



## **CC5051NI Databases**

**50% Individual Coursework**

**Autumn 2023**

**Student Name: Rakesh Chaurasiya**

**London Met ID: 22085821**

**Assignment Submission Date: January 15, 2024**

*I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.*

# Table of Contents

<b>1. Introduction:</b>	<b>1</b>
1.1. Current business activities and operation	1
1.2. Business Rules	1
1.3. Identification of Entities and Attributes	2
<b>2. Initial ERD</b>	<b>4</b>
2.1. Primary and Foreign Keys	4
2.2. Initial ERD	5
<b>3. Normalization</b>	<b>7</b>
3.1. UNF	7
3.2. 1NF	7
3.3. 2NF	7
3.3.1. Customer-1	7
3.3.2. Order-1	8
3.3.3. Product-1	8
3.4. 3NF	9
3.4.1. Customer-2	9
3.4.2. Product-2	10
3.4.3. Remaining Entities	11
<b>4. Final ERD</b>	<b>13</b>
<b>5. Implementation</b>	<b>14</b>
<b>6. Database Querying</b>	<b>29</b>
6.1. Information Queries	29
6.2. Transaction Queries	32
<b>7. Drop Query</b>	<b>37</b>
<b>8. Dump File Creation</b>	<b>38</b>

<b>9. Critical Evaluation .....</b>	<b>39</b>
<b>9.1. Critical Assessment of the Module .....</b>	<b>39</b>
<b>9.2. Critical Evaluation of the Coursework .....</b>	<b>39</b>

## Table of Figures

Figure 1 : Individual relationships between Customer, Order, and Product entities .....	4
Figure 2 : Initial ERD.....	5
Figure 3 : Final ERD .....	13
Figure 4 : Connecting to system, creating user Gadget_Emporium with password 123456, and granting it CONNECT and RESOURCE privileges.....	14
Figure 5 : Connecting to Gadget_Emporium user .....	14
Figure 6 : Creating and describing Customer_Category table.....	15
Figure 7 : Creating and describing Customer table .....	15
Figure 8 : Creating and describing Order table.....	16
Figure 9 : Creating and describing Vendor table .....	16
Figure 10 : Creating and describing Product table .....	17
Figure 11 : Creating and describing Product_Order table.....	17
Figure 12 : Creating and describing Customer_Product_Order table.....	18
Figure 13 : Showing all created tables in Gadget_Emporium.....	18
Figure 14 : Inserting values into Customer_Category table.....	19
Figure 15 : Displaying the values inserted into Customer_Category table .....	19
Figure 16 : Inserting values into Customer table .....	19
Figure 17 : Displaying the values inserted into Customer table.....	20
Figure 18 : Inserting values into Order table.....	20
Figure 19 : Displaying the values inserted into Order table .....	21
Figure 20 : Inserting values into Vendor table .....	21
Figure 21 : Displaying the values inserted into Vendor table.....	22
Figure 22 : Inserting values into Product table .....	22
Figure 23 : Displaying values inserted into Product table.....	23
Figure 24 : Inserting values into Product_Order table .....	24
Figure 25 : Displaying the values inserted into Product_Order table.....	25
Figure 26 : Inserting values into Customer_Product_Order table.....	26
Figure 27 : Displaying the values inserted into Customer_Product_Order table .....	27
Figure 28 : Committing the insertion of values into all the tables .....	28
Figure 29 : Listing all the customers that are also staff of the company .....	29

Figure 30 : Listing all the orders made for any particular product between the dates 01-05-2023 till 28-05-2023 .....	29
Figure 31 : Listing all the customers with their order details and also the customers who have not ordered any products yet .....	30
Figure 32 : Listing all product details that have the second letter 'a' in their product name and have a stock quantity more than 50 .....	31
Figure 33 : Finding out the customer who has ordered recently .....	31
Figure 34 : Showing the total revenue of the company for each month .....	32
Figure 35 : Creating a temporary table Order_Total to store Order_ID and Total_Amount .....	33
Figure 36 : Finding those orders that are equal or higher than the average order total value, and dropping the Order_Total table .....	34
Figure 37 : Listing the details of vendors who have supplied more than 3 products to the company .....	34
Figure 38 : Showing the top 3 product details that have been ordered the most .....	35
Figure 39 : Finding out the customer who has ordered the most in August with his/her total spending on that month .....	35
Figure 40 : Dropping all the created tables in order .....	37
Figure 41 : Creating a dump file of the coursework .....	38

## **Table of Tables**

Table 1 : Entities with their attributes, datatypes, and descriptions ..... 3

Table 3 : Entities with their Primary and Foreign Keys in Initial ERD..... 4

## 1. Introduction:

Gadget Emporium is an e-commerce business founded by Mr. John in 1990 A.D. which is situated in Arlington, Texas, USA. This business is involved in retailing a large variety of electronic devices and gadgets like Headphones, Smart Phones, Smart Watches, Speakers, Laptops, etc. The recent trend of online marketing has been growing huge, and the business is seeking to take advantage of this and launch its own e-commerce website where customers can browse and order multiple products, while enjoying a range of discount rates. The website has the provision of payment methods like Cash On Delivery, Credit Card, Debit Card, e-wallet, etc.

### 1.1. Current business activities and operation

- Gadget Emporium imports electronics directly from manufacturer and sells in their websites.
- Gadget Emporium maintains its supply chain through strong relationships with reputed companies like Apple, Samsung, Oppo, Vivo, JBL, Boat, Google, etc.
- The business has a user friendly e-commerce website with a secure and transparent payment process.
- The business has highly available telephone lines and email for customer support.
- The business promotes itself and boosts marketing through various ways like social media, television advertisement, seasonal offers.
- The business uses web traffic analyzing tools to gather data on customer preferences and ongoing trends.
- The company financial data proves its operational efficiency which is due to the implementation of effective marketing strategies and customer satisfaction, among other reasons.

### 1.2. Business Rules

- The database includes product details like product ID, name, stock level, price, description, category, and vendor details.

- The database keeps track of customers, including details like customer ID name, address, phone number, and category.
- The system records order details such as order Id, products included, customer details, order date, etc.
- A customer can place multiple orders, but an order can be placed by a single customer.
- An order must contain at least one product, and a product may be included in multiple orders.
- Multiple products may be ordered by multiple customers.
- Customers must belong to either Regular, Staff, or VIP categories, and are respectively granted discount rates of 0%, 5% and 10% on their total order amounts.
- After a customer checks out their order(s), an invoice is issued, which includes the details of the order, the customer who placed it, and the products purchased, along with the total amount and discounted amount.
- A customer can provide only one phone number, and two numbers are seen as belonging to two different customers.
- The system records vendor details such as vendor ID, name, address, and email address.
- A vendor is associated with a single phone number, and two different such numbers are recognized as belonging to two different vendors.

### 1.3. Identification of Entities and Attributes

Entities	Attributes	Descriptions	DATATYPES
<b>Customer</b>	Customer_ID	Unique identifier for a customer	NUMBER
	Customer_Name	Full name of customer	VARCHAR2
	Customer_Address	Address of customer	VARCHAR2
	Customer_Phone	Phone number of customer	VARCHAR2
	Customer_Category	Category of customer (Regular/Staff/VIP)	VARCHAR2



	Discount	Discount granted to customer according to their category	NUMBER
<b>Order</b>	Order_ID	Unique identifier for an order	NUMBER
	Order_Date	Date placed of an order	DATE
	Order_Total	Total amount of an order	NUMBER
<b>Product</b>	Product_ID	Unique identifier for a product	NUMBER
	Product_Name	Name of a product	VARCHAR2
	Product_Price	Unit price of a product	NUMBER
	Product_Stock	Stock quantity of a product	NUMBER
	Product_Category	Category of a product	VARCHAR2
	Product_Description	Description of a product	VARCHAR2
	Order_Quantity	Quantity of a product included in an order	NUMBER
	Vendor_ID	Unique identifier for a vendor	NUMBER
	Vendor_Name	Name of a vendor	VARCHAR2
	Vendor_Address	Address of a vendor	VARCHAR2
	Vendor_Number	Phone number of a vendor	VARCHAR2

