Team 200 OK
December 07, 2022

Aishwarya Sasane
Anusha Raju
Maurvin Shah
Monisha Rahim
Varun Kapuria

Chapter 1: Requirement Analysis

Harbor Freight is America's go-to store for low prices on tools and equipment. It currently operates a chain of retail stores and the company is in need of an e-commerce website to enable its customers to buy its products over the internet. The two main requirements from Harbor Freight are to improve the customer shopping experience as well as the customer support experience. The company aims to achieve an increase in sales through its online website.

Harbor Freight carries over 4,000 products, specializing in air compressors, generators, wrenches, drills, saws, hand tools, tool storage, welding supplies, and automotive tools. Every product has a unique identifier, product name, product overview, and price per unit. Each product

would also have a return period by which the product could be returned. Each product can have multiple units. Every product unit has a manufacturing date and the availability of that unit should also be tracked. Harbor Freight sells gift cards as products. A gift could be purchased by one customer and gifted to another customer or used by themselves. Each gift card has a code and gift card amount. Once a gift card is activated, it cannot be used again.

Each product belongs to a particular department. Each department can have multiple subcategories. For example, the product 'Hose' belongs to the category 'Lawn and Garden' and the sub-category 'Garden Tools'. Each category and sub-category uniquely identify a department. The product units are available in the stores and they are shipped to the customers from the store. Harbor freight does not have separate warehouses. All the products are stored in the multiple stores that harbor freight has. Every store has an address, phone number and store operating hours. Some stores have different open and close times for days of the week.

Customers can register themselves on the website. They should provide their name, email, phone number, and password to register. Customers can add addresses from the website. A customer can have multiple addresses and each address has the street address, city, state, country, zip code, and a tag to identify whether the address is a home, office, or other address. Harbor Freight also has its membership club. Some customers choose to be a member by paying the membership fee. The membership fee is due one year from the date purchased.

Customers can order products from the website. Details about the order such as Order placement date, shipping date, order status, estimated delivery date, actual delivery date, payment mode, and order value are stored. An order uses a particular shipping method. There are different shipping types such as Flat Rate, Express, and Truck through which the orders are shipped. Each shipping type has a price and the number of days it takes to deliver.

A customer can apply one coupon at most and only once. But, the same coupon can be used by multiple customers. All coupons have different discounts associated with them and are valid only for a particular period. The coupons also have a minimum cart value associated with them. For example, a coupon code called "SALE15" provides a 15% discount on an order value of more than 1000 dollars. In addition, Harbor Freight also provides deals on products. A deal is applied to a product or a product category. The deal has a start date and an end date. For example, there could be a summer deal on gardening products. Also, some deals are specific to members only. Customers can also enroll for a special type of credit card exclusively provided for Harbour Freight customers. The credit card comes with a credit limit up to which the customers can use them. The credit cards have a unique card number and CVV. The company wants to track the credit card purchase date as well as the validity.

When customers buy a product from the website, they can review the products once they are delivered to them. A review should have ratings on a scale of 1 to 5 stars and a description. Every review should be associated with a product that was bought as part of an order. Each product purchased by the customer can have only one review.

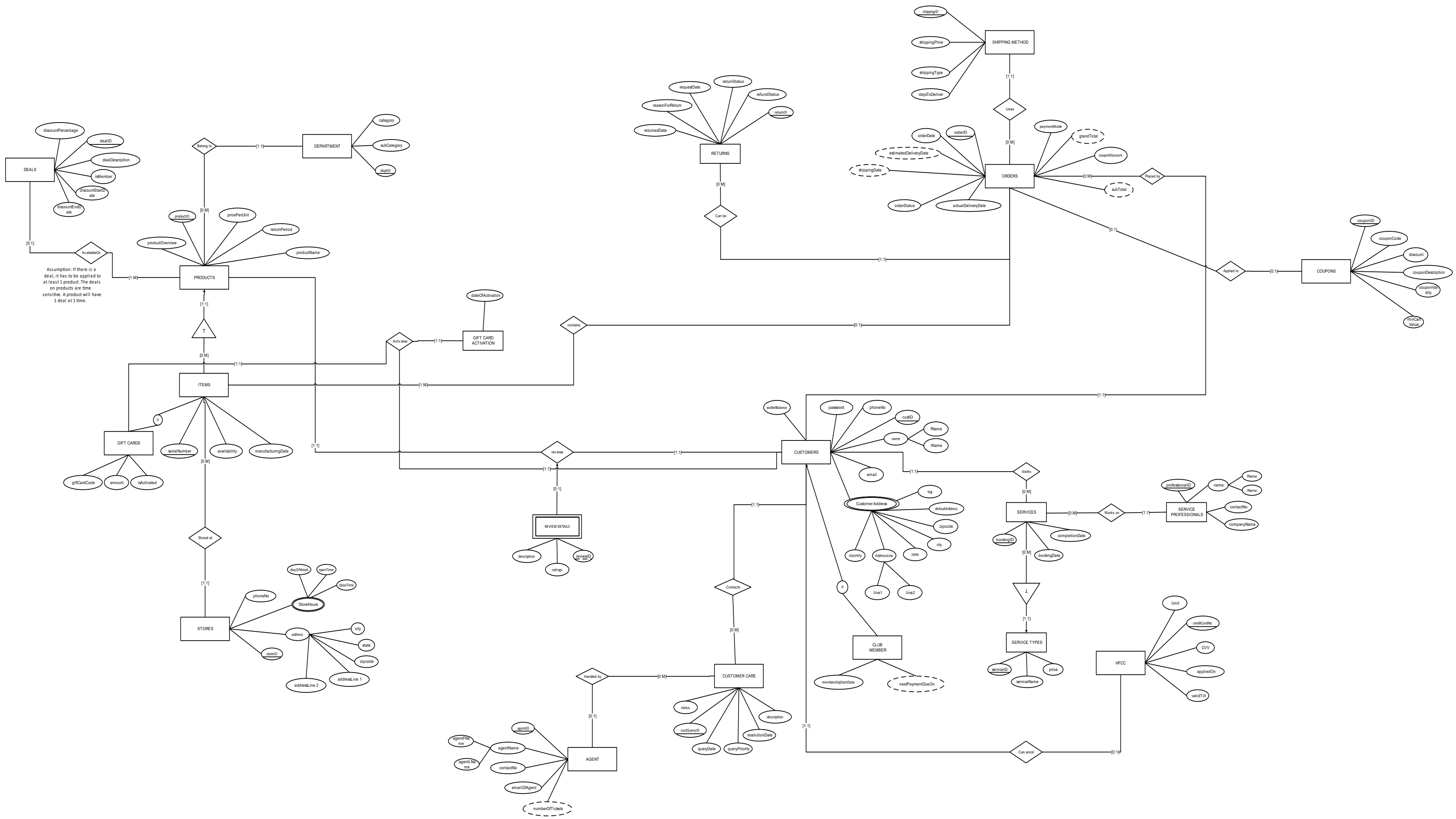
The customers have the flexibility to return a product within the return period. The customer is required to fill out a return request form where they have to provide the reason for the return. The return request date and the actual return date of the product are captured. Customers can check the status of their returns and subsequent refunds on the website. One customer can return many products, but one return order can be associated with only 1 customer.

The customers can reach out to customer care via the website. They should be able to provide a description of the problem. The date of creation, query priority, status, and resolution date are captured. The query is assigned to agents. The agents have a name, contact number, email and the number of tickets they are working on.

Harbour Freight provides services to customers such as repair, regular maintenance etc. Each service has a name and a price associated with it. When a customer books a service, the booking date and completion date are recorded. Harbour Freight hires third party service professionals to work on these services. The name, contact number and company name of the service professionals are stored.

Chapter 2: Conceptual Design

ER Diagram



DEALS
discountPercentage
dealID
dealDescription
isMember
discountStartDate
discountEndDate
[0:1]
AvailableOn
[1:M]
Assumption: If there is a deal, it has to be applied to at least 1 product. The deals on products are time sensitive. A product will have 1 deal at 1 time.
Belong to
[1:1]
DEPARTMENT
category
subCategory
deptID
[0:M]
PRODUCTS
productID
pricePerUnit
returnPeriod
productOverview
productName
[1:1]
T
[0:M]
ITEMS
[1:1]
D
GIFT CARDS
giftCardCode
amount
isActivated
serialNumber
availability
manufacturingDate
[0:M]
Stored at
[1:1]
STORES
phoneNo
StoreHours
dayOfWeek
openTime
closeTime
address
city
state
zipcode
addressLine 1
addressLine 2
storeID
Activates
[1:1]
GIFT CARD ACTIVATION
dateOfActivation
contains
[1:M]
[0:1]
reviews
[1:1]
[1:1]
[0:1]
REVIEW DETAILS
description
ratings
reviewID
RETURNS
requestDate
returnStatus
refundStatus
returnID
reasonForReturn
returnedDate
[0:M]
Can be
[1:1]
SHIPPING METHOD
shippingID
shippingPrice
shippingType
daysToDeliver
[1:1]
Uses
[0:M]
ORDERS
orderDate
orderID
estimatedDeliveryDate
shippingDate
orderStatus
actualDeliveryDate
paymentMode
grandTotal
couponDiscount
subTotal
[0:M]
Placed by
[0:1]
Applied to
[0:1]
COUPONS
couponID
couponCode
discount
couponDescription
couponValidity
minCartValue
[1:1]
CUSTOMERS
walletBalance
password
phoneNo
custID
name
fName
lName
email
Customer Address
tag
defaultAddress
zipcode
city
state
AddressLine
line1
line2
country
D
CLUB MEMBER
membershipStartDate
nextPaymentDueOn
[1:1]
books
[0:M]
SERVICES
bookingID
bookingDate
completionDate
[0:M]
Works on
[1:1]
SERVICE PROFESSIONALS
professionalID
name
fName
lName
contactNo
companyName
[0:M]
⊥
[1:1]
SERVICE TYPES
serviceID
serviceName
price
HFCC
limit
creditCardNo
CVV
appliedOn
validTill
[1:1]
Can enroll
[0:1]
[1:1]
Contacts
[0:M]
CUSTOMER CARE
status
custQueryID
queryDate
queryPriority
resolutionDate
description
Handled by
[0:M]
[0:1]
AGENT
agentID
agentName
agentFName
agentLName
contactNo
emailOfAgent
numberOfTickets

