Introduction

Fashion E-Commerce Database

A F-ECOM database needs to store information's about the products (ID, item name, brand name, description, MRP, discount, images), its customers (ID, name, DOB, gender, email, phone number, address), categories of the product(ID, name, gender), Seller of the products (ID, name, location), cart of the customer (ID, quantity, price), Order (ID, Status), Transaction(ID, Status).

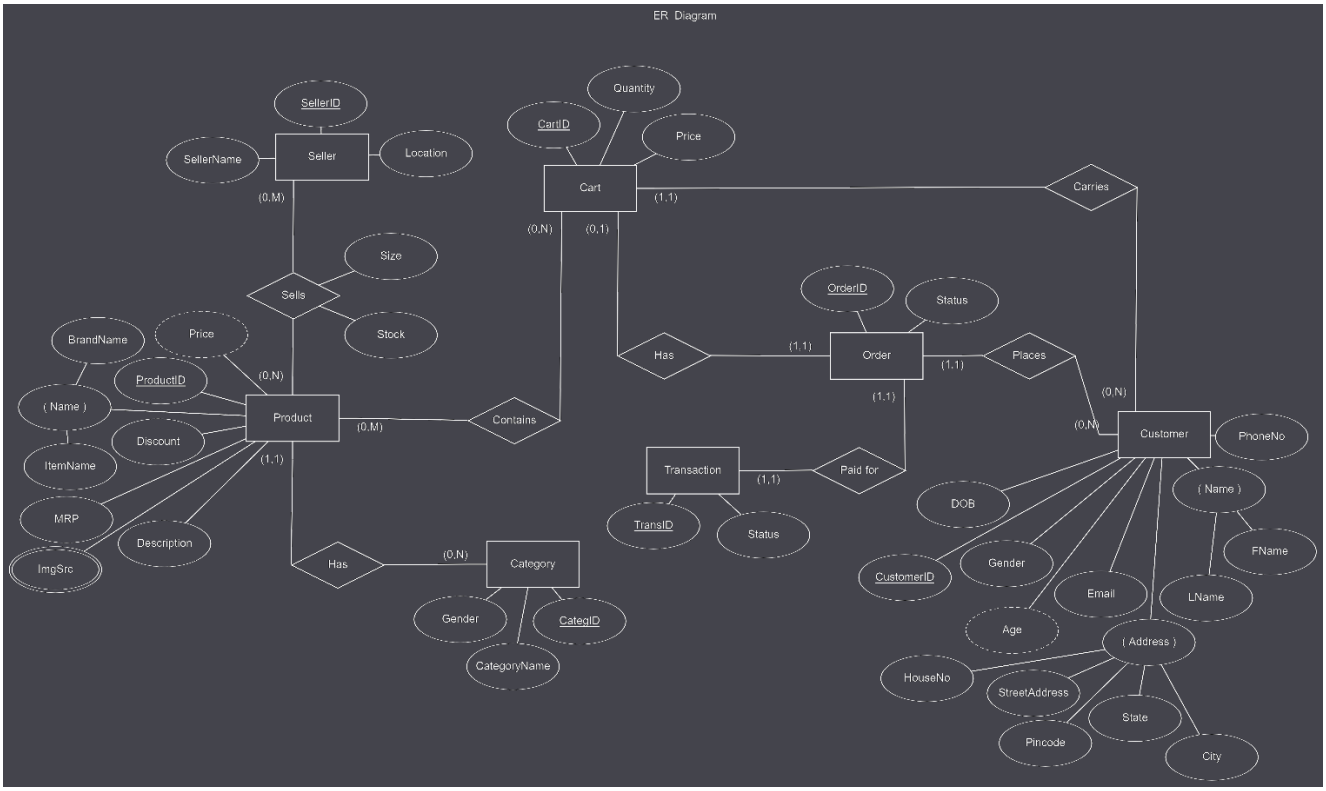
Key points:

