CC5051NI Databases

50% Individual Coursework

Autumn 2023

Student Name: Shirshak Aryal

London Met ID: 22085620

Assignment Submission Date: January 14, 2024

Word Count: 4877

Table of Contents

1. Introduction 1

1.1. About the Company 1

1.2. Current Business Activities and Operations 1

1.3. Business Rules 1

1.4. Entities and Attributes 3

2. Initial ERD (Entity Relationship Diagram) 5

2.1. Primary and Foreign Keys 6

2.2. ERD (Entity Relationship Diagram) 6

3. Normalization 8

3.1. UNF (Un-Normalized Form) 8

3.2. 1NF (1st Normalized Form) 8

3.3. 2NF (2nd Normalized Form) 9

3.3.1. Customer-1 9

3.3.2. Order-1 9

3.3.3. Product-1 10

3.3.4. All entities after 2NF 10

3.4. 3NF (3rd Normalized Form) 11

3.4.1. Customer-2 11

3.4.2. Order-2 11

3.4.3. Product-2 12

3.4.4. Remaining Bridge Entities 13

3.4.5. All 3NF entities 13

3.5. Final 3NF Entities 14

4. Final ERD (Entity Relationship Diagram) 15

5. Implementation 18

5.1. Creating the Gadget_Emporium User 18

5.2. Creating the Tables 19

5.3. Inserting Data into the Tables 25

6. Querying the Database 37

6.1. Information Queries 37

6.2. Transaction queries 40

7. Critical Evaluation 46

7.1. Critical Evaluation of the Module 46

7.2. Critical Evaluation of the Coursework 46

8. Drop Queries and Dump File Creation 48

8.1. Drop Queries 48

8.2. Dump File Creation 49

Table of Figures

Figure 1: One-to-many optional relationship between Customer and Order 5
Figure 2: Optional many-to-many relationship between Order and Product............... 5
Figure 3: Initial ERD – 1 6
Figure 4: Initial ERD – 2 7
Figure 5: Final ERD – 1 15
Figure 6: Final ERD – 2 16
Figure 7: Connecting to the System and creating the Gadget_Emporium user........ 18
Figure 8: Granting CONNECT and RESOURCE privileges to Gadget_Emporium .. 18
Figure 9: Connecting to Gadget_Emporium 18
Figure 10: Creating and describing the Customer table 19
Figure 11: Creating and describing the Order table 20
Figure 12: Creating and describing the Vendor table 20
Figure 13: Creating and describing the Product table 21
Figure 14: Creating and describing the Customer_Order table 22
Figure 15: Creating and describing the Order_Product table 23
Figure 16: Creating and describing the Customer_Order_Product table 24
Figure 17: Listing all tables created in Gadget_Emporium 24
Figure 18: Inserting values into the Customer table 25
Figure 19: Displaying the values inserted into the Customer table 25
Figure 20: Inserting values into the Order table 26
Figure 21: Displaying the values inserted into the Order table 27
Figure 22: Inserting values into the Vendor table 27
Figure 23: Displaying the values inserted into the Vendor table 28
Figure 24: Inserting values into the Product table 28
Figure 25: Displaying the values inserted into the Product table 29
Figure 26: Inserting values into the Customer_Order table 30
Figure 27: Displaying the values inserted into the Customer_Order table 31
Figure 28: Inserting values into the Order_Product table 32
Figure 29: Displaying the values inserted into the Order_Product table 33
Figure 30: Inserting values into the Customer_Order_Product table 34
Figure 31: Displaying the values inserted into the Customer_Order_Product table . 35
Figure 32: Committing the insertions of values into the tables 36

