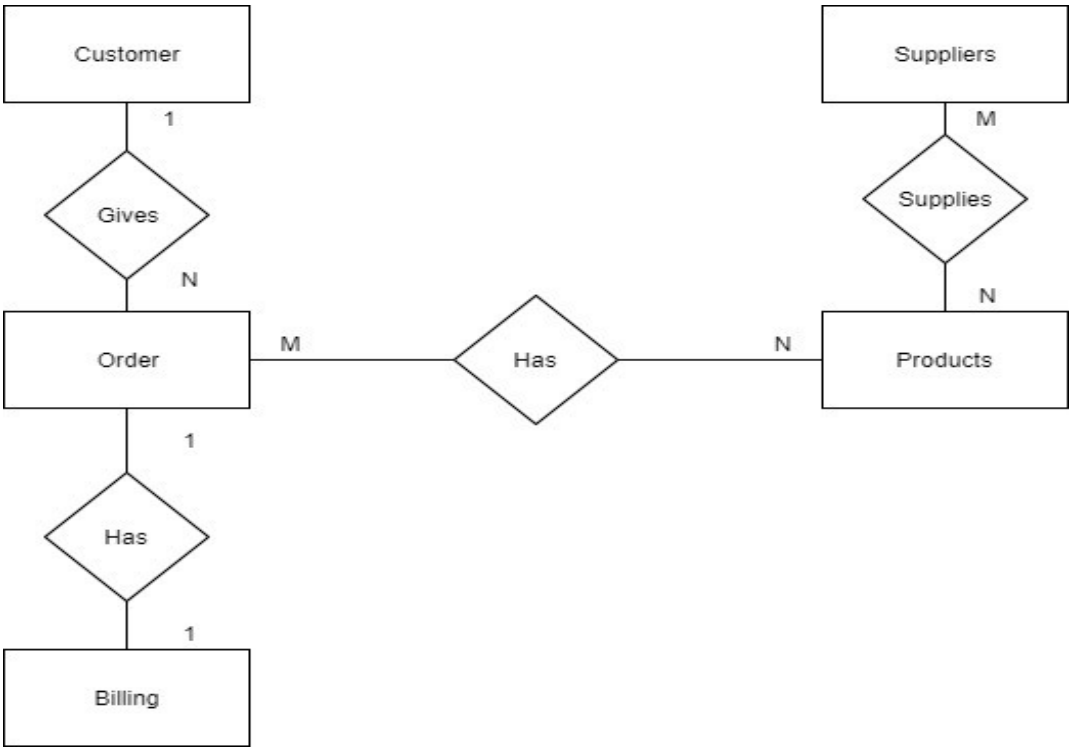
E-Commerce

Project Description: E-commerce websites provide an easy way to sell products to a large customer base. While using the website, users expect to find what they are looking for quickly and easily. A well-designed database is needed to meet the customer expectations. This database will store the product details and their availability for sale, it will have the customer details who will order the products, it will have order details of the products being sold.

