CPS510 Term Project

Documentation

Group 77 | E-commerce Database System

Ayaz Nakhuda | 500952842 | Section 2

David Ferris | 500969121 | Section 2

Ngoc Huyen Oanh Phung | Section 2

Documentation Index

Project Introduction

Application Overview

The application that we will be developing throughout the duration of this course is an e-commerce database system which is the backbone of some of the most revolutionary web applications in the world such as Amazon. E-commerce applications such as online shopping applications, have revolutionized the commercial shopping industry and the responsibility of an e-commerce database system is to provide the necessary operations and information to run traditional stores in a digital format. The operations and information needed include inventory management, customers' purchase record, product supplier information, general customer information and most importantly, allowing customers to purchase products. From these operations we can begin to see some of the entities that should be present within the database system such as the customers, the products available to be purchased, and the orders placed to the application. These entities and more will be explored later in this report.

In addition, to the entities of the database it is important to mention the overall functionality of the application. The functionality of this e-commerce system is quite simply understood by looking at the functionality of any popular e-commerce website such as Amazon or Aliexpress. To put it briefly, the system should provide the ability for users to buy available products and for a potential business to manage inventory of products produced by multiple sources. The functionality of the application is explored further in the *Functionality* section of this report.

One last important aspect of this application that should be noted in this overview is that we have decided this database system will provide support for having multiple sources of products or, in other words, multiple sellers. For this reason, the application must be able to differentiate products that are from different suppliers (sellers) which makes the database system more general as well as extensible.

System Description

As was touched on briefly during the *Application Overview* section, the information that must be managed by this e-commerce database system is the traditional business metrics needed to run physical commercial companies. In a general sense, this database will store information that relates to customers' purchases on an e-commerce platform. To elaborate on this, the system must maintain data that either facilitates the functionality of an e-commerce platform or provides key business insights for application owners. In terms of the information relating to functionality, the system must maintain information about the products which are available for purchase, customers who have purchased products or are able to make purchases, the products that a customer is intending to buy, and the products which have been purchased by customers. These pieces of information facilitate the key functionality of the e-commerce system such as product purchasing which will be touched on in the *Functionality* section of the report. On the other hand with respect to business metrics the system should maintain information regarding the purchase history of specific customers as well as the suppliers of the various products. This information allows application owners to gain key insights into things such as product popularity and pricing models for specific products.

From these information requirements some of the entities that should be present within the system as well as some of the attributes of these become clear and are outlined below.

Database System Entities:

Order: Describes a transaction between a customer and the e-commerce platform.

Order_Details: Describes the details in terms of products and quantities of a transaction between a customer and the e-commerce platform.

Customer: Customers are those who have purchased a product on the e-commerce platform that has deployed this database.

Product: A product in the system will be represented by its description as a string in the database.

Product_Categories: Describes the categories which products within the system fall into.

Seller: The seller of a purchased product will be represented in string form.

Shipper: The shipping services available through the platform for order delivery.

Cart: Relates a customer to a cart entity so they can accumulate products for later purchase.

Cart_Products: The customer's current collection of items awaiting check-out (purchase).

A Sample of Database System Entity Attributes:

Order_number: The ***Order_number*** is a unique number which is issued for each invoice. It will be represented as a 6 digit integer which is unique for each invoice. The ***Order_number*** is an attribute of the order entity.

Order_date: The ***Order_date*** is the date which the invoice was created. It will be represented as a string. The ***Order_date*** is an attribute of the Order entity.

Card_No: The ***Card_No*** is an integer which details the payment information associated with a customer's payment method. It will be represented as an integer. The ***Card_No*** is an attribute of the Customer entity.

CustomerID: Customers can be represented with a unique 5 digit integer. CustomerID is an attribute of the Customer entity.

Region: The ***Region*** is the nation that the customer is based in at the time of placing the order. The ***Region*** will be represented as a string and is an attribute of the Customer entity.

Billing_Address: The ***Billing_Address*** is a string that contains information about the address associated with a customer's credit or debit card account. This will be represented as a string and it is an attribute of the Customer entity.

Product_ID**:* The ***Product_ID is an integer which represents each individual product that is offered on the e-commerce platform. The ***Product_ID*** is an attribute of the Product entity.

Description: The description of a purchased product will be represented as a string. The Description is an attribute of the Product entity.

Quantity: This represents the quantity of an item that a customer has purchased. The Quantity will have integer values. The Quantity is an attribute of the Order_Details entity.

unitPrice: The unitPrice represents the price of an individual item that was purchased. It is represented as an integer. unitPrice is an attribute of Product.

Category: The Category represents which category that a product belongs in. It will be represented as a string and it is an attribute of the Product_Categories entity.

Functionality

The purpose of this e-commerce database system is to enable customers to explore a wide range of products and make purchases that are accurately recorded and processed while enabling the owners of the e-commerce platform to manage the inventory of available products and product suppliers. In order to facilitate these requirements our system must provide a variety of functions that are separated into four categories the first being *Customer Management*, the second is *Product Purchasing*, the third being *Order History Insight*, and finally the fourth is *Inventory Management*. The details of the functionality that fits in each category is outlined below.

Customer Management:

The Customer Management functionality refers to the need for the e-commerce database system to allow for the dynamic storage of customers along with their associated information. Customers need to be able to be added and removed from the database system at any point in time and by extension their corresponding information must be dynamic as well. That is, attributes such as payment information, delivery address, and billing address must not only be maintained for each unique customer but also must support dynamic updates which must also propagate throughout the database where this information is used.

Product Purchasing:

Potentially the most important and central functionality for this e-commerce system is to provide customers the ability to purchase products and have their orders recorded correctly and efficiently within the system. To fulfill this requirement first customers must be able to view the products which are available for purchase on the e-commerce platform. Thus, the products must be stored with correct stock information that can be updated at any point but this will be addressed later in the *Inventory Management* section. There are generally two ways to purchase products on an e-commerce platform, customers can either purchase a single item commonly seen as “Buy Now” option or customers can purchase a collection of items typically through the means of a “Shopping Cart”. Due to this the database system must not only maintain record of the products purchased by a customer but also the current state of that customer's “Shopping Cart” and update this as items are added or removed dynamically.

Order History Insight:

The *Order History Insight* functionality is mostly focused towards the application owners as its purpose is generally to form some type of insights on the performance of the e-commerce platform and specific products but it can also be valuable for customers to be able to view previous orders. All purchases on the platform must be recorded within the database to allow customers to verify purchases as well as the e-commerce platform owners to extract valuable information such as the performance of the platform and even the shopping habits of specific

customers. The system should also allow customers and owners to check if customers have purchased a specific product. The order history functionality is also helpful for customers as it can enable them to quickly place re-orders for sets of products they order frequently.

Inventory Management:

Finally, the *Inventory Management* functionality is another one of the most important aspects of this e-commerce system along with the *Product Purchasing* as it relates to the accurate storage and dynamic updating of available products within the system. The database system needs to update relevant product information such as the stock quantity when purchases are made on the platform in order to maintain the integrity of the data. In addition to managing pre-existing products in the system this system must have the ability to add and remove products from the database and when doing so it must also assign all necessary attributes for the product such as the supplier and category tags. These attributes will, of course, need to be provided by the person who is adding the product to the system. This functionality coupled with the functionality outlined in the *Product Purchasing* section provides the core elements of the e-commerce system and are the back-bone of popular e-commerce platforms.

Key Functions Summary:

- ***Add and Remove customers from the system***
    - ***Propagate additions and removals through all relevant areas of the system***
- ***Query for customers based on customer information***
- ***Perform updates on customer information***
    - ***Also should propagate through relevant areas of system***
- ***Query system for the list of products available for purchase (allow customers to browse products) - potentially view products by specific tags as well***
- ***Allow customers to purchase products one at a time or as a collection (shopping cart)***
- ***View the orders that have been placed in the system, i.e., allow orders/ a specific order for a specific customer to be viewed***
- ***Allow customers to reorder specific orders from their history***
- ***Add or Remove available products from the system***
- ***Add or Remove product suppliers from the system***
- ***Update stock quantities for products manually and when orders are placed***

System ER Models

E-Commerce System Entities & Attributes

The following section outlines the entities that will be present within our e-commerce system. Most of these are strong entities but some weak entities are also included in order to correctly describe the functionality of the system. It is important to note that the entities on this list may change throughout the development of the system to facilitate functionality improvements for the system. The list of entities is followed by a list of attributes for these entities.

Customer

- ***CustomerID:*** Customers can be represented with a unique 5 digit integer.
- ***CustomerName:*** Customers name represented as a string.
- ***BillingAddress:*** The Billing Address is a string that contains information about the address associated with a customer's credit or debit card account.
- ***Region:*** Customers location in terms of region.
- ***DeliveryAddress:*** Address where the customer expects to receive purchased products.
- ***PaymentInformation:*** composite attribute that is used to determine how a customer pays for products, necessary information for charging a customer for products.
    - Card_holder
    - Card_no

Order

- ***Order_number:*** The order number is a unique number which is issued for each invoice. It will be represented as a 6 digit integer which is unique for each invoice.
- ***Order_Date:*** The ***Order_Date*** is the date which the invoice was created. It will be represented as a string.
- ***ShipperID:*** The shipping service provider which will be used for this order (uses the ShippingID attribute from the Shipper entity)
- ***CustomerID:*** Foreign attribute that relates an order to a specific customer. CustomerID from the customer entity

Order_Details ~ (Multi-Valued Attribute of Order implemented as Weak entity)

- ***Order_number:*** The order number is a foreign attribute relating the details of an order to the order entity.
- ***Product_ID:*** Foreign attribute that relates an ordered product to an order.
- ***Quantity:*** Integer attribute representing the quantity of a product in a customer order

Product

- ***ProductID:*** Unique identifier for a product available on the system
- ***Description:*** The description of a product will be represented as a string.
- ***unitPrice:*** The unitPrice represents the price of an individual product that is available in the system. It is represented as an integer.

Cart

- ***CustomerID:*** Foreign attribute that relates the cart to a specific customer. CustomerID from the customer entity
- ***Cart_ID:*** Unique identifier for a customer's cart entity.

Cart_Products ~ (Multi-Valued Attribute of Cart implemented as Weak entity)

- ***Cart_ID:*** Foreign attribute that relates the cart_products to a specific customers cart
- ***Product_ID:*** Foreign attribute that relates the selected product to a customers cart
- ***Quantity:*** Integer attribute representing the quantity of a product in a customer cart

Seller

- ***SellerID:*** Unique identifier for a seller of a product/s.
- ***SellerName:*** The name of the seller will be represented as a string.
- ***SellerEmail:*** The email address of the seller represented as a string.

Product_Categories

- ***ProductID:*** Unique identifier for the product being referenced.
- ***Category:*** A string which contains the category name that a product belongs in.

Shipper

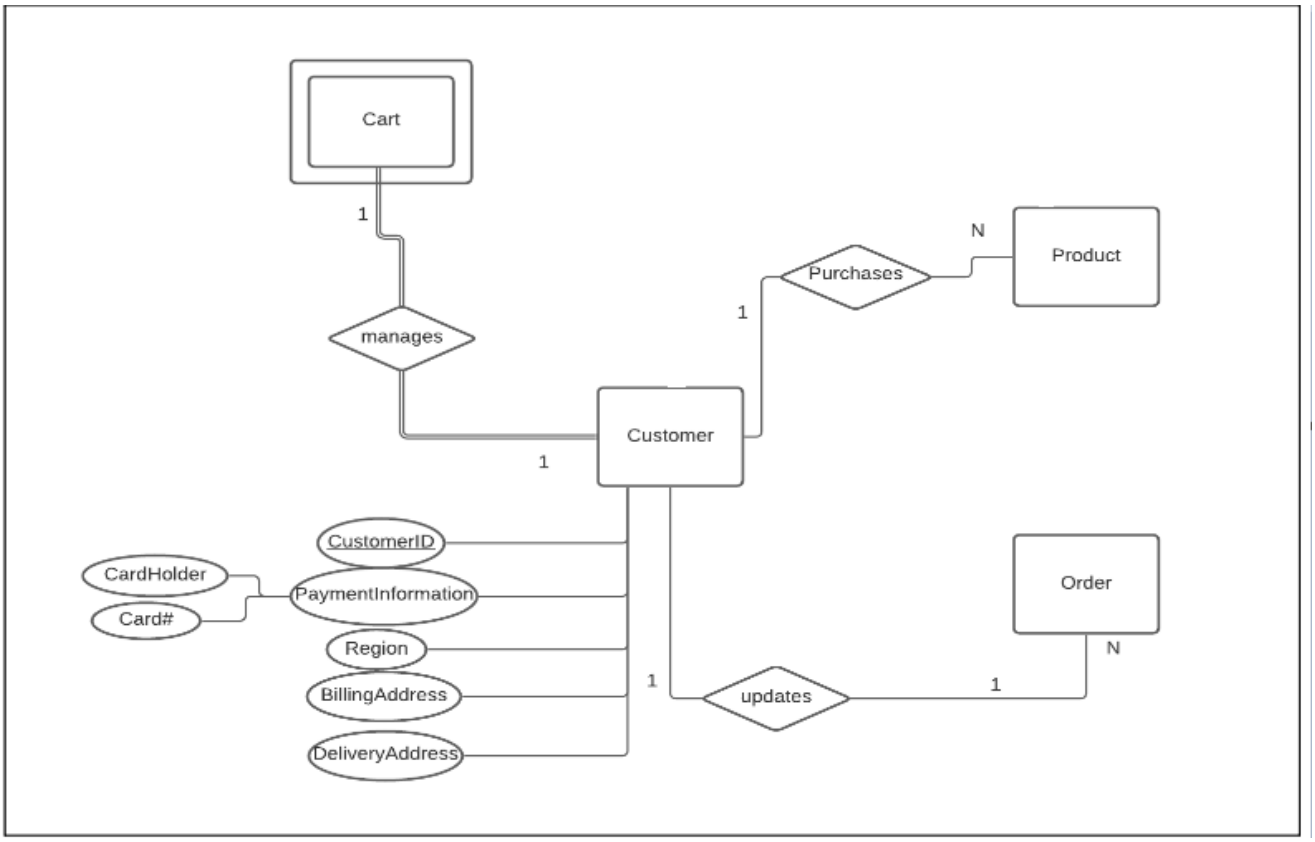
- ***ShipperID:*** Unique identifier for the shipper.
- ***ServiceProvider:*** The name of the shipment service provider of the order represented as a string.