- No two customers should have same phone number, which means a phone number can be registered only once.
- Every details of the customer are required at the time of registration.
- A seller can only have a single location
- Size and stock of the products from the seller need to be recorded.
- Some of the products might have zero MRP and zero discount, which will be considered as the free- limited time -sale.
- A product can have multiple images
- A customer can only have a single cart at any instance of time, but database should store his previous carts which is stored for making user experience better.
- An order is directly based on the customer and cart.
- Cart must contain all the product added by the cart owner.
- An order can have only one transaction
- Both total price and total quantity must be present in the cart details.
- Categories are based on the gender. (e.g. Both 'm' and 'f' should have sandals)
- A customer can have as many orders as he/she wish
- A product belongs to a single category
- When a product is added into a customer's cart, both price and quantity must be updated. This must be updated even when the customer removes the product from his/her cart.
- Transaction status and order status currently has only 2 values[Paid/not-paid and delivered/not-delivered].

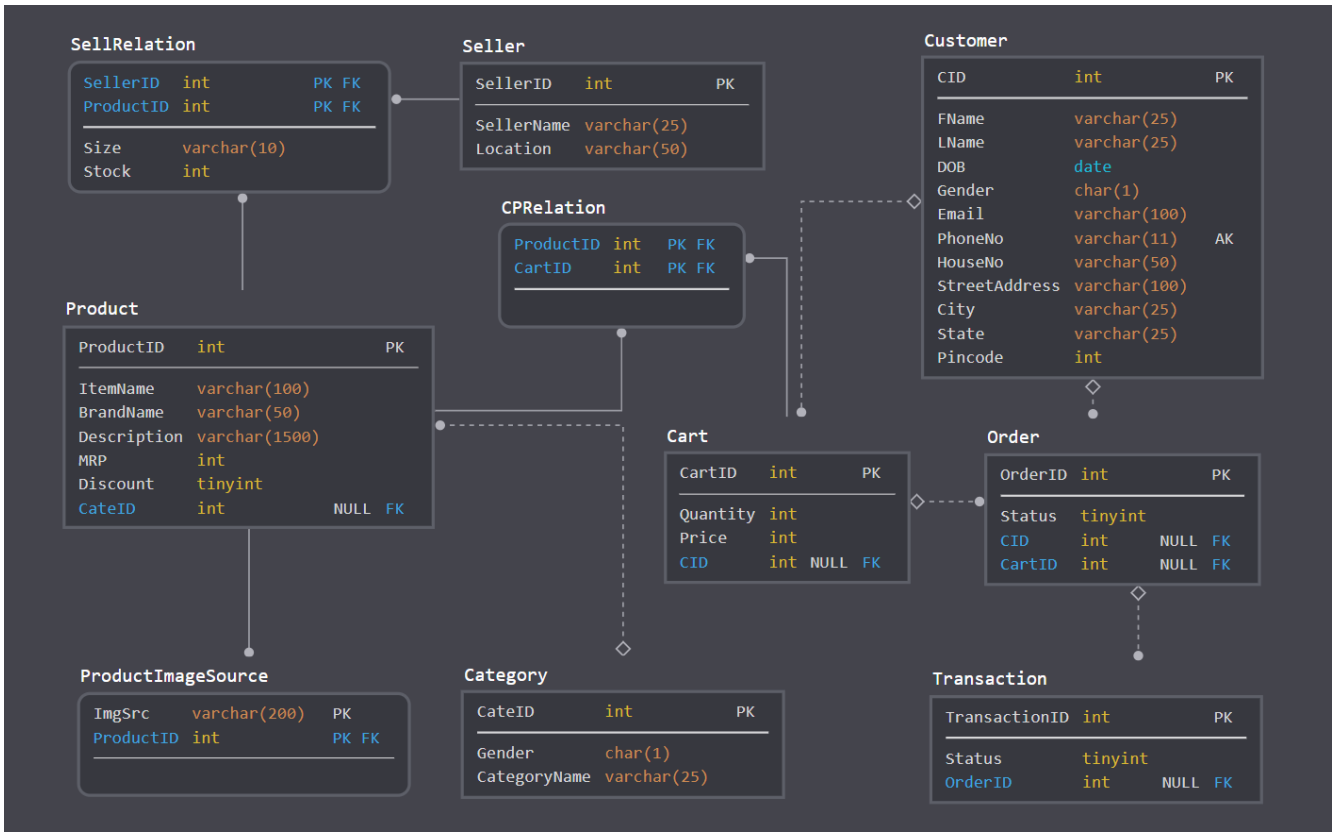
Data Model

- Data type of Product.Discount is chosen as tinyint, since the discount will never exceed the integer value 100.
- Both statuses are considered as tinyint data-type.
- DOB is considered as date data-type.