Data Dictionary

Schema Construct	Construct Description	Other Information
AGENTS	**Entity Class to model Agent Information**	
agentID	The agent identification number	Identifying Attribute
agentFName	First name of the agent	Attribute cannot contain NULL values
agentLName	Last name of the agent	Attribute cannot contain NULL values
contactNo	Contact number of the agent	Has to be exactly 10 numbers
emailOfAgent	Email address of the agent	Attribute cannot contain NULL values
numberOfTickets	The number of tickets handled by the agent	Cannot contain negative values
CLUB MEMBER	**Entity Class for Clubmembers**	
custID	Customer Identification Number	Identifying Attribute
membershipStartDate	Date when the membership starts on	
nextPaymentDueOn	Payment due date for the membership	One year after start date
COUPONS	**Entity Class for Coupons**	
couponID	The Coupon identification number	Identifying Attribute
couponCode	Coupon code that customer will put in	
couponDescription	Coupon description	
couponValidity	Till what date the coupon is valid for	
minCartValue	Minimum cart value needed for coupon to be valid	
discount	Discount percent	Discount percent should be less than 100
CUSTOMER ADDRESS	**Entity Class to model the Customer Address information**	
addressID	Address identification number	Identifying Attribute
custID	customer Identification Number	
city	Name of the City	Attribute cannot contain NULL values

country	The country name	Attribute cannot contain NULL values
defaultAddress	Whether the address is saved as default address by the customer	By deafult its no
line1	Address Line 1	Attribute cannot contain NULL values
line2	Address Line 2	
state	The state name	Attribute cannot contain NULL values
tag	Tags like Home/Work/Other	
zipcode	The zipcode of the city	Attribute cannot contain NULL values; should have exactly 5 digits
CUSTOMER CARE	**Entity Class to model Customer Care Service information**	
custQueryID	Customer query Identification number	Identifying Attribute
description	Description of the query	
queryDate	date of creation of query	
resolutionDate	date of resolution of query	
status	Status of the ticket eg. "Open"and "Closed"and "Pending"	Default is open
custID	Customer identification number	
agentID	Agent identification number	
queryPriority	priority of the query	
CUSTOMERS	**Entity class to model the customer information**	
custID	Customer identification number	Identifying Attribute
fName	First name of the customer	Attribute cannot contain NULL values
lName	Last name of the customer	Attribute cannot contain NULL values
phoneNo	Contact number of the customer	Attribute cannot contain NULL values; Phone number will be exactly 10 digits
email	Email of customer	Each email will be unique and not null; should be in email format
password	Password of customer	Minimum 8 characters; Attribute cannot contain NULL values
walletBalance	Balance in the customer's wallet	Deafult value is 0
DEALS	**Entity class to store information about the Deals**	

dealID	Deal identification number	Identifying Attribute
dealDescription	Deal description	Attribute cannot contain NULL values
discountStartDate	Start of the discount date	Attribute cannot contain NULL values
discountEndDate	End of the discount date	Attribute cannot contain NULL values
discountPercentage	Discount percent	Attribute cannot contain NULL values
isMember	Whether the deal is just for members or all customers. 1=Only for members 0 = All customers	
DEALS ON PRODUCTS	**Entity class to store information about the Deals**	
dealID	Deal identification number	Identifying Attribute
productID	Product identification number	Identifying Attribute
DEPARTMENTS	**Entity Class for storing Product Departments**	
deptID	Department identification number	Identifying Attribute
category	Category of the department	Attribute cannot contain NULL values
subCatergory	Sub Category of the department	Attribute cannot contain NULL values
GIFT_CARD_ACTIVATION	**Entity Class for storing gift card activation details by customer**	
serialNumber	Identification code for Gift Card	Identifying Attribute
custID	Identification code for Customer	Identifying Attribute
dateOfActivation	Date when the gift card is activated	Attribute cannot contain NULL values
GIFT_CARDS	**Entity Class for storing Gift Cards**	
serialNumber	Identification code for Items	Identifying Attribute
giftCardCode	Numeric code of the Gift Card	Attribute sould be unique; should be exactly 16 digits
amount	Amount of the gift card	Attribute cannot contain NULL values
isActivated	To check if the card is activated by the customer	Deafult value is no
HFCC	**Entity class to store details of the Harbour Freight credit cards purchased by customers**	

limit	How much balance is available in the card	Attribute cannot contain NULL values
creditCardNo	Unique 16 digit numeric number of purchased by the customer	Identifying Attribute; length should be 16 digits
custID	Customer Identification Number	
CVV	Security number	Length is 3 digits
validTill	The expiry date	Attribute cannot contain NULL values
appliedOn	Date the customer applied for the card	Attribute cannot contain NULL values
ITEMS	**Entity Class for storing Items**	
serialNumber	Serial Number of Item	Identifying Attribute
availability	Item available or not	
manufacturingDate	Date it was manufactured	
productID	Customer Identification Number	
storeID	Store Identification Number	
ITEMS IN ORDERS	**Entity Class for storing Items**	
serialNumber	Serial Number of Item	Identifying Attribute
orderID	Order Identification number	Identifying Attribute
ORDERS	**Entity Class for orders placed by customers**	
orderID	Order Identification number	Identifying Attribute
orderDate	Date when the order was placed	Attribute cannot contain NULL values
orderStatus	Status whether deliver was successful or not	
shippingDate	Date when product is shipped	
estimatedDeliveryDate	Estimated Date of delivery	Attribute cannot contain NULL values
actualDeliveryDate	Actual date when the delivery was made	
paymentMode	Credit/Debit/COD/PayPal	
couponDiscount		
subtotal	sum of unit prices + deals	Attribute cannot contain NULL values
grandTotal	(subtotal - coupon discount) + shipping cost	Attribute cannot contain NULL values

couponID	Which coupon was applied to the purchase	
custID	Customer identification number	
shippingID	Shipping Method Identification Number	
PRODUCTS	**Entity class to store information about the products**	
productID	Product Identification number	Identifying Attribute
productOverview	Description of the product	Attribute cannot contain NULL values
pricePerUnit	Price of the product	Attribute cannot contain NULL values
productName	Name of the product	Attribute cannot contain NULL values
returnPeriod	No. of days after orderand till which the product can be returned. Can be different for different products according to business logic.	Attribute cannot contain NULL values
deptID	Department identification number	
RETURNS	**Entity class to track returns in an order**	
returnID	Return Identification number	Identifying Attribute
returnStatus	Status of the return request eg. "Initiated"and "In process"and "Declined"and "Cancelled"and "Completed"	Default is Initiated
reasonForReturn	Reason why the product was returned	Attribute cannot contain NULL values
requestDate	Date when the return request was made	Attribute cannot contain NULL values
returnedDate	Date when product was actually returned	Attribute cannot contain NULL values
refundStatus	Status of Refund eg 'Initiated'and 'Completed'and 'Failed'	Default is Initiated
serialNumber	Serial Number of the item	
orderID	Order Identification number	
REVIEW_DETAILS	**Entity Class to model Review Information**	
reviewID	Review identification number	Identifying Attribute
custID	Customer identification number	
description	The description of the review	Attribute cannot contain NULL values
productID	Product Identification number	

ratings	The ratings for the given product	Values are strictly 1,2,3,4, or 5
SERVICES	**Entity Class for storing Services**	
bookingID	Booking ID of that Service	Identifying Attribute
serviceID	Identification of Service Type	
custID	Customer identification number	
professionalID	ID of the servicing professional	
bookingDate	Date the customer booked the service	Attribute cannot contain NULL values
completionDate	Date the service was completed	
SERVICE_PROFESSIONALS	**Entity Class for storing people doing the servicing**	
professionalID	ID of the servicing professional	Identifying Attribute
fName	First Name of the servicing professional	Attribute cannot contain NULL values
lName	Last Name of the servicing professional	Attribute cannot contain NULL values
contactNo	Contact Number of the servicing professional	Length should be exactly 10 digits
companyName	Company that the servicing professional belongs to	Attribute cannot contain NULL values
SERVICE_TYPES	**Entity Class for storing types of services**	
serviceID	Identification of Service Type	Identifying Attribute
serviceName	Name of the Service	Attribute cannot contain NULL values
price	Price of the Service	Attribute cannot contain NULL values
SHIPPING_METHOD	**Entity class for storing information of all types of shipping available**	
shippingID	ID of Shipping Method table	Identifying Attribute
shippingPrice	The price for the type of shipping method	Attribute cannot contain NULL values
shippingType	The type of shipping eg."Flat rate"and "Express"	Attribute cannot contain NULL values
daysToDeliver	How many days will it take to deliver depending on type of shipping	
STORES	**Entity class to store information about the offline stores**	

storeID	Store Identification number	Identifying Attribute
addressLine1	Free text address field 1	Attribute cannot contain NULL values
addressLine2	Free text address field 2	
city	City the store is present in	Attribute cannot contain NULL values
phoneNo	Contact number of the store	Attribute cannot contain NULL values; length is 10 digits
state	State the store is present in	Attribute cannot contain NULL values
zipcode	Zipcode of the area the store is present in	Attribute cannot contain NULL values; Should be exactly 5 digits
STORE_HOURS	**Entity class to store information about the offline stores**	
dayOfWeek	Day of the week	Identifying Attribute
storeID	Store Identification number	Identifying Attribute
openTime	Time the store opens	Attribute cannot contain NULL values
closeTime	Time the store closes	Attribute cannot contain NULL values