Customer Entity Relationships

This section outlines the main relationships that exist between the Customer entity and the other entities of the system. The relationships illustrated through the ER diagram in this section include:

- Customer purchasing a Product
- Customer managing their Cart
- Customer managing/updating their Order

Note that some relationships are intentionally left out of this entities section such as the relationship between Customer and Order entities as these are saved for the Order entity section, the following sections may follow this same form. Below is the ER diagram to illustrate the relationships for the Customer entity for more information on entity attributes see the *E-Commerce System Entities & Attributes* section of this report:

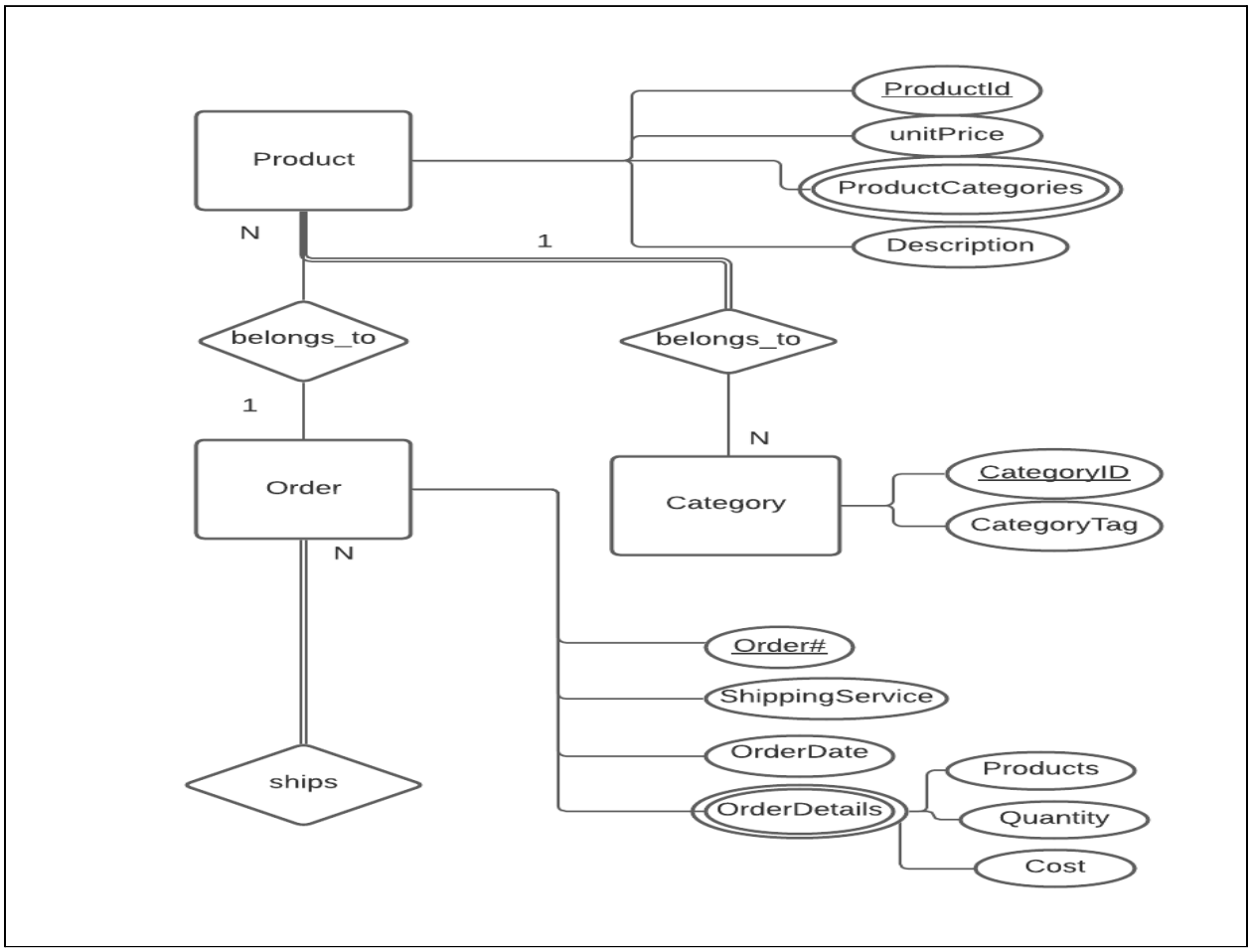


Product Entity Relationships

This section outlines the main relationships that exist between the Product entity and the other entities of the system. The relationships illustrated through the ER diagram in this section include:

- Products belonging to Orders
- Products belonging to Categories

Below is the ER diagram to illustrate the relationships for the Product entity for more information on entity attributes see the *E-Commerce System Entities & Attributes* section of this report:

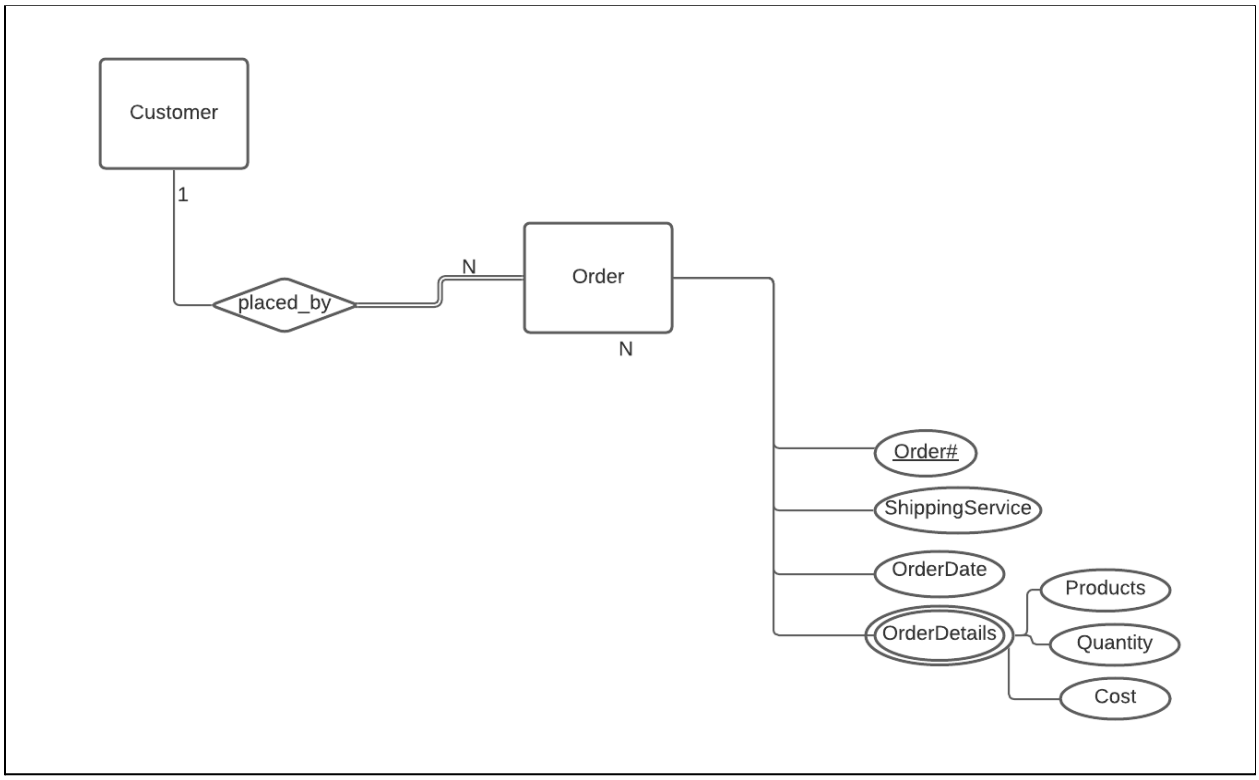


Order Entity Relationships

This section outlines the main relationships that exist between the Order entity and the other entities of the system. The relationships illustrated through the ER diagram in this section include:

- Order being placed by a Customer

Below is the ER diagram to illustrate the relationships for the Order entity for more information on entity attributes see the *E-Commerce System Entities & Attributes* section of this report:

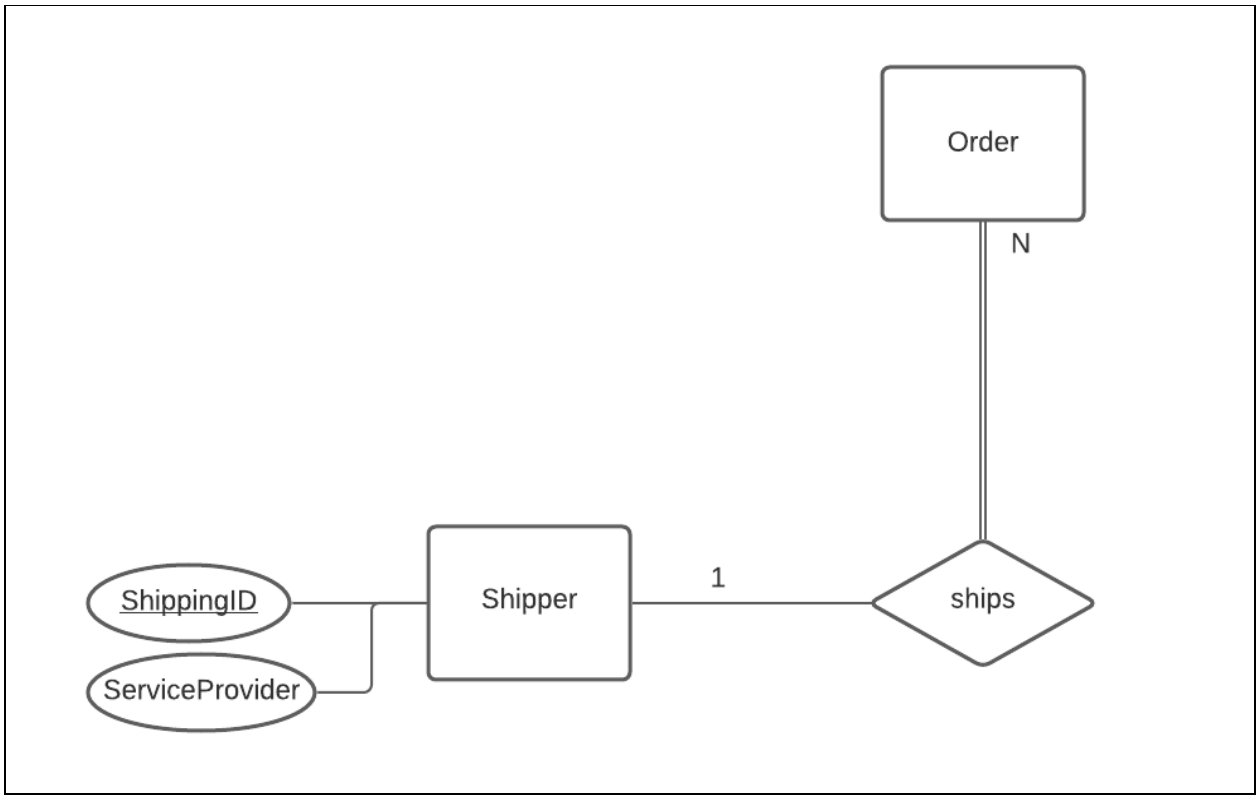


Shipper Entity Relationships

This section outlines the main relationships that exist between the Shipper entity and the other entities of the system. The relationships illustrated through the ER diagram in this section include:

- Shipper ships a customer's Order

Below is the ER diagram to illustrate the relationships for the Shipper entity for more information on entity attributes see the *E-Commerce System Entities & Attributes* section of this report:

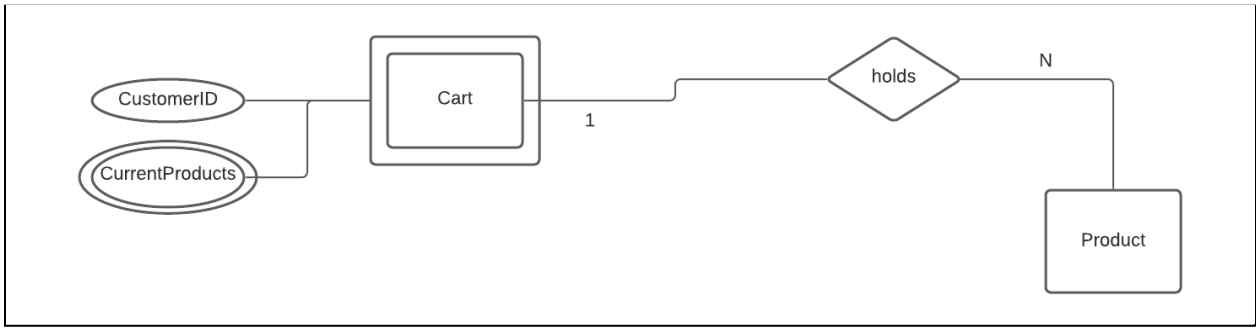


Cart Entity Relationships

This section outlines the main relationships that exist between the Cart entity and the other entities of the system. The relationships illustrated through the ER diagram in this section include:

- Cart holds a customer's selected Product

Below is the ER diagram to illustrate the relationships for the Cart entity for more information on entity attributes see the *E-Commerce System Entities & Attributes* section of this report:

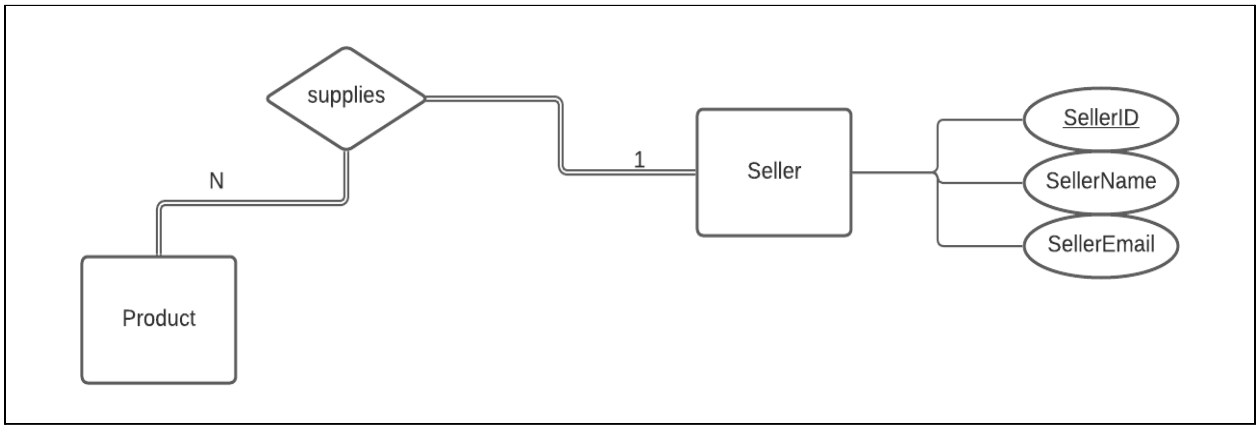


Seller Entity Relationships

This section outlines the main relationships that exist between the Seller entity and the other entities of the system. The relationships illustrated through the ER diagram in this section include:

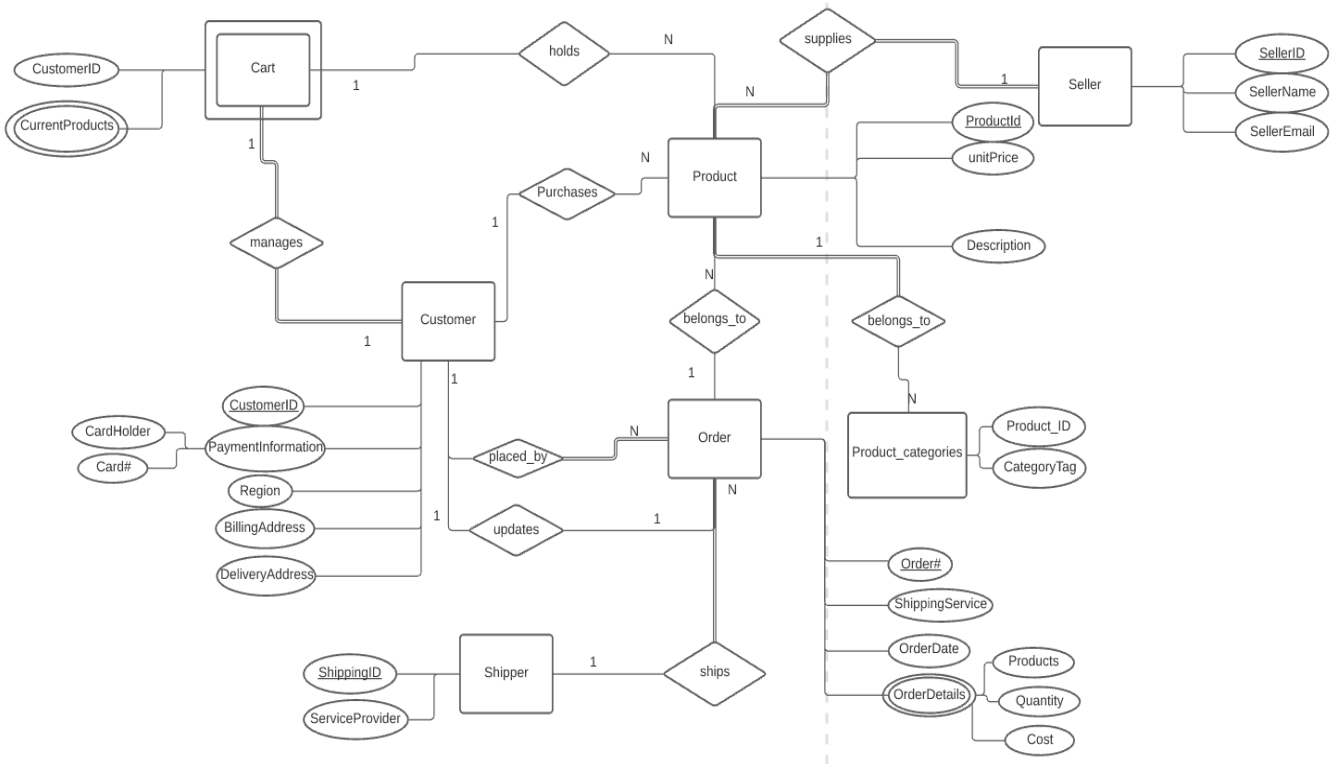
- Seller supplies one or more Products

Below is the ER diagram to illustrate the relationships for the Seller entity for more information on entity attributes see the *E-Commerce System Entities & Attributes* section of this



E-Commerce Overall System ER Diagram

This final section combines the diagrams and relationships from the previous sections of the individual entities in order to provide a clear high-level overview of the e-commerce system. This ER diagram outlines all entities, attributes, and relationships that make up our e-commerce system.


CustomerID
CurrentProducts
Cart
1
holds
N
supplies
1
Seller
SellerID
SellerName
SellerEmail
N
ProductId
unitPrice
1
Purchases
N
Product
1
manages
1
Description
N
Customer
belongs_to
belongs_to
1
1
N
CustomerID
CardHolder
PaymentInformation
Card#
Region
BillingAddress
DeliveryAddress
placed_by
N
Order
Product_categories
Product_ID
CategoryTag
1
updates
1
N
Order#
ShippingService
OrderDate
OrderDetails
Products
Quantity
Cost
ShippingID
ServiceProvider
Shipper
1
ships

System Normal Forms (3NF)

In this portion we outline the normal forms which each of our tables for the E-commerce system is in and where necessary we will perform decomposition of tables to ensure all tables are eventually in the third normal form (3NF). Exploring the various tables for our e-commerce database system revealed that a majority of our tables were already in 3NF and thus did not require any decomposition, however, in order to demonstrate this process for 2NF and 3NF we will introduce some new attributes and assumptions in order to perform the required decomposition for a few tables.

Product Table

Relational Schema:
Product(product_id, seller_id, unit_price, product_name, product_description)

```
CREATE TABLE Product (
    product_ID int NOT NUll PRIMARY KEY AUTO_INCREMENT,
    unit_price float NOT NULL,
    description_ VARCHAR2(30) NOT NULL
);
```

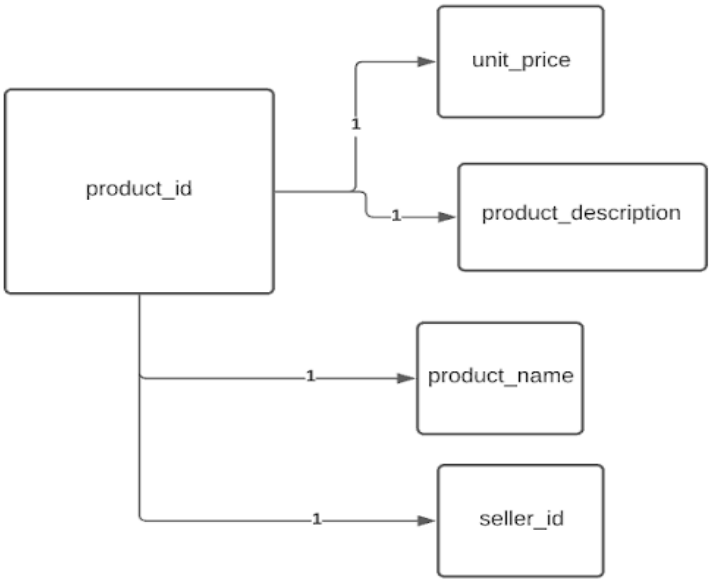
The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

$FD_{product}$ = {
product_id→ unit_price
product_id→ product_name
product_id→ product_description
product_id→ seller_id
}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the product table:

The Product table is in the 1st normal form (1NF) because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the product_ID
- Order of the stored data does not matter

The Product table is in the 2nd normal form (2NF) because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Product table is in 3rd normal form (3NF) because:

- Table is in 2nd normal form
- No transitive dependencies

Since the product table has been shown to already be in 3NF no decomposition is required for this table.

Customer Table

Relational Schema:
Customer(customer_id, customer_name, region, card_holder, card_no, billing_address, delivery_address)

```
CREATE TABLE Customer (
    customer_ID int NOT NUll PRIMARY KEY,
    customer_name VARCHAR2(30),
    region VARCHAR2(30),
    billing_address VARCHAR2(30) NOT NULL,
    delivery_address VARCHAR2(30) NOT NULL
);
```

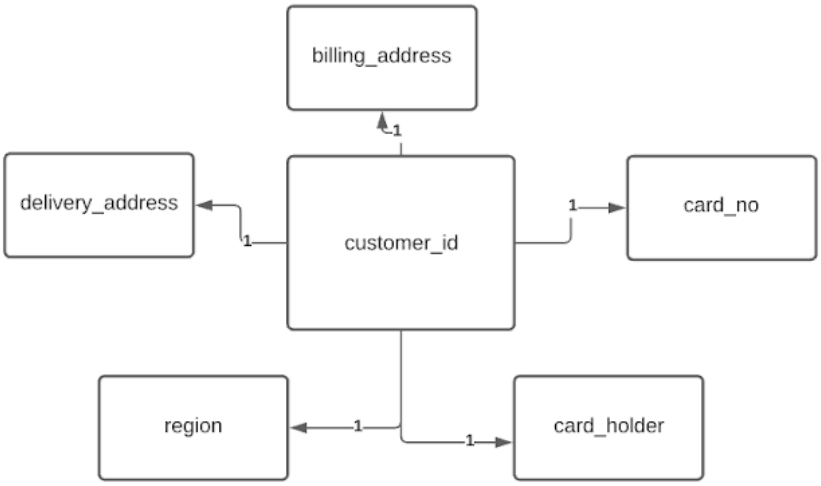
The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

$FD_{customer}$ = {
customer_id → customer_name
customer_id → region
customer_id → card_holder
customer_id → card_no
customer_id → billing_address
customer_id → delivery_address
}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the customer table:

The Customer table is in the 1st normal form (1NF) because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the customer_ID
- Order of the stored data does not matter

The Customer table is in the 2nd normal form (2NF) because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Customer table is in 3rd normal form (3NF) because:

- Table is in 2nd normal form
- No transitive dependencies

Since the customer table has been shown to already be in 3NF no decomposition is required for this table.

Seller Table (Example of Decomposition for 3NF)

Relational Schema:
Seller(seller_id, seller_name, seller_email)

```
CREATE TABLE Seller (
    seller_ID int NOT NUll PRIMARY KEY AUTO_INCREMENT,
    seller_name VARCHAR2(30),
    seller_email  VARCHAR2(30)

);
```

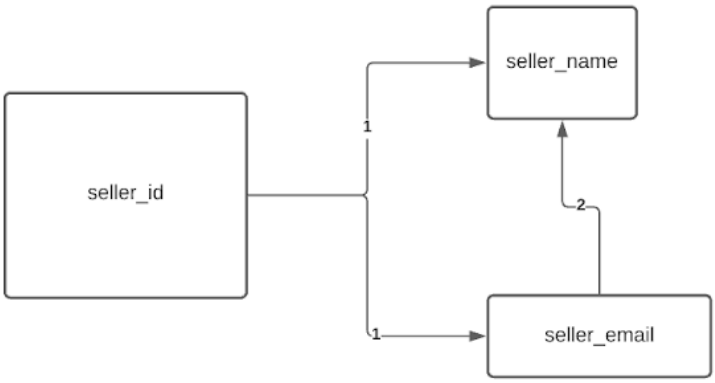
For the purposes of this assignment in order to perform decomposition on this table we will assume that the seller_email attribute for each seller must be unique, that is no two sellers can share the same email. With this assumption we then can see that a transitive functional dependency as now seller_email → seller_name and seller_id → seller_email. The functional dependencies for this table are shown below.

Functional Dependencies:

FD_{seller} = {
seller_id → seller_name
seller_id → seller_email
seller_email → seller_name
}

With this we construct the functional dependency diagram for the seller table as follows:

Functional Dependency Diagram



From this diagram we are able to determine the following information for the normal form of the seller table before any decomposition:

The Seller table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the seller_ID
- Order of the stored data does not matter

The Seller table is in the 2nd normal form because:

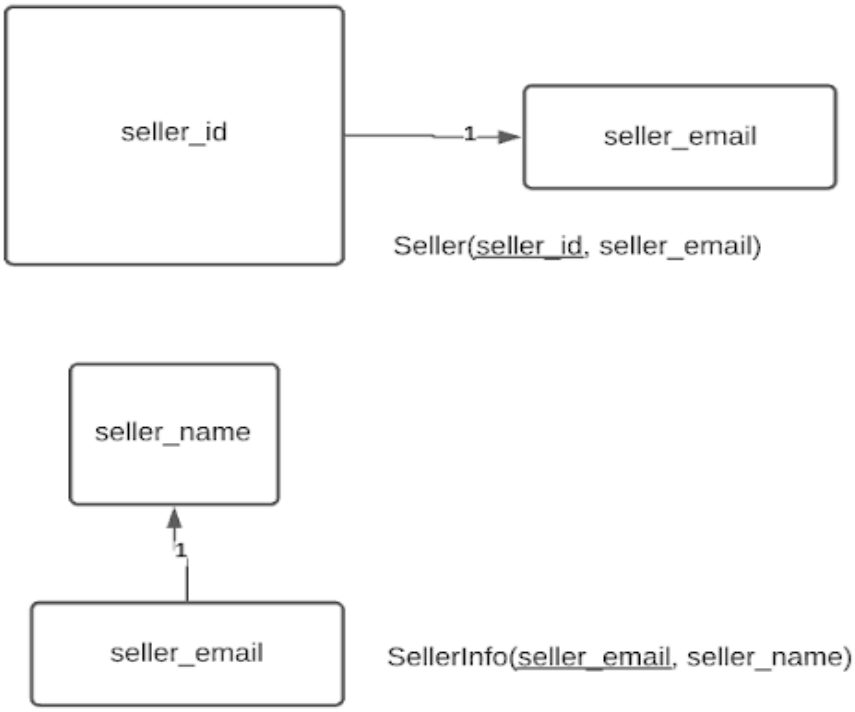
- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Seller table is NOT in 3rd normal form because:

- There is a Transitive dependency between seller_email and seller_name

Thus, in order to achieve 3NF for the seller table we must perform decomposition on the table, separating the relationship between seller_email and seller_name into a separate table such that the transitive dependency is removed. This decomposition is shown below:

Decomposed Functional Dependency Diagram



As we can see by decomposing the original seller table into two tables, one for the seller where we implement the relationship between seller_id and seller_email and one for the seller information where we implement the relationship between seller_email and seller_name attributes we have successfully removed the transitive dependency. In doing so we successfully have put the seller table into 3NF.

Decomposed Functional Dependencies:

FD_{seller} = {
seller_id → seller_email
}
$FD_{sellerInfo}$ = {
seller_email → seller_name
}

From this diagram we are able to determine the following information for the normal form of the decomposed seller table:

The Seller table is in the 1st normal form (1NF) because:

- Each attribute is single valued

- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the customer_ID
- Order of the stored data does not matter

The Seller table is in the 2nd normal form (2NF) because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Seller table is in 3rd normal form (3NF) because:

- Table is in 2nd normal form
- No transitive dependencies

Category Table

Relational Schema:

Category(category_id, category_tag)

```
CREATE TABLE Category (
    category_ID int NOT NUll PRIMARY KEY,
    category_tag VARCHAR2(30)

);
```

The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

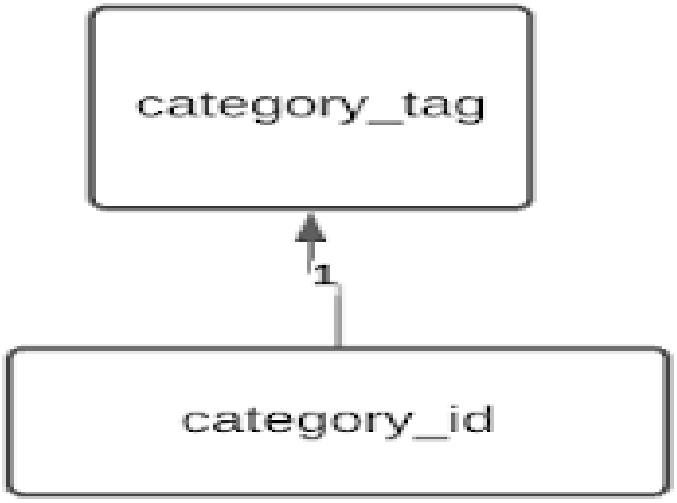
$FD_{category}$ = {

category_id→ category_tag

}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Category table:

The Category table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the category_ID
- Order of the stored data does not matter

The Category table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Category table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

Since the category table has been shown to already be in 3NF no decomposition is required for this table.