ER Model



Relational Model



FD and Normalization

FD

- ProductID → ItemName, ProductID → BrandName, ProductID → Description, ProductID → MRP, ProductID → Discount, ProductID → CateID
- CartID → Quantity, CartID → Price, CartID → CID
- CID → FName, CID → LName, CID → DOB, CID → Gender, CID → Email, CID → PhoneNo, CID → HouseNo, CID → StreetAddress, CID → City, CID → State, CID → Pincode
- OrderID → Status, OrderID → CID, OrderID → CartID
- TransactionID → Status, TransactionID → OrderID
- SellerID → SellerName, SellerID → Location
- CategoryID → Gender, CategoryID → CategoryName
- {SellerID, ProductID} → Size, {SellerID, ProductID} → Stock
- {BrandName, ItemName } → ProductID, {BrandName, ItemName } → BrandName, {BrandName, ItemName } → ItemName, {BrandName, ItemName } → Description, {BrandName, ItemName } → MRP, {BrandName, ItemName } → Discount, {BrandName, ItemName } → CateID

Testing for lossless join property

Since the relations are already in their normal forms, no further decompositions are involved. Therefor there is no need to test for lossless join property

Candidate Keys

1. Product

   On applying attribute closure,

   $\{ProductID\}^+$ = {ProductID, ItemName, BrandName, Description, MRP, Discount, CateID}

   On applying attribute closure,

   $\{BrandName, ItemName\}^+$ = {ProductID, ItemName, BrandName, Description, MRP, Discount, CateID}

   We can see that both {Productid} and {BrandName, ItemName} covers all the attributes of the relation.

   Therefore, candidate keys = {Productid, {BrandName, ItemName}}

   We choose primary key = {ProductID}

2. Cart

   On applying attribute closure,

   $\{CartID\}^+$ = {CartID, Quantity, Price, CID}

   Since {CartID} covers all the attributes of the relation, it is the candidate key.

3. Customer
   On applying attribute closure,
   {CID}$^+$ = {CID, FName, LName, DOB, Gender, Email, PhoneNo, HouseNo, StretAddress, City, State, Pincode}
   Since {CID} covers all the attributes of the relation, it is the candidate key.

4. Order
   On applying attribute closure,
   {OrderID}$^+$ = {OrderID, Status, CID, CartID}
   Since {OrderID} covers all the attributes of the relation, it is the candidate key.

5. Transaction
   On applying attribute closure,
   {TransactionID}$^+$ = {TransactionID, Status, OrderID}
   Since {TransactionID} covers all the attributes of the relation, it is the candidate key.

6. Seller
   On applying attribute closure,
   {SellerID}$^+$ = {SellerID, SellerName, Location}
   Since {SellerID} covers all the attributes of the relation, it is the candidate key.

7. Category
   On applying attribute closure,
   {CateID}$^+$ = {CateID, Gender, CategoryName}
   Since {CateID} covers all the attributes of the relation, it is the candidate key.

8. CPRelation
   On applying attribute closure,
   {ProductID, CartID}$^+$ = {ProductID, CartID}
   Since {ProductID, CartID} covers all the attributes of the relation, it is the candidate key.

9. SellRelation
   On applying attribute closure,
   {SellerID, ProductID}$^+$ = { SellerID, ProductID, size, stock }
   Since {SellerID, ProductID} covers all the attributes of the relation, it is the candidate key.

10. ProductImageSource
    On applying attribute closure,
    {ImgSrc, ProductID}$^+$ = {ImgSrc, ProductID}
    Since {ImgSrc, ProductID} covers all the attributes of the relation, it is the candidate key.

Normalization

Since all the relations have been created by converting the ER diagram to relational schema, the relations are in 3NF and BCNF.

There could be a case where 'ProductName' is added into a relation 'SellRelation'. Since only 'ProductID' is enough for functionally determining the 'ProductName', partial dependency on the primary key would be created and this case would violate 2NF. But all relations are normalized.