Chapter 3: Relational Design

Summary of Tables

Relational Schema

Appendix: SQL statements to create tables and define constraints

```
--AGENTS
CREATE TABLE AGENTS(
agentID varchar(10),
agentFName varchar(100) NOT NULL,
agentLName varchar(100) NOT NULL,
contactNo char(10),
emailOfAgent varchar(100) NOT NULL,
numberOfTickets int DEFAULT 0,
constraint check_phone_agent check (regexp_like (contactNo, '[0-9]{10}')),
CONSTRAINT PK_Agent PRIMARY KEY (agentID)
);
```

```
--CLUB_MEMBER
CREATE TABLE CLUB_MEMBER(
custID varchar(10),
membershipStartDate DATE,
nextPaymentDueOn DATE,
CONSTRAINT PK_ClubMember PRIMARY KEY (custID),
CONSTRAINT FK_PK_ClubMember FOREIGN KEY (custID) REFERENCES CUSTOMERS
(custID) on delete cascade
);

--COUPONS
CREATE TABLE COUPONS(
couponCode varchar(20),
couponDescription varchar(100),
couponValidity DATE,
discount float,
CONSTRAINT PK_CouponCode PRIMARY KEY (couponCode)
);

--CUSTOMER_ADDRESS
CREATE TABLE CUSTOMER_ADDRESS(
addressID varchar(10),
custID varchar(10),
city varchar(100) not null,
country varchar(100) not null,
defaultAddress char(3) default 'No',
line1 varchar(100) not null,
line2 varchar(100),
state char(2) not null,
tag varchar(10),
zipcode char(5),
constraint cust_add_pkey primary key(addressID),
constraint cust_add_fkey foreign key(custID) references CUSTOMERS(custID) on
delete cascade,
constraint check_zip check (regexp_like (zipcode, '[0-9]{5}'))
);

--CUSTOMER_CARE
CREATE TABLE CUSTOMER_CARE(
custQueryID varchar(10),
description varchar(100),
queryType varchar(100),
querySubType varchar(100),
status varchar(50) DEFAULT 'Open',
custID varchar(10),
agentID varchar(10),
Constraint check_query CHECK (queryType IN ('Online experience','Store
experience', 'Products','Services')),
Constraint check_querysub CHECK (querySubType IN
('Orders','Replacement','Account','Other','Refund')),
Constraint check_status CHECK (status IN ('Open','Close','Pending')),
CONSTRAINT PK_CustomerCare PRIMARY KEY (custQueryID),
```

```
CONSTRAINT FK_CUSTOMERCARE_CUSTOMERS FOREIGN KEY (custID) REFERENCES
CUSTOMERS(custID) on delete cascade,
CONSTRAINT FK_CustomerCare_Agents FOREIGN KEY (agentID) REFERENCES
AGENTS(agentID) on delete set null
);

--CUSTOMERS
CREATE TABLE CUSTOMERS(
custID varchar(10),
fName varchar(100) not null,
lName varchar(100) not null,
phoneNo char(10) not null,
email varchar(100) not null unique,
password varchar(20) not null,
walletBalance number(8,2) default 0,
constraint cust_pkey primary key(custID),
constraint check_phone_cust check (regexp_like (phoneNo, '[0-9]{10}')),
constraint check_email_cust check (email like '%@%.%'),
constraint check_pwd_length check (length(password)>=8)
);

--DEALS
CREATE TABLE DEALS(
dealID varchar(10),
dealDescription varchar(100) NOT NULL,
discountStartDate DATE NOT NULL,
discountEndDate DATE NOT NULL,
discountPercentage number(5,2) NOT NULL,
isMember char(3),
CONSTRAINT check_isMember CHECK (isMember IN ('Yes', 'No')),
CONSTRAINT PK_Deals PRIMARY KEY (dealID)
);

--DEALS_ON_PRODUCTS
CREATE TABLE DEALS_ON_PRODUCTS(
dealID varchar(10),
productID varchar(10),
CONSTRAINT PK_DOP PRIMARY KEY (dealID, productID),
CONSTRAINT FK_DOP_DEALS FOREIGN KEY (dealID) REFERENCES DEALS(dealID) on
delete set null,
CONSTRAINT FK_DOP_PRODUCTS FOREIGN KEY (productID) REFERENCES
PRODUCTS(productID) on delete set null
);

--DEPARTMENTS
CREATE TABLE DEPARTMENTS(
deptID varchar(10),
category varchar(100) NOT NULL,
subCategory varchar(100) NOT NULL,
CONSTRAINT PK_DEPT PRIMARY KEY (deptID),
CONSTRAINT category_types CHECK (category IN ('Power Tools','Hand
Tools','Automotive','Welding','Plumbing','Electrical','Hardware','Material
Handling','Painting','Lighting','Safety','Home and Security',
```

```
'Lawn and Garden','Air Tools','Generators'))
);

--GIFT_CARDS
CREATE TABLE GIFT_CARDS(
serialNumber varchar(20),
giftCardCode char(16) UNIQUE,
amount number(3) NOT NULL,
isActivated char(3) DEFAULT 'No',
CONSTRAINT  check_Activation CHECK (isActivated IN ('Yes','No')),
constraint check_giftcard_length check (length(giftCardCode )=16),
Constraint amount_types CHECK (amount IN ('25','50','75','100')),
CONSTRAINT PK_GIFT_CARDS PRIMARY KEY (serialNumber),
CONSTRAINT FK_GIFT_CARDS FOREIGN KEY (serialNumber) REFERENCES
ITEMS(serialNumber) on delete cascade
);

--GIFT_CARDS_ACTIVATION
CREATE TABLE GIFT_CARD_ACTIVATION(
serialNumber varchar(20),
custID varchar(10),
dateOfActivation DATE NOT NULL,
CONSTRAINT PK_GIFT_CARDS_ACTIVATION PRIMARY KEY (serialNumber , custID),
CONSTRAINT FK_GIFT_CARDS_ACTIVATION_GIFT_CARDS FOREIGN KEY (serialNumber)
REFERENCES GIFT_CARDS(serialNumber) on delete cascade,
CONSTRAINT FK_GIFT_CARDS_ACTIVATION_CUSTOMERS FOREIGN KEY (custID) REFERENCES
CUSTOMERS(custID) on delete cascade
);


--HFCC
CREATE TABLE HFCC(
limit number(6) NOT NULL,
creditCardNo char(16),
custID varchar(10),
CVV char(3),
validTill DATE NOT NULL,
appliedOn DATE NOT NULL,
constraint creditcard_length check (length(creditCardNo)=16),
constraint cvv_length check (length(CVV )=3),
CONSTRAINT PK_HFCC PRIMARY KEY (creditCardNo),
CONSTRAINT FK_HFCC FOREIGN KEY (custID) REFERENCES CUSTOMERS(custID) on
delete cascade
);

--ITEMS
CREATE TABLE ITEMS(
serialNumber varchar(20),
availability char(3),
manufacturingDate DATE,
productID varchar(10),
storeID varchar(10),
```

```
CONSTRAINT check_isAvailable CHECK (availability IN ('Yes', 'No')),
CONSTRAINT PK_ITEMS PRIMARY KEY (serialNumber),
CONSTRAINT FK_ITEMS_PRODUCTS FOREIGN KEY (productID) REFERENCES
PRODUCTS(productID) on delete cascade,
CONSTRAINT FK_ITEMS_STORES FOREIGN KEY (storeID) REFERENCES STORES(storeID)
on delete cascade
);


--ITEMS_IN_ORDERS
create table ITEMS_IN_ORDERS
(
orderID varchar(50),
serialNumber varchar(20),
CONSTRAINT PK_ITEMS_ON_ORDERS PRIMARY KEY (orderID,serialNumber),
CONSTRAINT FK_ITEMS_ON_ORDERS_ORDERS FOREIGN KEY (orderID) REFERENCES
ORDERS(orderID) on delete cascade,
CONSTRAINT FK_ITEMS_ON_ORDERS_ITEMS FOREIGN KEY (serialNumber) REFERENCES
ITEMS(serialNumber) on delete set null
);

--ORDERS
CREATE TABLE ORDERS(
orderID varchar(50),
orderDate DATE NOT NULL,
orderStatus varchar(20) NOT NULL,
shippingDate DATE,
estimatedDeliveryDate DATE NOT NULL,
actualDeliveryDate DATE,
paymentMode varchar(20),
couponDiscount decimal(8,2),
subtotal decimal(8,2) NOT NULL,
grandTotal decimal(8,2) NOT NULL,
couponID varchar(20),
custID varchar(10),
shippingID varchar(10),
CONSTRAINT PK_ORDERS PRIMARY KEY (orderID),
CONSTRAINT FK_ORDERS_COUPONS FOREIGN KEY (couponID) REFERENCES
COUPONS(couponID) on delete set null,
CONSTRAINT FK_ORDERS_CUSTOMERS FOREIGN KEY (custID) REFERENCES
CUSTOMERS(custID) on delete cascade,
CONSTRAINT FK_ORDERS_SHIPPING FOREIGN KEY (shippingID) REFERENCES
SHIPPING_METHOD(shippingID) on delete set null
);


--PRODUCTS
CREATE TABLE PRODUCTS(
productID varchar(10),
productOverview varchar(1000) NOT NULL,
pricePerUnit FLOAT NOT NULL,
productName varchar(100) NOT NULL,
returnPeriod int NOT NULL,
```

```
reviewID varchar(10),
deptID varchar(10),
CONSTRAINT PK_PRODUCTS PRIMARY KEY (productID),
CONSTRAINT FK_RID_PRODUCTS FOREIGN KEY (reviewID) REFERENCES
REVIEW_DETAILS(reviewID) on delete set null,
CONSTRAINT FK_DID_PRODUCTS FOREIGN KEY (deptID) references
DEPARTMENTS(deptID) on delete set null
);

--RETURNS
CREATE TABLE RETURNS(
returnID varchar(10),
returnStatus varchar(20) DEFAULT 'initiated',
reasonForReturn varchar(150) NOT NULL,
requestDate DATE NOT NULL,
returnedDate DATE NOT NULL,
refundStatus varchar(10) DEFAULT 'initiated',
serialNumber varchar(20),
orderID varchar(20),
Constraint check_returnstatus CHECK (returnStatus IN ('Initiated','In
process', 'Completed','Declined','Cancelled')),
Constraint check_refundstatus CHECK (refundStatus IN ('Initiated','In
process', 'Completed','Declined','Cancelled')),
CONSTRAINT PK_RETURNS PRIMARY KEY (returnID),
CONSTRAINT FK_RETURNS_ORDERS FOREIGN KEY (orderID) REFERENCES ORDERS(orderID)
on delete set null,
CONSTRAINT FK_RETURNS_ITEMS FOREIGN KEY (serialNumber) REFERENCES
ITEMS(serialNumber) on delete set null
);

--REVIEW_DETAILS
CREATE TABLE REVIEW_DETAILS(
custID varchar(10),
description varchar(2000) NOT NULL,
productID varchar(10),
ratings int,
reviewID varchar(10),
CONSTRAINT ratingsLimit CHECK (ratings BETWEEN 1 and 5),
CONSTRAINT PK_REVIEW_DETAILS PRIMARY KEY (reviewID),
CONSTRAINT FK_REVIEW_DETAILS_CUSTOMERS FOREIGN KEY (custID) REFERENCES
CUSTOMERS(custID) on delete cascade,
CONSTRAINT FK_REVIEW_DETAILS_PRODUCTS FOREIGN KEY (productID) REFERENCES
PRODUCTS(productID) on delete cascade
);

-- SERVICE_PROFESSIONALS
CREATE TABLE SERVICE_PROFESSIONALS(
professionalID varchar(10),
fName varchar(100) NOT NULL,
lName varchar(100) NOT NULL,
contactNo varchar(10),
companyName varchar(100) NOT NULL,
constraint check_phone_service_professionals check (regexp_like (contactNo,
```

```
'[0-9]{10}')),
CONSTRAINT PK_SERVICE_PROFESSIONALS PRIMARY KEY (professionalID)
);


-- SERVICE_TYPES
CREATE TABLE SERVICE_TYPES(
serviceID varchar(10),
serviceName varchar(100) NOT NULL,
price number(5,2) NOT NULL,
CONSTRAINT checkTypes CHECK (serviceName  IN ('Repair', 'Routine
Maintainance', 'Diagnostics', 'Oiling', 'Replacement of Parts',
'Fabrication')),
CONSTRAINT PK_SERVICE_TYPES PRIMARY KEY (serviceID)
);


-- SERVICES
CREATE TABLE SERVICES(
custID varchar(10),
professionalID varchar(10),
serviceID varchar(10),
bookingID varchar(10),
bookingDate DATE NOT NULL,
completionDate DATE,
CONSTRAINT PK_SERVICES PRIMARY KEY (bookingID),
CONSTRAINT FK_SERVICES_CUSTOMERS FOREIGN KEY (custID) REFERENCES
CUSTOMERS(custID) on delete set null,
CONSTRAINT FK_SERVICES_SERVICEPROFESSIONAL FOREIGN KEY (professionalID)
REFERENCES SERVICE_PROFESSIONALS(professionalID) on delete set null,
CONSTRAINT FK_SERVICE_SERVICESTYPES FOREIGN KEY (serviceID) REFERENCES
SERVICE_TYPES(serviceID) on delete set null
);


-- SHIPPING_METHOD
CREATE TABLE SHIPPING_METHOD(
shippingID varchar(10),
shippingPrice number(5,2) NOT NULL,
shippingType varchar(50) NOT NULL,
CONSTRAINT check_shippingType CHECK (shippingType IN ('Flat Rate', 'Express',
'Truck')),
CONSTRAINT PK_SHIPPING_METHOD PRIMARY KEY (shippingID)
);


-- STORE_HOURS
CREATE TABLE STORE_HOURS(
dayOfWeek varchar(20),
openTime TIMESTAMP WITH TIME ZONE NOT NULL,
closeTime TIMESTAMP WITH TIME ZONE NOT NULL,
storeID varchar(10),
CONSTRAINT PK_STORES_HOURS PRIMARY KEY (dayOfWeek, storeID),
```

```
CONSTRAINT FK_STORES_HOURS FOREIGN KEY (storeID) REFERENCES STORES(storeID)
on delete cascade
);


-- STORES
CREATE TABLE STORES(
addressLine1 varchar(100) NOT NULL,
addressLine2 varchar(100),
city varchar(100) NOT NULL,
phoneNo char(10) NOT NULL,
state char(2) NOT NULL,
storeID varchar(10),
zipcode varchar(5) NOT NULL,
Constraint check_state2 CHECK (state IN
('AL','AK','AZ','AR','CA','CO','CT','DE','FL','GA','HI','ID','IL','IN','IA','
KS','KY','LA','ME','MD','MA','MI','MN','MS','MO','MT','NE','NV','NH','NJ','NM
','NY','NC','ND','OH','OK','OR','PA','RI','SC','SD','TN','TX','UT','VT','VA',
'WA','WV','WI','WY')),
constraint check_zip_stores check (regexp_like (zipcode, '[0-9]{5}')),
constraint zip_length check (length(zipcode)=5),
CONSTRAINT PK_STORES PRIMARY KEY (storeID)
);
```