Table 1 : Entities with their attributes, datatypes, and descriptions

## 2. Initial ERD

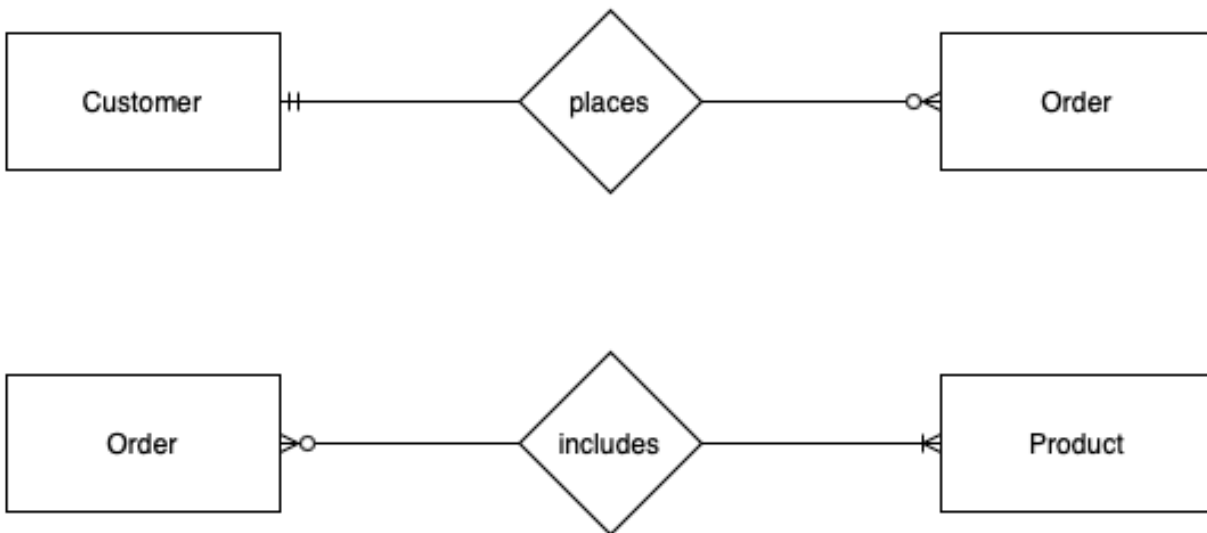


Figure 1 : Individual relationships between Customer, Order, and Product entities

The picture above shows individual relationships between the Customer, Order, and Product tables. Since a customer can place zero or many orders but an order can be placed by only one customer, Customer has a one-to-many optional relationship with Order. Similarly, since an Order has to include at least one product but a product may be included in zero or more orders, Order has an optional many-to-many relationship with Product.

### 2.1. Primary and Foreign Keys

Entities	Primary Key	Foreign Key(s)
<b>Customer</b>	Customer_ID	
<b>Order</b>	Order_ID	Customer_ID, Product_ID
<b>Product</b>	Product_ID	Order_ID

Table 2 : Entities with their Primary and Foreign Keys in Initial ERD

## 2.2. Initial ERD

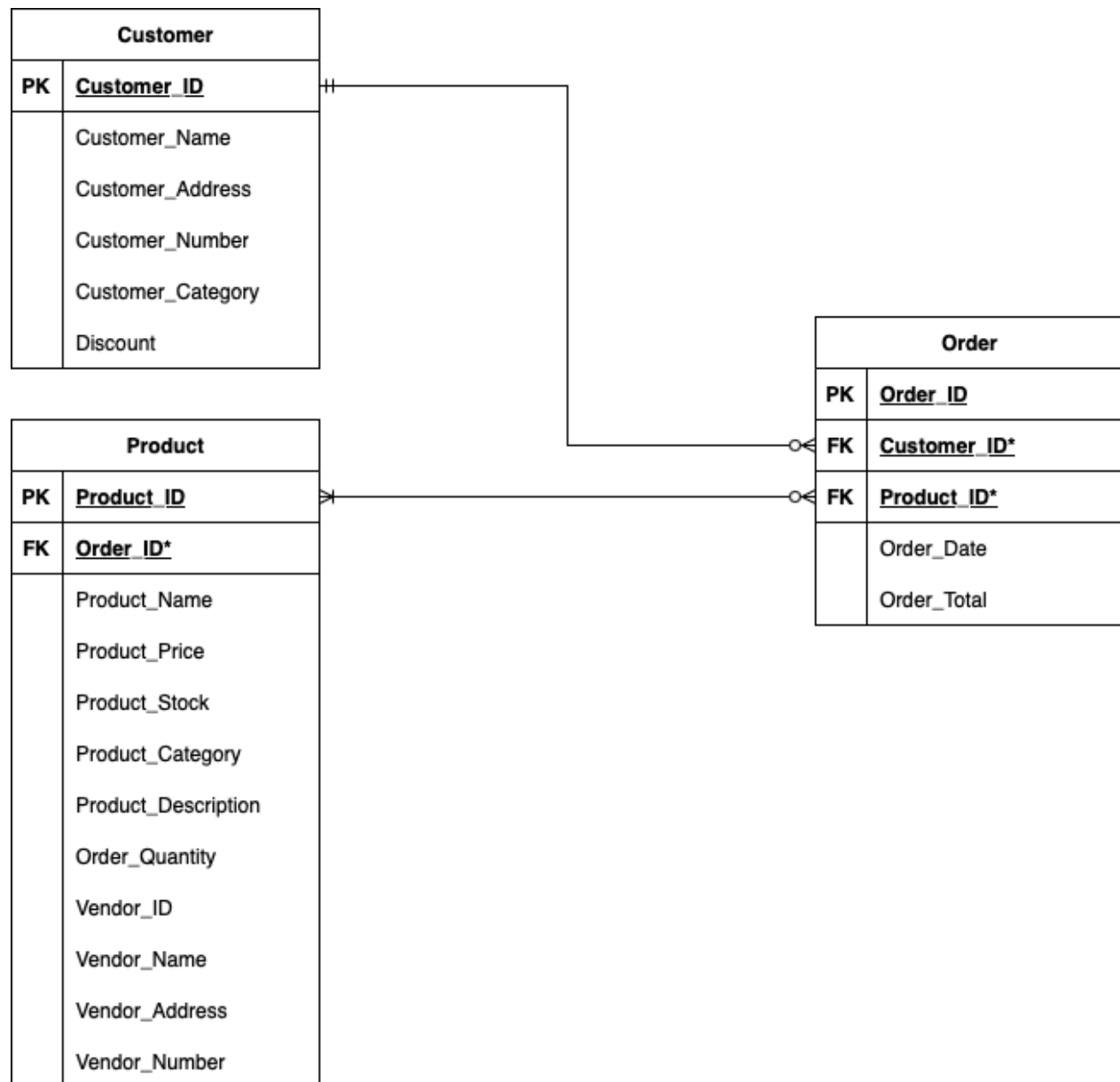


Figure 2 : Initial ERD

The scenario given in the question can be studied to derive the entities and attributes seen in the Initial ERD here. The Customer entity has Customer\_ID as the primary key and no foreign keys. The Order table has Order\_ID as the primary key, and Customer\_ID and Product\_ID as the foreign keys. The Product table has Product\_ID as the primary key with Order\_ID as the foreign key.

The Customer table has a one-to-many optional relationship with the Order table because a customer may place multiple or no orders, but an order can be placed by

exactly one customer only. The Product table has a many-to-many optional relationship with the Order table, since a product may be included in multiple or no orders, and an order has to have at least one product included in it but may contain more.

### 3. Normalization

#### 3.1. UNF

**Customer** (Customer\_ID, Customer\_Name, Customer\_Address, Customer\_Number, Customer\_Category, Discount {Order\_ID, Order\_Date, {Product\_ID, Product\_Name, Product\_Price, Product\_Stock, Product\_Category, Product\_Description, Order\_Quantity, Vendor\_ID, Vendor\_Name, Vendor\_Address, Vendor\_Number}}})

Here, the Customer\_ID attribute was selected as the candidate key, for which the attributes of Order were a repeating group. Further, the attributes of Product were a repeating group for the Order entity. These repeating groups were separated from the repeating data in 1NF.