Shipper Table

Relational Schema:
Shipper(shipper_id, service_provider)

```
CREATE TABLE shipper (
    shipper_ID int NOT NUll PRIMARY KEY,
    service_provider VARCHAR2(30)

);
```

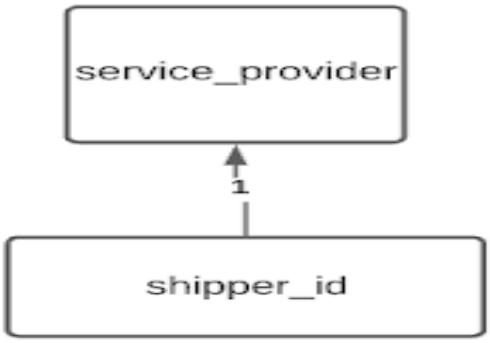
The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

$FD_{shipper}$ = {
shipper_id → service_provider
}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Shipper table:

The shipper table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the shipper_ID
- Order of the stored data does not matter

The shipper table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The shipper table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

Since the shipper table has been shown to already be in 3NF no decomposition is required for this table.

Orders Table

Relational Schema:

Orders(order_number, customer_id, shipper_id, order_date)

```
CREATE TABLE orders (
    order_number int NOT NUll PRIMARY KEY,
    customer_ID int REFERENCES Customer(customer_ID),
    shipper_ID int REFERENCES Shipper(shipper_ID),
    order_date date
);
```

The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

FD_{orders} = {

order_number→ customer_id

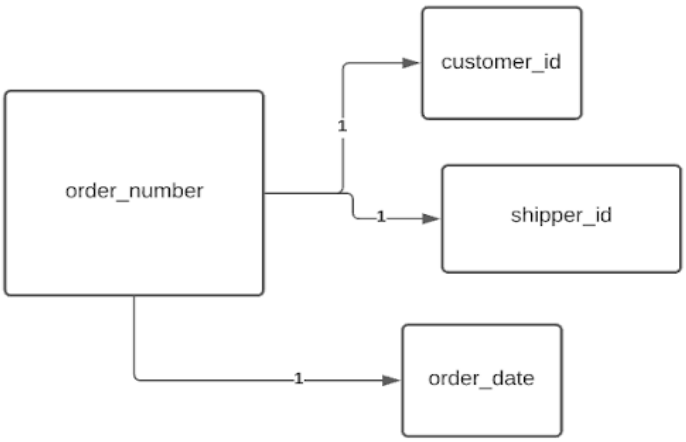
order_number→ shipper_id

order_number→ order_date

}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Orders table:

The Orders table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the order_number
- Order of the stored data does not matter

The Orders table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Orders table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

Since the orders table has been shown to already be in 3NF no decomposition is required for this table.

Product_Categories Table

Relational Schema:
Product_Categories(product_id, category)

```
CREATE TABLE product_categories (
    product_ID int REFERENCES product(product_ID),
    category VARCHAR2(30) NOT NULL
);
```

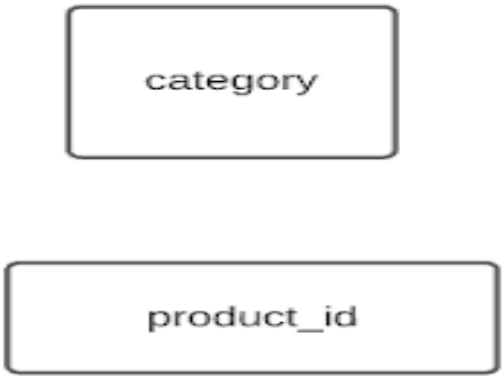
The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

$FD_{product\_categories} = \{\} \rightarrow$ No functional dependencies

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Product_Categories table:

The product_categories table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the product_ID
- Order of the stored data does not matter

The product_categories table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The product_categories table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

Since the product_categories table has been shown to already be in 3NF no decomposition is required for this table.

Cart Table

Relational Schema:

Cart(cart_id, customer_id)

```
CREATE TABLE cart (
    cart_ID int NOT NULL PRIMARY KEY, -- this customer id has that cart id
    customer_ID int REFERENCES Customer(Customer_ID)

);
```

The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

FD_{cart} = {

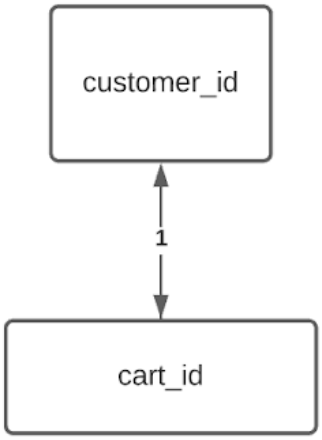
cart_id → customer_id

customer_id → cart_id

}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Cart table:

The Cart table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the cart_ID
- Order of the stored data does not matter

The Cart table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Cart table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

Since the Cart table has been shown to already be in 3NF no decomposition is required for this table.

Cart_Products Table (Example of Decomposition for 2NF)

The cart_products table is a table which does not have a primary key and instead utilizes two foreign keys, cart_id and product_id, to relate the product contents of a cart to the cart entity. Since a cart can have multiple products and the same product can belong to multiple carts we see that there is no functional dependency between the two foreign keys in either direction.

Relational Schema:
Cart_Products(cart_id, product_id, quantity)

For the purposes of this assignment we will modify the structure of the Cart_products table so that we may force the table to not be in 2NF so that we may perform decomposition on it. In order to do this we will add a new attribute to the table, productAvailability which represents the total amount of the product available on the platform. This changes the relational schema described above to the following:

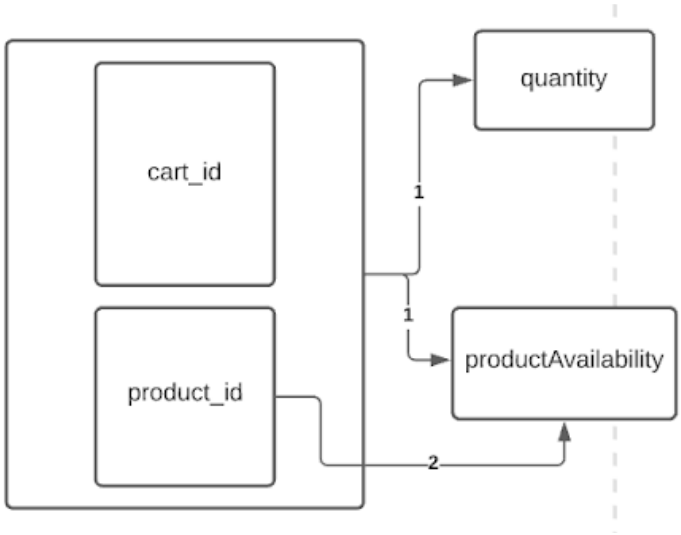
Updated Relational Schema:
Cart_Products(cart_id, product_id, quantity, productAvailability)

This change then affects the functional dependencies of the table and changes the functional dependencies for the Cart_Products table to the following:
Functional Dependencies:

$FD_{cart\_products}$ = {
(cart_id, product_id) → quantity
(cart_id, product_id) → productAvailability
product_id → productAvailability
}

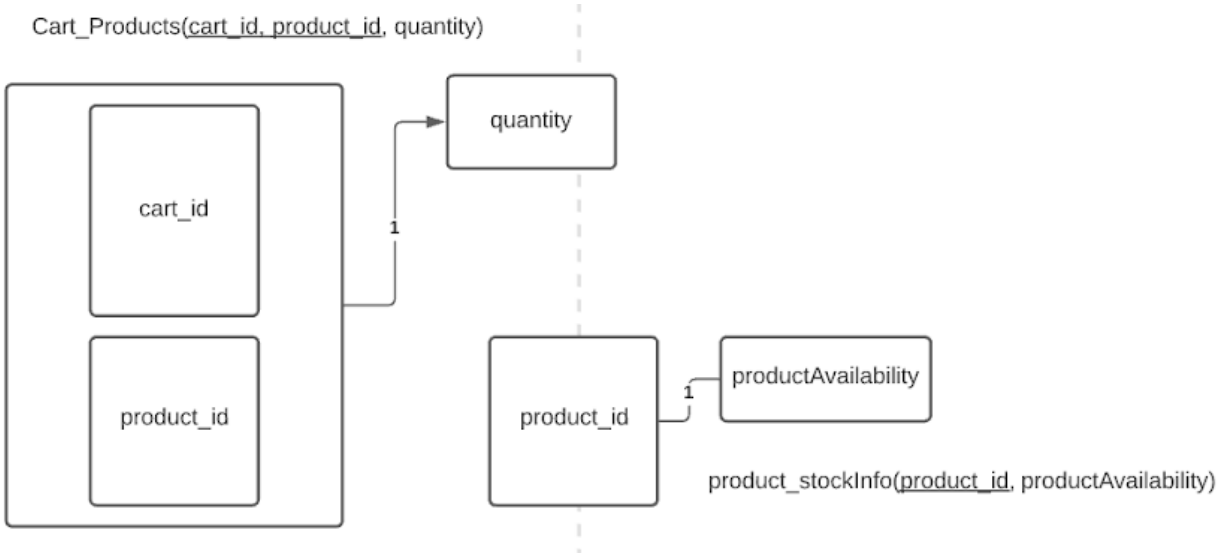
As we can see from the updated functional dependencies for the cart_products table we now have a partial dependency between the primary key for the table, which in this case is the multi-valued key (cart_id, product_id), and the productAvailability attribute. Since we are able to remove the cart_id attribute from the key and still have a functional dependency between the product_id and the productAvailability attributes we know there is a partial dependency present. From these functional dependencies we construct a dependency diagram before any decomposition as follows:

Functional Dependency Diagram



From this diagram along with the functional dependencies we see that this table is only in 1NF and not 2NF since the attribute values are all atomic values but a partial dependency exists. Thus, to put this table into 2NF we must perform decomposition on the table and separate the relationship between the product_id and productAvailability attributes into their own table. In doing this we retrieve the following updated functional dependency diagram:

Updated Functional Dependency Diagram



By applying the above decomposition we have successfully put the cart_products as well as the new product_stockInfo into 2NF and 3NF since there are no transitive dependencies. Therefore, from the above diagram we retrieve the following functional dependencies:

Updated Functional Dependencies:

$FD_{cart\_products}$ = {
(cart_id, product_id) → quantity
}
$FD_{product\_stockInfo}$ = {
product_id → productAvailability
}

Thus, from this diagram of the functional dependencies along with the set of functional dependencies we are able to determine the following information about the normal form for the decomposed Product_Categories table:

The cart_products table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the cart_ID and product_id multi-valued key
- Order of the stored data does not matter

The cart_products table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The cart_products table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

Order_Details Table

The order_details table is another table, similar to the cart_products table, which does not have a primary key and instead utilizes two foreign keys, order_number and product_id, to relate the product contents of an order to the order entity. Since an order can have multiple products and the same product can belong to multiple orders we see that there is no functional dependency between the two foreign keys in either direction.

Relational Schema:

Order_Details(<u>order_number, product_id</u>, quantity)

```
CREATE TABLE order_details (
    order_number int REFERENCES orders(order_number), -- order number relates its details to higher level order info
    product_ID int REFERENCES product(product_ID),
    quantity int NOT NULL


);
```

The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

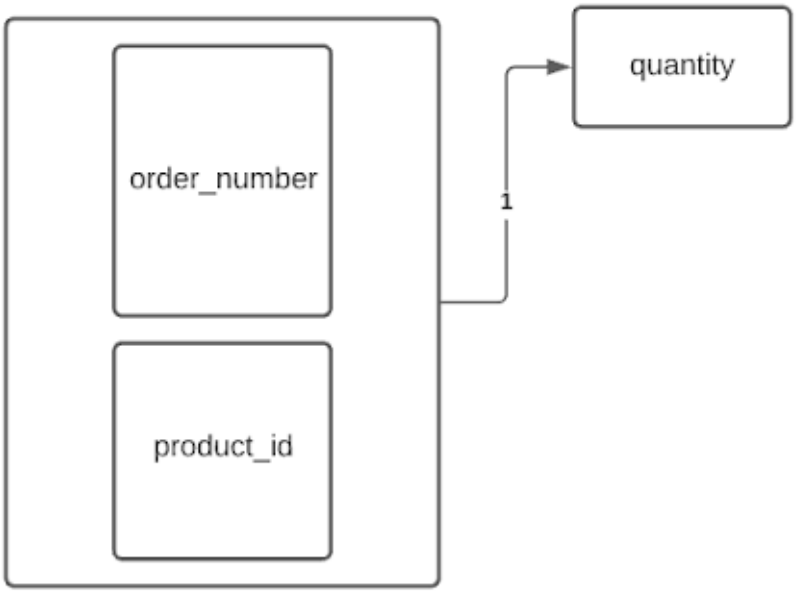
$FD_{order\_details}$ = {

(order_number, product_id) →quantity

}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Cart table:

The Order_Details table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the order_number and product_id multi-valued key
- Order of the stored data does not matter

The Order_Details table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Order_Details table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

Since the Order_Details table has been shown to already be in 3NF no decomposition is required for this table.

Bernstein's Algorithm and BCNF

In this portion, similar to the previous section, we outline the normal forms for each of the tables within our e-commerce database system. Specifically, we will be showing that all of the tables within our system are in BCNF and for some tables we will be introducing some new assumptions or attributes in order to facilitate the demonstration of 3NF derivation using Bernstein's algorithm and BCNF decomposition. To draw attention to these demonstrations the two tables we choose to perform these processes on are presented first along with the demonstration and the remainder of the report outlines how each of the remaining tables already conform to the BCNF guidelines. Also, note that for tables which we had performed decomposition for the previous section, we have kept their decomposition process where possible, i.e., where additional changes did not need to be made for this section.