Appendix: SQL Statements for Sequences and Triggers for ID Generation

```
--Address ID
CREATE SEQUENCE addressid_sequence
MINVALUE 1
MAXVALUE 2000000;

CREATE OR REPLACE TRIGGER gen_address_id
  BEFORE INSERT
  ON CUSTOMER_ADDRESS
  FOR EACH ROW
BEGIN
      :new.addressID :=  to_char(addressid_sequence.NEXTVAL);
END;

--Agent_ID
CREATE SEQUENCE agentID2_sequence
MINVALUE 101
MAXVALUE 999;

create or replace TRIGGER gen_agent_id
  BEFORE INSERT
  ON AGENTS
  FOR EACH ROW
BEGIN
      :new.agentID :=  to_char(agentID2_sequence.NEXTVAL);
```

```
END;

--Coupon_ID
CREATE SEQUENCE couponid_sequence
MINVALUE 1
MAXVALUE 10000;

create or replace TRIGGER gen_coupon_id
  BEFORE INSERT
  ON COUPONS
  FOR EACH ROW
BEGIN
      :new.couponID :=  to_char(couponid_sequence.NEXTVAL);
END;

--Cust_ID
CREATE SEQUENCE custid_sequence
MINVALUE 100001
MAXVALUE 999999;

create or replace TRIGGER gen_cust_id
  BEFORE INSERT
  ON CUSTOMERS
  FOR EACH ROW
BEGIN
      :new.custID :=  to_char(custid_sequence.NEXTVAL);
END;

--Customercare_ID
CREATE SEQUENCE customercareid_sequence
MINVALUE 1
MAXVALUE 999999;

create or replace TRIGGER gen_customercare_id
  BEFORE INSERT
  ON CUSTOMER_CARE
  FOR EACH ROW
BEGIN
      :new.CUSTQUERYID :=  to_char(customercareid_sequence.NEXTVAL);
END;

--Department_ID
CREATE SEQUENCE deptid_sequence
MINVALUE 101
MAXVALUE 999;

create or replace TRIGGER gen_department_id
  BEFORE INSERT
  ON DEPARTMENTS
  FOR EACH ROW
BEGIN
      :new.deptID :=  to_char(deptid_sequence.NEXTVAL);
```

```
END;

--HFCC Number
CREATE SEQUENCE HFCCnumber_sequence
MINVALUE 1000000000000000
MAXVALUE 9999999999999999;

CREATE OR REPLACE TRIGGER hfcc_num_id
  BEFORE INSERT
  ON HFCC
  FOR EACH ROW
BEGIN
      :new.creditcardno :=  to_char(HFCCnumber_sequence.NEXTVAL);
END;

--Order_ID
CREATE SEQUENCE productid_sequence
MINVALUE 1
MAXVALUE 10000000000000;

create or replace TRIGGER gen_order_id
  BEFORE INSERT
  ON ORDERS
  FOR EACH ROW
BEGIN
      :new.orderID :=  to_char(orderID_sequence.NEXTVAL);
END;

--Product_ID
CREATE SEQUENCE productid_sequence
MINVALUE 1001
MAXVALUE 9999;

create or replace TRIGGER gen_product_id
  BEFORE INSERT
  ON PRODUCTS
  FOR EACH ROW
BEGIN
      :new.productID :=  to_char(productid_sequence.NEXTVAL);
END;

--Service_ID
CREATE SEQUENCE serviceid_sequence
MINVALUE 1
MAXVALUE 999999;

create or replace TRIGGER gen_service_id
  BEFORE INSERT
  ON SERVICES
  FOR EACH ROW
BEGIN
      :new.bookingID :=  to_char(serviceid_sequence.NEXTVAL);
```

```
END;

--ServiceProfessionalID
CREATE SEQUENCE serviceprofessionalid_sequence
MINVALUE 1
MAXVALUE 1000;

create or replace TRIGGER gen_serviceprofessional_id
  BEFORE INSERT
  ON SERVICE_PROFESSIONALS
  FOR EACH ROW
BEGIN
      :new.professionalID :=
to_char(serviceprofessionalid_sequence.NEXTVAL);
END;

--ServiceTypes_ID
CREATE SEQUENCE servicetypesid_sequence
MINVALUE 1
MAXVALUE 100;

create or replace TRIGGER gen_servicetypes_id
  BEFORE INSERT
  ON SERVICE_TYPES
  FOR EACH ROW
BEGIN
      :new.serviceID :=  to_char(servicetypesid_sequence.NEXTVAL);
END;

--ShippingMethod_ID
CREATE SEQUENCE shippingmethodid_sequence
MINVALUE 1
MAXVALUE 10;

create or replace TRIGGER gen_shippingmethod_id
  BEFORE INSERT
  ON SHIPPING_METHOD
  FOR EACH ROW
BEGIN
      :new.ShippingID :=  to_char(shippingmethodid_sequence.NEXTVAL);
END;

--Store_ID
CREATE SEQUENCE storeid_sequence
MINVALUE 1
MAXVALUE 9999;

create or replace TRIGGER gen_store_id
  BEFORE INSERT
  ON STORES
  FOR EACH ROW
BEGIN
```

```
      :new.storeID :=  to_char(storeid_sequence.NEXTVAL);
END;
```

Chapter 4: Queries

Query 1: Frequently purchased together

This code displays the top 3 products that were purchased along with a particular product in the past orders. It also displays the number of times the second product was ordered along with the first product. This could help the customers locate related products easily, and help the business sell more products in a single order.

```
WITH
order_products AS(
    SELECT orderid, p.productname
    FROM  items_in_orders io
    JOIN items i ON io.serialnumber=i.serialnumber
    JOIN products p ON i.productid=p.productid
),
ordered_together AS(
    SELECT a.productname, b.productname AS second_product, COUNT(*) AS
times_ordered_together,
    row_number() OVER(PARTITION BY a.productname ORDER BY COUNT(*) DESC) as
rn
    FROM order_products a
    JOIN order_products b ON a.orderid=b.orderid
    WHERE a.productname<>b.productname
    GROUP BY a.productname, b.productname
)
SELECT productname, second_product, times_ordered_together, rn AS
rank_of_second_product
FROM ordered_together
WHERE rn<=3;
```

PRODUCTNAME	SECOND_PRODUCT	TIMES_ORDERED_TOGETHER	RANK_OF_SECOND_PRODUCT
Garden Fork	Shovel	4	1
Hose	Nozzle	5	1
Hose	Hose Mender	3	2
Hose	Leaf Rake	1	3
Hose Mender	Hose	3	1

Query 2: Top deals on best selling products

This query finds the top deals on best selling products. Best selling products are described as the products that are sold more times than the

average number of times a product is sold. This helps customers find the best deals.

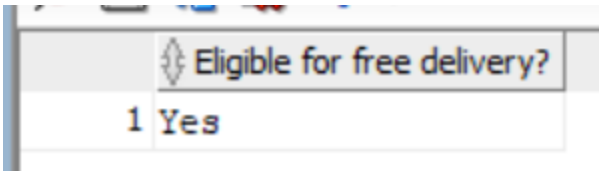
```
WITH
average_times_ordered AS(
    SELECT ROUND(AVG(TotalNumberOfTimesOrdered),2) AS
AverageNumberOfTimesOrdered
    FROM (
    SELECT p.productid, COUNT(*)AS TotalNumberOfTimesOrdered
    FROM items_in_orders io
    JOIN items i ON io.serialnumber = i.serialnumber
    JOIN products p ON p.productid = i.productid
    GROUP BY p.productid)t
    ),
best_selling_products AS(
    SELECT p.productid, p.productname, p.priceperunit, COUNT(*) AS
NumberOfTimesOrdered
    FROM items_in_orders io
    JOIN items i ON io.serialnumber = i.serialnumber
    JOIN products p ON p.productid = i.productid
    HAVING COUNT(*) >  (SELECT AverageNumberOfTimesOrdered FROM
average_times_ordered)
    GROUP BY p.productid, p.productname, p.priceperunit
    )
SELECT * FROM (
    SELECT bsp.productname, d.discountPercentage,
(bsp.priceperunit*d.discountPercentage)/100 AS discountValue,
    row_number() OVER(ORDER BY (bsp.priceperunit*d.discountPercentage)/100
DESC) AS rank_of_values
    FROM deals_on_products dop
    JOIN best_selling_products bsp ON dop.productid = bsp.productid
    JOIN deals d ON dop.dealid = d.dealid
    )
WHERE rank_of_values<=3;
```

PRODUCTNAME	DISCOUNTPERCENTAGE	DISCOUNTVALUE	RANK_OF_VALUES
Shovel	10	2.5	1
Nozzle with Soap Dispenser	10	1.5	2
Spray Gun	10	1.5	3

Query 3: Eligible for free delivery

The code displays if a product is eligible for free delivery for a particular customer. The product is eligible for free delivery, if there are items available for that product in a store in the same pincode as the customer. This helps customers easily find products that are eligible for free delivery and they are more likely to order those products.

```
SELECT CASE WHEN avail>0 THEN 'Yes' ELSE 'No' END AS "Eligible for free
delivery?" FROM (
    SELECT COUNT(*) AS avail
    FROM customer_address ca
    JOIN stores s ON ca.zipcode=s.zipcode
    JOIN items i ON s.storeid=i.storeid
    JOIN products p ON i.productid=p.productid
    WHERE ca.custid='100007' AND availability=1 AND p.productid='1006'
) ;
```



	Eligible for free delivery?
1	Yes

Query 4: Total money saved through deals and coupons

The code displays the total money saved by the customer through all deals or coupons used by him in the past orders.This promotes brand loyalty and encourages repeat orders from customers.

```
WITH discount AS(
    SELECT custid, SUM((p.priceperunit*d.discountPercentage)/100) AS
discountValue
    FROM orders o
    JOIN items_in_orders io ON io.orderid = o.orderid
    JOIN items i ON i.serialnumber = io.serialnumber
    JOIN products p ON p.productid = i.productid
    JOIN deals_on_products dop ON dop.productid = p.productid
    JOIN deals d ON dop.dealid = d.dealid
    GROUP BY custid),
coupon AS(
    SELECT custid, SUM(coupondiscount) AS couponDiscount
    FROM orders
    GROUP BY custid)
SELECT discount.custid, discount.discountValue + coupon.couponDiscount AS
TotalMoneySaved
FROM discount JOIN coupon ON discount.custid = coupon.custid;
```

	CUSTID	TOTALMONEYSAVED
1	100009	9.9
2	100006	40.9
3	100012	19.3
4	100011	21.8
5	100015	22.05
6	100018	24.3

Query 5: Low stock alert

The code displays the products for which there are 5 or less items left in stock. This is used to display an alert on the website for products which are low on stock. It helps customers buy a product that they don't wnat to miss.

```
SELECT p.productid, COUNT(*)
FROM products p
JOIN items i ON p.productid=i.productid
WHERE availability=1
GROUP BY p.productid
HAVING COUNT(*)<=5;
```

	PRODUCTID	Items Left
1	1084	4
2	1086	1
3	1027	1
4	1083	3
5	1085	5
6	1004	4

Query 6: Top products of the season

This code is used to display the top products purchased in the current season. It will check the top products sold in the same season in the past years.It helps customers easily find products that are sold in a season. Eg. Gardening related products are sold relatively frequently in summer.

```
WITH display_current_season As (
    SELECT CASE
    WHEN EXTRACT(MONTH FROM current_date) IN (6,7,8) THEN 'Summer'
    WHEN EXTRACT(MONTH FROM current_date) IN (9,10,11) THEN 'Fall'
    WHEN EXTRACT(MONTH FROM current_date) IN (12,1,2) THEN 'Winter'
    WHEN EXTRACT(MONTH FROM current_date) IN (3,4,5) THEN 'Spring' END AS
current_season
    FROM dual
),
order_season AS (
    SELECT o.orderid, productid, CASE
    WHEN EXTRACT(MONTH FROM orderdate) IN (6,7,8) THEN 'Summer'
    WHEN EXTRACT(MONTH FROM orderdate) IN (9,10,11) THEN 'Fall'
    WHEN EXTRACT(MONTH FROM orderdate) IN (12,1,2) THEN 'Winter'
    WHEN EXTRACT(MONTH FROM orderdate) IN (3,4,5) THEN 'Spring' END AS
order_season
    FROM orders o
```

```
    JOIN items_in_orders io ON o.orderid=io.orderid
    JOIN items i ON io.serialnumber=i.serialnumber
),
display_season_products AS (
    SELECT current_season, productid, COUNT(DISTINCT orderid) AS
total_orders,
    row_number() OVER(PARTITION BY current_season ORDER BY COUNT(DISTINCT
orderid) DESC) AS rank_of_product
    FROM display_current_season dcs
    JOIN order_season os ON dcs.current_season=os.order_season
    GROUP BY current_season, productid
)
SELECT * FROM display_season_products WHERE rank_of_product<=3;
```