#### 3.2. 1NF

- **Customer-1** (Customer\_ID, Customer\_Name, Customer\_Address, Customer\_Number, Customer\_Category, Discount)
- **Order-1** (Order\_ID, Customer\_ID\*, Order\_Date)
- **Product-1** (Product\_ID, Order\_ID\*, Customer\_ID\*, Product\_Name, Product\_Price, Product\_Stock, Product\_Category, Product\_Description, Order\_Quantity, Vendor\_ID, Vendor\_Name, Vendor\_Address, Vendor\_Number)

#### 3.3. 2NF

##### 3.3.1. Customer-1

**Customer-1** has no partial dependencies because it has only one key attribute. So, it is already in 2NF.

- **Customer-2** (Customer\_ID, Customer\_Name, Customer\_Address, Customer\_Number, Customer\_Category, Discount)

### 3.3.2. Order-1

**Order-1** has 2 key attributes. So, there may be partial dependencies.

Checking for 2NF,

- **Order\_ID** → Order\_Date
- **Customer ID** → (New table is not formed because only one attribute is here.)
- **Order\_ID, Customer ID** → X

Therefore, the new tables are

- **Order-2** (Order ID, Order\_Date)
- **Customer-Order-2** (Order ID\*, Customer ID\*)

### 3.3.3. Product-1

**Product-1** has 3 key attributes. So, there may be partial dependencies.

Checking for 2NF,

- **Product\_ID** → Product\_Name, Product\_Price, Product\_Stock, Product\_Category, Product\_Description, Vendor\_ID, Vendor\_Name, Vendor\_Address, Vendor\_Number
- **Customer\_ID** → (New table is not formed because only one attribute is here.)
- **Order\_ID** → (New table is not formed because only one attribute is here.)
- **Product\_ID, Customer\_ID** → X
- **Product\_ID, Order\_ID** → Order\_Quantity
- **Customer\_ID, Order\_ID** → (This entity has already been formed above.)
- **Product\_ID, Customer\_ID, Order\_ID** → X

New tables formed are,

- **Product-2** (Product ID, Product\_Name, Product\_Price, Product\_Stock, Product\_Category, Product\_Description, Vendor\_ID, Vendor\_Name, Vendor\_Address, Vendor\_Number)
- **Product-Order-2** (Product ID\*, Order ID\*, Order\_Quantity)
- **Customer-Product-2** (Customer ID\*, Product ID\*)

- **Customer-Order-Product-2** (Customer\_ID\*, Order\_ID\*, Product\_ID\*)

#### All 2NF tables:

- **Product-2** (Product\_ID, Product\_Name, Product\_Price, Product\_Stock, Product\_Category, Product\_Description, Vendor\_ID, Vendor\_Name, Vendor\_Address, Vendor\_Number)
- **Product-Order-2** (Product\_ID\*, Order\_ID\*, Order\_Quantity)
- **Customer-Product-2** (Customer\_ID\*, Product\_ID\*)
- **Customer-Order-Product-2** (Customer\_ID\*, Order\_ID\*, Product\_ID\*)
- **Order-2** (Order\_ID, Order\_Date)
- **Customer-Order-2** (Order\_ID\*, Customer\_ID\*)
- **Customer-2** (Customer\_ID, Customer\_Name, Customer\_Address, Customer\_Number, Customer\_Category, Discount)

### 3.4. 3NF

#### 3.4.1. Customer-2

Checking for transitive dependency in **Customer-2**,

**Customer-2** (Customer\_ID, Customer\_Name, Customer\_Address, Customer\_Number, Customer\_Category, Discount)

- Customer\_ID → Customer\_Name, Customer\_Address, Customer\_Number → X

This assumption is made because the customer's name and address are not unique to each customer. Phone numbers are unique to each customer but can be altered in the future by customers themselves, therefore increasing the chances of loss in data integrity.

- Customer\_ID → Customer\_Category → Discount
- Customer\_ID → Discount → Customer\_Category

The above assumptions are made because a customer is granted a discount rate based on the category they belong to, and each category has a unique discount rate.

Separating them based on the former assumption,

- **Customer\_ID** → Customer\_Category
- **Customer\_Category** → Discount

Therefore, the new 3NF tables are,

- **Customer-3** (Customer\_ID, Customer\_Name, Customer\_Address, Customer\_Number, Customer\_Category\*)
- **Customer-Category-3** (Customer\_Category, Discount)

### 3.4.2. Product-2

Checking for transitive dependencies in **Product-2**,

- **Product\_ID** → Product\_Name, Product\_Price, Product\_Stock, Product\_Category, Product\_Description, Vendor\_Name, Vendor\_Address, Vendor\_Number → X

This assumption is made because the product name, price, stock, category, description, vendor name and vendor address are not always unique to a product. Vendor number may be unique to each product, but it can be changed in the future by the vendor.

- **Product\_ID** → Vendor\_ID → Vendor\_Name, Vendor\_Address, Vendor\_Number

This assumption is made because vendor ID is unique for each vendor, so it can identify their name, address, and number.

Separating it,

- **Product\_ID** → Vendor\_ID
- **Vendor\_ID** → Vendor\_Name, Vendor\_Address, Vendor\_Number

New 3NF tables are formed

- **Product-3** (Product\_ID, Product\_Name, Product\_Price, Product\_Stock, Product\_Category, Product\_Description, Vendor\_ID\*)



- **Vendor-3** (Vendor\_ID, Vendor\_Name, Vendor\_Address, Vendor\_Number)

### 3.4.3. Remaining Entities

For the remaining entities,

- **Product-Order-2** (Product\_ID\*, Order\_ID\*, Order\_Quantity)
- **Customer-Product-2** (Customer\_ID\*, Product\_ID\*)
- **Customer-Order-Product-2** (Customer\_ID\*, Order\_ID\*, Product\_ID\*)
- **Order-2** (Order\_ID, Order\_Date, Order\_Total)
- **Customer-Order-2** (Order\_ID\*, Customer\_ID\*)

There are no transitive dependencies because there is either only one non-key attribute or none. So, they are already in 3NF. i.e.

- **Product-Order-3** (Product\_ID\*, Order\_ID\*, Order\_Quantity)
- **Customer-Product-3** (Customer\_ID\*, Product\_ID\*)
- **Customer-Order-Product-3** (Customer\_ID\*, Order\_ID\*, Product\_ID\*)
- **Order-3** (Order\_ID, Order\_Date)
- **Customer-Order-3** (Order\_ID\*, Customer\_ID\*)

**All tables in 3NF:**

- **Customer-3** (Customer\_ID, Customer\_Name, Customer\_Address, Customer\_Number, Customer\_Category\*)
- **Order-3** (Order\_ID, Order\_Date)
- **Product-3** (Product\_ID, Product\_Name, Product\_Price, Product\_Stock, Product\_Category, Product\_Description, Vendor\_ID\*)
- **Customer-Category-3** (Customer\_Category, Discount)
- **Product-Order-3** (Product\_ID\*, Order\_ID\*, Order\_Quantity)
- **Customer-Order-3** (Order\_ID\*, Customer\_ID\*)
- **Customer-Product-3** (Customer\_ID\*, Product\_ID\*)
- **Customer-Order-Product-3** (Customer\_ID\*, Order\_ID\*, Product\_ID\*)
- **Vendor-3** (Vendor\_ID, Vendor\_Name, Vendor\_Address, Vendor\_Number)

Since **Customer-Order-Product-3** is the main bridge entity, **Customer-Order-3** and **Customer-Product-3** can be omitted because they have no other attributes except foreign keys.

Therefore, the final 3NF entities needed are:

- **Customer-3** (Customer\_ID, Customer\_Name, Customer\_Address, Customer\_Number, Customer\_Category\*)
- **Order-3** (Order\_ID, Order\_Date)
- **Product-3** (Product\_ID, Product\_Name, Product\_Price, Product\_Stock, Product\_Category, Product\_Description, Vendor\_ID\*)
- **Customer-Category-3** (Customer\_Category, Discount)
- **Product-Order-3** (Product\_ID\*, Order\_ID\*, Order\_Quantity)
- **Customer-Order-Product-3** (Customer\_ID\*, Order\_ID\*, Product\_ID\*)
- **Vendor-3** (Vendor\_ID, Vendor\_Name, Vendor\_Address, Vendor\_Number)

