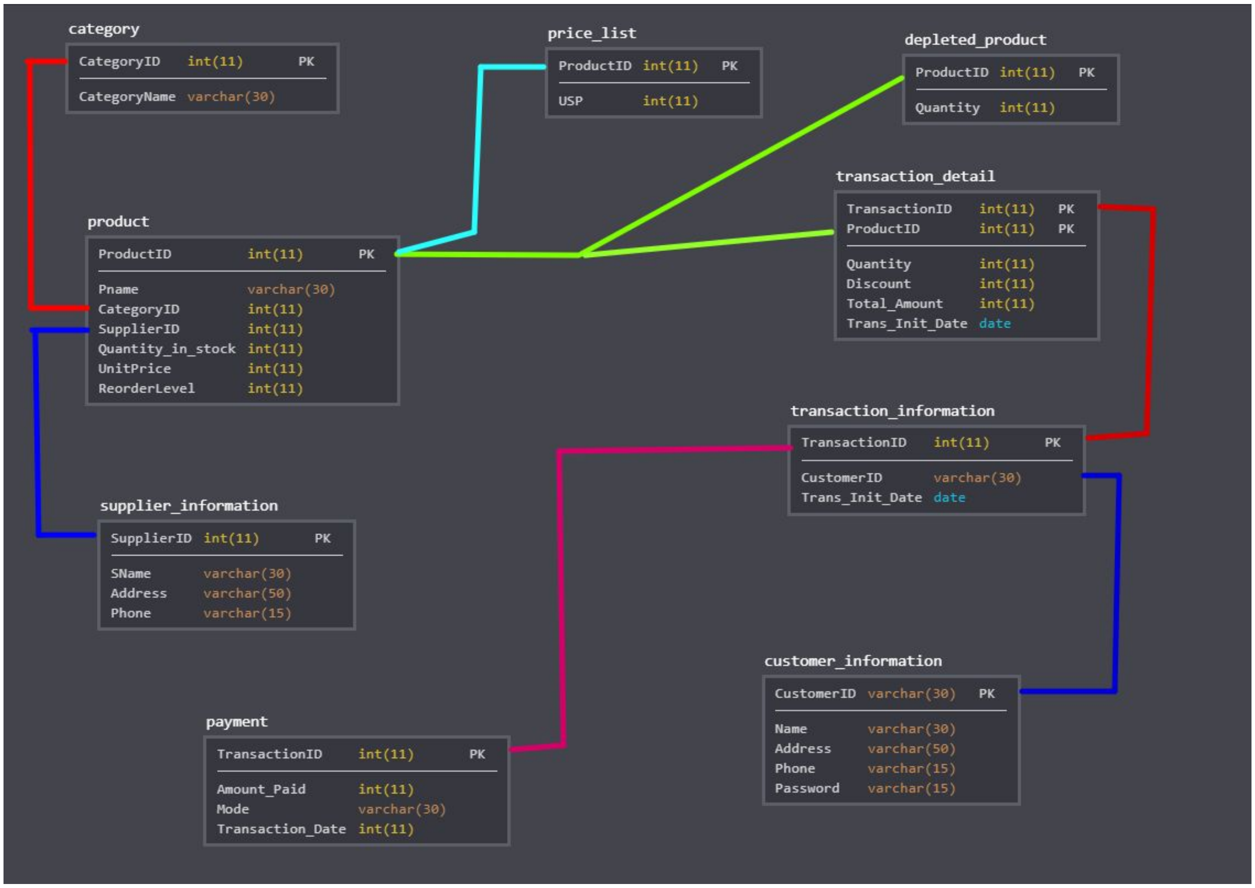
Product ID	Pname	Category ID	supplier ID	Quantity_in_stock	UnitPrice	Reorderlevel

Payment

TransactionID ————————-	Amount_Paid	Mode	Transaction_Date

Price_list

ProductID ————————-	Quantity



category

CategoryID	int(11)	PK
CategoryName	varchar(30)	

price_list

ProductID	int(11)	PK
USP	int(11)	

depleted_product

ProductID	int(11)	PK
Quantity	int(11)	

transaction_detail

TransactionID	int(11)	PK
ProductID	int(11)	PK
Quantity	int(11)	
Discount	int(11)	
Total_Amount	int(11)	
Trans_Init_Date	date	

product

ProductID	int(11)	PK
Pname	varchar(30)	
CategoryID	int(11)	
SupplierID	int(11)	
Quantity_in_stock	int(11)	
UnitPrice	int(11)	
ReorderLevel	int(11)	

transaction_information

TransactionID	int(11)	PK
CustomerID	varchar(30)	
Trans_Init_Date	date	

supplier_information

SupplierID	int(11)	PK
SName	varchar(30)	
Address	varchar(50)	
Phone	varchar(15)	

customer_information

CustomerID	varchar(30)	PK
Name	varchar(30)	
Address	varchar(50)	
Phone	varchar(15)	
Password	varchar(15)	

payment

TransactionID	int(11)	PK
Amount_Paid	int(11)	
Mode	varchar(30)	
Transaction_Date	int(11)	

2.5 Functional Dependencies

Every table has a functional dependency from the ID(primary key) to all the other attributes.

Attribute (A)	Attributes Determined by Attribute A (B)
ProductID	Pname, Quantity_in_stock, UnitPrice, ReorderLevel
SupplierID	SName, Address, Phone
TransactionID	Amount_Paid, Mode, Transaction_Date
CategoryID	CategoryName
ProductID	USP
TransactionID	Trans_Init_Date
ProductID	Quantity
CustomerID	Name, Address, Phone, Password
TransactionID, ProductID	Quantity, Discount, Total_Amount

2.6 Normalization

a) First Normal Form (1NF) :

4 criteria in order for the database to be in 1NF

1. Only have single (atomic) valued attributes
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. The order in which data is stored, does not matter.

Multiple values are restricted from being entered into the relation

The domain for each field is determined by the ‘Type’ while creating the relation.

Although column names repeat across relations, all column names in a single relation are

unique.
All 7 relations satisfy the four conditions.

b) Second Normal Form (2NF) :

- 1NF
- Should not have Partial Dependency.

c) Third Normal Form (3NF) :

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

Starting with the products table:
In the products table the productID is the primary key. If we put the USP of the product as well in the same table it will cause a transitive dependency therefore we created a separate table price_list.