	CURRENT_SEASON	PRODUCTID	TOTAL_ORDERS	RANK_OF_PRODUCT
1	Winter	1004	5	1
2	Winter	1006	2	2
3	Winter	1001	2	3

Query 7: Average ratings for product

The code is used to display an average of the ratings given to a product by customers who have purchased it in the past. The average will only be displayed if there are a minimum of 5 reviews available for that product.It helps the customer get an idea about the quality of the product.

```
SELECT rd.productid, ROUND(AVG(ratings),2) AS avg_rating
FROM review_details rd
JOIN orders o ON rd.custid=o.custid
JOIN items_in_orders io ON o.orderid=io.orderid
JOIN items i ON io.serialnumber=i.serialnumber
GROUP BY rd.productid
HAVING COUNT(DISTINCT reviewid)>=5;
```

	PRODUCTID	AVG_RATING
1	1004	3.99
2	1003	3.88

Query 8: Average query resolution time

The code displays the average query resolution time for the agent assigned to his query. This can be displayed to the customer for him to know when his query will be resolved. It helps in setting the right expectations to customers in terms of when their query would be resolved.

```
SELECT cc.agentid, agentfname,
COALESCE(ROUND(AVG((resolutiondate-querydate)*24)),0) AS
ResolutionTimeinHours
FROM customer_care cc
JOIN agents a ON a.agentid = cc.agentid
WHERE custid='100007' AND cc.agentid='104'
GROUP BY cc.agentid, agentfname;
```

	AGENTID	AGENTFNAME	RESOLUTIONTIMEINHOURS
1	104	Rebecca	45

Query 9: Products eligible for return

The code displays the items ordered by the customer which are still eligible to be returned. Each product has a specific return period (can be 7 days for some products, 14 or 28 for some and so on). We also display the days remaining to request a return for the item. We also check that item has not been returned already.

```
SELECT o.orderid, p.productid, p.productname, i.serialnumber ,
ROUND((current_date - actualdeliverydate),0) as return_period_ending_in
FROM orders o
JOIN items_in_orders io ON o.orderid=io.orderid
JOIN items i ON io.serialnumber=i.serialnumber
JOIN products p ON i.productid=p.productid
WHERE (current_date - actualdeliverydate)<=returnperiod
AND orderstatus ='Delivered' AND custid='100007'
AND NOT EXISTS
(
    SELECT orderid, serialnumber
    FROM returns r
    WHERE o.orderid=r.orderid AND io.serialnumber=r.serialnumber
);
```

	ORDERID	PRODUCTID	PRODUCTNAME	SERIALNUMBER	RETURN_PERIOD_ENDING_IN
1	17	1009	Leaf Rake	10080	13
2	19	1007	Leaf Blowers	10103	13
3	17	1006	Shovel	10097	13
4	19	1006	Shovel	10100	13

Query 10: Recommend next best product

The code is used to recommend the next best products that can be purchased by

a customer based on the products purchased in his last order. The next best products are determined by checking what products were purchased in the next order by other customers in the past.

```
WITH all_orders AS (
    SELECT o.orderid, custid, orderdate, productname
    FROM orders o
    JOIN items_in_orders io ON o.orderid=io.orderid
    JOIN items i ON io.serialnumber=i.serialnumber
    JOIN products p ON i.productid=p.productid
),
next_best_product AS (
    SELECT custid, productname , orderid,
    LEAD(productname) OVER(PARTITION BY custid ORDER BY ORDERDATE) AS "Next
Ordered",
    LEAD(orderid) OVER(PARTITION BY custid ORDER BY ORDERDATE) AS "Next Order
ID"
    FROM all_orders
)
SELECT productname, "Next Ordered" , count(*)
FROM next_best_product
WHERE "Next Ordered" IS NOT NULL AND orderid<>"Next Order ID"
AND productname IN
(
    SELECT productname
    FROM ORDERS o
    JOIN items_in_orders io ON o.orderid=io.orderid
    JOIN items i ON io.serialnumber=i.serialnumber
    JOIN products p ON i.productid=p.productid
    WHERE custid='100018' AND
    orderdate =
        (SELECT MAX(orderdate) FROM orders where custid='100018')
)
GROUP BY productname, "Next Ordered"
ORDER BY COUNT(*) DESC
FETCH FIRST 3 ROWS ONLY
;
```

	PRODUCTNAME	Next Ordered	Number of Times Ordered
1	Long Drum Winch	Nozzle with Soap Dispenser	2
2	Long Drum Winch	Greenhouse	1
3	Long Drum Winch	Lithium-Ion Battery	1

Chapter 5: Triggers and Procedures

Trigger 1: Update number of tickets for agent

This trigger will increment the number of tickets handled by the agent after a customer query is assigned to him. This will help in keeping a track of the performance of the agent.

```
CREATE OR REPLACE TRIGGER update_numAgentTickets
AFTER UPDATE OF AGENTID
ON CUSTOMER_CARE
DECLARE
CURSOR numTicketsCursor IS
  SELECT agentid, count(*) as numTickets
    FROM CUSTOMER_CARE cc
  GROUP BY agentid;
BEGIN
    FOR updateNTCounter IN numTicketsCursor LOOP
       UPDATE AGENTS
       SET numberOfTickets = updateNTCounter.numTickets
       WHERE agentid = updateNTCounter.agentid;
    END LOOP;
END;
/
```

Trigger 2: Update wallet balance of customer

This trigger will update the wallet balance of the customer after he activates a valid gift card from his account. This will be updated only if the gift card has not been activated already.

```
CREATE OR REPLACE TRIGGER update_walletBalance
BEFORE INSERT
ON GIFT_CARD_ACTIVATION
FOR EACH ROW
DECLARE
    gc_amount GIFT_CARDS.amount%type;
    gc_isactivated GIFT_CARDS.isactivated%type;
BEGIN
       SELECT amount, isactivated
       INTO gc_amount, gc_isactivated
       FROM GIFT_CARDS
       WHERE serialnumber=:new.serialnumber;

       IF (gc_isactivated='No') THEN
           UPDATE CUSTOMERS
           SET walletBalance = walletBalance + gc_amount
           WHERE custid = :new.custid;

           UPDATE GIFT_CARDS
           SET isactivated='Yes'
           WHERE serialnumber = :new.serialnumber;
```

```
       ELSE
            raise_application_error(-20008, 'Error: Gift card already
activated!');
       END IF;
END;
/
```

Procedure 1: Smart assignment of customer agents

This procedure will assign an agent smartly to a customer query. In the future, the priority of the query will be calculated using machine learning algorithms, by mining the text in the query. In this project, we are assigning a priority randomly to the query. The queries with high priority will be handled by agents who have demonstrated high performance in the past (based on the query resolution time), i.e. the quickest agents will be assigned to the high priority queries. For all queries, the agent with the lowest open ticket count will be assigned.

```
CREATE OR REPLACE PROCEDURE update_AgentID
    (p_custqueryid IN CUSTOMER_CARE.custqueryid%type) AS

queryAgent AGENTS.agentid%type;
randomPriority number;
CURSOR queryCursor IS
SELECT custqueryid FROM CUSTOMER_CARE;
BEGIN
     --assigning random priority to queries
     SELECT round(dbms_random.value(1,3))
     INTO randomPriority
     FROM dual;

     UPDATE CUSTOMER_CARE
     SET queryPriority=
        CASE WHEN randomPriority=1 THEN 'Low'
        WHEN randomPriority=2 THEN 'Medium'
        WHEN randomPriority=3 THEN 'High'
        END
        WHERE custqueryid=p_custqueryid;

     IF randomPriority=3 THEN
     SELECT agentid
     INTO queryAgent
     FROM
     (
       SELECT a.agentid, COUNT(CASE WHEN status='Open' THEN 1 END) AS
openTickets,
       row_number() OVER(ORDER BY COUNT(CASE WHEN status='Open' THEN 1 END))
AS lowestOpenTickets
       FROM AGENTS a LEFT JOIN CUSTOMER_CARE cc ON a.agentid=cc.agentid
       WHERE a.agentid IN (
         SELECT agentid
```

```
         FROM CUSTOMER_CARE
         GROUP BY agentid
         HAVING AVG(RESOLUTIONDATE-QUERYDATE)<=(SELECT AVG(RESOLUTIONDATE-
QUERYDATE) FROM CUSTOMER_CARE))
         GROUP BY a.agentid
     ) WHERE lowestOpenTickets=1;

     ELSE
     SELECT agentid
     INTO queryAgent
     FROM
     (
         SELECT a.agentid, COUNT(CASE WHEN status='Open' THEN 1 END) AS
openTickets,
         row_number() OVER(ORDER BY COUNT(CASE WHEN status='Open' THEN 1
END)) AS lowestOpenTickets
         FROM AGENTS a LEFT JOIN CUSTOMER_CARE cc ON a.agentid=cc.agentid
         GROUP BY a.agentid
     ) WHERE lowestOpenTickets=1;

     END IF;

     UPDATE CUSTOMER_CARE
     SET agentid = queryAgent
     WHERE custqueryid=p_custqueryid;
END;
/
```

Chapter 6: User Interface

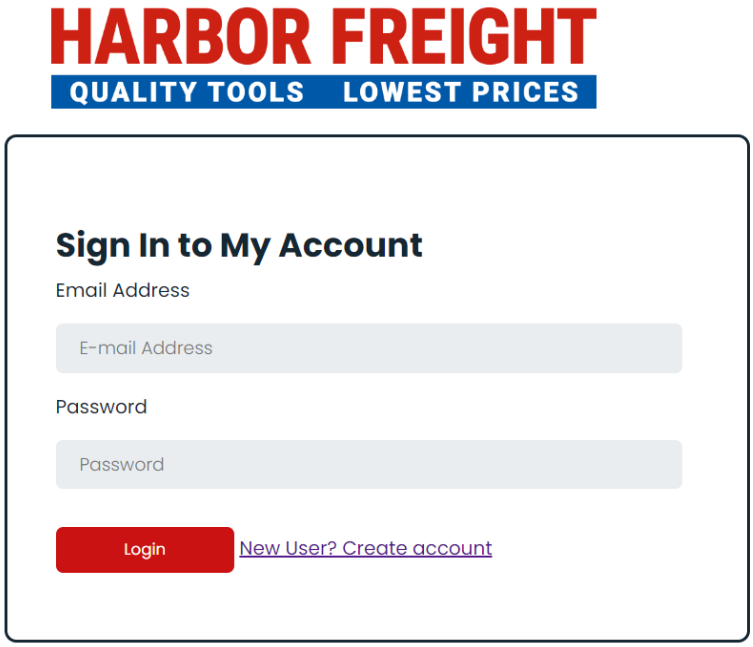
Link: http://ec2-35-89-31-2.us-west-2.compute.amazonaws.com/harbour/index.php
Use following credentials:

1. Email: mitchell.bush@gmail.com
   Password: qw69B56W

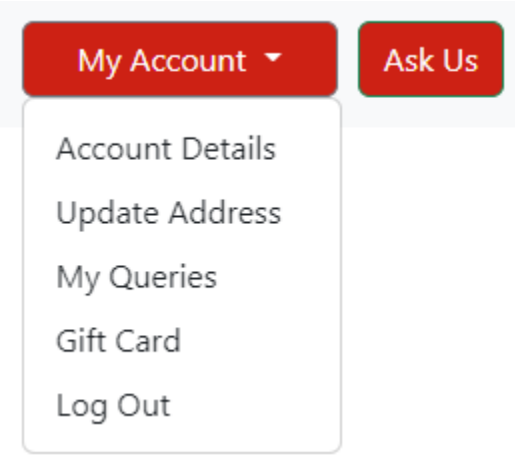
2. Email: Anusha@gmail.com
   Password: Anusha123

Login Page

The customer can login to his account using his registered email id and corresponding password.