## 4. Final ERD

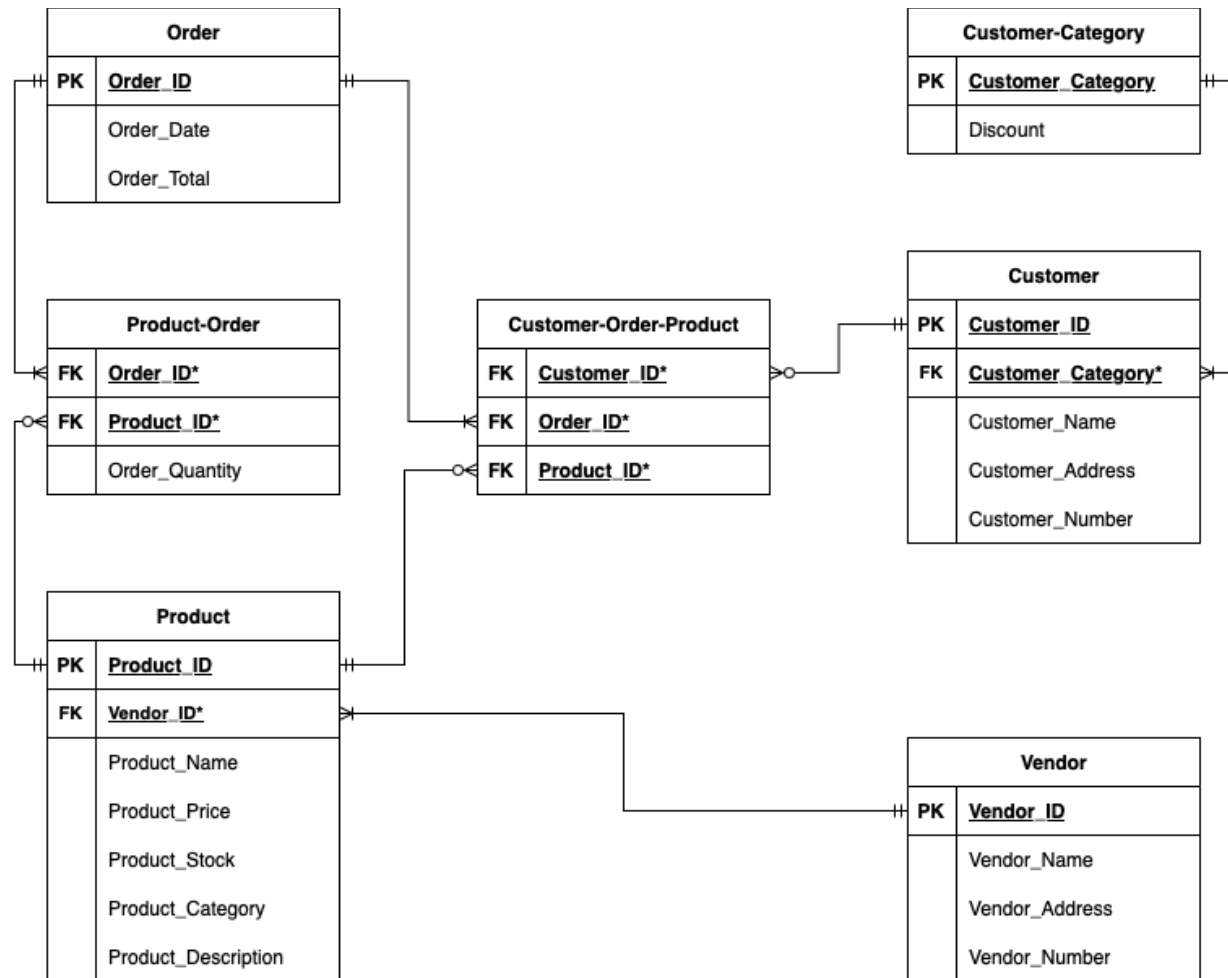


Figure 3 : Final ERD

## 5. Implementation

```
SQL*Plus: Release 11.2.0.2.0 Production on Wed Jan 10 20:49:20 2024  
Copyright (c) 1982, 2010, Oracle. All rights reserved.  
  
SQL> CONNECT SYSTEM/12345  
Connected.  
SQL> CREATE USER Gadget_Emporium IDENTIFIED BY 123456;  
  
User created.  
  
SQL> GRANT CONNECT, RESOURCE TO Gadget_Emporium;  
  
Grant succeeded.  
  
SQL> |
```

*Figure 4 : Connecting to system, creating user Gadget\_Emporium with password 123456, and granting it CONNECT and RESOURCE privileges*

First, the system was connected to, the user Gadget\_Emporium was created, and the CONNECT and RESOURCE privileges were granted to it.

```
SQL> CONNECT Gadget_Emporium;  
Enter password:  
Connected.  
SQL> |
```

*Figure 5 : Connecting to Gadget\_Emporium user*

The Gadget\_Emporium user was connected to successfully.

Below is the description of creating, describing, and inserting records into each final 3NF table. Every attribute has been specified as NOT NULL, since all of them are mandatory to be recorded. The email and contact number attributes of Customer and Vendor tables have been specified as UNIQUE as well, since those things are unique to each customer and vendor.

```
SQL> CREATE TABLE Customer_Category
  2 (Customer_Category VARCHAR2(20) PRIMARY KEY,
  3 Discount NUMBER NOT NULL);

Table created.

SQL> DESC Customer_Category;
Name                                         Null?    Type
-----
CUSTOMER_CATEGORY                          NOT NULL VARCHAR2(20)
DISCOUNT                                  NOT NULL NUMBER

SQL>
```

Figure 6 : Creating and describing Customer\_Category table

The Customer\_Category table was created and described as seen here.

```
SQL> CREATE TABLE Customer
  2 (Customer_ID NUMBER PRIMARY KEY,
  3 Customer_Name VARCHAR2(30) NOT NULL,
  4 Customer_Address VARCHAR2(30) NOT NULL,
  5 Customer_Number VARCHAR2(30) NOT NULL UNIQUE,
  6 Customer_Category VARCHAR2(20) NOT NULL,
  7 FOREIGN KEY (Customer_Category) REFERENCES Customer_Category(Customer_Category));

Table created.

SQL> DESC Customer;
Name                                         Null?    Type
-----
CUSTOMER_ID                               NOT NULL NUMBER
CUSTOMER_NAME                             NOT NULL VARCHAR2(30)
CUSTOMER_ADDRESS                           NOT NULL VARCHAR2(30)
CUSTOMER_NUMBER                            NOT NULL VARCHAR2(30)
CUSTOMER_CATEGORY                          NOT NULL VARCHAR2(20)
```

Figure 7 : Creating and describing Customer table

Next, the Customer table was created and described.

```
SQL> CREATE TABLE "ORDER"
  2  (Order_ID NUMBER PRIMARY KEY,
  3  Order_Date DATE NOT NULL);
```

Table created.

```
SQL> DESC "ORDER";
```

Name	Null?	Type
ORDER_ID	NOT NULL	NUMBER
ORDER_DATE	NOT NULL	DATE

Figure 8 : Creating and describing Order table

The next setp was the creation of the Order table. The table name was enclosed in double quotes since Order is a reserved keywork in SQL and so cannot be used as a table name.

```
SQL> CREATE TABLE Vendor
  2  (Vendor_ID NUMBER PRIMARY KEY,
  3  Vendor_Name VARCHAR2(30) NOT NULL,
  4  Vendor_Address VARCHAR2(30) NOT NULL,
  5  Vendor_Number VARCHAR2(30) NOT NULL UNIQUE);
```

Table created.

```
SQL> DESC Vendor;
```

Name	Null?	Type
VENDOR_ID	NOT NULL	NUMBER
VENDOR_NAME	NOT NULL	VARCHAR2(30)
VENDOR_ADDRESS	NOT NULL	VARCHAR2(30)
VENDOR_NUMBER	NOT NULL	VARCHAR2(30)

Figure 9 : Creating and describing Vendor table

The Vendor table was created and described next.

```
SQL> CREATE TABLE Product
 2 (Product_ID NUMBER PRIMARY KEY,
 3 Product_Name VARCHAR2(30) NOT NULL,
 4 Product_Description VARCHAR2(30) NOT NULL,
 5 Product_Category VARCHAR2(30) NOT NULL,
 6 Product_Price NUMBER NOT NULL,
 7 Product_Stock NUMBER NOT NULL,
 8 Vendor_ID NUMBER NOT NULL,
 9 FOREIGN KEY (Vendor_ID) REFERENCES Vendor(Vendor_ID));
```

Table created.

```
SQL> DESC Product;
```

Name	Null?	Type
PRODUCT_ID	NOT NULL	NUMBER
PRODUCT_NAME	NOT NULL	VARCHAR2(30)
PRODUCT_DESCRIPTION	NOT NULL	VARCHAR2(30)
PRODUCT_CATEGORY	NOT NULL	VARCHAR2(30)
PRODUCT_PRICE	NOT NULL	NUMBER
PRODUCT_STOCK	NOT NULL	NUMBER
VENDOR_ID	NOT NULL	NUMBER

Figure 10 : Creating and describing Product table

The previous step was followed by creating the Product table and describing it.

```
SQL> CREATE TABLE Product_Order
 2 (Product_ID NUMBER NOT NULL,
 3 Order_ID NUMBER NOT NULL,
 4 Order_Quantity NUMBER NOT NULL,
 5 FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID),
 6 FOREIGN KEY (Order_ID) REFERENCES "ORDER"(Order_ID),
 7 PRIMARY KEY (Product_ID, Order_ID));
```

Table created.

```
SQL> DESC Product_Order;
```

Name	Null?	Type
PRODUCT_ID	NOT NULL	NUMBER
ORDER_ID	NOT NULL	NUMBER
ORDER_QUANTITY	NOT NULL	NUMBER

Figure 11 : Creating and describing Product\_Order table

The Product\_Order bridge entity was next created and described.

```
SQL> CREATE TABLE Customer_Product_Order
  2 (Customer_ID NUMBER NOT NULL,
  3 Product_ID NUMBER NOT NULL,
  4 Order_ID NUMBER NOT NULL,
  5 FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID),
  6 FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID),
  7 FOREIGN KEY (Order_ID) REFERENCES "ORDER"(Order_ID),
  8 PRIMARY KEY (Customer_ID, Product_ID, Order_ID));
```