Category and Product are two separate entities to avoid transitive dependency.

Supplier Information and customer information are independent of other tables and the ID in each table is the primary key.

Transaction detail initially consisted of the payment and transaction_information but was later made into different tables for the same reason. The transaction information maps the transaction ID with the customer ID. Payment has the amount and mode of payment. Transaction detail comprises the product, quantity and date of transaction.

The trigger sets products to enter the depleted product table once the quantity falls below the minimum required.

Since the database is in 3NF form it does not have any functional dependency problems.

2.7 DDL for Tables

1. Product:

```
CREATE TABLE `product` (
  `ProductID` int(11) NOT NULL,
  `Pname` varchar(30) NOT NULL,
  `CategoryID` int(11) NOT NULL,
  `SupplierID` int(11) NOT NULL,
  `Quantity_in_stock` int(11) NOT NULL,
  `UnitPrice` int(11) NOT NULL,
  `ReorderLevel` int(11) NOT NULL,
  PRIMARY KEY (`ProductID`),
  KEY `product_ibfk_2` (`CategoryID`),
  KEY `product_ibfk_3` (`SupplierID`),
  CONSTRAINT `product_ibfk_2` FOREIGN KEY (`CategoryID`) REFERENCES
     `category` (`CategoryID`),
  CONSTRAINT `product_ibfk_3` FOREIGN KEY (`SupplierID`) REFERENCES
     `supplier_information` (`SupplierID`)
)
```

2.Transaction Detail:

```
CREATE TABLE `transaction_detail` (
  `TransactionID` int(11) NOT NULL,
  `ProductID` int(11) NOT NULL,
  `Quantity` int(11) NOT NULL,
  `Discount` int(11) NOT NULL DEFAULT '0',
  `Total_Amount` int(11) NOT NULL,
  `Trans_Init_Date` date NOT NULL,
  PRIMARY KEY (`TransactionID`,`ProductID`),
  KEY `td_ibfk_2` (`ProductID`),
  CONSTRAINT `td_ibfk_2` FOREIGN KEY (`ProductID`) REFERENCES
     `product` (`ProductID`)
)
```

3.Transaction Information:

```
CREATE TABLE `transaction_information` (
  `TransactionID` int(11) NOT NULL AUTO_INCREMENT,
  `CustomerID` varchar(30) NOT NULL,
  `Trans_Init_Date` date NOT NULL,
  PRIMARY KEY (`TransactionID`),
  KEY `ti_ibfk_1` (`CustomerID`),
  CONSTRAINT `ti_ibfk_1` FOREIGN KEY (`CustomerID`) REFERENCES
     `customer_information` (`CustomerID`)
)
```

4.Price_List:

```
CREATE TABLE `price_list` (
  `ProductID` int(11) NOT NULL,
  `USP` int(11) NOT NULL,
  PRIMARY KEY (`ProductID`)
)
```

5.Customer Information:

```
CREATE TABLE `customer_information` (
  `CustomerID` varchar(30) NOT NULL,
  `Name` varchar(30) NOT NULL,
  `Address` varchar(50) NOT NULL,
  `Phone` varchar(15) NOT NULL,
  `Password` varchar(15) NOT NULL,
  PRIMARY KEY (`CustomerID`)
)
```

6.Depleted Product:

```
CREATE TABLE `depleted_product` (
  `ProductID` int(11) NOT NULL,
  `Quantity` int(11) NOT NULL,
  PRIMARY KEY (`ProductID`)
)
```

7.Category:

```
CREATE TABLE `category` (
  `CategoryID` int(11) NOT NULL,
  `CategoryName` varchar(30) NOT NULL,
  PRIMARY KEY (`CategoryID`)
)
```

8.Payment

```
CREATE TABLE `payment` (
  `TransactionID` int(11) NOT NULL,
  `Amount_Paid` int(11) NOT NULL,
  `Mode` varchar(30) NOT NULL,
  `Transaction_Date` int(11) NOT NULL,
  PRIMARY KEY (`TransactionID`)
)
```

9.Supplier Information

```
CREATE TABLE `supplier_information` (
  `SupplierID` int(11) NOT NULL,
  `SName` varchar(30) NOT NULL,
  `Address` varchar(50) NOT NULL,
  `Phone` varchar(15) NOT NULL,
  PRIMARY KEY (`SupplierID`)
)
```

2.8 Triggers

1. Quantity

   ‘max_min_quantity’: This trigger checks if the quantity ordered of a particular product is less than maximum and more than the minimum allowed. Also decreases the quantity in stock of the product ordered. BEFORE INSERT ON ‘TRANSACTION_DETAIL’.

```
DELIMITER $$
CREATE TRIGGER `max_min_quantity_update` BEFORE UPDATE ON
`transaction_detail` FOR EACH ROW BEGIN
declare var1 int;
declare var2 int;
select ReorderLevel into var1 from product where ProductID = NEW.ProductID;
select Quantity_in_stock into var2 from product where ProductID = NEW.ProductID;
if new.Quantity<var1 THEN
signal sqlstate '45000' set message_text = 'Less than min quantity';
end if;
if new.Quantity>var2 THEN
signal sqlstate '45000' set message_text = 'More than max quantity';
end if;
update product set Quantity_in_stock = Quantity_in_stock - NEW.Quantity where
ProductID=NEW.ProductID;
END
$$
DELIMITER ;
```

2. ‘Depleted_check_update’ :

   It checks if the ‘quantity_in_stock’ is less than ‘ReorderLevel . If yes, it inserts that particular product in ‘depleted_product’ table. Also, removes a product from the ‘depleted_product’ table when the quantity of product is increased. BEFORE UPDATE OR INSERT ON ‘PRODUCT’.

```
CREATE TRIGGER `depleted_check_update` BEFORE UPDATE ON `product` FOR
EACH ROW BEGIN
Declare finished integer default 0;
Declare cust varchar(30);
declare flag integer default 0;
Declare c1 cursor for select ProductID from depleted_product;
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET finished = 1;

if NEW.Quantity_in_stock < NEW.ReorderLevel THEN
insert into depleted_product(ProductID,Quantity)
values(NEW.ProductID,NEW.Quantity_in_stock);


else
open c1;
get_cust: LOOP
Fetch c1 into cust;
if finished=1 then
leave get_cust;
end if;
if cust=NEW.ProductID then
set finished=1;
set flag=1;
end if;
end loop get_cust;
close c1;
if flag=1 then
Delete from depleted_product where ProductID=NEW.ProductID;
END if;
end if;
END
$$
```

3. ‘Decreased_quantity’ :

   In case a transaction fails, the quantity of product has to be restored to its original level before the transaction. This trigger is used to do that.