DDL

```
DROP DATABASE IF EXISTS `FECOM`;
CREATE DATABASE `FECOM`;
USE `FECOM`;

CREATE TABLE IF NOT EXISTS `Customer`(
 `CID`          int NOT NULL ,
 `FName`        varchar(25) NOT NULL ,
 `LName`        varchar(25) NOT NULL ,
 `DOB`          date NOT NULL,
 `Gender`       char(1) NOT NULL CHECK(`Gender`='M' or `Gender`='W'or `Gender`='O'),
 `Email`        varchar(100) NOT NULL ,
 `PhoneNo`      varchar(11) NOT NULL ,
 `HouseNo`      varchar(50) NOT NULL ,
 `StreetAddress` varchar(100) NOT NULL ,
 `City`         varchar(25) NOT NULL ,
 `State`        varchar(25) NOT NULL ,
 `Pincode`      int NOT NULL ,
 UNIQUE (`PhoneNo`),
 PRIMARY KEY (`CID`)
);

CREATE TABLE IF NOT EXISTS `Cart`(
 `CartID`   int NOT NULL CHECK(`CartID`<9999),
 `Quantity` int NOT NULL ,
 `Price`    int NOT NULL ,
 `CID`      int ,

PRIMARY KEY (`CartID`),
CONSTRAINT `FK_CID` FOREIGN KEY (`CID`) REFERENCES `Customer` (`CID`) ON DELETE SET NULL ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS `Category`(
 `CateID`       int NOT NULL ,
```

```
 `Gender`      char(1) NOT NULL CHECK(`Gender`='M' or `Gender`='W' or `Gender`='O'),
 `CategoryName` varchar(25) NOT NULL ,

PRIMARY KEY (`CateID`)
);

CREATE TABLE IF NOT EXISTS `Product`(
 `ProductID`   int NOT NULL ,
 `ItemName`    varchar(100) NOT NULL ,
 `BrandName`   varchar(50) NOT NULL ,
 `Description` varchar(1500) NOT NULL ,
 `MRP`         int NOT NULL ,
 `Discount`    tinyint NOT NULL DEFAULT 0 ,
 `CateID`      int ,

PRIMARY KEY (`ProductID`),
CONSTRAINT `FK_CateID` FOREIGN KEY (`CateID`) REFERENCES `Category` (`CateID`) ON DELETE SET NULL ON UPDATE CASCA
DE
);

CREATE TABLE IF NOT EXISTS `CPRelation`(
 `ProductID` int NOT NULL ,
 `CartID`    int NOT NULL ,

PRIMARY KEY (`ProductID`, `CartID`),
CONSTRAINT `FK_ProductID` FOREIGN KEY (`ProductID`) REFERENCES `Product` (`ProductID`) ON DELETE CASCADE ON UPDA
TE CASCADE,
CONSTRAINT `FK_CartID` FOREIGN KEY (`CartID`) REFERENCES `Cart` (`CartID`) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS `Order`(
 `OrderID` int NOT NULL ,
 `Status`  tinyint NOT NULL ,
 `CID`     int ,
 `CartID`  int ,

PRIMARY KEY (`OrderID`),
CONSTRAINT `FK_CID2` FOREIGN KEY (`CID`) REFERENCES `Customer` (`CID`) ON DELETE SET NULL ON UPDATE CASCADE,
CONSTRAINT `FK_CartID2` FOREIGN KEY (`CartID`) REFERENCES `Cart` (`CartID`) ON DELETE SET NULL ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS `ProductImageSource`(
 `ImgSrc`    varchar(200) NOT NULL ,
 `ProductID` int NOT NULL ,
```

```
PRIMARY KEY (`ImgSrc`, `ProductID`),
CONSTRAINT `FK_ProductID2` FOREIGN KEY (`ProductID`) REFERENCES `Product` (`ProductID`) ON DELETE CASCADE ON UPD
ATE CASCADE
);

CREATE TABLE IF NOT EXISTS `Seller`(
 `SellerID`   int NOT NULL ,
 `SellerName` varchar(25) NOT NULL ,
 `Location`   varchar(50) NOT NULL ,

PRIMARY KEY (`SellerID`)
);

CREATE TABLE IF NOT EXISTS `SellRelation`(
 `SellerID`  int NOT NULL ,
 `ProductID` int NOT NULL ,
 `Size`      varchar(10) NOT NULL ,
 `Stock`     int NOT NULL ,

PRIMARY KEY (`SellerID`, `ProductID`),
CONSTRAINT `FK_ProductID3` FOREIGN KEY (`ProductID`) REFERENCES `Product` (`ProductID`) ON DELETE CASCADE ON UPD
ATE CASCADE,
CONSTRAINT `FK_SellerID` FOREIGN KEY (`SellerID`) REFERENCES `Seller` (`SellerID`) ON DELETE CASCADE ON UPDATE CASCA
DE
);

CREATE TABLE IF NOT EXISTS `Transaction`(
 `TransactionID` int NOT NULL ,
 `Status`        tinyint NOT NULL ,
 `OrderID`       int ,

PRIMARY KEY (`TransactionID`),
CONSTRAINT `FK_OrderID` FOREIGN KEY (`OrderID`) REFERENCES `Order` (`OrderID`) ON DELETE SET NULL ON UPDATE CASC
ADE
);
```