Table created.

```
SQL> DESC Customer_Product_Order;
```

Name	Null?	Type
CUSTOMER_ID	NOT NULL	NUMBER
PRODUCT_ID	NOT NULL	NUMBER
ORDER_ID	NOT NULL	NUMBER

Figure 12 : Creating and describing Customer\_Product\_Order table

The step above was followed by the creation and description of the Customer\_Product\_Order bridge entity.

```
SQL> SELECT table_name FROM user_tables;
```

```
TABLE_NAME
```

```
-----
CUSTOMER_CATEGORY
CUSTOMER
ORDER
VENDOR
PRODUCT
PRODUCT_ORDER
CUSTOMER_PRODUCT_ORDER
```

```
7 rows selected.
```

Figure 13 : Showing all created tables in Gadget\_Emporium

The picture above shows the names of all tables created in the steps above.



```
SQL> INSERT ALL
  2 INTO Customer_Category VALUES ('R', 0)
  3 INTO Customer_Category VALUES ('S', 0.05)
  4 INTO Customer_Category VALUES ('V', 0.1)
  5 SELECT * FROM DUAL;

3 rows created.
```

Figure 14 : Inserting values into Customer\_Category table

The Customer\_Category was filled with 3 rows of data as seen here.

```
SQL> SELECT * FROM Customer_Category;
```

CUSTOMER_CATEGORY	DISCOUNT
R	0
S	.05
V	.1

Figure 15 : Displaying the values inserted into Customer\_Category table

The inserted values in the Customer\_Category were then displayed.

```
SQL> INSERT ALL
  2 INTO Customer VALUES (1, 'Jack', '123 Main St', '555-1234', 'S')
  3 INTO Customer VALUES (2, 'Emily', '456 Oak Ave', '555-5678', 'R')
  4 INTO Customer VALUES (3, 'John', '789 Pine Ln', '555-9012', 'V')
  5 INTO Customer VALUES (4, 'Sarah', '101 Maple Dr', '555-3456', 'S')
  6 INTO Customer VALUES (5, 'Michael', '202 Cedar Rd', '555-7890', 'R')
  7 INTO Customer VALUES (6, 'Amanda', '303 Elm Blvd', '555-2345', 'S')
  8 INTO Customer VALUES (7, 'David', '404 Birch Ct', '555-6789', 'S')
  9 INTO Customer VALUES (8, 'Jessica', '505 Walnut Ave', '555-0123', 'V')
 10 INTO Customer VALUES (9, 'Brian', '606 Spruce St', '555-4567', 'S')
 11 INTO Customer VALUES (10, 'Lisa', '707 Pine Ln', '555-8901', 'R')
 12 INTO Customer VALUES (11, 'Chris', '808 Oak Ave', '555-2378', 'V')
 13 INTO Customer VALUES (12, 'Megan', '909 Cedar Rd', '555-6239', 'R')
 14 SELECT * FROM DUAL;
```

Figure 16 : Inserting values into Customer table

The Customer table was populated with 12 rows of data as shown.

```
SQL> SELECT * FROM Customer;
```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_ADDRESS	CUSTOMER_NUMBER	CUSTOMER_CATEGORY
1	Jack	123 Main St	555-1234	S
2	Emily	456 Oak Ave	555-5678	R
3	John	789 Pine Ln	555-9012	V
4	Sarah	101 Maple Dr	555-3456	S
5	Michael	202 Cedar Rd	555-7890	R
6	Amanda	303 Elm Blvd	555-2345	S
7	David	404 Birch Ct	555-6789	S
8	Jessica	505 Walnut Ave	555-0123	V
9	Brian	606 Spruce St	555-4567	S
10	Lisa	707 Pine Ln	555-8901	R
11	Chris	808 Oak Ave	555-2378	V
12	Megan	909 Cedar Rd	555-6239	R

Figure 17 : Displaying the values inserted into Customer table

The values inserted into the Customer table were then displayed.

```
SQL> INSERT ALL
  2 INTO "ORDER" VALUES (1, '12-FEB-23')
  3 INTO "ORDER" VALUES (2, '23-MAY-23')
  4 INTO "ORDER" VALUES (3, '05-AUG-23')
  5 INTO "ORDER" VALUES (4, '04-MAY-23')
  6 INTO "ORDER" VALUES (5, '07-APR-23')
  7 INTO "ORDER" VALUES (6, '02-JUN-23')
  8 INTO "ORDER" VALUES (7, '14-MAY-23')
  9 INTO "ORDER" VALUES (8, '25-MAR-23')
 10 INTO "ORDER" VALUES (9, '10-AUG-23')
 11 INTO "ORDER" VALUES (10, '22-MAY-23')
 12 SELECT * FROM DUAL;
```

Figure 18 : Inserting values into Order table

Next, the Order table was filled with 10 rows of data as in the picture above.

```
SQL> SELECT * FROM "ORDER";
```

ORDER_ID	ORDER_DAT
1	12-FEB-23
2	23-MAY-23
3	05-AUG-23
4	04-MAY-23
5	07-APR-23
6	02-JUN-23
7	14-MAY-23
8	25-MAR-23
9	10-AUG-23
10	22-MAY-23

```
10 rows selected.
```

Figure 19 : Displaying the values inserted into Order table

The inserted values were then displayed as shown.

```
SQL> INSERT ALL
  2 INTO Vendor VALUES (1, 'TechGlobe Solutions', '123 Main St', '555-1234')
  3 INTO Vendor VALUES (2, 'ElectronicXpress', '456 Oak Ave', '555-5678')
  4 INTO Vendor VALUES (3, 'CircuitMasters', '789 Pine Ln', '555-9012')
  5 INTO Vendor VALUES (4, 'SparkTech', '101 Maple Dr', '555-3456')
  6 INTO Vendor VALUES (5, 'Silicon Valley', '202 Cedar Rd', '555-7890')
  7 INTO Vendor VALUES (6, 'Quantum Components', '303 Elm Blvd', '555-2345')
  8 INTO Vendor VALUES (7, 'PowerTech', '404 Birch Ct', '555-6789')
  9 INTO Vendor VALUES (8, 'MicroChip World', '505 Walnut Ave', '555-0123')
 10 INTO Vendor VALUES (9, 'ElectroCraft', '606 Spruce St', '555-4567')
 11 INTO Vendor VALUES (10, 'RealTech Electronics', '707 Pine Ln', '555-8901')
 12 SELECT * FROM DUAL;

10 rows created.
```

Figure 20 : Inserting values into Vendor table

The Vendor table was next populated with 10 rows of data.

```
SQL> SELECT * FROM Vendor;
```

VENDOR_ID	VENDOR_NAME	VENDOR_ADDRESS	VENDOR_NUMBER
1	TechGlobe Solutions	123 Main St	555-1234
2	ElectronicXpress	456 Oak Ave	555-5678
3	CircuitMasters	789 Pine Ln	555-9012
4	SparkTech	101 Maple Dr	555-3456
5	Silicon Valley	202 Cedar Rd	555-7890
6	Quantum Components	303 Elm Blvd	555-2345
7	PowerTech	404 Birch Ct	555-6789
8	MicroChip World	505 Walnut Ave	555-0123
9	ElectroCraft	606 Spruce St	555-4567
10	RealTech Electronics	707 Pine Ln	555-8901

```
10 rows selected.
```