   BEFORE DELETE ON ‘TRANSACTION_INFORMATION’.

```
DELIMITER ;
DELIMITER $$
CREATE TRIGGER `decrease_quantity` BEFORE DELETE ON
`transaction_information` FOR EACH ROW BEGIN
```

```
Declare finished integer default 0;
Declare cust integer;
Declare quant integer default 0;
Declare c1 cursor for select ProductID from transaction_detail where
TransactionID=OLD.TransactionID;
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET finished = 1;

CREATE TEMPORARY TABLE IF NOT EXISTS my_temp_table
SELECT ProductID, Quantity from transaction_detail where
TransactionID=OLD.TransactionID;
open c1;
get_cust: LOOP
Fetch c1 into cust;
if finished=1 then
leave get_cust;
end if;
Select Quantity into quant from my_temp_table where ProductID=cust;
Update product set quantity_in_stock=quantity_in_stock+quant where ProductID=cust;
end loop;
close c1;
Delete from transaction_detail where transactionID=OLD.TransactionID;
END
$$
DELIMITER ;
```

2.9 Indexes

- Indexes for table `category`

```
ALTER TABLE `category`
  ADD PRIMARY KEY (`CategoryID`);
```

- Indexes for table `customer_information`

```
ALTER TABLE `customer_information`
  ADD PRIMARY KEY (`CustomerID`);
```

- Indexes for table `depleted_product`

```
ALTER TABLE `depleted_product`
  ADD PRIMARY KEY (`ProductID`);
```

- Indexes for table `payment`

```
ALTER TABLE `payment`
  ADD PRIMARY KEY (`TransactionID`);
```

- Indexes for table `price_list`

```
ALTER TABLE `price_list`
  ADD PRIMARY KEY (`ProductID`);
```

- Indexes for table `product`

```
ALTER TABLE `product`
  ADD PRIMARY KEY (`ProductID`),
  ADD KEY `product_ibfk_2` (`CategoryID`),
  ADD KEY `product_ibfk_3` (`SupplierID`);
```

- Indexes for table `supplier_information`

```
ALTER TABLE `supplier_information`
  ADD PRIMARY KEY (`SupplierID`);
```

- Indexes for table `transaction_detail`

```
ALTER TABLE `transaction_detail`
  ADD PRIMARY KEY (`TransactionID`,`ProductID`),
  ADD KEY `td_ibfk_2` (`ProductID`);
```

- Indexes for table `transaction_information`

```
ALTER TABLE `transaction_information`
  ADD PRIMARY KEY (`TransactionID`),
  ADD KEY `ti_ibfk_1` (`CustomerID`);
```

- AUTO_INCREMENT for table `transaction_information`

```
ALTER TABLE `transaction_information`
  MODIFY `TransactionID` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=29;
```

- Constraints for table `product`

```
ALTER TABLE `product`
  ADD CONSTRAINT `product_ibfk_2` FOREIGN KEY (`CategoryID`)
REFERENCES `category` (`CategoryID`),
  ADD CONSTRAINT `product_ibfk_3` FOREIGN KEY (`SupplierID`) REFERENCES
`supplier_information` (`SupplierID`);
```

- Constraints for table `transaction_detail`

```
ALTER TABLE `transaction_detail`
  ADD CONSTRAINT `td_ibfk_2` FOREIGN KEY (`ProductID`) REFERENCES
`product` (`ProductID`);
```

- Constraints for table `transaction_information`

```
ALTER TABLE `transaction_information`
  ADD CONSTRAINT `ti_ibfk_1` FOREIGN KEY (`CustomerID`) REFERENCES
`customer_information` (`CustomerID`);
```

3. Implementation

3.1 Tools and technologies used

a) Backend

We have used **PHP** for our backend.

PHP stands for Hypertext Preprocessor and is a server-side programming language.

There are many reasons to use **PHP** for server side programming, firstly it is a free language with no licensing fees so the cost of using it is minimal.

A good benefit of using **PHP** is that it can interact with many different database languages including **MySQL**.

PHP also has very good online documentation with a good framework of functions in place. This makes the language relatively easy to learn and very well supported online. There are countless forums and tutorials on various PHP methods and problems so it is usually very easy to find help if you need it.

MySQL

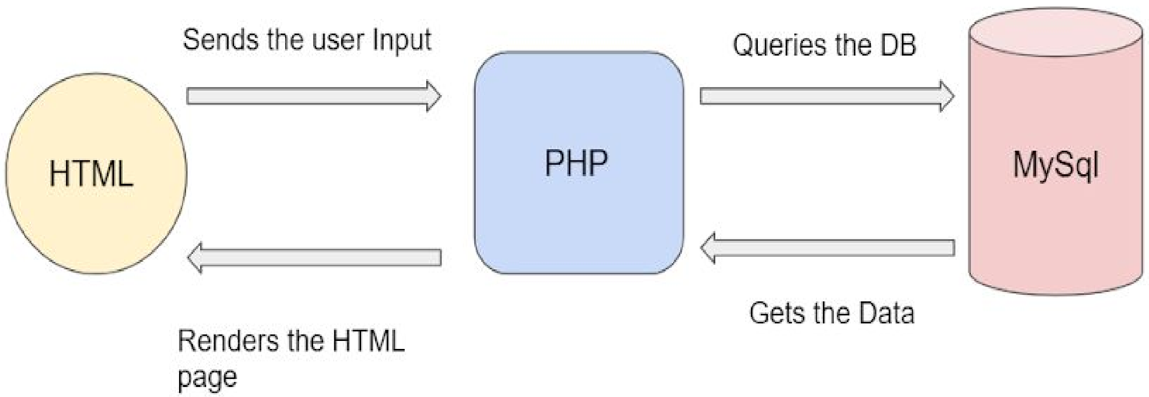
We have used mysql as the database for this project. We have used mysql in the course labs as well and find it easy to use, therefore we have used it in this project

b) Frontend

For the frontend we have used HTML, CSS and javascript. These are easy to use and work well with PHP.