DML

Most of the inserted data is not mentioned here due to space constraint.

For the complete code, visit https://github.com/arg-z/fashionECom-DBM.

```
insert into `Category` values
(8635, 'M', 'Casual Shoes'),
```

```
(1880, 'W', 'Leggings'),
(1750, 'W', 'Flip Flop & Slippers'),
(9482, 'M', 'Tshirts'),
(8236, 'M', 'Track Pants'),
(7541, 'M', 'Casual Sandals'),
(6754, 'M', 'Formal Shoes'),
(1140, 'W', 'Sarees'),
(5632, 'M', 'Socks'),
(8494, 'M', 'Jackets & Coats');

INSERT INTO `Customer` VALUES
(9487 , 'Rita' , 'Barron' , '2003-09-
30' , 'M' , 'ritabarron@wemail.com' , '8027797792' , '1300' , 'East Dowling Road' , 'Anchorage' , 'AK' , 99518),
(8014 , 'Elizabeth' , 'Saucedo' , '2016-06-
22' , 'm' , 'elizabethsaucedo@wemail.com' , '7644815897' , '736' , 'Middle Turnpike East' , 'Manchester' , 'CT' , 06040),
(6219 , 'Amanda' , 'Smith' , '1997-05-
06' , 'M' , 'amandasmith@wemail.com' , '9856296919' , '70' , 'Orchard Shore Road' , 'Colchester' , 'VT' , 05446),
(1512 , 'Jenny' , 'Davis' , '2006-09-
16' , 'W' , 'jennydavis@wemail.com' , '7277506540' , '2306' , 'Edinburgh Drive' , 'Montgomery' , 'AL' , 36116),
(9236 , 'William' , 'Cowan' , '2010-03-
11' , 'W' , 'williamcowan@wemail.com' , '8472166117' , '13583' , 'West 68th Avenue' , 'Arvada' , 'CO' , 80004),
(6921 , 'Lisa' , 'Black' , '1999-05-
13' , 'm' , 'lisablack@wemail.com' , '8161673607' , '4385' , 'Wares Ferry Road' , 'Montgomery' , 'AL' , 36109);

INSERT INTO `Product` VALUES
(4695 , 'Lightly Distressed Skinny Jeans' , 'DNMX' , 'Logo placement;;5-pocket styling;;Cotton Blend;;Belt loops with button-
loop closure;;Machine wash cold;;Low Rise;;' , 0 , 0, 4504),
(9356 , 'Washed Mid-
Rise Jeans' , 'PEOPLE' , 'Machine wash;;75.5% Cotton, 23.5% Polyester, 1% Elastane;;Mid Rise;;' , 1699 , 50, 4504),
(9543 , 'Lightly Washed Joggers with Drawstring Fastening' , 'DNMX' , 'Whiskers;;5-
pocket styling;;Cotton Blend;;Belt loops;;Stretchable fabric;;Machine wash;;Mid Rise;;' , 1499 , 49, 4504),
(2642 , 'Mid-Wash Slim Fit Jeans' , 'FLYING MACHINE' , '5-
pocket styling;;100% Cotton;;Zip fly with pocket styling;;Machine wash;;Mid Rise;;' , 2199 , 59, 4504),
(4014 , 'Skinny Fit Jeans with Whiskers' , 'DNMX' , 'Zip fly button closure;;Belt loops;;Cotton Blend;;5-
pocket styling;;Line dry;;Machine wash;;Mid Rise;;' , 1299 , 30, 4504),