Figure 21 : Displaying the values inserted into Vendor table

The values inserted into the Vendor table were then displayed as seen here.

```
SQL> INSERT ALL
2 INTO Product VALUES (1, 'Samsung S3', 'Smartphone', 'Productivity', 120, 100, 1)
3 INTO Product VALUES (2, 'Panasonic TV', 'UHD Television', 'Entertainment', 200, 75, 1)
4 INTO Product VALUES (3, 'Sony Camera', 'DSLR Camera', 'Cameras', 150, 60, 2)
5 INTO Product VALUES (4, 'Asus Laptop', 'Gaming Laptop', 'Gaming', 180, 70, 3)
6 INTO Product VALUES (5, 'Acer Monitor', 'HD Computer Monitor', 'Gaming', 100, 80, 3)
7 INTO Product VALUES (6, 'Apple iPad', 'iPad with M1 Chip', 'Productivity', 300, 55, 5)
8 INTO Product VALUES (7, 'Canon Camera', 'Cinematic Camera', 'Cameras', 170, 65, 3)
9 INTO Product VALUES (8, 'Dell Keyboard', 'Bluetooth Keyboard', 'Productivity', 50, 90, 4)
10 INTO Product VALUES (9, 'Casio Calculator', 'Scientific Calculator', 'Productivity', 80, 95, 4)
11 INTO Product VALUES (10, 'Toshiba Hard Drive', '1TB Hard Drive', 'Productivity', 70, 85, 4)
12 INTO Product VALUES (11, 'AOC Monitor', '4K Resolution Monitor', 'Gaming', 110, 55, 3)
13 INTO Product VALUES (12, 'JBL Speakers', 'Bass-boosted Speakers', 'Entertainment', 120, 75, 1)
14 INTO Product VALUES (13, 'Microsoft Surface', 'Tablet/Mini Laptop', 'Productivity', 400, 65, 4)
15 INTO Product VALUES (14, 'Nikon Camera', 'Ultra-ZoomIn Camera', 'Cameras', 250, 70, 7)
16 INTO Product VALUES (15, 'Razer Mouse', 'High Quality Gaming Mouse', 'Gaming', 90, 80, 7)
17 INTO Product VALUES (16, 'Vizio TV', 'HD Television', 'Entertainment', 300, 95, 7)
18 INTO Product VALUES (17, 'AOC Earbuds', 'Wireless Earbuds', 'Entertainment', 80, 60, 7)
19 INTO Product VALUES (18, 'Philips Hue Lights', 'Smart Lights', 'Lighting', 200, 55, 6)
20 INTO Product VALUES (19, 'Acer Chromebook', 'Notebook Laptop', 'Productivity', 150, 65, 8)
21 INTO Product VALUES (20, 'Corsair Keyboard', 'Gaming Keyboard', 'Gaming', 110, 75, 9)
22 INTO Product VALUES (21, 'iPhone 13', 'Smartphone', 'Productivity', 190, 100, 1)
23 INTO Product VALUES (22, 'Macbook Air', 'Laptop', 'Productivity', 220, 75, 10)
24 SELECT * FROM DUAL;
```

Figure 22 : Inserting values into Product table

The Product table was then populated with 22 rows of data, as depicted in the picture above.

```
SQL> SELECT * FROM Product;
```

PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION	PRODUCT_CATEGORY	PRODUCT_PRICE	PRODUCT_STOCK	VENDOR_ID
1	Samsung S3	Smartphone	Productivity	120	100	1
2	Panasonic TV	UHD Television	Entertainment	200	75	1
3	Sony Camera	DSLR Camera	Cameras	150	60	2
4	Asus Laptop	Gaming Laptop	Gaming	180	70	3
5	Acer Monitor	HD Computer Monitor	Gaming	100	80	3
6	Apple iPad	iPad with M1 Chip	Productivity	300	55	5
7	Canon Camera	Cinematic Camera	Cameras	170	65	3
8	Dell Keyboard	Bluetooth Keyboard	Productivity	50	90	4
9	Casio Calculator	Scientific Calculator	Productivity	80	95	4
10	Toshiba Hard Drive	1TB Hard Drive	Productivity	70	85	4
11	AOC Monitor	4K Resolution Monitor	Gaming	110	55	3
12	JBL Speakers	Bass-boosted Speakers	Entertainment	120	75	1
13	Microsoft Surface	Tablet/Mini Laptop	Productivity	400	65	4
14	Nikon Camera	Ultra-ZoomIn Camera	Cameras	250	70	7
15	Razer Mouse	High Quality Gaming Mouse	Gaming	90	80	7
16	Vizio TV	HD Television	Entertainment	300	95	7
17	AOC Earbuds	Wireless Earbuds	Entertainment	80	60	7
18	Philips Hue Lights	Smart Lights	Lighting	200	55	6
19	Acer Chromebook	Notebook Laptop	Productivity	150	65	8
20	Corsair Keyboard	Gaming Keyboard	Gaming	110	75	9
21	iPhone 13	Smartphone	Productivity	190	100	1
22	Macbook Air	Laptop	Productivity	220	75	10

22 rows selected.

Figure 23 : Displaying values inserted into Product table

The values inserted into the Product table were then displayed.

```
SQL> INSERT ALL
  2 INTO Product_Order VALUES (1, 1, 5)
  3 INTO Product_Order VALUES (1, 2, 8)
  4 INTO Product_Order VALUES (1, 3, 2)
  5 INTO Product_Order VALUES (2, 4, 3)
  6 INTO Product_Order VALUES (2, 5, 6)
  7 INTO Product_Order VALUES (3, 6, 4)
  8 INTO Product_Order VALUES (3, 7, 9)
  9 INTO Product_Order VALUES (4, 8, 7)
 10 INTO Product_Order VALUES (5, 9, 1)
 11 INTO Product_Order VALUES (5, 10, 5)
 12 INTO Product_Order VALUES (6, 1, 2)
 13 INTO Product_Order VALUES (7, 2, 8)
 14 INTO Product_Order VALUES (7, 3, 3)
 15 INTO Product_Order VALUES (8, 4, 6)
 16 INTO Product_Order VALUES (8, 5, 4)
 17 INTO Product_Order VALUES (9, 6, 7)
 18 INTO Product_Order VALUES (9, 7, 1)
 19 INTO Product_Order VALUES (10, 8, 9)
 20 INTO Product_Order VALUES (10, 9, 5)
 21 SELECT * FROM DUAL;

19 rows created.
```

Figure 24 : Inserting values into Product\_Order table

Next, 19 rows of data were inserted into the Product\_Order bridge entity as illustrated here.

```
SQL> SELECT * FROM Product_Order;
```

PRODUCT_ID	ORDER_ID	ORDER_QUANTITY
1	1	5
1	2	8
1	3	2
2	4	3
2	5	6
3	6	4
3	7	9
4	8	7
5	9	1
5	10	5
6	1	2
7	2	8
7	3	3
8	4	6
8	5	4
9	6	7
9	7	1
10	8	9
10	9	5

19 rows selected.

Figure 25 : Displaying the values inserted into Product\_Order table

The values inserted into the Product\_Order table were then displayed.

```
SQL> INSERT ALL
  2 INTO Customer_Product_Order VALUES (1, 1, 1)
  3 INTO Customer_Product_Order VALUES (2, 1, 2)
  4 INTO Customer_Product_Order VALUES (5, 1, 3)
  5 INTO Customer_Product_Order VALUES (4, 2, 4)
  6 INTO Customer_Product_Order VALUES (7, 2, 5)
  7 INTO Customer_Product_Order VALUES (1, 3, 6)
  8 INTO Customer_Product_Order VALUES (7, 3, 7)
  9 INTO Customer_Product_Order VALUES (2, 4, 8)
 10 INTO Customer_Product_Order VALUES (5, 5, 9)
 11 INTO Customer_Product_Order VALUES (5, 5, 10)
 12 INTO Customer_Product_Order VALUES (2, 6, 1)
 13 INTO Customer_Product_Order VALUES (4, 7, 2)
 14 INTO Customer_Product_Order VALUES (7, 7, 3)
 15 INTO Customer_Product_Order VALUES (1, 8, 4)
 16 INTO Customer_Product_Order VALUES (4, 8, 5)
 17 INTO Customer_Product_Order VALUES (2, 9, 6)
 18 INTO Customer_Product_Order VALUES (1, 9, 7)
 19 INTO Customer_Product_Order VALUES (7, 10, 8)
 20 INTO Customer_Product_Order VALUES (5, 10, 9)
 21 SELECT * FROM DUAL;

19 rows created.
```

Figure 26 : Inserting values into Customer\_Product\_Order table

Finally, the Customer\_Product\_Order table was populated with 19 rows of data as described by the picture above.



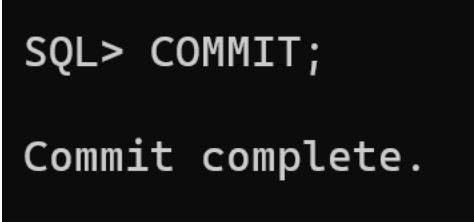
```
SQL> SELECT * FROM Customer_Product_Order;
```

CUSTOMER_ID	PRODUCT_ID	ORDER_ID
1	1	1
2	1	2
5	1	3
4	2	4
7	2	5
1	3	6
7	3	7
2	4	8
5	5	9
5	5	10
2	6	1
4	7	2
7	7	3
1	8	4
4	8	5
2	9	6
1	9	7
7	10	8
5	10	9

```
19 rows selected.
```

Figure 27 : Displaying the values inserted into Customer\_Product\_Order table

The values inserted into the Customer\_Product\_Order table were then displayed.

A terminal window with a black background and white text. The first line shows the SQL command 'SQL> COMMIT;' and the second line shows the output 'Commit complete.'

```
SQL> COMMIT;  
  
Commit complete.
```

*Figure 28 : Committing the insertion of values into all the tables*

The insertion of all the records into each table above was then made permanent by committing.