3.2 Application Architecture

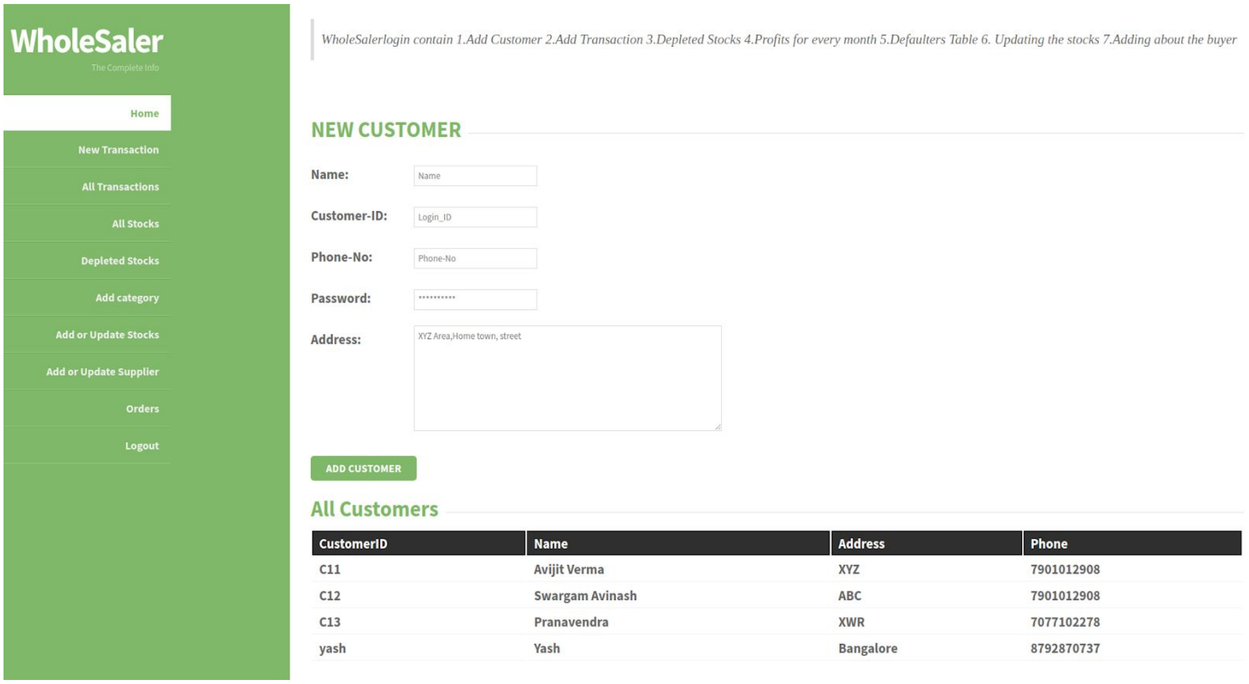
PHP is the server side application. This renders the specific HTML pages. The user input is received by the frontend in terms of data in a form. This data is then sent to the backend server to process it. It could be a login request or a transaction process. The details are then processed by the backend server and sent to the database to store or retrieved from the database and displayed to the user. The complete flow is from the presentation layer to the business layer and then the database.



3.3 Snapshots

Wholesaler:

The Wholesaler has the total overview of everything. Stocks, Transactions , Orders etc are all handled by the wholesaler. Wholesalers can onboard new customers as well. This is the homepage for the wholesaler.



WholeSalerlogin contain 1.Add Customer 2.Add Transaction 3.Depleted Stocks 4.Profits for every month 5.Defaulters Table 6. Updating the stocks 7.Adding about the buyer

NEW CUSTOMER

Name: Name

Customer-ID: Login_ID

Phone-No: Phone-No

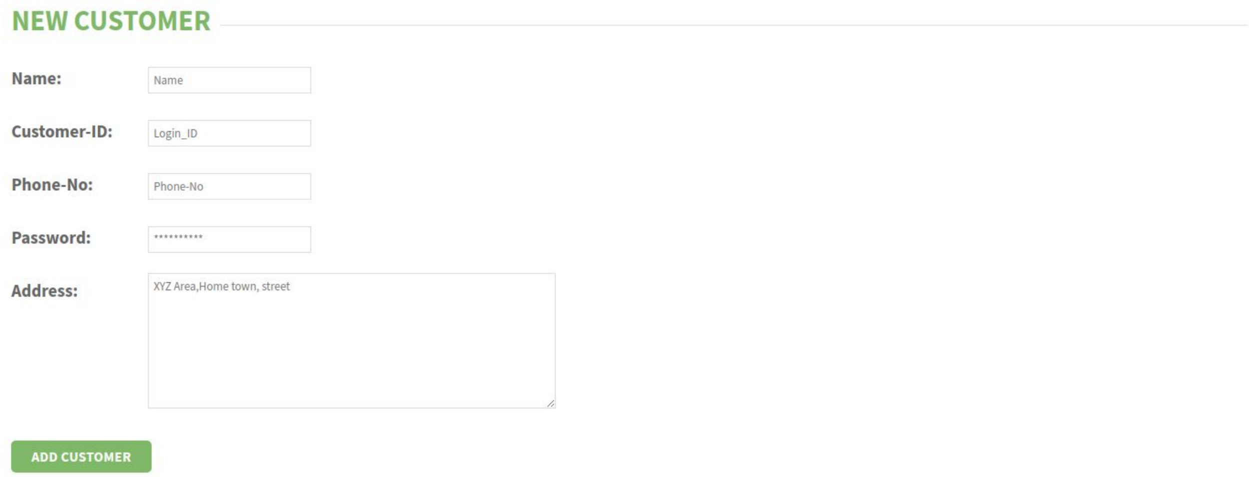
Password: **********

Address: XYZ Area,Home town, street

All Customers

CustomerID	Name	Address	Phone
C11	Avijit Verma	XYZ	7901012908
C12	Swargam Avinash	ABC	7901012908
C13	Pranavendra	XWR	7077102278
yash	Yash	Bangalore	8792870737

Registering a Customer:



NEW CUSTOMER

Name: Name

Customer-ID: Login_ID

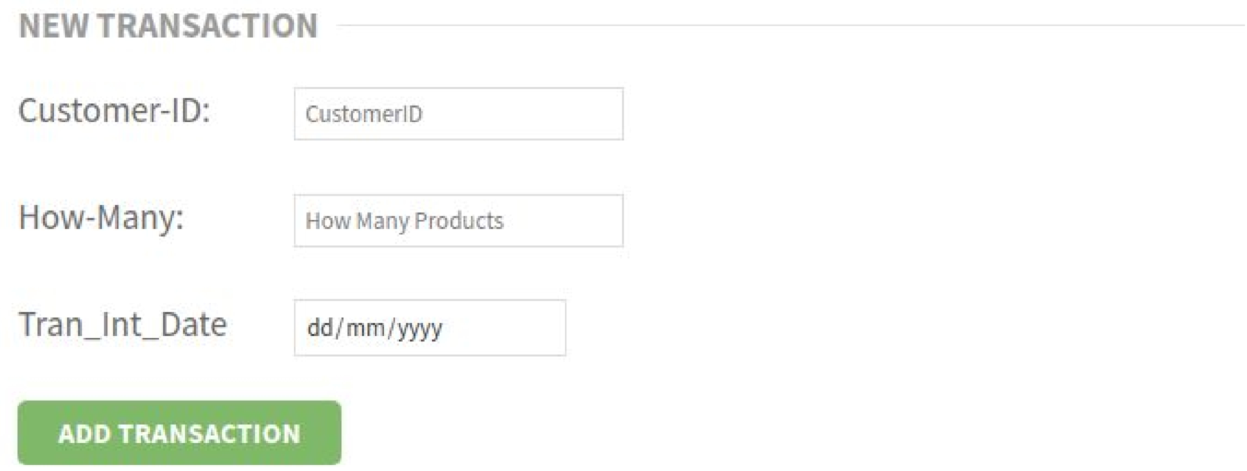
Phone-No: Phone-No

Password: **********

Address: XYZ Area,Home town, street

This form is used to get the details of customers to register them to our management system. The customer ID has to be unique for the given user.

Creating a new Transaction :



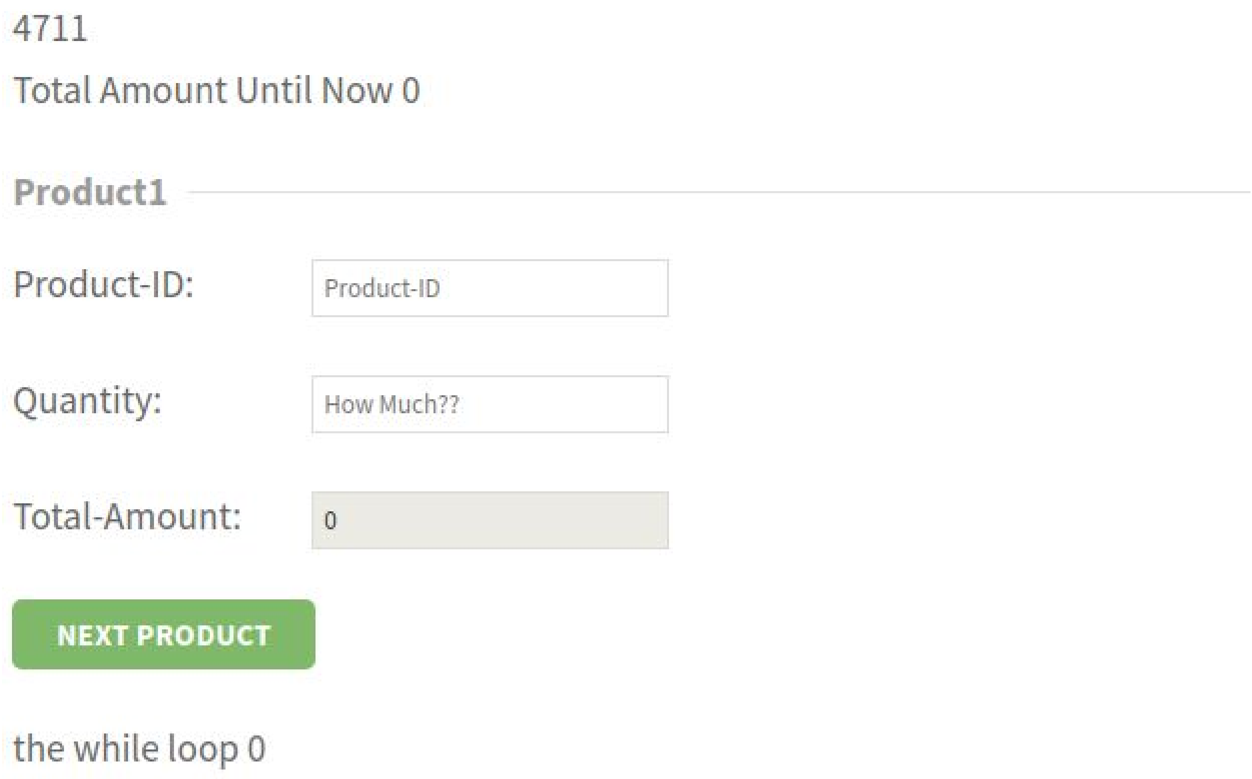
NEW TRANSACTION

Customer-ID: CustomerID

How-Many: How Many Products

Tran_Int_Date dd/mm/yyyy

This form is used to create a new transaction. The user has to enter the CUstomer ID, Number of Products and the Transaction Date.