Customer Table (Example of BCNF Decomposition)

Relational Schema:

Customer(customer_id, customer_name, region, card_holder, card_no, billing_address, delivery_address)

```
CREATE TABLE Customer (
    customer_ID int NOT NUll PRIMARY KEY,
    customer_name VARCHAR2(30),
    region VARCHAR2(30),
    billing_address VARCHAR2(30) NOT NULL,
    delivery_address VARCHAR2(30) NOT NULL
);
```

Functional Dependencies:

$FD_{customer}$ = {

customer_id → customer_name

customer_id → region

customer_id → card_holder

customer_id → card_no

customer_id → billing_address

customer_id → delivery_address

}

For the purposes of this assignment in order to be able to provide an example of decomposition to BCNF for one of our tables we make a slight adjustment to the above schema. The change we make is to add a new attribute postal_code which represents the postal code for the customer and we assume that the postal code uniquely identifies the customer's region attribute. That is, there is a functional dependency, postal_code → region. Therefore, the updated relational schema for this table is presented below:

Updated Relational Schema:
Customer(customer_id, customer_name, region, card_holder, card_no, billing_address, delivery_address, postal_code)

With this updated relational schema created from the newly added attribute as well as the assumption outlined above we determine the updated functional dependencies of this table to be as follows:

Functional Dependencies:

$FD_{customer}$ = {
customer_id → customer_name
customer_id → region
customer_id → card_holder
customer_id → card_no
customer_id → billing_address
customer_id → delivery_address
customer_id → postal_code
postal_code→ region
}

Now, checking each of the LHS of the functional dependencies presented above we determine that customer_id is a key, however, postal_code is not. This is determined using the closure for the two attributes as follows:

$Customer\_id^+$ = {customer_id, customer_name, region, card_holder, card_no, billing_address, delivery_address, postal_code}

$Postal\_code^+$ = {postal_code, region}

Since the closure of customer_id contains all attributes for the relation it is a key however, the closure for postal_code does not and therefore is not a key. Thus, BCNF is violated due to the functional dependency of postal_code→ region.

With this we can now perform decomposition according to the rules for BCNF and form two new relations which we will call R1 and R2 in which we separate the dependency violating BCNF into R1 and the remaining dependencies into R2. Then, the relational schemas of R1 and R2 are as follows:

R1(postal_code, region)

R2(customer_id, customer_name, card_holder, card_no, billing_address, delivery_address, postal_code)

Now, we know that R1 is in BCNF but we must check once again that R2 is in BCNF or else we must perform further decomposition. We check this the same way as before by using the functional dependencies and closures for the LHS attributes:

Functional Dependencies:

FD_{R1} = {
customer_id → customer_name
customer_id → card_holder
customer_id → card_no
customer_id → billing_address
customer_id → delivery_address
customer_id → postal_code
}

$Customer\_id^{+}$ = {customer_id, customer_name, card_holder, card_no, billing_address, delivery_address, postal_code}

From the above closures we see that R1 is in fact in BCNF as the only attribute on the LHS of the functional dependencies (customer_id) is also a key since all attributes are contained within its closure.

From the functional dependencies of the table we are able to see that the following is true for the normal forms of the customer table:

The Customer table is in the 1st normal form (1NF) because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the customer_ID
- Order of the stored data does not matter

The Customer table is in the 2nd normal form (2NF) because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key, no partial dependencies

The Customer table is in 3rd normal form (3NF) because:

- Table is in 2nd normal form
- No transitive dependencies

The Customer table is in BCNF:

Since the customer table has now also been decomposed above into the two relations R1 and R2 where all functional dependencies between attributes have a key as the determinant we now also see that the customer table has been successfully decomposed into BCNF.

Seller Table (Example of Decomposition for 3NF: Bernstein's Algorithm)

Relational Schema:

Seller(<u>seller_id</u>, seller_name, seller_email)

```
CREATE TABLE Seller (
    seller_ID int NOT NUll PRIMARY KEY AUTO_INCREMENT,
    seller_name VARCHAR2(30),
    seller_email  VARCHAR2(30)

);
```

For the purposes of this assignment in order to perform decomposition on this table we will assume that the seller_email attribute for each seller must be unique, that is no two sellers can share the same email. With this assumption we then can see that a transitive functional dependency as now seller_email → seller_name and seller_id → seller_email. The functional dependencies for this table are shown below.

Functional Dependencies:

FD_{seller} = {

seller_id → seller_name

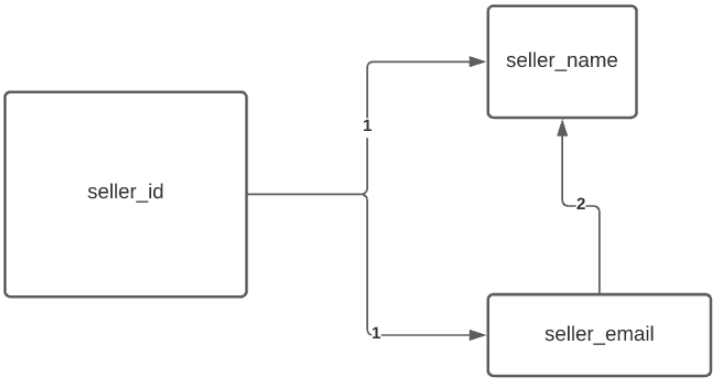
seller_id → seller_email

seller_email → seller_name

}

With this we construct the functional dependency diagram for the seller table as follows:

Functional Dependency Diagram



From this diagram we are able to determine the following information for the normal form of the seller table before any decomposition:

The Seller table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the seller_ID
- Order of the stored data does not matter

The Seller table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Seller table is NOT in 3rd normal form because:

- There is a Transitive dependency between seller_email and seller_name

Now, to put this relation into 3NF instead of performing decomposition via a functional dependency diagram as we did last time we will now apply Bernstein's algorithm to this table in order to obtain a decomposed 3NF relation which is lossless join and dependency preserving. The steps of this process are outlined below:

Step 1: Determine The Functional Dependencies

As we have shown above for the seller table with the added assumption for the seller_email the seller table has the following functional dependencies:

FD_{seller} = {
seller_id → seller_name
seller_id → seller_email
seller_email → seller_name
}

Step 2a: Break RHS and find redundancies

Since the RHS of our functional dependencies shown above are already broken apart as much as possible we do not need to separate them further, thus we can begin to look for redundancies using closure for the functional dependencies:

FD_{seller} = {
seller_id → seller_name
seller_id → seller_email
seller_email → seller_name
}

seller_id → seller_name: $seller\_id^+$ = {seller_id, seller_email, seller_name} → redundancy (since we get seller_name)

seller_id → seller_email: $seller\_id^+$ = {seller_id, seller_name} → No redundancy (Do not get seller_email)

seller_email → seller_name: $seller\_email^+$ = {seller_email} → No redundancy

Now, we are able to remove all the redundant dependencies we have just found and in doing so we obtain the new functional dependencies to be:

FD_{seller} = {
seller_id → seller_email
seller_email → seller_name
}

Step 2b: Find and remove partial dependencies(or minimizing LHS)

Since, the LHS of all dependencies are all single attributes and not composite of any kind it is seen that no partial dependencies exist within this set of functional dependencies. To further illustrate this we will once again look at the closures for the attributes without the mentioned FD:

seller_id → seller_email: $seller\_id^+$={seller_id}
seller_email → seller_name: $seller\_email^+$={seller_email}

Since each LHS contains only one attribute it can be seen that it is not possible for this relation to have any partial dependencies.

Step 3: Find Keys:

In order to perform this task we will follow the provided formula for simplifying the process as follows:

- If an attribute is not in LHS and is not in LHR: it is part of the key
- If an attribute is only in LHS but not in RHS: it is part of the key
- If an attribute is only in RHS and not in LHS: it is NOT part of the key

Applying the above rules to the functional dependencies for our seller table it can be seen that the attribute seller_name only appears on the RHS and never on the LHS and thus {seller_name} is not part of the key. For the seller_id attribute we see that it only appears on the LHS and never on the RHS and thus {seller_id} is part of the key.

Thus, we see that {seller_id} should be part of any keys for the seller table, but, now we must consider the combination of attributes for keys, we only consider the attribute seller_email since we have shown that seller_name is not a part of the key therefore:

$Seller\_id^{+}$ = {seller_id, seller_email, seller_name} → this is a key
$Seller\_id,Seller\_email^{+}$ = {seller_id, seller_email, seller_name} → this is a key

Step 4: Make Tables

Finally, using the reduced functional dependencies we will create a table for each dependency and if necessary we will create a relation to ensure the primary key is included. Recall the functional dependencies:

FD_{seller} = {
seller_id → seller_email
seller_email → seller_name
}

Thus, we create the following relation tables to decompose into 3NF:

R1(seller_id, seller_email) with FD: seller_id → seller_email

R2(seller_email, seller_name) with FD: seller_email → seller_name

Since, R1 already includes one of the keys we found above, being {seller_id}, there is no need to add a relationship for the key to the decomposition. With this the seller table has been successfully decomposed into 3NF using the Bernstein Algorithm.

The Seller table is in the 1st normal form (1NF) because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the customer_ID
- Order of the stored data does not matter

The Seller table is in the 2nd normal form (2NF) because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Seller table is in 3rd normal form (3NF) because:

- Table is in 2nd normal form
- No transitive dependencies

The Seller table is in BCNF:

To show that the two relations are also in BCNF we look again at the closures of the two relations:

R1: FD: {seller_id → seller_email }

$Seller\_id^+$ = {seller_id, seller_email} ; Thus, seller_id is a key

$Seller\_email^+$ = {seller_email} ; Thus, seller_email is not a key

Since all the LHS' for R1's Functional dependencies are keys we see that this table is in BCNF as there are no non key attributes determining another attribute

R2: FD: {seller_email → seller_name}

$Seller\_email^+$ = {seller_email, seller_name} ; Thus, seller_email is a key

$Seller\_name^+$ = {seller_name} ; Thus, seller_name is not a key

Since all the LHS' for R2's Functional dependencies are keys we see that this table is in BCNF as there are no non key attributes determining another attribute

Product Table

Relational Schema:

Product(product_id, seller_id, unit_price, product_name, product_description)

```
CREATE TABLE Product (
    product_ID int NOT NUll PRIMARY KEY AUTO_INCREMENT,
    unit_price float NOT NULL,
    description_ VARCHAR2(30) NOT NULL
);
```

The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

$FD_{product}$ = {

product_id→ unit_price

product_id→ product_name

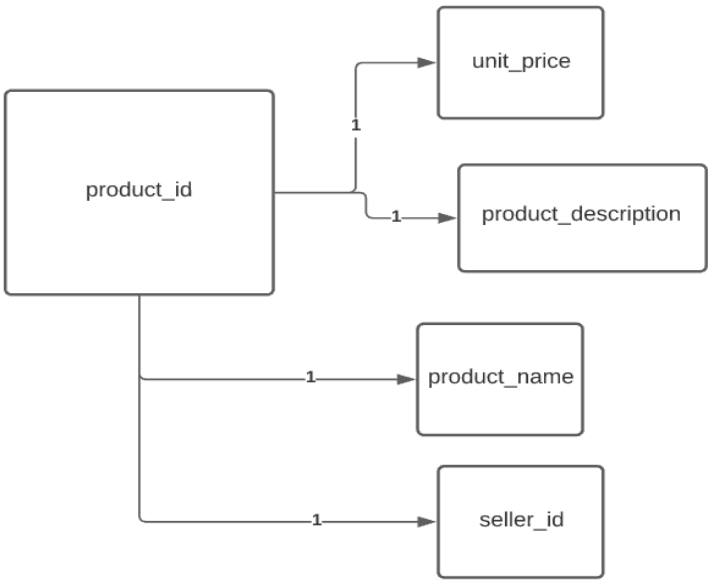
product_id→ product_description

product_id→ seller_id

}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the product table:

The Product table is in the 1st normal form (1NF) because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the product_ID
- Order of the stored data does not matter

The Product table is in the 2nd normal form (2NF) because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Product table is in 3rd normal form (3NF) because:

- Table is in 2nd normal form
- No transitive dependencies

The Product table is in BCNF:

Since the product table has been shown to already be in 3NF we can now analyze the functional dependencies to determine whether this table is in BCNF. In order to do this we look at the closure for all the attributes which appear on the LHS of a functional dependency for this relation. If all these attributes are keys then we are in BCNF. Since, only one attribute appears on the LHS (product_id) we only consider this attribute:

$Product\_id^{+}$ = {product_id, unit_price, product_name, seller_id, product_description} → since product_id has a functional dependency with all attributes (product_id is a key)

Therefore, it can be seen from the closure of the Product_id attribute that it is indeed a key as the closure includes all attributes of the relation. Since this is the only attribute we must consider, we see that the product table is in BCNF.

Category Table

Relational Schema:

Category(<u>category_id</u>, category_tag)

```
CREATE TABLE Category (
    category_ID int NOT NUll PRIMARY KEY,
    category_tag VARCHAR2(30)

);
```

The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

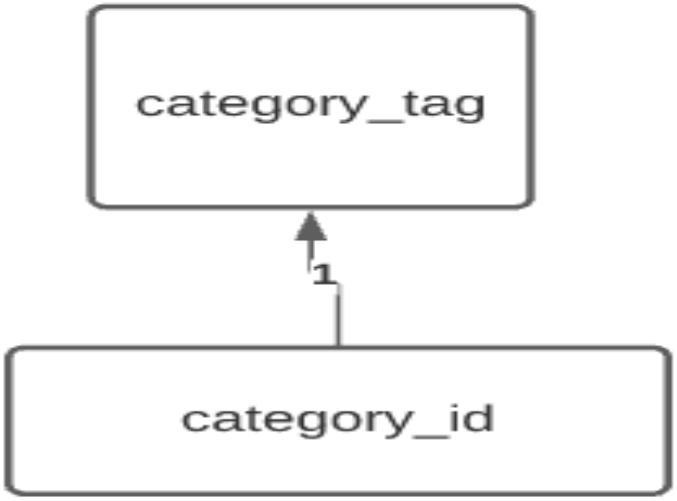
$FD_{category}$ = {

category_id→ category_tag

}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Category table:

The Category table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the category_ID
- Order of the stored data does not matter

The Category table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Category table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

The Category table is in BCNF:

Since the category table has been shown to already be in 3NF we can now analyze the functional dependencies to determine whether this table is in BCNF. In order to do this we look at the closure for all the attributes which appear on the LHS of a functional dependency for this relation. If all these attributes are keys then we are in BCNF. Since, only one attribute appears on the LHS (category_id) we only consider this attribute:

Category_id^{+} = {category_id, category_tag} → since category_id has a functional dependency with all attributes (category_id is a key)

Therefore, it can be seen from the closure of the Category_id attribute that it is indeed a key as the closure includes all attributes of the relation. Since this is the only attribute we must consider, we see that the category table is in BCNF.