## 6. Database Querying

### 6.1. Information Queries

#### Listing the customers who are also the staff of the company

```
SQL> SPPOOL "\\Mac\Home\Desktop\coursework_spool.txt"
SQL> SELECT * FROM Customer WHERE Customer_Category = 'S';
```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_ADDRESS	CUSTOMER_NUMBER	CUSTOMER_CATEGORY
1	Jack	123 Main St	555-1234	S
4	Sarah	101 Maple Dr	555-3456	S
6	Amanda	303 Elm Blvd	555-2345	S
7	David	404 Birch Ct	555-6789	S
9	Brian	606 Spruce St	555-4567	S

Figure 29 : Listing all the customers that are also staff of the company

All details from the Customer table were selected where the Customer\_Category had a value of 'S'.

#### Listing all the orders made for any particular product between the dates 01-05-2023 and 28-05-2023

```
SQL> SELECT Order_ID, Product_ID, Order_Date FROM
2 "ORDER" NATURAL JOIN Product_Order
3 WHERE Order_Date BETWEEN '01-MAY-23' AND '28-MAY-23';
```

ORDER_ID	PRODUCT_ID	ORDER_DAT
2	1	23-MAY-23
4	2	04-MAY-23
7	3	14-MAY-23
10	5	22-MAY-23
2	7	23-MAY-23
4	8	04-MAY-23
7	9	14-MAY-23

7 rows selected.

Figure 30 : Listing all the orders made for any particular product between the dates 01-05-2023 till 28-05-2023

The Order and Product tables were naturally joined, followed by selecting the Order\_ID, Product\_ID, and Order\_Date attributes where the Order\_Date had values between '01-MAY-23' and '28-MAY-23'.

**Listing all the customers with their order details and also the customers who have not ordered any products yet**

```
SQL> SELECT c.Customer_ID, c.Customer_Name, cpo.Order_ID, o.Order_Date FROM
  2 Customer c LEFT JOIN Customer_Product_Order cpo ON c.Customer_ID = cpo.Customer_ID
  3 LEFT JOIN "ORDER" o ON o.Order_ID = cpo.Order_ID
  4 ORDER BY Order_ID;
```

CUSTOMER_ID	CUSTOMER_NAME	ORDER_ID	ORDER_DAT
2	Emily	1	12-FEB-23
1	Jack	1	12-FEB-23
4	Sarah	2	23-MAY-23
2	Emily	2	23-MAY-23
7	David	3	05-AUG-23
5	Michael	3	05-AUG-23
4	Sarah	4	04-MAY-23
1	Jack	4	04-MAY-23
7	David	5	07-APR-23
4	Sarah	5	07-APR-23
2	Emily	6	02-JUN-23
1	Jack	6	02-JUN-23
7	David	7	14-MAY-23
1	Jack	7	14-MAY-23
7	David	8	25-MAR-23
2	Emily	8	25-MAR-23
5	Michael	9	10-AUG-23
5	Michael	9	10-AUG-23
5	Michael	10	22-MAY-23
3	John		
12	Megan		
10	Lisa		
11	Chris		
6	Amanda		
8	Jessica		
9	Brian		

26 rows selected.

*Figure 31 : Listing all the customers with their order details and also the customers who have not ordered any products yet*

The Customer and Customer\_Product\_Order tables were left joined on Customer\_ID, followed by left joining the Order table on Order\_ID. Then, the Customer\_ID, Customer\_Name, Order\_ID and Order\_Date were selected, and the outputs were sorted by Order\_ID in ascending order.

### Listing the details of products that have 'a' as the second character in their name and have a stock quantity more than 50

```
SQL> SELECT * FROM Product
2 WHERE Product_Name LIKE '_a%' AND Product_Stock > 50;
```

PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION	PRODUCT_CATEGORY	PRODUCT_PRICE	PRODUCT_STOCK	VENDOR_ID
1	Samsung S3	Smartphone	Productivity	120	100	1
2	Panasonic TV	UHD Television	Entertainment	200	75	1
7	Canon Camera	Cinematic Camera	Cameras	170	65	3
9	Casio Calculator	Scientific Calculator	Productivity	80	95	4
15	Razer Mouse	High Quality Gaming Mouse	Gaming	90	80	7
22	Macbook Air	Laptop	Productivity	220	75	10

Figure 32 : Listing all product details that have the second letter 'a' in their product name and have a stock quantity more than 50

All details from the Product table were selected, where the values of Product\_Name had 'a' as the second letter while simultaneously having a value greater than 50 in the Product\_Stock attribute.

### Finding out the customer who has ordered recently

```
SQL> SELECT * FROM
2 (SELECT Customer_ID, Customer_Name, Order_ID, Order_Date FROM
3 Customer NATURAL JOIN Customer_Product_Order NATURAL JOIN "ORDER"
4 ORDER BY Order_Date DESC)
5 WHERE ROWNUM = 1;
```

CUSTOMER_ID	CUSTOMER_NAME	ORDER_ID	ORDER_DAT
5	Michael	9	10-AUG-23

Figure 33 : Finding out the customer who has ordered recently

The Customer, Customer\_Product\_Order, and Order tables were naturally joined, followed by selecting the Customer\_ID, Customer\_Name, Order\_ID and Order\_Date attributes. The output was sorted by Order\_Date in descending order, and only the topmost row was displayed.

## 6.2. Transaction Queries

### Finding the total revenue of the company in each month

```
SQL> SELECT TO_CHAR(Order_Date, 'YYYY-MM') AS Order_Month,
2  SUM(Product_Price * Order_Quantity * (1 - Discount)) AS Monthly_Total FROM
3  "ORDER" NATURAL JOIN Customer_Product_Order
4  NATURAL JOIN Product_Order
5  NATURAL JOIN Customer
6  NATURAL JOIN Customer_Category
7  NATURAL JOIN Product
8  GROUP BY TO_CHAR(Order_Date, 'YYYY-MM');

ORDER_M  MONTHLY_TOTAL
-----  -
2023-05      4965.5
2023-02        1170
2023-04        1330
2023-03      1858.5
2023-06        1130
2023-08      1174.5

6 rows selected.
```

Figure 34 : Showing the total revenue of the company for each month

The Order, Customer\_Product\_Order, Product\_Order, Customer, Customer\_Category, and Product Tables were naturally joined, followed by selecting the Order\_Date (only the year and month) and Monthly\_Total (as the sum of the products of Product\_Price, Order\_Quantity, and (1 – Discount)) attributes. The Monthly\_Total attribute was grouped by the year-and-month form of the Order\_Date attribute.

### Finding the orders that are equal to or higher than the average order total value

```
SQL> CREATE TABLE Order_Total AS
  2 (SELECT Order_ID,
  3 SUM(Product_Price * Order_Quantity * (1 - Discount)) AS Total_Amount FROM
  4 "ORDER" NATURAL JOIN Customer_Product_Order
  5 NATURAL JOIN Product_Order
  6 NATURAL JOIN Customer
  7 NATURAL JOIN Customer_Category
  8 NATURAL JOIN Product
  9 GROUP BY Order_ID);
```

Table created.

```
SQL> SELECT * FROM Order_Total ORDER BY Total_Amount DESC;
```

ORDER_ID	TOTAL_AMOUNT
2	2252
8	1858.5
7	1358.5
5	1330
1	1170
6	1130
4	855
3	724.5
10	500
9	450

10 rows selected.

Figure 35 : Creating a temporary table Order\_Total to store Order\_ID and Total\_Amount

A temporary table called Order\_Total was created to store the Order\_ID and Total\_Amount attributes. This was done by naturally joining the Order, Customer\_Product\_Order, Product\_Order, Customer, Customer\_Category, and Product tables, followed by the selection of the two attributes. The Total\_Amount attribute was calculated like in the previous query, i.e., the sum of the products of Product\_Price, Order\_Quantity, and (1 – Discount) attributes, grouped by Order\_ID.

The values set into the table were then displayed.

```
SQL> SELECT * FROM Order_Total
  2  WHERE Total_Amount >= (SELECT AVG(Total_Amount) FROM Order_Total)
  3  ORDER BY Total_Amount DESC;

ORDER_ID TOTAL_AMOUNT
-----
2         2252
8         1858.5
7         1358.5
5         1330
1         1170

SQL> DROP TABLE Order_Total;

Table dropped.
```

Figure 36 : Finding those orders that are equal or higher than the average order total value, and dropping the Order\_Total table

Next, all the orders from the Order\_Total table were selected where the Total\_Amount value was higher than or equal to the average of all Total\_Amount values. The table was then dropped as it was of no further use.