4711

Total Amount Until Now 0

Product1

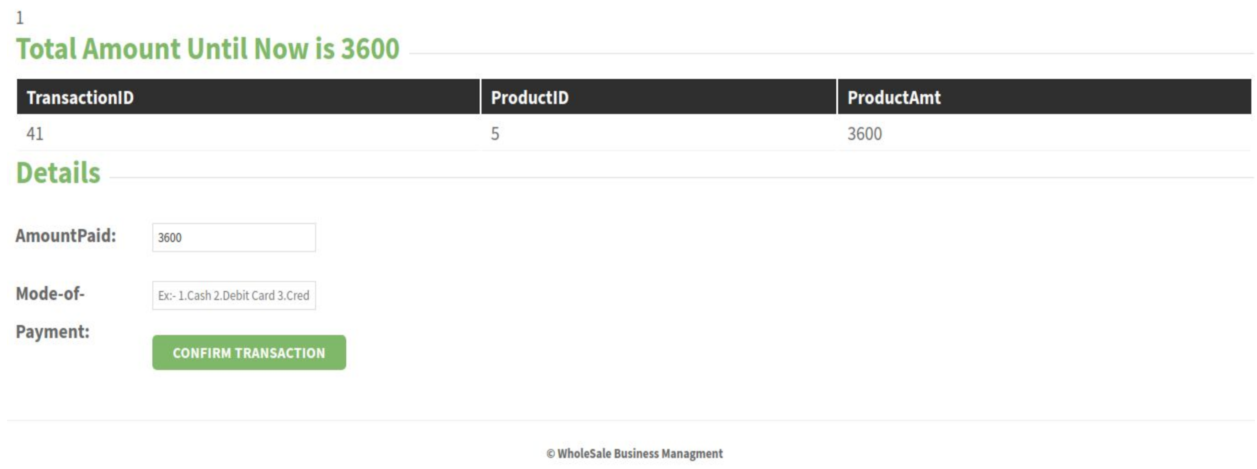
Product-ID: Product-ID

Quantity: How Much??

Total-Amount: 0

the while loop 0

Once the initial details ae filled we are brought to add the product ID and the quantity



1

Total Amount Until Now is 3600

TransactionID	ProductID	ProductAmt
41	5	3600

Details

AmountPaid: 3600

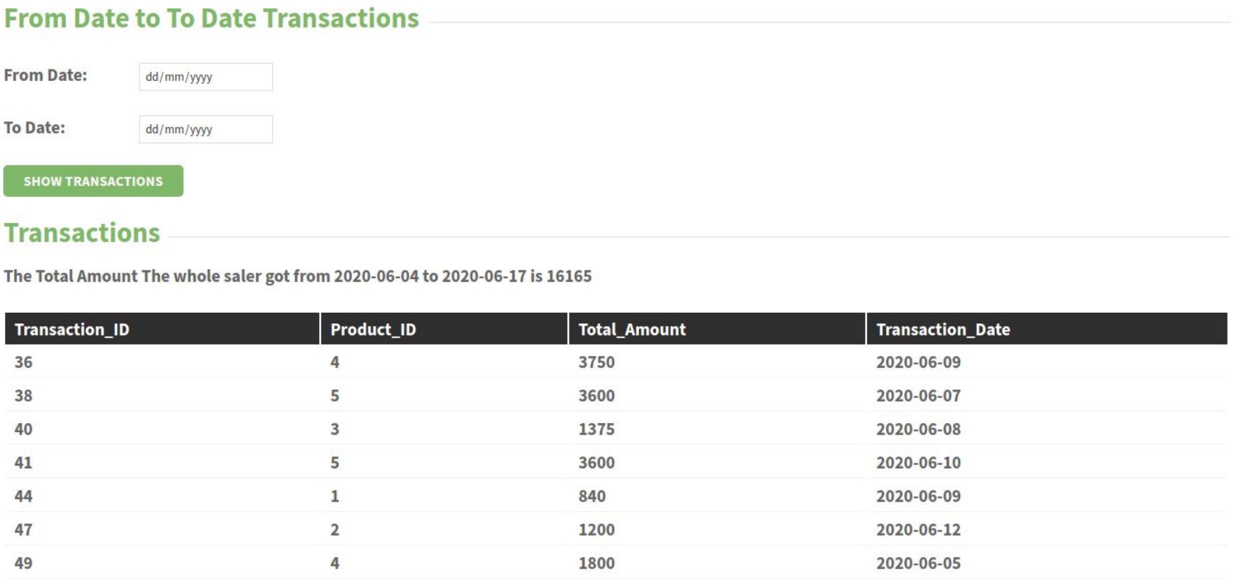
Mode-of-Payment: Ex:- 1.Cash 2.Debit Card 3.Cred

© WholeSale Business Managment

This is the payment page where the wholesaler enters the mode of payment along with the amount to confirm the transaction, else it is a fail and get deleted.

At any point in the transaction if there is an error the whole transaction fails and rolls back.

List of Transactions :



From Date to To Date Transactions

From Date: dd/mm/yyyy

To Date: dd/mm/yyyy

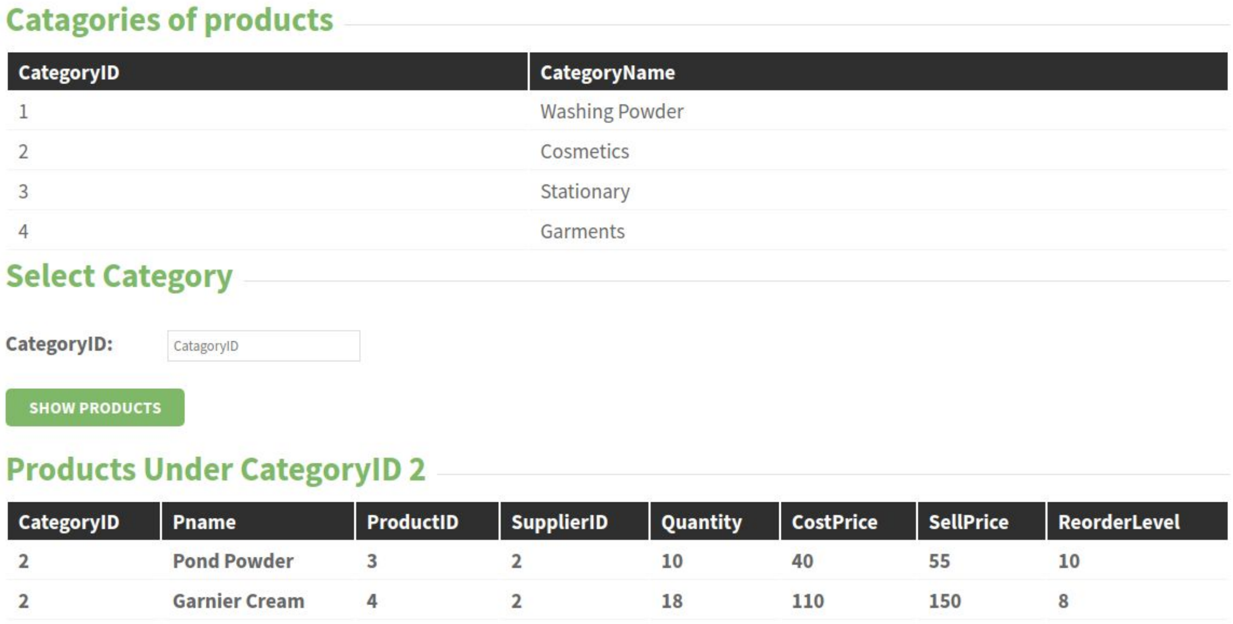
Transactions

The Total Amount The whole saler got from 2020-06-04 to 2020-06-17 is 16165

Transaction_ID	Product_ID	Total_Amount	Transaction_Date
36	4	3750	2020-06-09
38	5	3600	2020-06-07
40	3	1375	2020-06-08
41	5	3600	2020-06-10
44	1	840	2020-06-09
47	2	1200	2020-06-12
49	4	1800	2020-06-05

The transactions can be listed based on a start and end date.