Shipper Table

Relational Schema:
Shipper(<u>shipper_id</u>, service_provider)

```
CREATE TABLE shipper (
    shipper_ID int NOT NUll PRIMARY KEY,
    service_provider VARCHAR2(30)

);
```

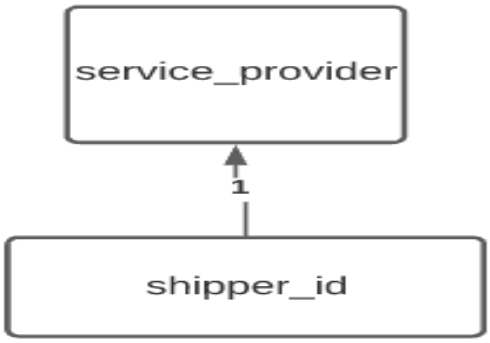
The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

$FD_{shipper}$ = {
shipper_id → service_provider
}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Shipper table:

The shipper table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the shipper_ID
- Order of the stored data does not matter

The shipper table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The shipper table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

The Shipper table is in BCNF:

Since the shipper table has been shown to already be in 3NF we can now analyze the functional dependencies to determine whether this table is in BCNF. In order to do this we look at the closure for all the attributes which appear on the LHS of a functional dependency for this relation. If all these attributes are keys then we are in BCNF. Since, only one attribute appears on the LHS (shipper_id) we only consider this attribute:

Shipper_id$^+$ = {shipper_id, service_provider} → since shipper_id has a functional dependency with all attributes (shipper_id is a key)

Therefore, it can be seen from the closure of the Shipper_id attribute that it is indeed a key as the closure includes all attributes of the relation. Since this is the only attribute we must consider, we see that the shipper table is in BCNF.

Orders Table

Relational Schema:

Orders(<u>order_number</u>, customer_id, shipper_id, order_date)

```
CREATE TABLE orders (
    order_number int NOT NUll PRIMARY KEY,
    customer_ID int REFERENCES Customer(customer_ID),
    shipper_ID int REFERENCES Shipper(shipper_ID),
    order_date date
);
```

The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

FD_{orders} = {

<u>order_number</u>→ customer_id

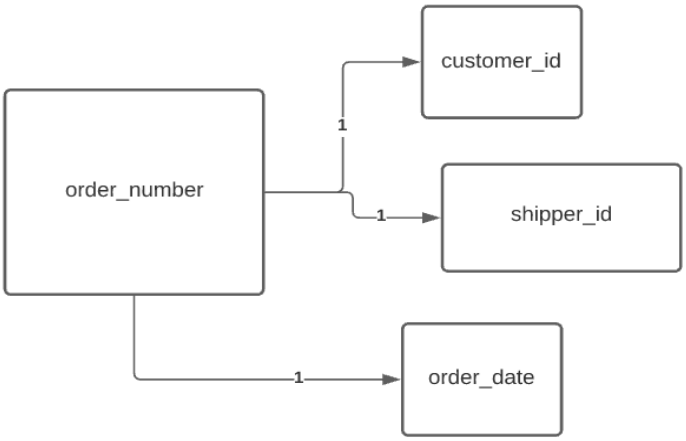
<u>order_number</u>→ shipper_id

<u>order_number</u>→ order_date

}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Orders table:

The Orders table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the order_number
- Order of the stored data does not matter

The Orders table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Orders table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

The Order table is in BCNF:

Since the orders table has been shown to already be in 3NF we can now analyze the functional dependencies to determine whether this table is in BCNF. In order to do this we look at the closure for all the attributes which appear on the LHS of a functional dependency for this relation. If all these attributes are keys then we are in BCNF. Since, only one attribute appears on the LHS (order_number) we only consider this attribute:

Order_number^{+} = {order_number, customer_id, shipper_id, order_date} → since order_number has a functional dependency with all attributes (order_number is a key)

Therefore, it can be seen from the closure of the Order_number attribute that it is indeed a key as the closure includes all attributes of the relation. Since this is the only attribute we must consider, we see that the orders table is in BCNF.

Product_Categories Table

Relational Schema:
Product_Categories(product_id, category)

```
CREATE TABLE product_categories (
    product_ID int REFERENCES product(product_ID),
    category VARCHAR2(30) NOT NULL
);
```

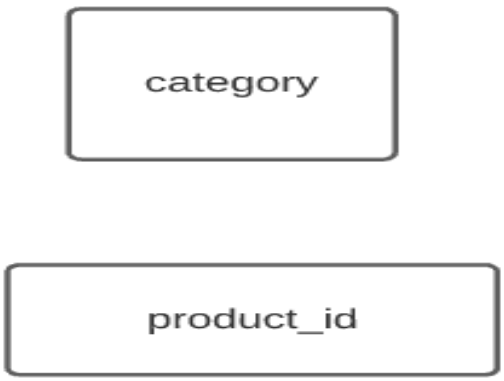
The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

$FD_{product\_categories}$ = {} → No functional dependencies

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Product_Categories table:

The product_categories table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the product_ID
- Order of the stored data does not matter

The product_categories table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The product_categories table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

The Product_categories table is in BCNF:

Since the product_categories table has been shown to already be in 3NF we can now analyze the functional dependencies to determine whether this table is in BCNF. In order to do this we look at the closure for all the attributes which appear on the LHS of a functional dependency for this relation. If all these attributes are keys then we are in BCNF. Since, there are no functional dependencies for this table then by default the key is (product_id, category) and the table is in BCNF

Therefore, it can be seen that the Product_categories table is indeed in BCNF.

Cart Table

Relational Schema:

Cart(<u>cart_id</u>, customer_id)

```
CREATE TABLE cart (
    cart_ID int NOT NULL PRIMARY KEY, -- this customer id has that cart id
    customer_ID int REFERENCES Customer(Customer_ID)

);
```

The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

FD_{cart} = {

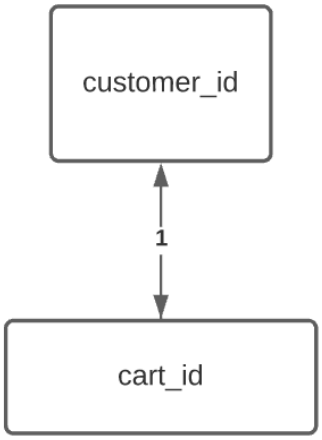
cart_id → customer_id

customer_id → cart_id

}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Cart table:

The Cart table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the cart_ID
- Order of the stored data does not matter

The Cart table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Cart table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

The Cart table is in BCNF:

Since the orders table has been shown to already be in 3NF we can now analyze the functional dependencies to determine whether this table is in BCNF. In order to do this we look at the closure for all the attributes which appear on the LHS of a functional dependency for this relation. If all these attributes are keys then we are in BCNF. Since, only two attributes appear on the LHS (cart_id and customer_id) we only consider these attributes:

$Cart\_id^+$ = {cart_id, customer_id} → since cart_id has a functional dependency with all attributes (cart_id is a key)

$Customer\_id^+$ = {customer_id, cart_id} → since customer_id has a functional dependency with all attributes (customer_id is a key)

Therefore, it can be seen from the closure of the cart_id and customer_id attributes that they are both indeed keys as the closure includes all attributes of the relation. Since these are the only attributes on the LHS of the functional dependencies, we see that the cart table is in BCNF.

Cart_Products Table (Example of Decomposition for 2NF)

The cart_products table is a table which does not have a primary key and instead utilizes two foreign keys, cart_id and product_id, to relate the product contents of a cart to the cart entity. Since a cart can have multiple products and the same product can belong to multiple carts we see that there is no functional dependency between the two foreign keys in either direction.

Relational Schema:
Cart_Products(<u>cart_id, product_id</u>, quantity)

For the purposes of this assignment we will modify the structure of the Cart_products table so that we may force the table to not be in 2NF so that we may perform decomposition on it. In order to do this we will add a new attribute to the table, productAvailability which represents the total amount of the product available on the platform. This changes the relational schema described above to the following:

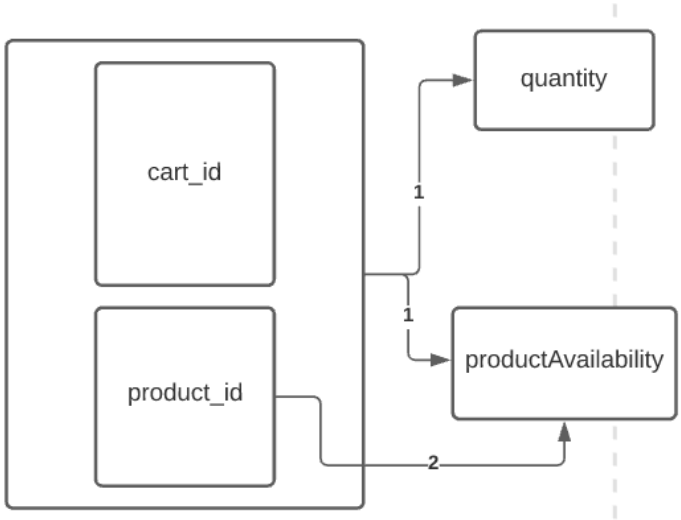
Updated Relational Schema:
Cart_Products(<u>cart_id, product_id</u>, quantity, productAvailability)

This change then affects the functional dependencies of the table and changes the functional dependencies for the Cart_Products table to the following:
Functional Dependencies:

$FD_{cart\_products}$ = {
(cart_id, product_id) → quantity
(cart_id, product_id) → productAvailability
product_id → productAvailability
}

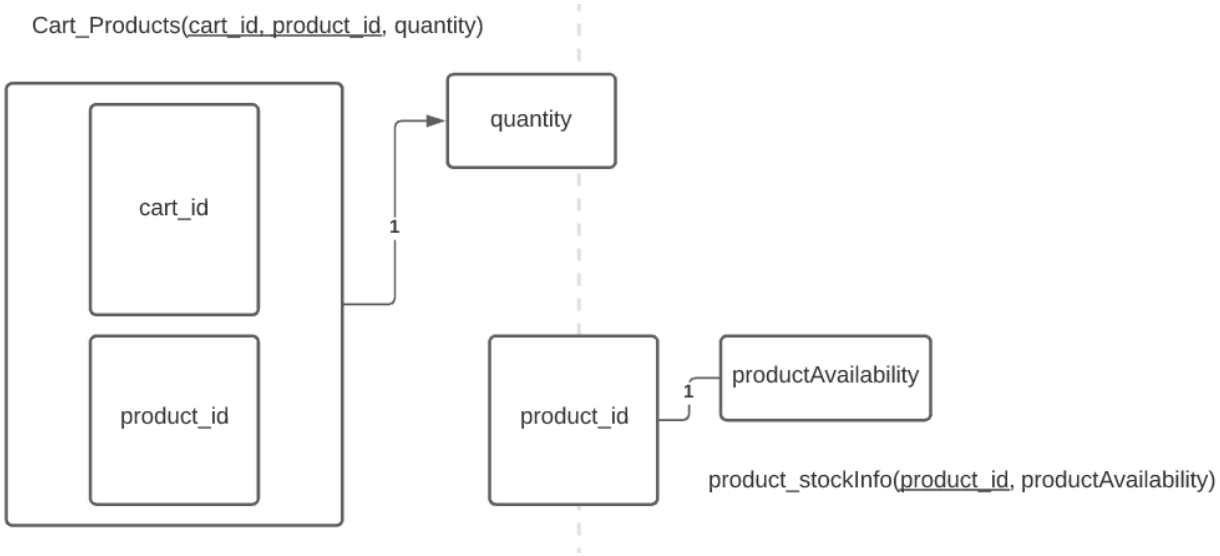
As we can see from the updated functional dependencies for the cart_products table we now have a partial dependency between the primary key for the table, which in this case is the multi-valued key (cart_id, product_id), and the productAvailability attribute. Since we are able to remove the cart_id attribute from the key and still have a functional dependency between the product_id and the productAvailability attributes we know there is a partial dependency present. From these functional dependencies we construct a dependency diagram before any decomposition as follows:

Functional Dependency Diagram



From this diagram along with the functional dependencies we see that this table is only in 1NF and not 2NF since the attribute values are all atomic values but a partial dependency exists. Thus, to put this table into 2NF we must perform decomposition on the table and separate the relationship between the product_id and productAvailability attributes into their own table. In doing this we retrieve the following updated functional dependency diagram:

Updated Functional Dependency Diagram



By applying the above decomposition we have successfully put the cart_products as well as the new product_stockInfo into 2NF and 3NF since there are no transitive dependencies. Therefore, from the above diagram we retrieve the following functional dependencies:

Updated Functional Dependencies:

$FD_{cart\_products}$ = {
(cart_id, product_id) → quantity
}
$FD_{product\_stockInfo}$ = {
product_id → productAvailability
}

Thus, from this diagram of the functional dependencies along with the set of functional dependencies we are able to determine the following information about the normal form for the decomposed Product_Categories table:

The cart_products table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the cart_ID and product_id multi-valued key
- Order of the stored data does not matter

The cart_products table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The cart_products table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

The Cart_proudcts table is in BCNF:

Since the cart_products table has been shown to already be in 3NF we can now analyze the functional dependencies to determine whether the two decomposed tables are in BCNF. In order to do this we look at the closure for all the attributes which appear on the LHS of a functional dependency for the two decomposed relations. If all these attributes are keys then we are in BCNF for both relations. First we will consider the table cart_products. Since, only one composite attribute appears on the LHS ((cart_id, product_id)) we only consider this attribute:

$(Cart\_id, Product\_id)^+$ = {cart_id, product_id, quantity} → since (Cart_id, Product_id) has a functional dependency with all attributes ((Cart_id, Product_id) is a key)

Therefore, it can be seen from the closure of the composite (cart_id, product_id) attribute that this is indeed a key as the closure includes all attributes of the relation. Since this is the only attribute on the LHS of the functional dependencies, we see that the decomposed cart_products table is in BCNF.

Next, we will consider the table product_stockInfo. Since, only one attribute appears on the LHS (product_id) we only consider this attribute:

$Product\_id^+$ = {product_id, productAvailability} → since product_id has a functional dependency with all attributes of this relation (product_id is a key)

Therefore, it can be seen from the closure of the product_id attribute that this is indeed a key as the closure includes all attributes of the relation. Since this is the only attribute on the LHS of the functional dependencies, we see that the product_stockInfo table is in BCNF.

Since both of the decomposed tables for the larger cart_products table is in BCNF we see that the cart_products table has been successfully decomposed into BCNF.

Order_Details Table

The order_details table is another table, similar to the cart_products table, which does not have a primary key and instead utilizes two foreign keys, order_number and product_id, to relate the product contents of an order to the order entity. Since an order can have multiple products and the same product can belong to multiple orders we see that there is no functional dependency between the two foreign keys in either direction.

Relational Schema:

Order_Details(order_number, product_id, quantity)

```
CREATE TABLE order_details (
    order_number int REFERENCES orders(order_number), -- order number relates its details to higher level order info
    product_ID int REFERENCES product(product_ID),
    quantity int NOT NULL


);
```

The functional dependencies determined in the previous assignment for this table are presented once again below along with a diagram to facilitate the visualization of the tables normal forms and any decomposition which must be performed.