Figure 33: Listing all customers that are also the staff of the company 37
Figure 34: Listing all orders made for any particular product between the dates 01-05-2023 and 28-05-2023 37
Figure 35: Listing all the customers with their order details and also the customers who have not ordered any products yet 38
Figure 36: Listing all product details that have the second letter ‘a’ in their product name and have a stock quantity more than 50 39
Figure 37: Finding the customer who has ordered recently...................................... 39
Figure 38: Showing the total revenue of the company for each month 40
Figure 39: Creating a temporary Ord_Total table with the order IDs and total amounts 41
Figure 40: Finding the orders that are equal or higher than the average order total value.. 42
Figure 41: Dropping the Ord_Total table 42
Figure 42: Listing the details of vendors who have supplied more than 3 products to the company.. 43
Figure 43: Showing the top 3 product details that have been ordered the most....... 44
Figure 44: Finding the customer who has ordered the most in August with his/her total spending on that month 45
Figure 45: Dropping all tables in order 48
Figure 46: Creating a dump file of the coursework 49

Table of Tables

Table 1: Attributes of the Customer entity 3
Table 2: Attributes of the Order entity 3
Table 3: Attributes of the Product entity 4
Table 4: Entities with their Primary and Foreign Keys before normalization 6
Table 5: Entities with their Primary and Foreign Keys after normalization 17

1. Introduction

1.1. About the Company

Gadget Emporium is an electronics marketplace founded in 2022 by entrepreneur and electronics enthusiast Mr. John. The head office of this marketplace is situated in 623 H. Cashew Street, Rivendell, USA. The business specializes in the retailing of electronic devices and accessories that include a diverse range of products, including TVs, laptops, smartphones, and more. As online marketing has picked up great momentum in the modern era, Gadget Emporium seeks to leverage this and provide a convenient and comprehensive online platform for the buying and selling of electronic gadgets with provision of online payment through mobile banking/credit and debit cards, as well as cash-on-delivery.

1.2. Current Business Activities and Operations

Gadget Emporium currently has a system in which the customer details are recorded when the customer creates an account for the website. Product details, on the other hand, are requested to the respective vendors and recorded. The business advertises itself in several ways including social media, billboards, noticeboards, electronics conventions, etc. Ongoing trends, seasonal trends and popular electronic gadgets are inferred and noted from sources like blogs, statistical information from websites, web traffic analyzing tools, etc. which are then utilized for maximum profit.

1.3. Business Rules

- The system is expected to record the details of electronic devices, such as product names, descriptions, categories, prices, and stock levels.
- A specific product must belong to a single category and each category can contain multiple products.
- The system should be able to record the details of all its customers such as their name, address, and phone number.

- Customers can belong to either one of three categories: Regular (R), Staff (S), and VIP (V). Customers in those categories are granted discounts of 0%, 5% and 10% respectively.
- Each customer's address is stored to be utilized in the delivery process.
- A customer must be associated with exactly one phone number; two different numbers are recorded as belonging to two different customers. However, customers will have the option to update their phone number in the future.
- Customers can browse the website and order one or more electronic gadgets online. An order can include more than one product and any one type of product can be included in multiple orders from multiple customers.
- The system should record the details of every order, including the details of the products purchased, quantities, unit prices, total amounts, and discounted total amounts.
- An order need not be confirmed immediately; an invoice is generated only once the order has been confirmed.
- Invoice details are obtained from the details of the orders themselves, including the customer and payment details (along with the discounted price).
- The discount rate is applied to the line total price of each product in an order, based on the category to which the customer belongs.
- The system should keep records of vendors that supply the electronic devices to the store. A particular product can be supplied by a single vendor, and a vendor can supply one or more products.
- A vendor must have exactly one email address associated with it, and two different addresses are recorded as belonging to two different vendors.
- The system should be able to track the stock quantities of each product in real time to prevent overselling, by recording inventory details like the stock quantity or availability status.
- The system should integrate with numerous payment gateways to ensure secure, seamless transactions for each order.
- Payment options must be either cash on delivery, credit/debit card or e-wallet. An order can be paid for using only one of the payment options.

1.4. Entities and Attributes

The business rules mentioned above can be used to derive the entities and attributes involved, which are shown below:

ENTITY	ATTRIBUTES	DATATYPES	DESCRIPTION
Customer	cust_ID	NUMBER	This is a unique identifier for each customer.
	cust_name	VARCHAR2	This is the full name of each customer.
	cust_address	VARCHAR2	This is the address (city and US state) of each customer.
	cust_phone	NUMBER	This is the phone number of each customer.
	cust_category	VARCHAR2	This is the category to which a customer belongs, i.e., Regular (R), Staff (S), and VIP (V).