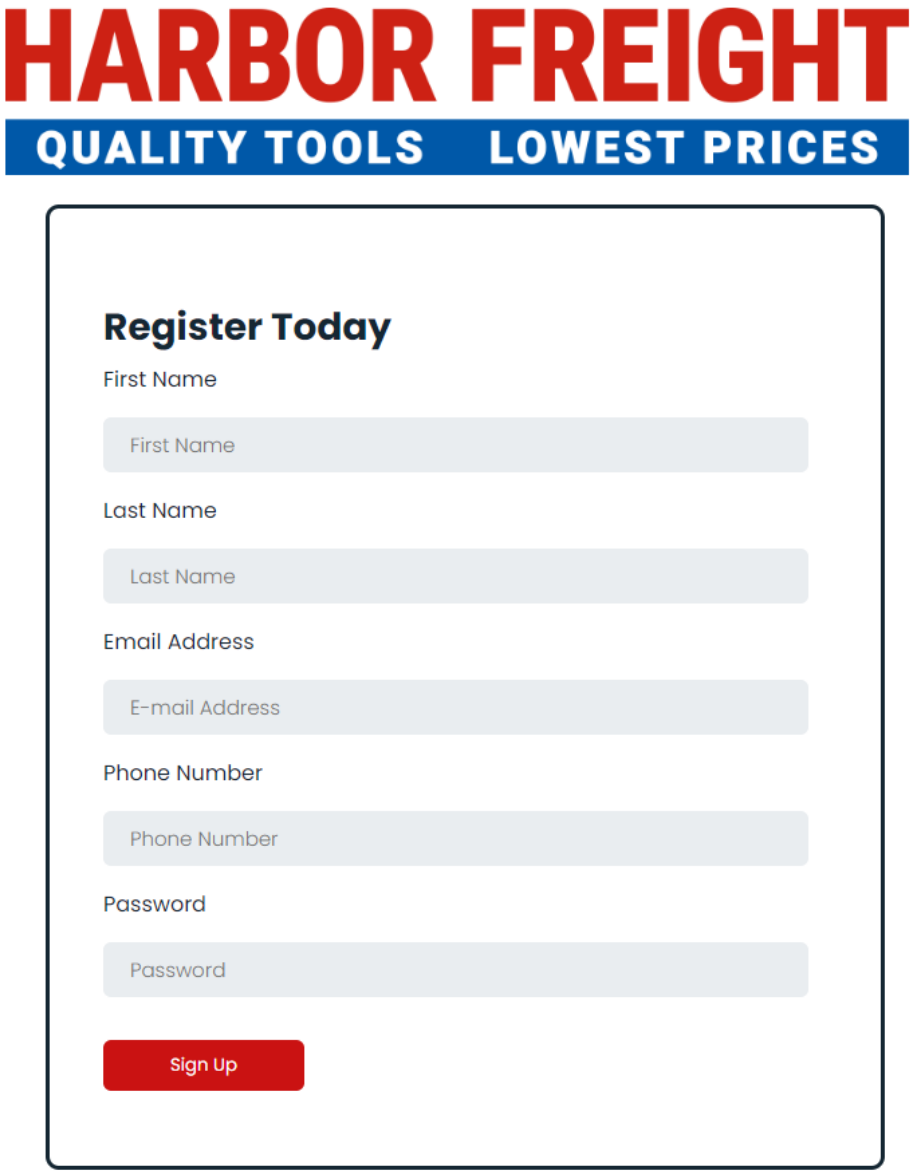


Navigation bar



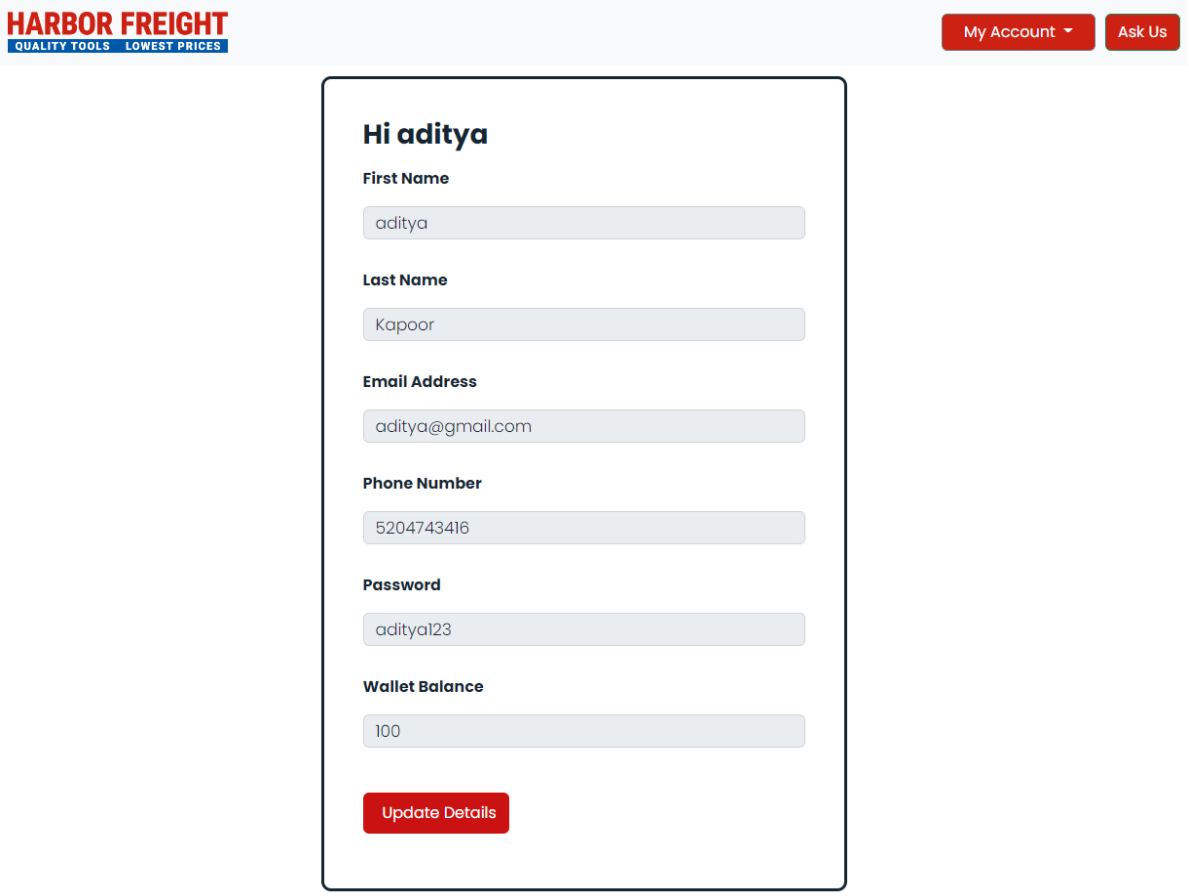
Create account page

If the customer does not already have an account, he can create a new account for himself by clicking on the ‘New User? Create account’ button from the login page.

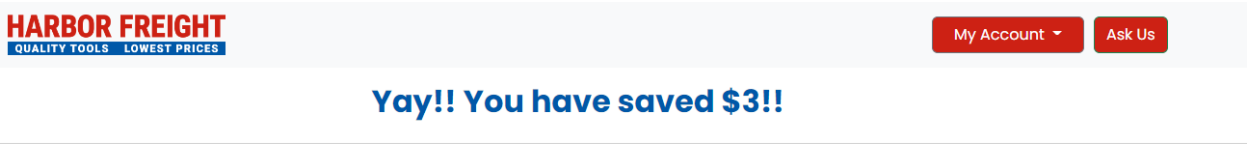


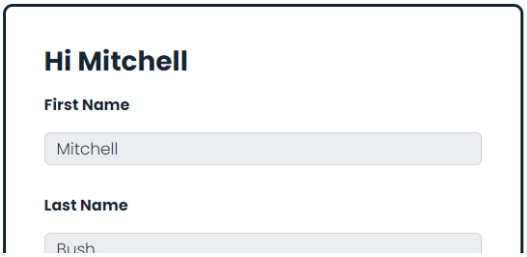
Account details page

The customer can view his current account details by going to ‘My Account’ and then selecting ‘Account Details’ from the dropdown. The customer can also update his account details by clicking on the ‘Update Details’ button.



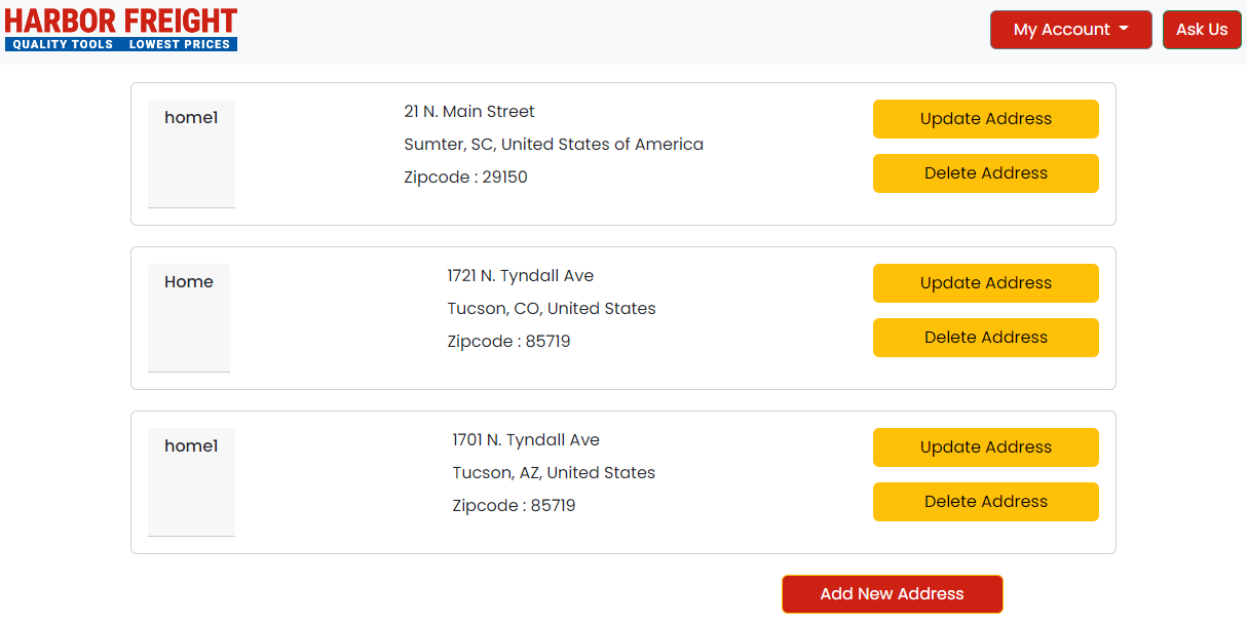
In this page, we also display the total money customer has saved throughout their journey with Harbor Freight.