Functional Dependencies:

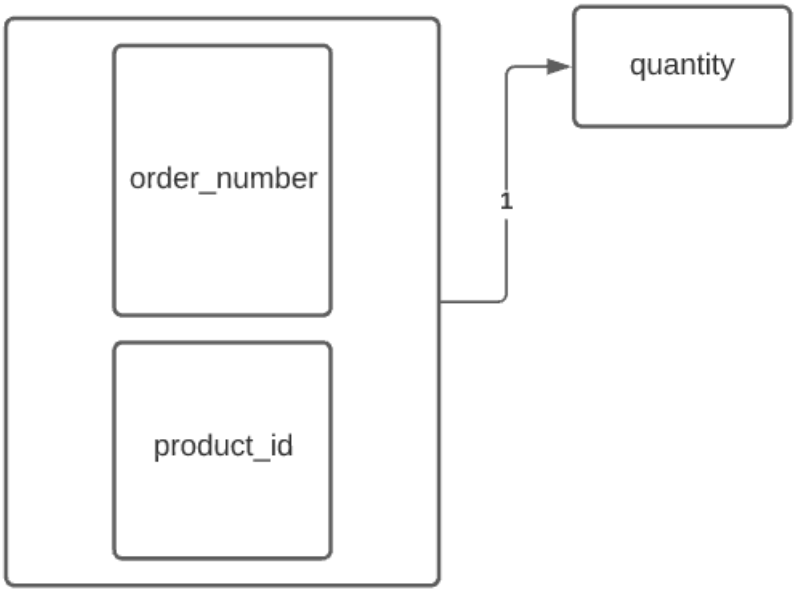
$FD_{order\_details}$ = {

(order_number, product_id) →quantity

}

Using the identified functional dependencies we create a diagram similar to the one presented in the lectures for this table, this is shown below.

Functional Dependency Diagram



From this diagram of the functional dependencies we are able to determine the following information about the normal form for the Cart table:

The Order_Details table is in the 1st normal form because:

- Each attribute is single valued
- Values of each attribute are of the same type
- Each row in the Product table is uniquely identified by the order_number and product_id multi-valued key
- Order of the stored data does not matter

The Order_Details table is in the 2nd normal form because:

- Table is in 1st normal form
- All attributes (non-key columns) dependent on the key

The Order_Details table is in 3rd normal form because:

- Table is in 2nd normal form
- No transitive dependencies

The Order_details table is in BCNF:

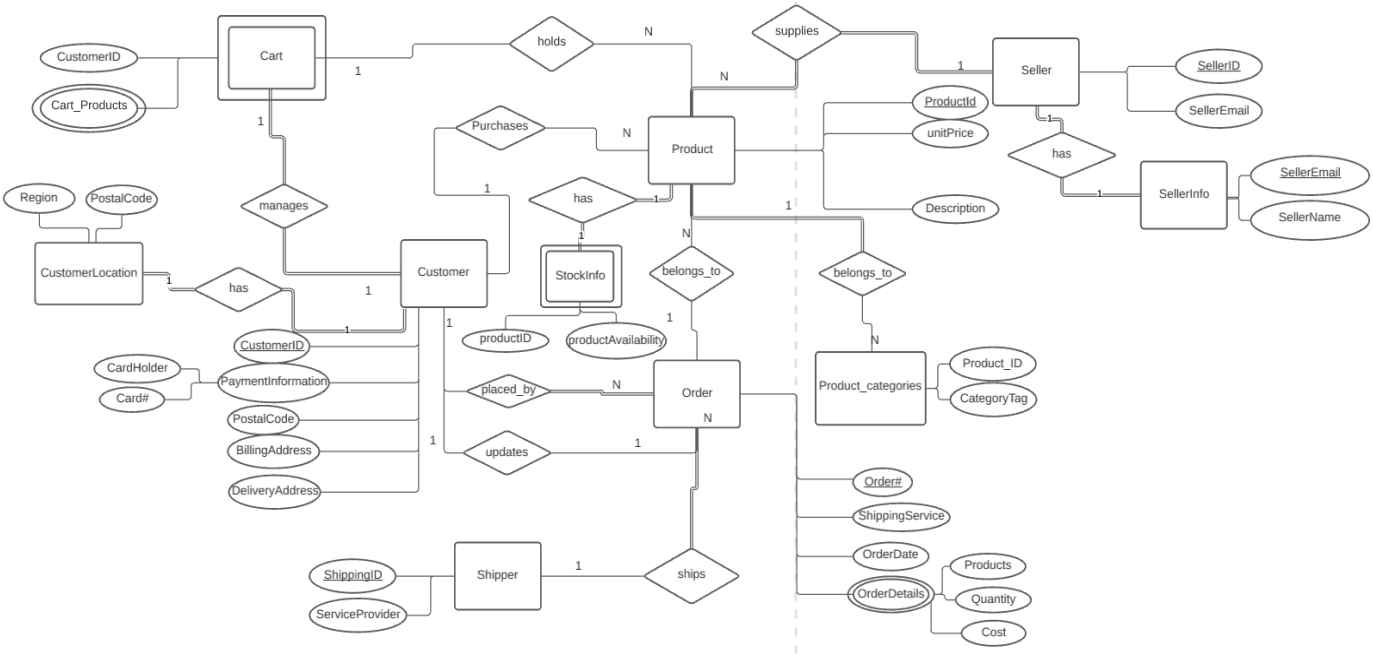
Since the order_details table has been shown to already be in 3NF we can now analyze the functional dependencies to determine whether this table is in BCNF. In order to do this we look at the closure for all the attributes which appear on the LHS of a functional dependency for this relation. If all these attributes are keys then we are in BCNF. Since, only one composite attribute appears on the LHS ((order_number, product_id)) we only consider this attribute:

$(\text{Order\_number, Product\_id})^{+}$ = {order_number, product_id, quantity} → since composite attribute (Order_number, Product_id) has a functional dependency with all attributes ((Order_number, Product_id) is a key)

Therefore, it can be seen from the closure of the (Order_number, Product_id) attribute that it is indeed a key as the closure includes all attributes of the relation. Since this is the only attribute we must consider, we see that the order_details table is in BCNF.

Updated System ER Model

In this section we present the updated ER model for the overall system with the assumptions used in the previous normalization sections. We do not present the relational schemas for each individual entity since this is presented in the above normalization sections instead this section is used to present the overall view of the system under the normalization assumptions. However, we will highlight the major differences between this model and the original model in terms of the added entities and relations created by the assumptions and new attributes from the normalization section. The ER diagram for the normalized system is shown below:



The first major change from the previously shown ER model for the system is the addition of the StockInfo entity and its relationship with the product entity. This entity and relation is introduced in order to accommodate the new productAvailabilty attribute shown in the decomposition of the cart_products table. The relation associates each product with its corresponding stock information. See the normalization section for more in depth information as to why this has been introduced.

Relational Schema:

StockInfo(Product_ID, productAvailability)

The second change from the previously shown ER model for the system is the addition of the SellerInfo entity and its relationship with the Seller entity. This entity and relation is

introduced in accordance with the assumption made in the normalization stage that the Seller_Email attribute uniquely identifies the Seller_Name attribute. The relation associates each seller entity with its corresponding seller information with the primary key as the Seller_Email attribute. See the normalization section for more in depth information as to why this has been introduced.

Relational Schema:

SellerInfo(Seller_Email, Seller_Name)

The final major change between the original diagram to our normalized diagram is the addition of the CustomerLocation entity and its relationship with the Customer entity. This entity and relation is introduced in order to accommodate the new Postal_Code attribute shown in the decomposition of the Customer table in the normalization section. This entity and relation is introduced in order to facilitate the BCNF form of the customer table with the new Postal_Code attribute which is assumed to uniquely identify the region attribute. The relation associates each Customer entity with their corresponding LocationInformation, in other words, their region.

Relational Schema:

CustomerLocation(Postal_Code, Region)

Query Relational Algebra

In this section of the documentation we present the list of SQL queries that we have used in the past assignments along with their associated Relational Algebra translations. Note that below the operator "||" is used as the Natural Join operation.

SQL Query	Relational Algebra
SELECT c.customer_id AS CID, c.customer_name AS CNAME, c.region AS CREGION, c.billing_address AS Bill_Addr, c.delivery_address AS Deliv_addr FROM customer c WHERE EXISTS (SELECT o.order_number FROM orders o WHERE o.customer_id = c.customer_id) GROUP BY c.customer_id, c.customer_name, c.region, c.billing_address, c.delivery_address ORDER BY c.customer_id ASC;	**Π**customer_id, customer_name, region, billing_address, delivery_address (***σ*** (Customer || Orders))
SELECT p.product_id, p.product_name, p.product_description, p.unit_price FROM product p WHERE EXISTS (SELECT pc.category FROM product_categories pc WHERE pc.product_id = p.product_id AND pc.category = 'Dairy') UNION (SELECT p1.product_id, p1.product_name, p1.product_description, p1.unit_price FROM product p1 WHERE EXISTS (SELECT s.seller_id FROM seller s WHERE s.seller_name = 'Christian Daniel' AND p1.seller_id = s.seller_id));	DairyProducts -> **Π**product_id, product_name, product_description, unit_price (***σ***category = 'Dairy' (Product || Product_Categories)) SellerProducts -> **Π**product_id, product_name, product_description, unit_price (***σ***seller_name = 'Christian Daniel' (Product || Seller)) Result -> DairyProducts ∪ SellerProducts
SELECT product_id, product_name, product_description, unit_price	Products -> **Π**product_id, product_name, product_description, unit_price(***σ*** (product))

FROM product MINUS (SELECT p.product_id, p.product_name, p.product_description, p.unit_price FROM product p, order_details OD WHERE OD.product_id = p.product_id);	OrderedProducts -> **Π**product_id, product_name, product_description, unit_price(***σ*** (product || order_details)) Result -> Products - OrderedProducts
SELECT C.customer_id, C.customer_name, COUNT(DISTINCT OD.product_id) AS NUM_PURCHASED_PRODUCTS FROM customer C, order_details OD WHERE EXISTS (SELECT O.order_number, O.customer_id FROM orders O WHERE C.customer_id = O.customer_id AND OD.order_number = O.order_number) GROUP BY C.customer_id, C.customer_name;	Customer_ID, Customer_name **F** COUNT product_id(Customer || Order_Details || Orders)
SELECT C.customer_id, C.customer_name, AVG(OD.quantity) AS AVG_Purchase_Quantity FROM customer C, order_details OD WHERE EXISTS (SELECT O.order_number, O.customer_id FROM orders O WHERE C.customer_id = O.customer_id AND OD.order_number = O.order_number) GROUP BY C.customer_id, C.customer_name;	Customer_ID, Customer_name **F** AVERAGE quantity (Customer || Order_Details || Orders)
SELECT customer_name, COUNT(O.order_number) AS Num_Orders FROM customer C, orders O WHERE C.customer_id = O.customer_id GROUP BY customer_name ORDER BY Num_Orders DESC;	Customer_name, **F** COUNT order_number (Customer || Orders)
SELECT customer_name, COUNT(OD.order_number) AS Num_Ordered FROM customer c, product p, orders O,	PlaystationProductSet -> ***σ***product_name=’Playstation 5’ (Product) Result -> Customer_name, **F** COUNT

order_details OD WHERE p.product_name = 'Playstation 5' AND O.order_number = OD.order_number AND p.product_id = OD.product_id AND c.customer_id = O.customer_id GROUP BY customer_name;	order_number(Orders || Order_Details || Product || Customer)
SELECT seller_name, SUM(OD.quantity) AS NUM_SOLD_PRODUCTS FROM seller s, product p, order_details OD WHERE p.product_id = OD.product_id AND p.seller_id = s.seller_id GROUP BY seller_name ORDER BY NUM_SOLD_PRODUCTS DESC;	Seller_name, **F** SUM quantity (Product || Order_Details || Seller)
SELECT DISTINCT product_name, product_description, unit_price AS Price FROM Product ORDER BY Price ASC;	**Π**product_name, product_description, unit_price(***σ*** (orders))
SELECT category, COUNT(product_ID) AS NumOfProducts FROM product_categories GROUP BY category ORDER BY NumOfProducts ASC;	Category, **F** COUNT product_ID (Product_Categories)
SELECT DISTINCT order_date, customer_id, order_number FROM orders WHERE order_date = '04-NOV-21' ORDER BY order_number;	**Π**order_date, customer_id, order_number(***σ***order_date='04-NOV-21' (orders))
SELECT DISTINCT seller.seller_ID, seller.seller_name, product.product_name, product.product_description, product.unit_price FROM Seller, Product WHERE seller.seller_ID = product.seller_id ORDER BY product.unit_price;	**Π**seller.seller_id, seller.seller_name, product.product_name, product.product_description, product.unit_price(***σ***seller.seller_id=product.seller_id (Product || Seller))

SELECT product_ID, SUM(quantity) AS NumOrdered FROM order_details GROUP BY product_id ORDER BY NumOrdered DESC;	product_ID, **F** SUM quantity (order_details)
SELECT cart.customer_ID, cart.cart_ID, product_ID, SUM(quantity) AS NumOfProduct FROM cart_products, cart WHERE cart.cart_ID = cart_products.cart_ID GROUP BY cart.customer_ID, cart.cart_ID, product_ID ORDER BY NumOfProduct DESC;	product_ID, cart.customer_ID, cart.cart_ID, **F** SUM quantity (Cart_products || Cart)
SELECT orders.shipper_ID, shipper.service_provider, COUNT(orders.shipper_ID) AS NumOrdersShipped FROM orders, shipper WHERE orders.shipper_ID = shipper.shipper_ID GROUP BY orders.shipper_ID, shipper.service_provider ORDER BY NumOrdersShipped;	orders.shipper_ID, shipper.service_provider **F** COUNT Shipper_id (Orders || Shipper)

E-Commerce System UI

In this final section of the documentation instructions for compilation and usage of the developed UI for the project is outlined. Functionality of the UI along with screenshots of example usage for the UI application.

Compilation and Usage

Important Note: The menu is developed to connect to a local db for this reason the credentials in the dburl must be changed to appropriate values when attempting to run the menu. Once you have connected to your local DB running the create tables command will initialize all necessary tables for the system and allow you to populate with the data you desire and run queries.

Compiling the Menu:
To compile the menu for the e-commerce db system use the command provided below from within the project directory:

```
javac -cp ojdbc8.jar; EcommerceUI.java
```

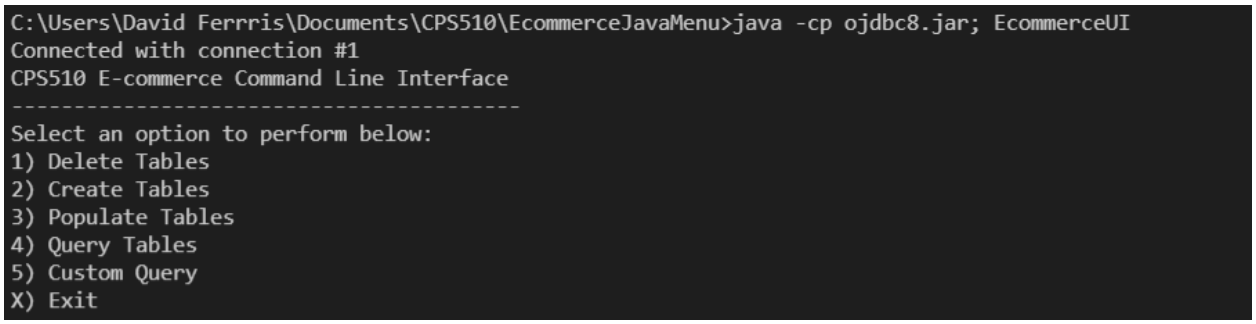
Running the Menu:
To run the menu for the e-commerce db system use the command provided below from within the project directory:

```
java -cp ojdbc8.jar; EcommerceUI
```

Functionality and Screenshots

Screenshots of the running UI application is presented below along with a description of the image and the functionality demonstrated. The images and corresponding descriptions below fully describe the provided functionality of the application.