### Finding the vendors who supply more than 3 products to the company

```
SQL> SELECT Vendor_ID, Vendor_Name, COUNT(Product_ID) AS Num_of_Products
  2  FROM Vendor NATURAL JOIN Product
  3  GROUP BY Vendor_ID, Vendor_Name HAVING COUNT(Product_ID) > 3;

VENDOR_ID VENDOR_NAME NUM_OF_PRODUCTS
-----
1 TechGlobe Solutions 4
7 PowerTech 4
3 CircuitMasters 4
4 SparkTech 4
```

Figure 37 : Listing the details of vendors who have supplied more than 3 products to the company

The Vendor and Product tables were naturally joined, followed by selecting the Vendor\_ID, Vendor\_Name, and COUNT(Product\_ID) attributes. The COUNT(Product\_ID) was grouped by Vendor\_ID and Vendor\_Name, which returned the number of different Product\_ID values associated with each Vendor\_ID/Vendor\_Name value. From the outcome, only those records were returned where the value of COUNT(Product\_ID) was greater than 3.



### Finding the details of the top 3 products that have been ordered the most

```

SQL> SELECT * FROM
  2 (SELECT Product_ID, Product_Name, SUM(Order_Quantity) AS Times_Ordered FROM
  3 "ORDER" NATURAL JOIN Product_Order
  4 NATURAL JOIN Product
  5 GROUP BY Product_ID, Product_Name
  6 ORDER BY SUM(Order_Quantity) DESC)
  7 WHERE ROWNUM <= 3;

```

PRODUCT_ID	PRODUCT_NAME	TIMES_ORDERED
1	Samsung S3	15
10	Toshiba Hard Drive	14
3	Sony Camera	13

Figure 38 : Showing the top 3 product details that have been ordered the most

The Order, Product\_Order, and Product tables were naturally joined, followed by selecting the Product\_ID, Product\_Name, and COUNT(Order\_ID) attributes. The COUNT(Order\_ID) attribute was grouped by Product\_ID and Product\_Name, therefore giving the number of different Order\_ID values associated with each Product\_ID/Product\_Name value. The outcome was then sorted by COUNT(Product\_ID) in descending order, and only the top 3 rows were displayed.

### Finding the customer who spent the most in August with his/her total spending on that month

```

SQL> SELECT * FROM
  2 (SELECT Customer_ID, Customer_Name, Order_ID, TO_CHAR(Order_Date, 'YYYY-MON') AS Order_Month,
  3 SUM(Product_Price * Order_Quantity * (1 - Discount)) AS Total_Amount FROM
  4 Customer NATURAL JOIN Customer_Category
  5 NATURAL JOIN Customer_Product_Order
  6 NATURAL JOIN Product_Order
  7 NATURAL JOIN "ORDER"
  8 NATURAL JOIN Product
  9 WHERE EXTRACT(MONTH FROM Order_Date) = 8
  10 GROUP BY Order_ID, TO_CHAR(Order_Date, 'YYYY-MON'), Customer_ID, Customer_Name
  11 ORDER BY SUM(Product_Price * Order_Quantity * (1 - Discount)) DESC)
  12 WHERE ROWNUM = 1;

```

CUSTOMER_ID	CUSTOMER_NAME	ORDER_ID	ORDER_MONTH	TOTAL_AMOUNT
7	David	3	2023-AUG	484.5

Figure 39 : Finding out the customer who has ordered the most in August with his/her total spending on that month

The Customer, Customer\_Category, Customer\_Product\_Order, Product\_Order, Order, and Product tables were naturally joined, followed by selecting the Customer\_ID, Customer\_Name, Order\_ID, Order\_Date, and Total\_Amount attributes. The

Total\_Amount attribute was calculated like in previous queries as the sum of the products of Product\_Price, Order\_Quantity, and (1 – Discount) attributes, grouped by Order\_ID, Order\_Date, Customer\_ID and then Customer\_Name. This returned the total spending of each customer per order. Then, all records where the month in Order\_Date was August were selected using the EXTRACT function, and sorted by Total\_Amount, out of which only the topmost record was displayed.

## 7. Drop Query

```
SQL> DROP TABLE Customer_Product_Order;

Table dropped.

SQL> DROP TABLE Product_Order;

Table dropped.

SQL> DROP TABLE Product;

Table dropped.

SQL> DROP TABLE Vendor;

Table dropped.

SQL> DROP TABLE "ORDER";

Table dropped.

SQL> DROP TABLE Customer;

Table dropped.

SQL> DROP TABLE Customer_Category;

Table dropped.

SQL> SELECT table_name FROM user_tables;

no rows selected
```

Figure 40 : Dropping all the created tables in order

## 8. Dump File Creation

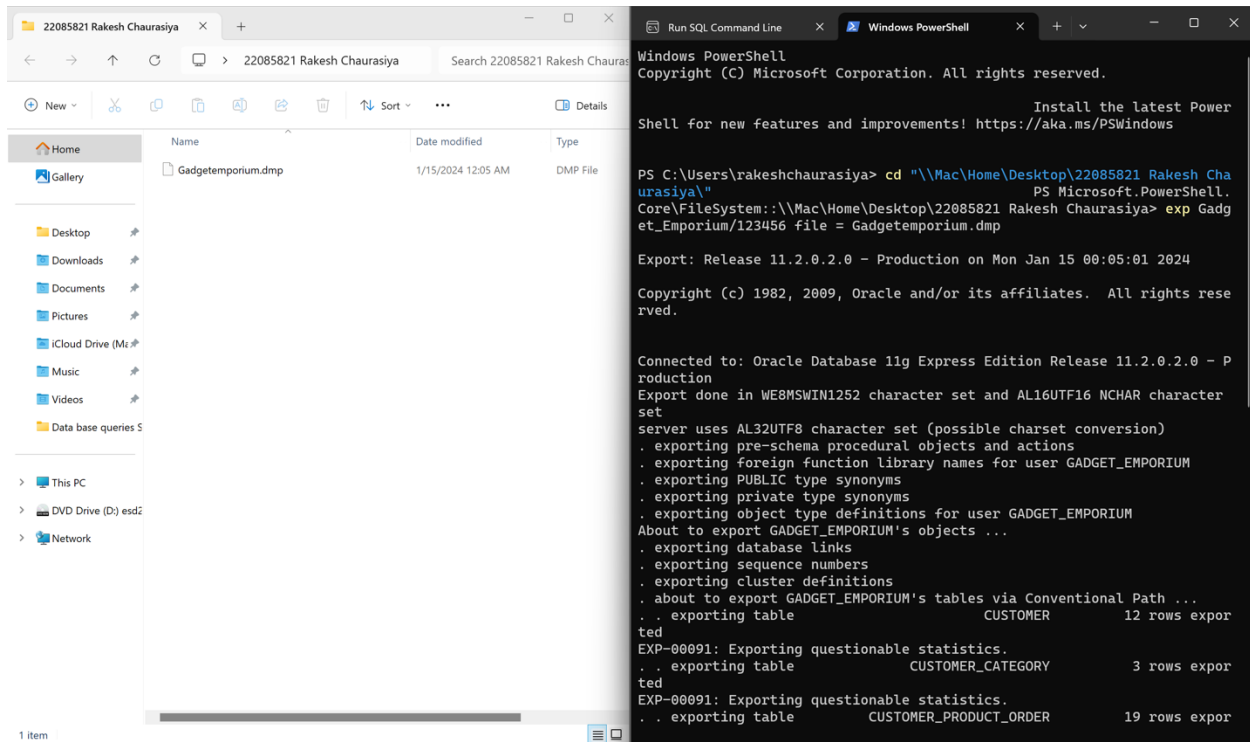


Figure 41 : Creating a dump file of the coursework

## **9. Critical Evaluation**

### **9.1. Critical Assessment of the Module**

The module cc5051NI databases provided a good knowledge on the basics of database creation, database management software, the importance of data and databases to a business, etc. The module also introduced and explained the concept of normalization, which is a fundamental concept in designing the schema for a database. It ensures the reduction of data redundancies and anomalies, but it still has some disadvantages like making the retrieval of data slower. Most of the concepts in the module were simple and easy, but some of them like normalization were very confusing. In short, the module was a great way of introducing the basics of databases.

### **9.2. Critical Evaluation of the Coursework**

Besides the module, this coursework also helped a lot in obtaining the practical experience in handling a database by using a database management software. The creation and description of each table was very simple and straightforward. Since the data inserted into the tables had to meet the requirements of the queries mentioned in the coursework, the insertion of data was really tedious and needed to be repeated many times in order to match the queries asked as well as to maintain the logic of the relationships between the tables. The queries were a little difficult but were quickly figured out and implemented. Overall, the coursework was a great way of getting started with practical knowledge on databases.