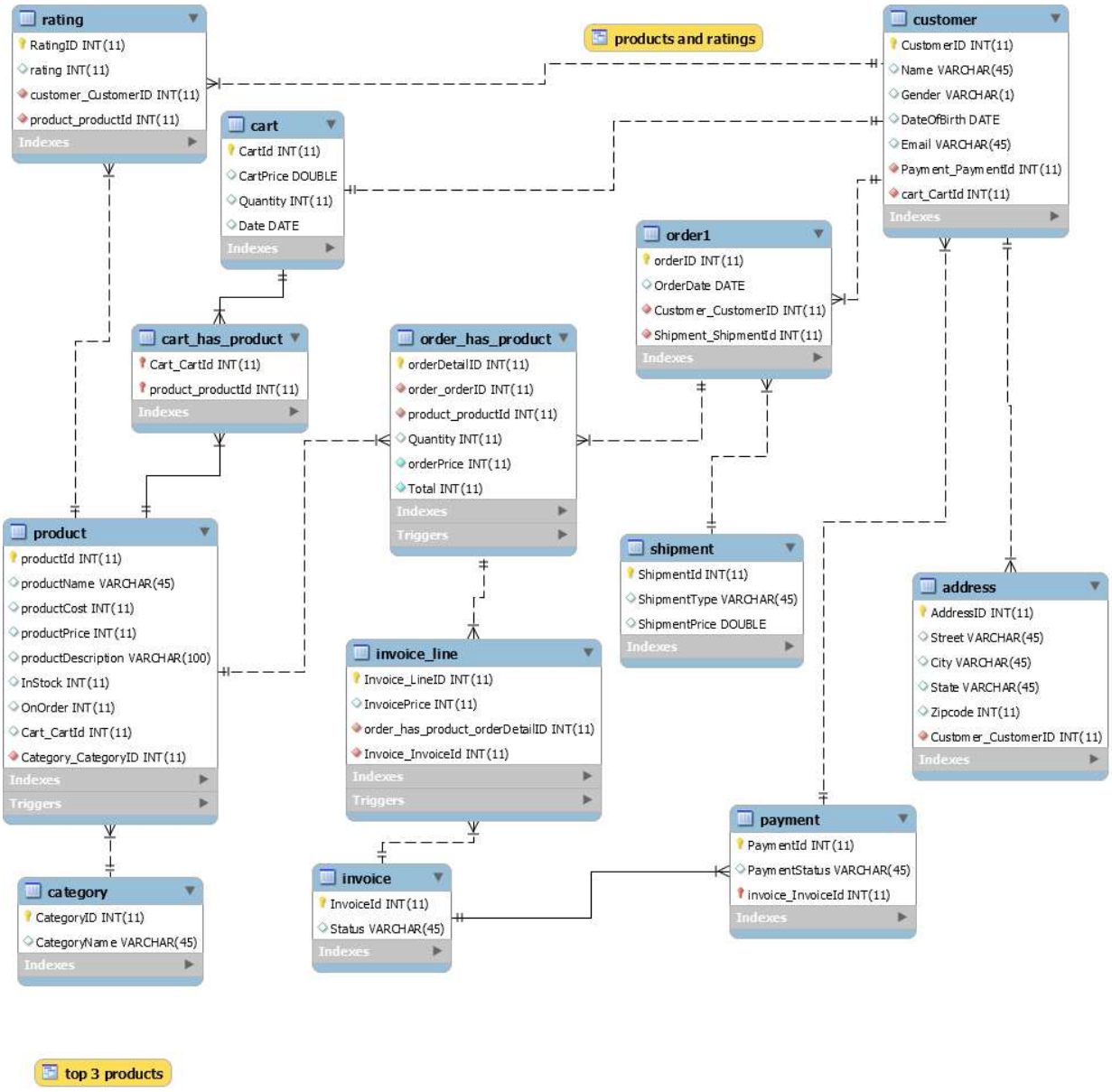
I have created the database keeping in mind of all the business requirements. The main tables of this design are Customer, Product, Order_has_Product. I have created various functionalities using functions, triggers, procedures, views and also have tried using some of the aggregate functins like AVG() and SUM().

ER Diagram:

First Draft



Final Draft


rating
RatingID INT(11)
rating INT(11)
customer_CustomerID INT(11)
product_productId INT(11)
Indexes
products and ratings
customer
CustomerID INT(11)
Name VARCHAR(45)
Gender VARCHAR(1)
DateOfBirth DATE
Email VARCHAR(45)
Payment_PaymentId INT(11)
cart_CartId INT(11)
Indexes
cart
CartId INT(11)
CartPrice DOUBLE
Quantity INT(11)
Date DATE
Indexes
order1
orderID INT(11)
OrderDate DATE
Customer_CustomerID INT(11)
Shipment_ShipmentId INT(11)
Indexes
cart_has_product
Cart_CartId INT(11)
product_productId INT(11)
Indexes
order_has_product
orderDetailID INT(11)
order_orderID INT(11)
product_productId INT(11)
Quantity INT(11)
orderPrice INT(11)
Total INT(11)
Indexes
Triggers
product
productId INT(11)
productName VARCHAR(45)
productCost INT(11)
productPrice INT(11)
productDescription VARCHAR(100)
InStock INT(11)
OnOrder INT(11)
Cart_CartId INT(11)
Category_CategoryID INT(11)
Indexes
Triggers
shipment
ShipmentId INT(11)
ShipmentType VARCHAR(45)
ShipmentPrice DOUBLE
Indexes
address
AddressID INT(11)
Street VARCHAR(45)
City VARCHAR(45)
State VARCHAR(45)
Zipcode INT(11)
Customer_CustomerID INT(11)
Indexes
invoice_line
Invoice_LineID INT(11)
InvoicePrice INT(11)
order_has_product_orderDetailID INT(11)
Invoice_InvoiceId INT(11)
Indexes
Triggers
payment
PaymentId INT(11)
PaymentStatus VARCHAR(45)
invoice_InvoiceId INT(11)
Indexes
category
CategoryID INT(11)
CategoryName VARCHAR(45)
Indexes
invoice
InvoiceId INT(11)
Status VARCHAR(45)
Indexes
top 3 products

Triggers:

1. **Before Insert on Order_has_product**

```
USE `ecommerce`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `ecommerce`.`order_has_product_BEFORE_INSERT`
BEFORE INSERT ON `ecommerce`.`order_has_product`
FOR EACH ROW
BEGIN
    if new.quantity < (select inStock from product p where new.product_productId = p.productId)
    then
    set New.orderprice = (select productPrice from product
                          where product.productId = new.product_productID );
    set NEW.total = New.orderprice * new.quantity;
    end if;
END$$
```