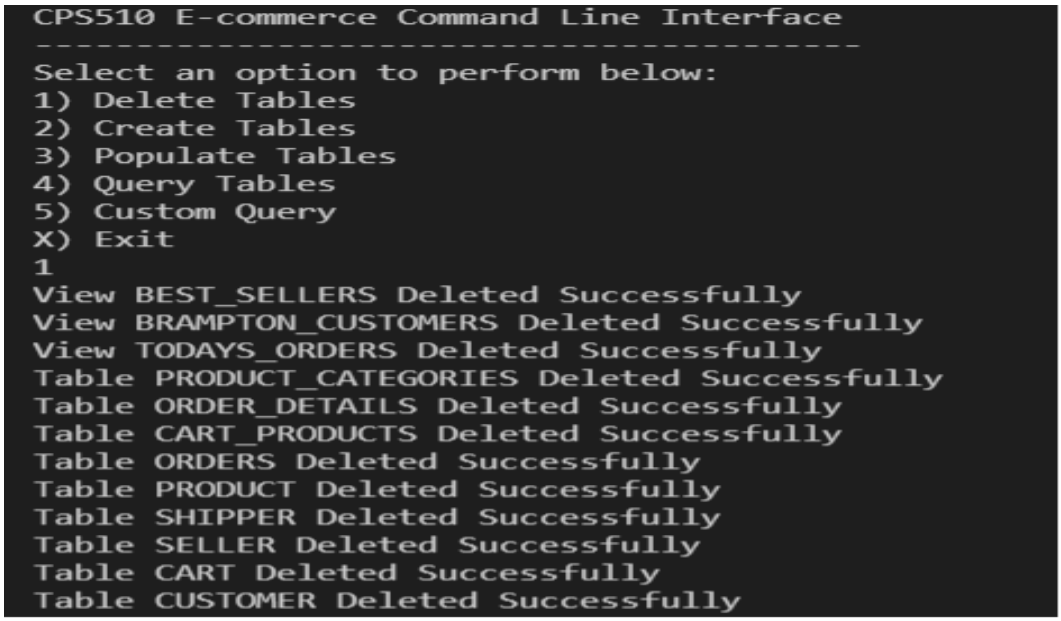
Menu Start Up:



This image highlights the start up view for the command line ui and shows the initial print out of the menu along with the options provided for the user. We provide 5 options for the user to execute based on a number value entered through stdin.

1) Delete Tables - Deletes the tables for the e-commerce system in the db
2) Create Tables - Creates the tables for the e-commerce system in the db
3) Populate Tables - Inserts a predefined set of dummy data into the tables for e-commerce system
4) Query Tables - Performs a set of predefined queries in order to generate a report style page with headers separating returned results for each Query
5) Custom Query - Allows the user to execute any SQL statement they desire on the db
X) Exit - Close the connection and exit the menu

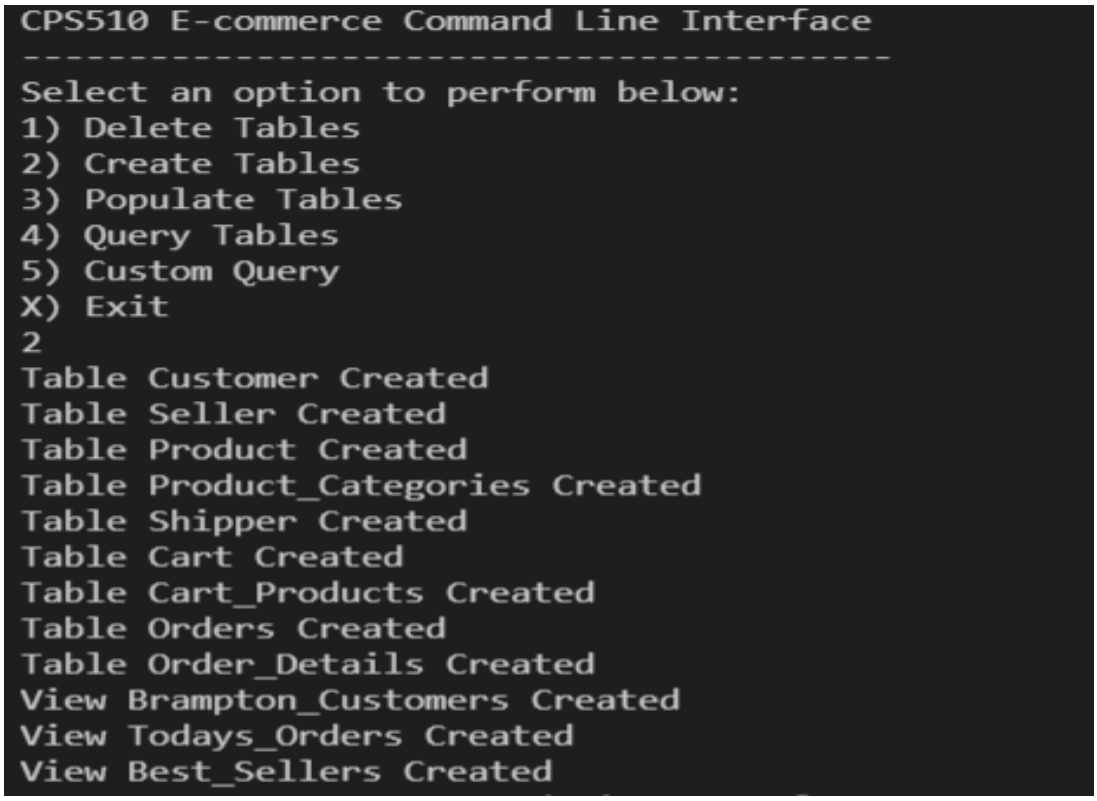
Delete Tables:



This image highlights the execution of the Delete Tables option - Drops all tables and views related to the e-commerce database system from the db cascading constraints in order to not violate any foreign key constraints.

Displays either a successful deletion print out or the error code and message generated by the db upon attempting statement execution - i.e., Table or View does not exist

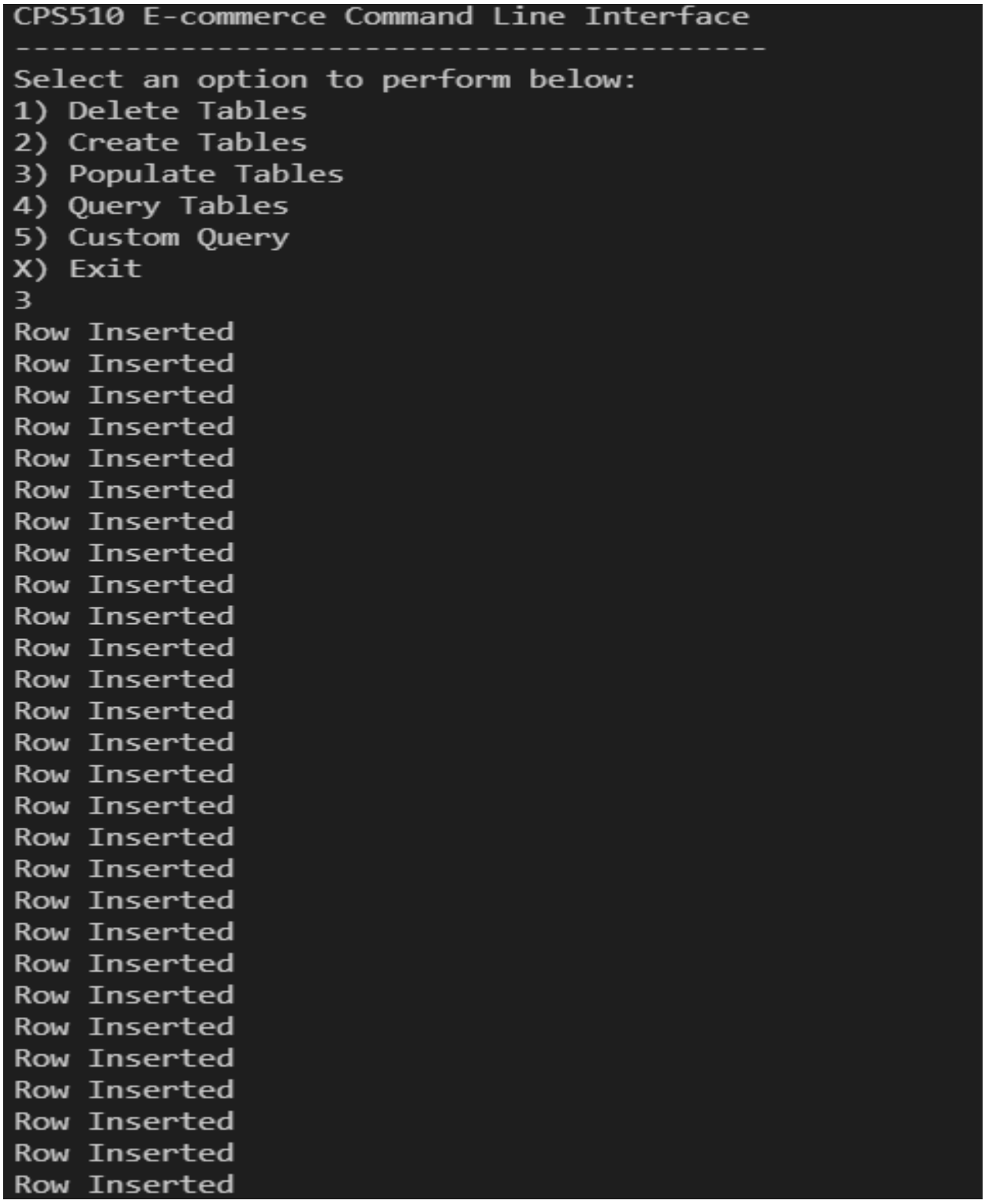
Create Tables:



This image highlights the execution of the Create Tables option - Creates all tables and views related to the e-commerce database system from the db.

Displays either a successful creation print out or the error code and message generated by the db upon attempting statement execution - i.e., Object already exists

Populate Tables:



This image highlights the execution of the Populate Tables option - Inserts a collection of predefined dummy data into the tables of the e-commerce db in order to facilitate report generation and querying.

Displays either a successful Row Inserted print out or the error code and message generated by the db upon attempting statement execution.

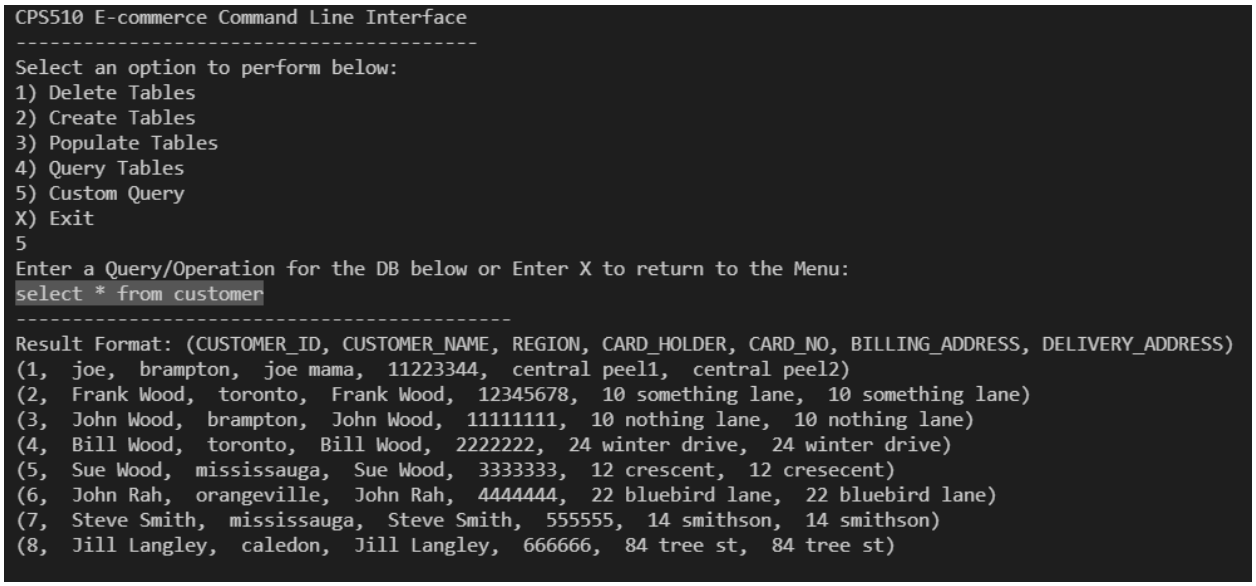
Query Tables:

```
CPS510 E-commerce Command Line Interface
----------------------------------------
Select an option to perform below:
1) Delete Tables
2) Create Tables
3) Populate Tables
4) Query Tables
5) Custom Query
X) Exit
4
Number of Customer Orders
------------------------------------------
Result Format: (CUSTOMER_NAME, NUM_ORDERS)
(joe,  2)
(John Wood,  2)
(Frank Wood,  2)

Amount of Playstations Ordered by Customer
------------------------------------------
Result Format: (CUSTOMER_NAME, NUM_ORDERED)
(joe,  1)
(Frank Wood,  1)

Seller Performance
------------------------------------------
Result Format: (SELLER_NAME, NUM_SOLD_PRODUCTS)
(Elizabeth Smith,  6)
(Christian Daniel,  3)

Active Customers (Customers who Order)
------------------------------------------
Result Format: (CID, CNAME, CREGION, BILL_ADDR, DELIV_ADDR)
(1,  joe,  brampton,  central peel1,  central peel2)
(2,  Frank Wood,  toronto,  10 something lane,  10 something lane)
(3,  John Wood,  brampton,  10 nothing lane,  10 nothing lane)
```

This image highlights the execution of the Query Tables option - Performs a collection of predefined queries in order to generate a report style output of return values with associated headers to describe the meaning of the results.

Displays either a successful result report with a corresponding header for each query along with the structure of the returned values in terms of the column names
or the error code and message generated by the db upon attempting statement execution - i.e., Object already exists

Custom Query:



This image highlights the execution of the Custom Query option - Allows the user to perform any SQL statement they need on the e-commerce db including queries and update/delete/insert statements. This facilitates the ability for users to select specific records from the db and manage the db as needed.

The custom query option brings the user to a sub-menu of sorts allowing them to either enter a SQL statement or press X to return to the main menu.
Upon entering a statement the user will also be returned to the main menu upon completion.

Displays either the results of a query or executes a CRUD command within the db or the error code and message if an improper statement is passed.