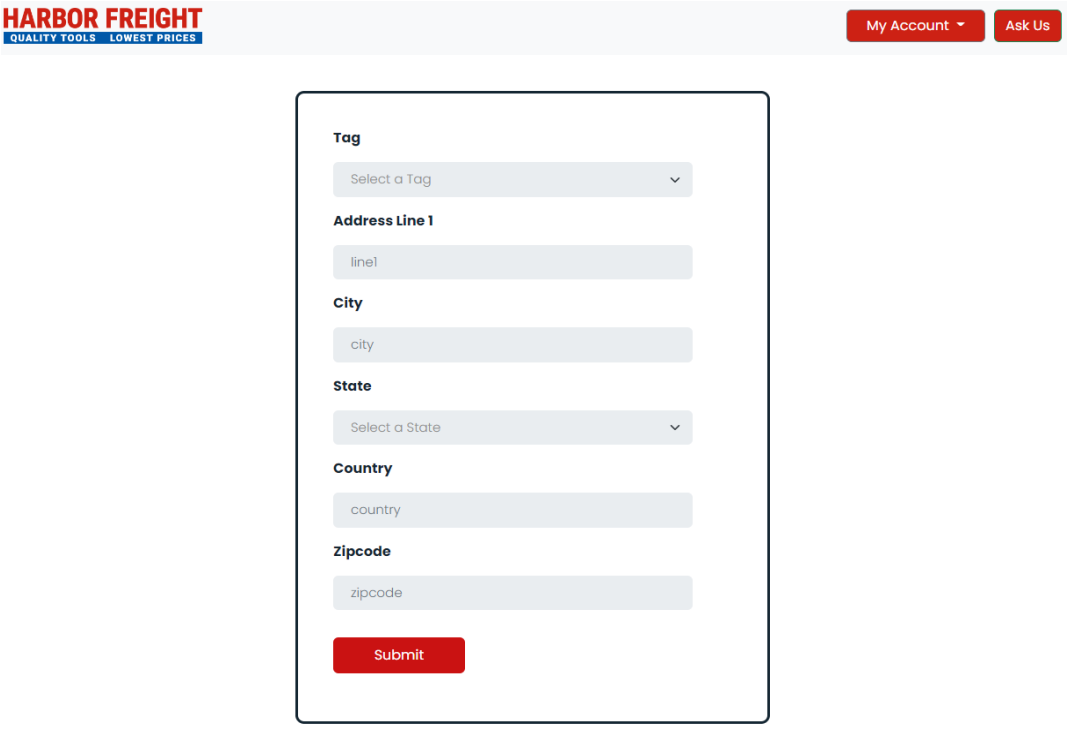


Customer address page

This page shows the saved addresses of the customer. We can reach this page by clicking on ‘My Account’ and then selecting the ‘Update Address’ from the dropdown. Customers can update their address, delete address and add new addresses as well.

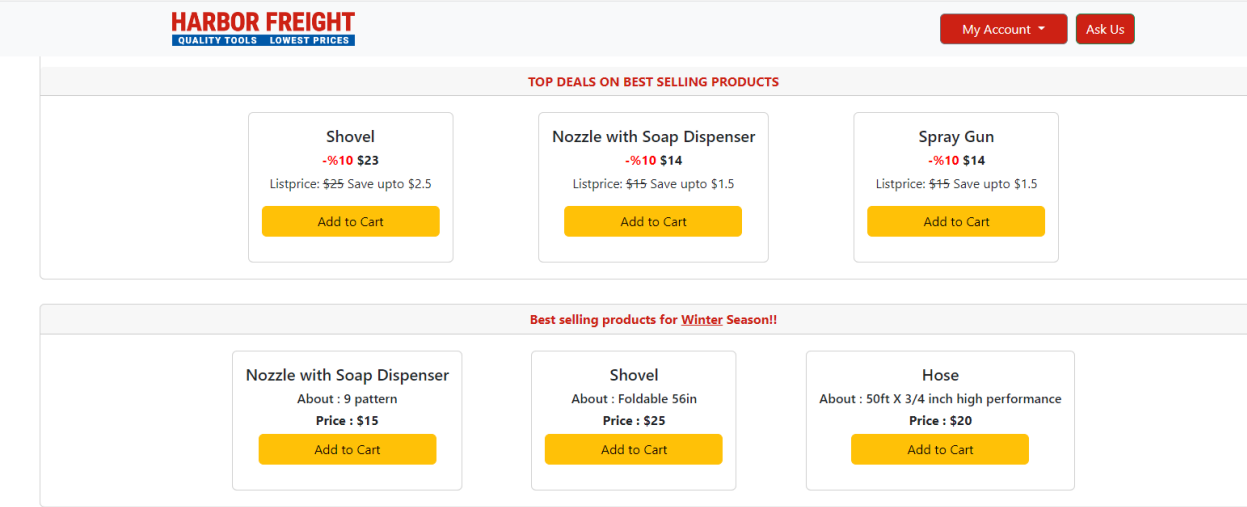


A customer can reach this page by clicking on the ‘Add New Address’ button on the address page. The customer can save a new address using this page.

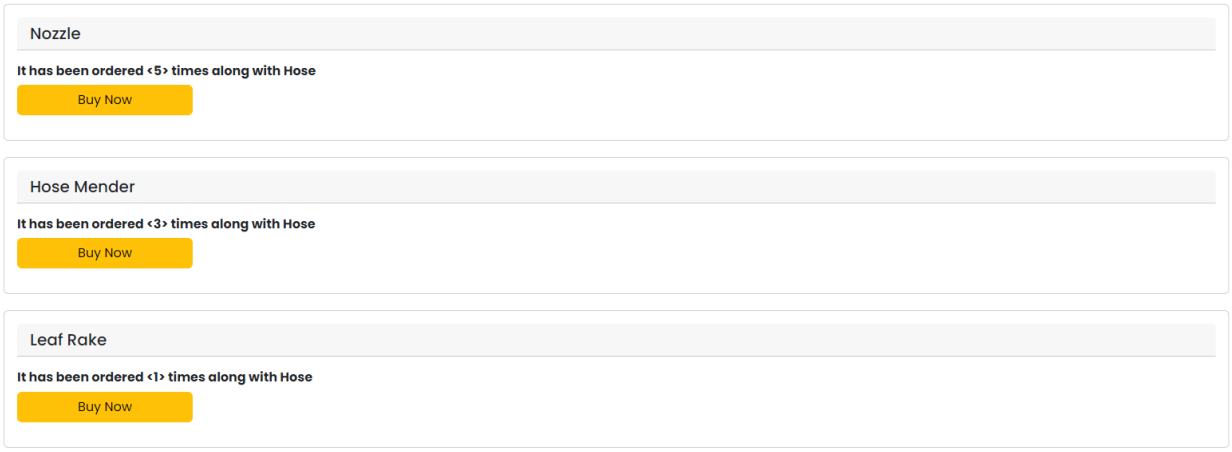


Products Page

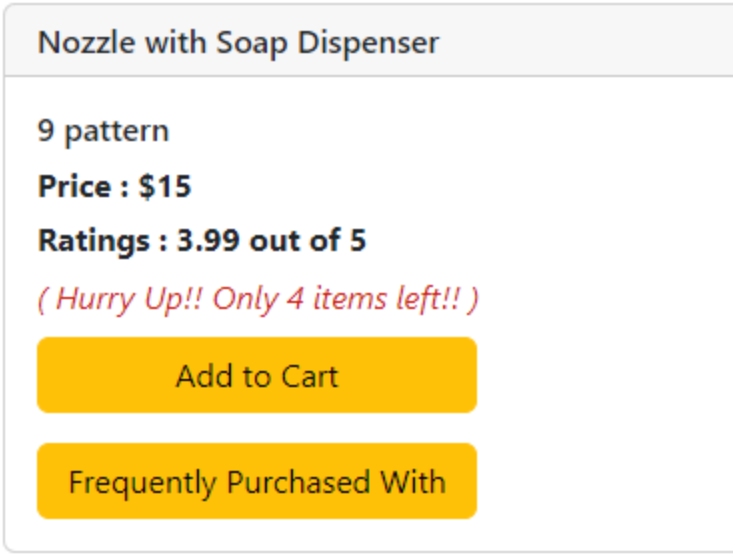
This is the default page displayed after the customer logs in. This page displays the top deals on best selling products and the best selling products for the current season.



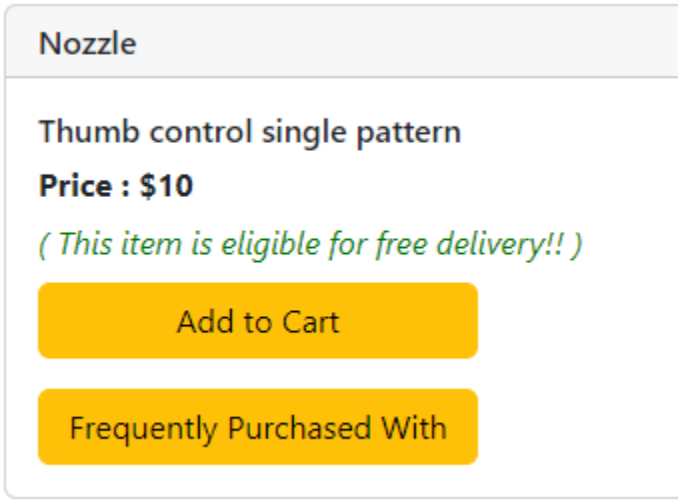
List of products that were frequently purchased with the product ‘Hose’. Customers can navigate to this page by clicking on the ‘Frequently purchased together’ button on a product



The following alert is displayed when there is low stock

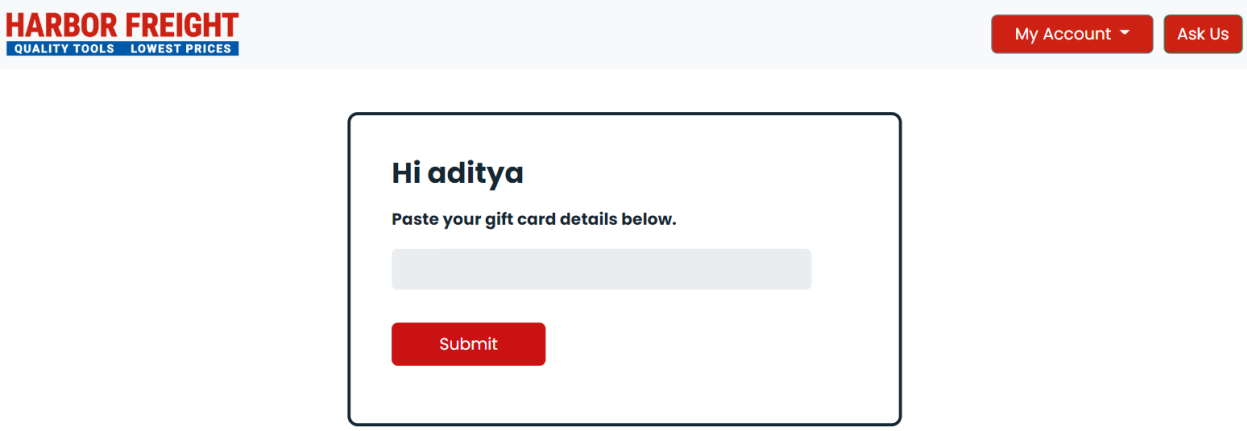


The following alert is displayed when the product is eligible for free delivery

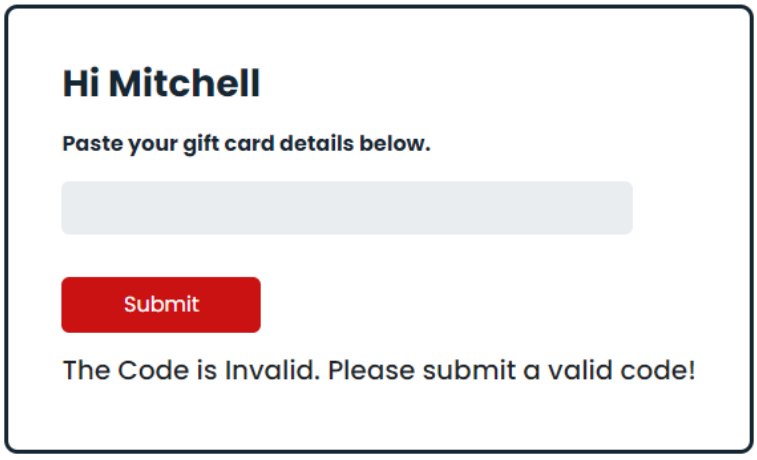


Gift card activation

Customers can navigate to the Gift card page from the ‘Account details’ button. They can enter the gift card code to activate. On submitting, if the gift card code is valid, the gift card amount is added to the customer’s wallet.