2. **After Insert on Order_has_product**

```
USE `ecommerce`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `ecommerce`.`order_has_product_AFTER_INSERT`
AFTER INSERT ON `ecommerce`.`order_has_product`
FOR EACH ROW
BEGIN
        if new.quantity < (select inStock from product p where new.product_productId = p.productId)
        then
        update product
        set product.inStock = product.instock - new.quantity,
            product.onOrder = product.onOrder + new.quantity
        where new.product_productId = product.productId;
        end if;

END$$
```

3. **Before Insert on Invoice_Line**

```
USE `ecommerce`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `ecommerce`.`invoice_line_BEFORE_INSERT`
BEFORE INSERT ON `ecommerce`.`invoice_line`
FOR EACH ROW
BEGIN
    Declare op int;
    declare sp int;

    set sp = (select shipmentPrice from shipment s, order1 od, order_has_product o
              where o.orderdetailid = new.order_has_product_orderDetailID
                                    and o.order_orderID = od.orderID
                                    and od.Shipment_ShipmentId = s.shipmentID);
    set op = (select orderPrice from order_has_product o
              where o.orderDetailId = new.order_has_product_orderDetailID);
    set new.invoicePrice = op + sp;


END$$

DELIMITER ;
```

4. Before Insert on Product

```
USE `ecommerce`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `ecommerce`.`product_BEFORE_INSERT`
BEFORE INSERT ON `ecommerce`.`product`
FOR EACH ROW
BEGIN
    SET NEW.productprice = NEW.productcost * 2;
END$$
```

5. Before Update on Product

```
USE `ecommerce`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `ecommerce`.`product_BEFORE_UPDATE`
BEFORE UPDATE ON `ecommerce`.`product`
FOR EACH ROW
BEGIN
        IF NEW.productcost <> OLD.productcost
    THEN
      SET NEW.productprice = NEW.productcost * 2;
  END IF ;
END$$
```

Procedures:

1. To change Invoice Status

```
DELIMITER $$
USE `ecommerce`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `changeInvoiceStatus`(IN param1 int)
BEGIN

        DECLARE xname VARCHAR(20);

        set xname = (select paymentStatus
                    from payment
                    where invoice_Invoiceid=param1);
        if
            xname = 'Paid'
        then
            if exists (select * from invoice where invoiceId = param1)
            then
                update invoice
                set status = xname where invoiceId = param1;
            end if ;
        End If ;
END$$

DELIMITER ;
```

2. **To calculate total orders for a Customer**

```
DELIMITER $$
USE `ecommerce`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `calctotalOrders`(IN param1 INT)
BEGIN
SELECT COUNT(*) as `Total Order`
from customer c inner join order1 on c.CustomerID = order1.Customer_CustomerID
where c.CustomerID = param1
group by c.CustomerID;
END$$

DELIMITER ;
```

Functions:

1. **To calculate total payment for a Customer**

```
DELIMITER $$
USE `ecommerce`$$
CREATE DEFINER=`root`@`localhost` FUNCTION `calcTotal`(Id int) RETURNS decimal(9,2)
BEGIN
  DECLARE profit DECIMAL(9,2);
  SET profit = (select Sum(i.invoicePrice)
                from customer c inner join order1 o on o.Customer_CustomerID=c.CustomerID
                inner join order_has_product op on op.order_orderID=o.orderID
                inner join invoice_line i on i.order_has_product_orderDetailID = op.orderDetailID
                where c.CustomerID = id);
  RETURN profit;
END$$

DELIMITER ;
```

2. **To calculate average rating for a product**

```
DELIMITER $$
USE `ecommerce`$$
CREATE DEFINER=`root`@`localhost` FUNCTION `calcAvgRatingForProduct`(Id int) RETURNS decimal(9,2)
BEGIN
  DECLARE avgRating DECIMAL(9,2);
  SET avgRating = (select Avg(rating.rating)
                   from rating
                   where product_productId = id);
  RETURN avgRating;
END$$

DELIMITER ;
```

Views:

1. To display Product and their Ratings

```
13  create view `Products and Ratings` as
14  select distinct productId, productName, productDescription,(select avg(rating)
15                                                               from rating
16                                                               where productId = rating.product_productId)
17                                                               as Rating
18  from product, rating
19  where productId = rating.product_productId;
20
```

Output:

Result Grid | Filter Rows: | Export:

productId	productName	productDescription	Rating
17	Nike	great shoes	3.4000
18	WildCraft	great bags	3.0000
19	Timex	great watches	5.0000
20	H&M	great clothes	5.0000
21	Zara	great clothes	3.5000
22	IPad	great Tabs	3.0000

2. To display the Top 3 Products

```
8   create view `Top 3 Products` as
9   select *
10  from product
11  Order By OnOrder desc
12  Limit 3;
13
```

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

productId	productName	productCost	productPrice	productDescription	InStock	OnOrder	Cart_CartId	Category_CategoryID
20	H&M	10	20	great clothes	25	20	12	13
22	IPad	15	30	great Tabs	30	15	9	15
18	WildCraft	10	20	great bags	5	8	10	12