Stocks :



Catagories of products

CategoryID	CategoryName
1	Washing Powder
2	Cosmetics
3	Stationary
4	Garments

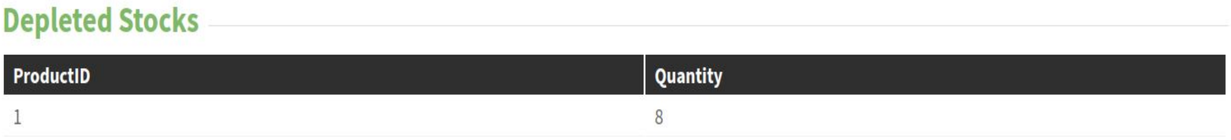
Select Category

CategoryID: CatagoryID

Products Under CategoryID 2

CategoryID	Pname	ProductID	SupplierID	Quantity	CostPrice	SellPrice	ReorderLevel
2	Pond Powder	3	2	10	40	55	10
2	Garnier Cream	4	2	18	110	150	8

This page gives the details about the stock in terms of Quantity, Price etc.
On selecting the category they are displayed.
Depleted Stocks

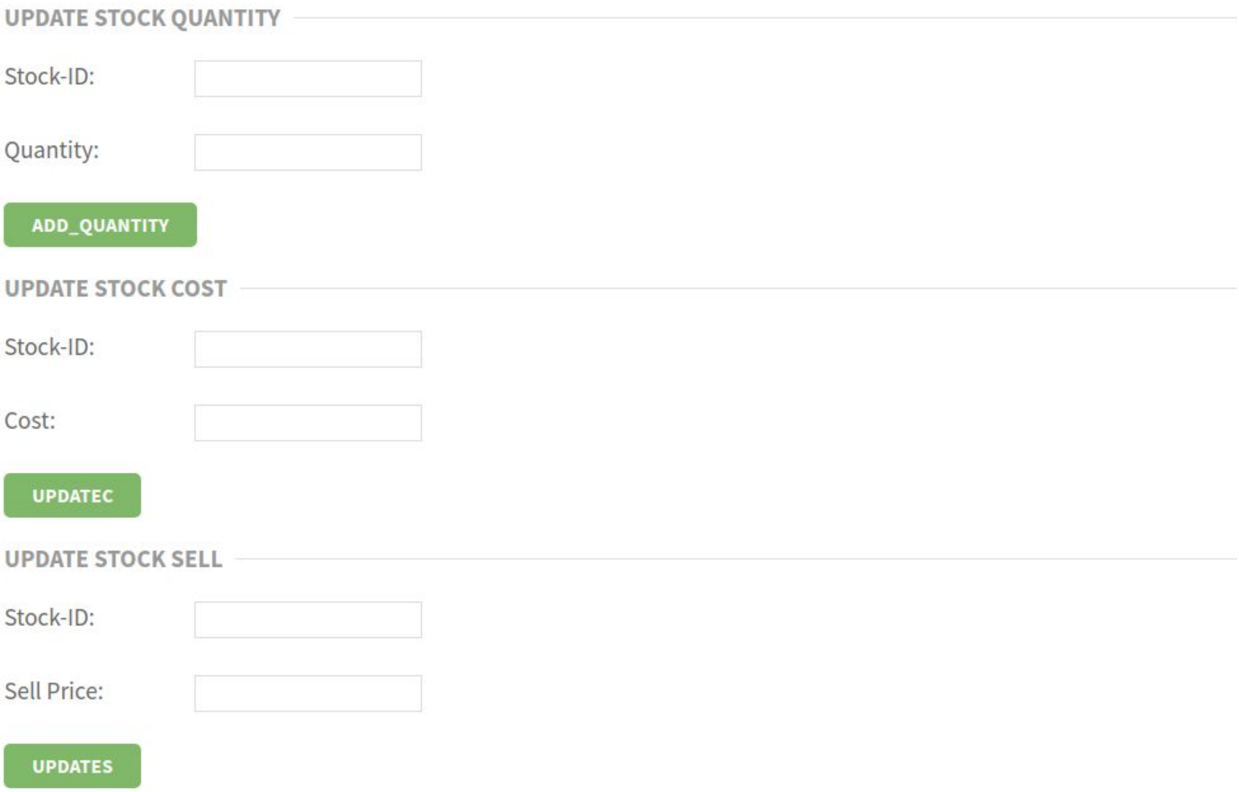


Depleted Stocks

ProductID	Quantity
1	8

If any product runs low on stock, it is displayed on this page.

Updating Values :