(3839 , 'Lightly Washed Mid-
Rise Slim Fit Jeans' , 'MUFTI' , 'Machine wash cold;;Mid Rise;;98% cotton, 2% elastane;;' , 2199 , 35, 4504),
(7175 , 'High-Neck Zip-Front Track Jacket' , 'PERFORMAX' , 'Regular Fit;;Moisture managed Quickdry technology;;Water-
Resistant;;Insert pockets;;Polyester;;Transfer detail;;Performax technology and garment features vary from style to style;;Ma
chine wash;;' , 1499 , 30, 8494);


INSERT INTO `ProductImageSource` VALUES
('https://assets.ajio.com/medias/sys_master/root/h8d/h1f/16496021209118/-78Wx98H-441025044-tintblue-
MODEL3.jpg' , 4695),
```

```
('https://assets.ajio.com/medias/sys_master/root/hbd/h27/16496024125470/-78Wx98H-441025044-tintblue-MODEL4.jpg' , 4695),
('https://assets.ajio.com/medias/sys_master/root/h28/h76/16496017965086/-78Wx98H-441025044-tintblue-MODEL2.jpg' , 4695),
('https://assets.ajio.com/medias/sys_master/root/hff/hac/16496030089246/-78Wx98H-441025044-tintblue-MODEL.jpg' , 4695),
('https://assets.ajio.com/medias/sys_master/root/h87/hdc/16496028909598/-473Wx593H-441025044-tintblue-MODEL.jpg' , 4695),
('https://assets.ajio.com/medias/sys_master/root/hc3/h82/14270494146590/-78Wx98H-460482018-blue-MODEL5.jpg' , 9356);

INSERT INTO `Seller` VALUES
(8122, 'Appario Retail' , 'CO'),
(8264, 'Cloudtail' , 'AL'),
(6406, 'STPL' , 'MD'),
(3954, 'Electrama' , 'FL'),
(2928, 'Darshita' , 'AZ'),
(2807, 'BasicDeal' , 'OK'),
(4496, 'weguarantee' , 'VT');

INSERT INTO SellRelation VALUES
(4496, 5952, '36' ,19),
(2807, 8798, '38' ,32),
(2928, 3931, '37' ,45),
(3954, 7415, '37' ,8),
(2928, 9936, '36' ,13),
(8122, 9229, '36' ,31),
(8122, 2451, '38' ,5);

insert into Cart values
(4155, 0, 0,7308),
(4152, 0, 0,7308),
(7028, 0, 0,3755),
(9880, 0, 0,8014),
(5685, 0, 0,6062);

insert into CPRelation values
(3521, 4309),
(4695, 5345),
(3890, 9627),
(4695, 6919),
(3521, 9627),
(1932, 4309),
(9356, 4152),
(3890, 4152),
```

```
(9356, 7028),
(7175, 6919);

insert into `Order` values
(4367, 1 , 8547, 4152),
(8507, 0 , 5786, 7028),
(3691, 1 , 3359, 4309),
(7033, 0 , 6876, 9627),
(8699, 1 , 6273, 5345),
(6660, 1 , 3359, 4478),
(3672, 1 , 3673, 6919);

insert into `Transaction` values
(1000, 1 , 3672),
(1001, 0 , 6660),
(1002, 0 , 3691),
(1003, 1 , 7033),
(1004, 0 , 4367),
(1005, 1 , 8507);
```

Triggers

Some of the much alike triggers are omitted from the report.