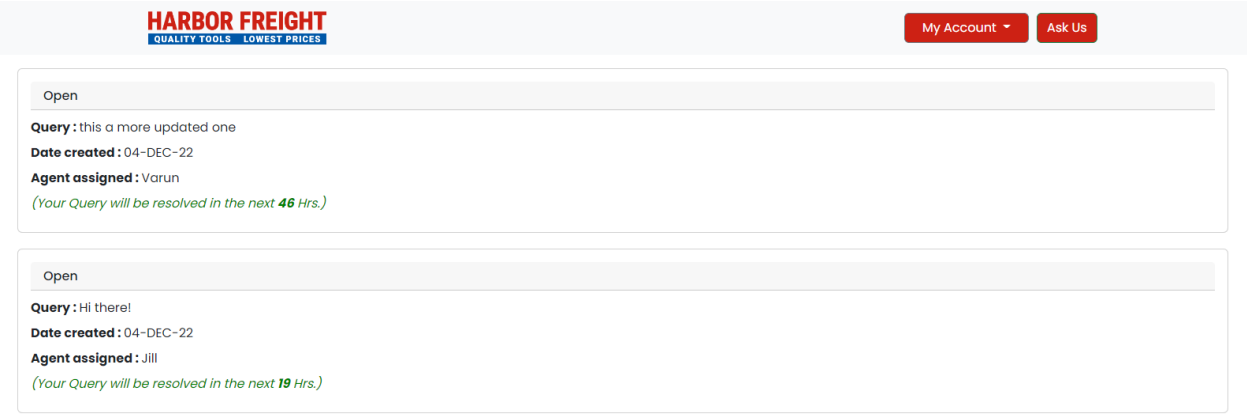


The following error is displayed when an invalid gift card code is entered



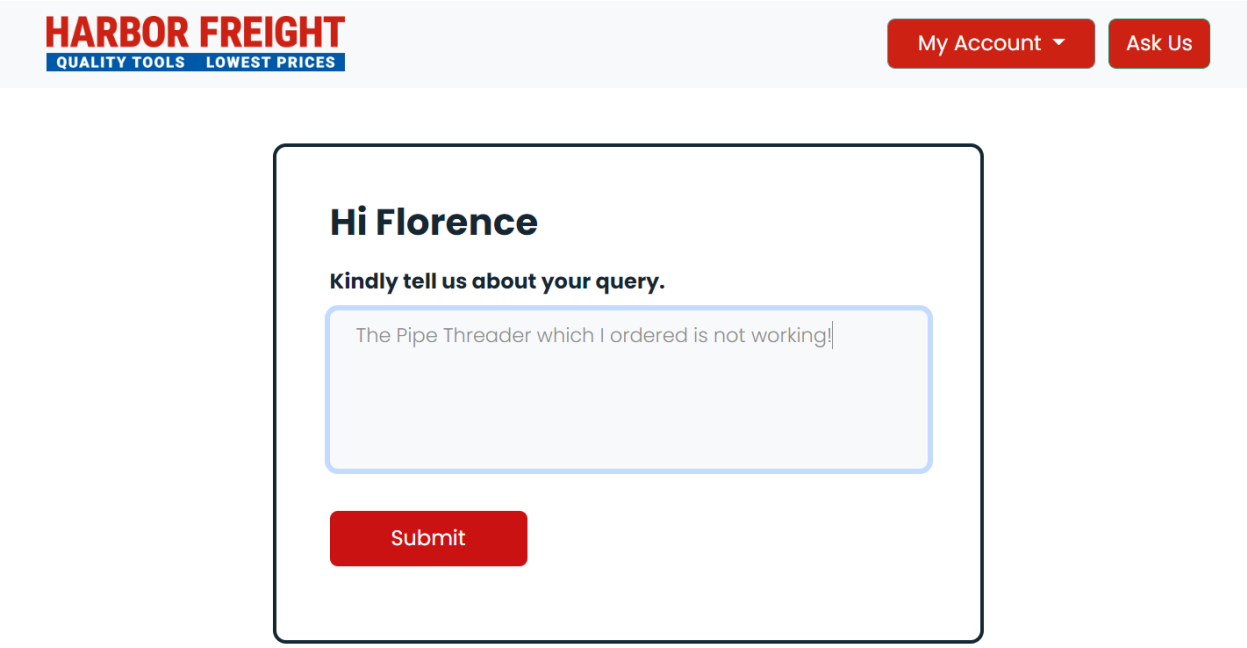
Customer tickets page

Customers can navigate to this page using ‘My Queries’ option under ‘Account Details’ button. Here, the customers can view the queries created by them and the associated details.

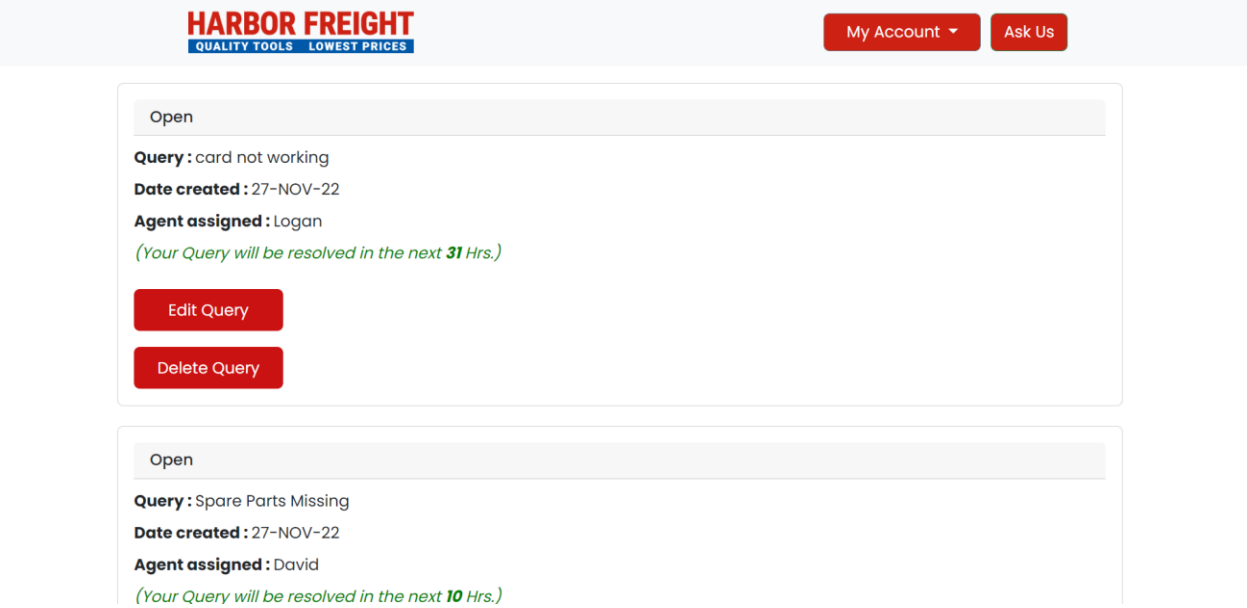


The Queries page

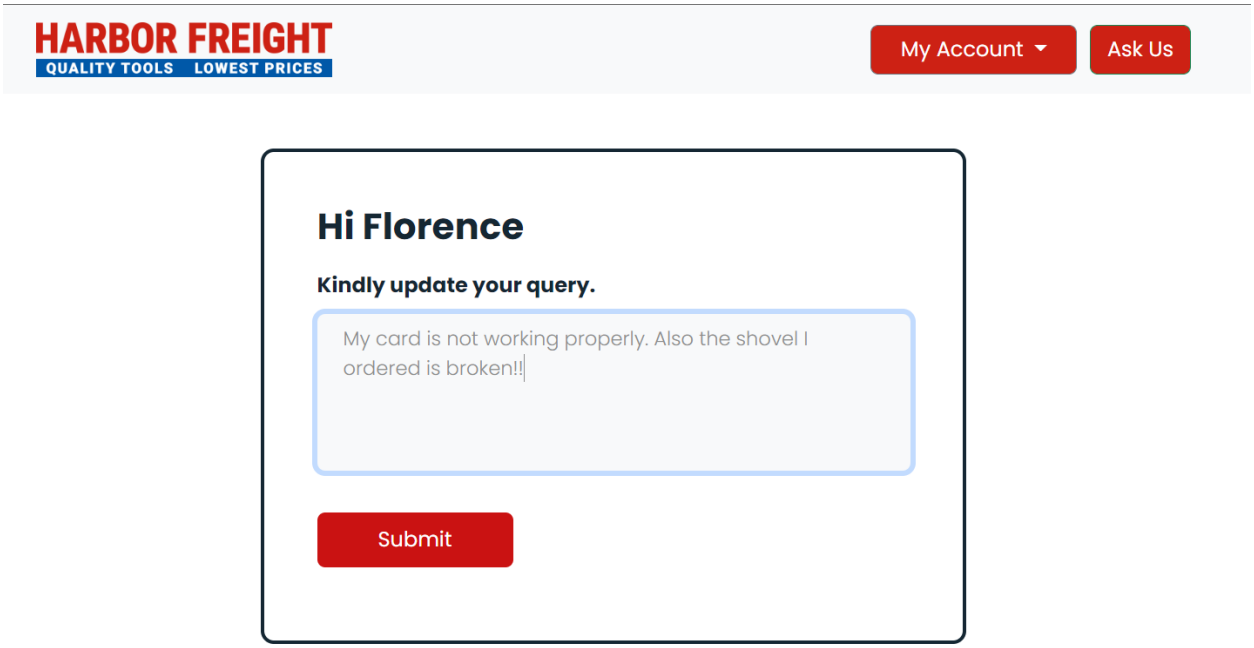
After clicking on the ‘Ask Us’ button on the top right corner of the page, the customer will be taken to a new page where the customer can add a new query.
After clicking on the Submit button the new query will be added to the list of queries.



After navigating to My Account > My Queries, the customer will be able to see the list of queries he has raised through “Ask Us”. Here the customer has an option to edit the query or delete the query as well.



After clicking on ‘Edit Query’ the customer will be taken to a new page where he can update the query and submit it.



Chapter 7: Implementation Plan

Harbor Freight is a large e-commerce platform with an average of about 30 million visitors per month, or 1 million visitors per day. The average user visits 2 pages/visit. Therefore, we assume that we would need to handle about 60 million visits per month + 10% leeway, producing 66 million visits per month. Assuming each visit on our website requires an average of 12 kb in php script, each visitor would use 24 kb. So for 66 million visits, we would require 66 million times 24 kb which is 1.58 TB. Therefore we recommend starting with a 1650GB plan. We plan to host the Harbor Freight website on Amazon's EC2 servers because we can scale seamlessly with Amazon EC2 auto scaling and its pay per use which could also save costs.

Steps & Hours

Steps	**Person Hours**
Database Creation	20
Data Insertion	10
Front-End Creation	40
Backend Development	30
Backend & Frontend Integration	20
Hosting on EC2	10
Training Employees	20
Testing	40
Maintenance	10
Potential Hurdles	40
Total	240

Expenses

Expenses	**Amount**	**Notes**
Amazon EC2 Instance - US West (Oregon)	$5,620	Windows Server with SQL Server Standard; 1 EC2 Instance
Storage (AWS)	$165 * 12 months = $1,980	1650GB Amazon Elastic Block Storage (EBS)

Personnel	$48,000	$40/hour for 240 hours total; 5 database consultants
Processor License (Oracle Database)	$17,500	Standard Edition 2
Software Update + License Support (Oracle)	$3,850	
Additional Fees	$4,000	Workspace, Food, Miscellaneous
Total	**$80,950**	

References

https://www.semrush.com/website/harborfreight.com/overview/
https://www.zippia.com/database-consultant-jobs/salary/
https://calculator.aws/#/addService/EC2
https://www.oracle.com/assets/technology-price-list-070617.pdf

Appendix A: Lessons Learned

- As a group with mixed technical backgrounds, the first thing we learned is to play according to each other's strengths and work as a team to accomplish various tasks.
- It was a great learning experience implementing all the concepts that we learned throughout the course
- It was insightful to implement the project as it would be implemented in the real-world
- The team had conflicts and learned how to navigate through difficult conversations