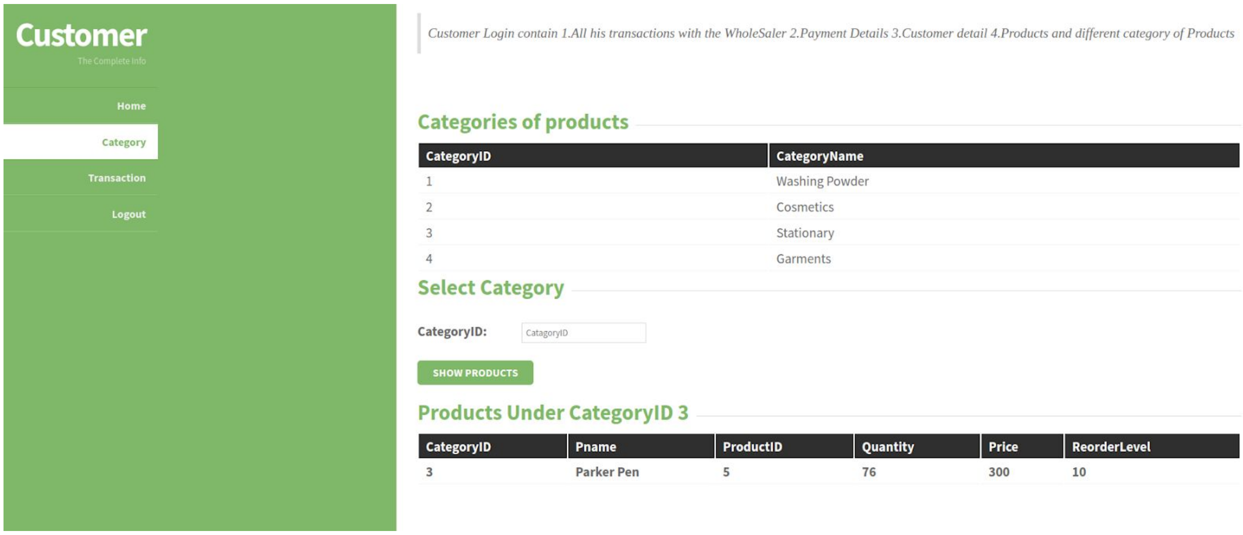


If any quantity needs to be updated, the quantity cost and selling price can be updated.

Customer:

Customer has its own login and can its on views and landing pages.

The customer can view the products as well as check the transactions made between a particular time frame.



Customer Login contain 1.All his transactions with the WholeSaler 2.Payment Details 3.Customer detail 4.Products and different category of Products

Categories of products

CategoryID	CategoryName
1	Washing Powder
2	Cosmetics
3	Stationary
4	Garments

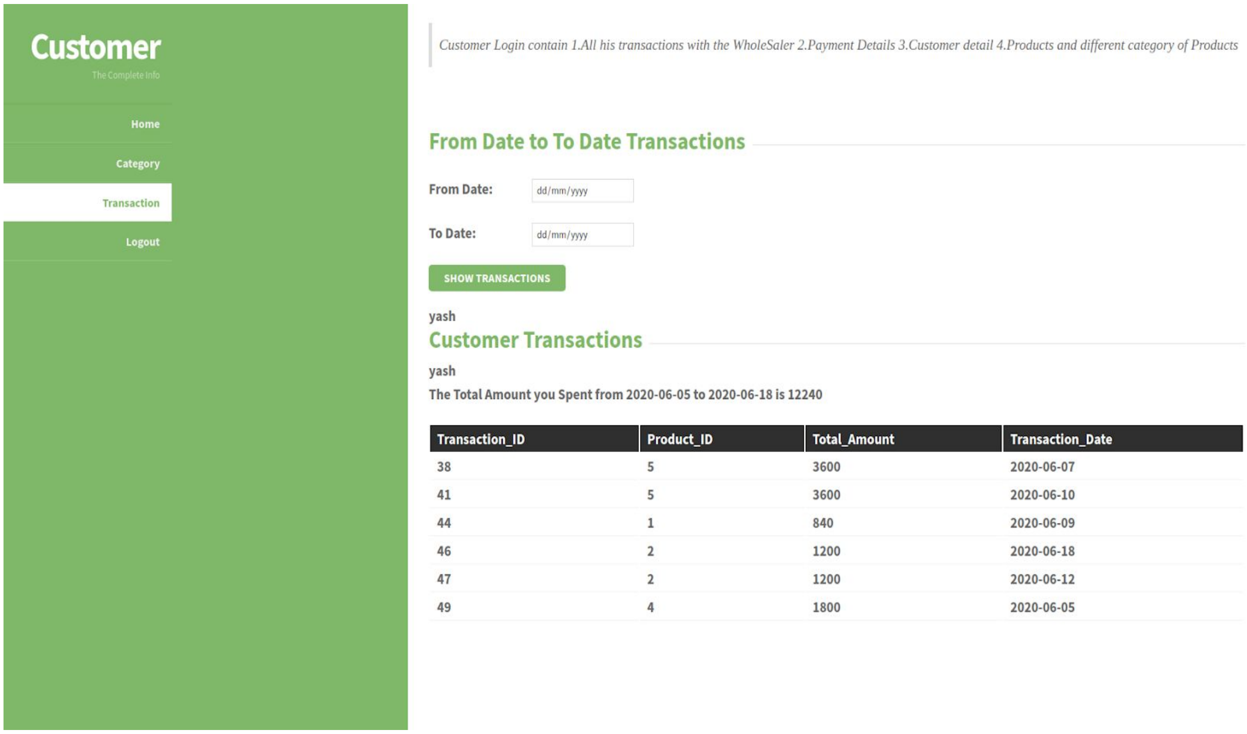
Select Category

CategoryID: CatagoryID

SHOW PRODUCTS

Products Under CategoryID 3

CategoryID	Pname	ProductID	Quantity	Price	ReorderLevel
3	Parker Pen	5	76	300	10



Customer Login contain 1.All his transactions with the WholeSaler 2.Payment Details 3.Customer detail 4.Products and different category of Products

From Date to To Date Transactions

From Date: dd/mm/yyyy

To Date: dd/mm/yyyy

SHOW TRANSACTIONS

yash

Customer Transactions

yash

The Total Amount you Spent from 2020-06-05 to 2020-06-18 is 12240

Transaction_ID	Product_ID	Total_Amount	Transaction_Date
38	5	3600	2020-06-07
41	5	3600	2020-06-10
44	1	840	2020-06-09
46	2	1200	2020-06-18
47	2	1200	2020-06-12
49	4	1800	2020-06-05

3.4 Contribution

- Jaidev: worked on frontend (HTML and the styling using css)
- Karthik Reddy: worked on Database, transactions and triggers
- Yash: Worked with the web application (PHP, HTML) and helped with the database

3.5 References

- Database Systems, 6 th Edition, by Ramez Elmasri and Shamkant B. Navathe.
- 'Wholesale Management System' by Lee Yee Fhong, National Technical University College of Malaysia, 2004.
- 'DESIGNING A LOGICAL DATA MODEL FOR A SALES AND INVENTORY MANAGEMENT SYSTEM' by Hari Krishna Mahat.
- Public domain information of 'White Clarke North America Wholesale Management Table'.