For the complete code, visit https://github.com/arg-z/fashionECom-DBM.

```
-- Check constraint for gender in Customer table
delimiter $$
create trigger TRI_CUST_GENDER before insert on `Customer`
for each row
begin
   if (new.`Gender` = 'm' or new.`Gender` = 'w' or new.`Gender` = 'o') then
      set new.`Gender` =  (ucase(new.`Gender`));
   elseif (new.`Gender` != 'M' or new.`Gender` != 'W' or new.`Gender` != 'O') then
      signal sqlstate '45000' set message_text = 'gender must in [\'M\',\'F\',\'O\',\'m\',\'f\',\'o\']';
   end if;
end;$$

-- Check constraint for cart id as `Cart`.`CartID`>9999 ( -> which implies that the id must be of 4 digit)
delimiter $$
create trigger TRI_CART_ID before insert on `Cart`
for each row
begin
```

```
  if (new.`CartID` >9999) then
     signal sqlstate '45000' set message_text = 'max digit of ID is 4';
  end if;
end;$$

-- Cart Price and Quantity have to be updated before inserting a product into the cart
delimiter $$
create trigger TRI_CART_UPDATE after insert on `CPRelation`
for each row
begin
DECLARE tempint int;
DECLARE tempmrp int;
DECLARE realmrp int;
DECLARE tempdisco int;
set tempint := (select `Quantity` from `Cart` where new.`CartID`=`Cart`.`CartID`) + 1;
set realmrp := (select `Price` from `Cart` where new.`CartID`=`Cart`.`CartID`) ;
set tempmrp := (select `MRP` from `Product` where new.`ProductID`=`Product`.`ProductID`);
set tempdisco := (select `Discount` from `Product` where new.`ProductID`=`Product`.`ProductID`);
 UPDATE `Cart` SET `Quantity`= tempint where new.CartID = cart.CartID;
 UPDATE `Cart` SET Price = tempmrp where new.CartID = cart.CartID;
 UPDATE `Cart` SET `Price` = realmrp + (tempmrp - (tempmrp*tempdisco/100)) where new.CartID = cart.CartID;
end;$$
```

SQL Queries - DQL

```
-- List the customers and their cart details whose cart either contains at least two items or is worth more than Rs.1500
select CID, FName, LName, Price, Quantity from Customer natural join
((SELECT * FROM Cart GROUP BY CartID HAVING SUM(Price)>1500)  union
(SELECT * FROM Cart GROUP BY CartID HAVING SUM(Quantity)>2 or SUM(Quantity)=2)
) as a1;

-- Some of the customers got the delivered item, but never paid for it.
List the customer's details with his/her phone number, orderID, transactionID, cartID and the price which they owe.
select e2.TransactionID, e2.OrderID,e2.CID, e2.CartID, e2.Quantity, e2.Price, cus.FName, cus.LName, cus.PhoneNo
from Customer as cus inner join
(select e1.TransactionID, e1.OrderID,e1.CID, e1.CartID, cca.Quantity, cca.Price from Cart as cca inner join
(select t.TransactionID, t.OrderID,o.CID, o.CartID  from `Transaction` as t inner join `Order`as o on t.OrderID = o.OrderID
where t.`Status` = 0 and o.`Status`=1) as e1 where cca.CartID = e1.CartID and Quantity !=0
) as e2 where e2.CID = cus.CID;

-- List out the products with the number of images available [only if the number is not zero].
select p.ProductID, p.ItemName, p.BrandName, kk.CountImages from Product as p join
(select count(*) as CountImages, ProductID from ProductImageSource as pis group by ProductID) as kk
```

```
where kk.ProductID = p.ProductID;

-- Get the most priced product in the catalog.
select max(MRP), ProductID, BrandName, ItemName from Product;

-- Get the 10 most priced product in the catalog
select ProductID, BrandName, ItemName, MRP from Product order by (MRP - (MRP*Discount)/100) desc limit 10;

-- List out all the customers with all of their carts
(select * from Customer left outer join Cart on Customer.CID = Cart.CID )
union
(select * from Customer right outer join Cart on Customer.CID = Cart.CID )
```

Conclusion

Inserted data is actually scraped from https://www.ajio.com using python. Code is present in the GitHub repo https://github.com/arg-z/fashionECom-DBM .

Well, there is no code to update each and every thing when the product is deleted from the db. Only the foreign keys will be updated which is dependent of that product primary key.