Table 1: Attributes of the Customer entity

The **Customer** entity stores the details of customers in the attributes mentioned in the table above. The **cust_ID** attribute stores a **number** that serves as a **unique identifier** for each customer. The **cust_name** attribute holds the **full name** of the customer, while the **cust_address** attribute denotes the **address** of the customer which is **used as delivery addresses**. The **cust_phone** and **cust_category** attributes store the **phone number** and **category (Regular, Staff, or VIP)** of each customer.

ENTITY	ATTRIBUTES	DATATYPES	DESCRIPTION
Order	ord_ID	NUMBER	This is the unique identifier for each order.
	ord_date	DATE	This is the date each order was placed in.
	discount_rate	NUMBER	This is the discount rate applied to the line totals of each product in an order, based on the category the customer belongs to.

Table 2: Attributes of the Order entity

The **Order** entity, on the other hand, stores the details of each order placed by customers. The **ord_ID** attribute stores a **number** that **uniquely identifies each order**. The **ord_date** and **discount_rate** attributes hold the **date placed** and **applied discount rate** on the **line totals of every product** of each order, respectively.

ENTITY	ATTRIBUTES	DATATYPES	DESCRIPTION
Product	prod_ID	NUMBER	This is a unique identifier for each product.
	prod_name	VARCHAR2	This is the name of each product.
	prod_desc	VARCHAR2	This is the description for each product.
	prod_category	VARCHAR2	This is the category each product belongs to.
	prod_price	NUMBER	This is the unit price of each product.
	prod_stock	NUMBER	This is the stock quantity of each product.
	order_qty	NUMBER	This is the quantity of a product included in an order.
	vendor_ID	NUMBER	This is a unique identifier of each vendor.
	vendor_name	VARCHAR2	This is the name of each vendor.
	vendor_address	VARCHAR2	This is the address (city, country) of each vendor.
	vendor_mail	VARCHAR2	This is the email address of each vendor.

Table 3: Attributes of the Product entity

Lastly, the **Product** entity stores the details of all products as depicted in the table above. The **prod_ID** attribute holds a **number** that acts as a **unique identifier** for **each product**. The **prod_name** and **prod_desc** attributes denote the **names** and **descriptions** of each product, respectively. The **prod_price** attribute stores the **unit price (in USD)** while the **prod_stock** attribute stores the available **stock quantity** of the product. The **order_qty** attribute stores the **quantity of the product** that was included **in a particular order**. The **vendor_ID** attribute stores a **number** that **uniquely identifies** the **vendor** that supplies the product to the company. The **vendor_name, vendor_address,** and **vendor_mail** attributes store the **name, address**, and **email address** of the vendor, respectively.

2. Initial ERD (Entity Relationship Diagram)

The three entities are related to each other as described by the diagrams below:

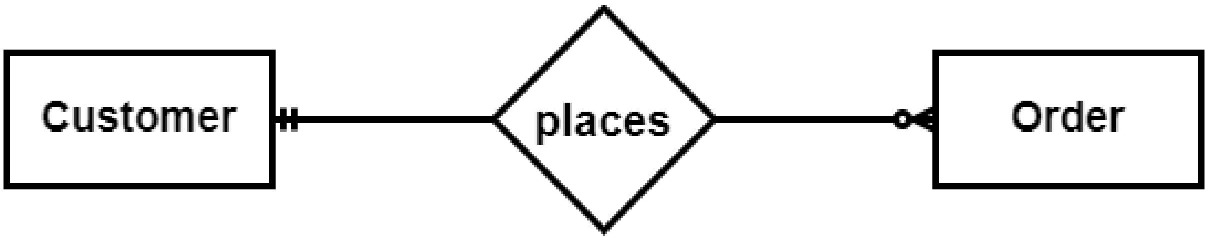


Figure 1: One-to-many optional relationship between Customer and Order

The diagram above shows that **Customer** has a **one-to-many optional** relationship with **Order**, since a **customer** can **place multiple orders** as well as **no orders**, while a **specific order** can be placed by **exactly one customer**.

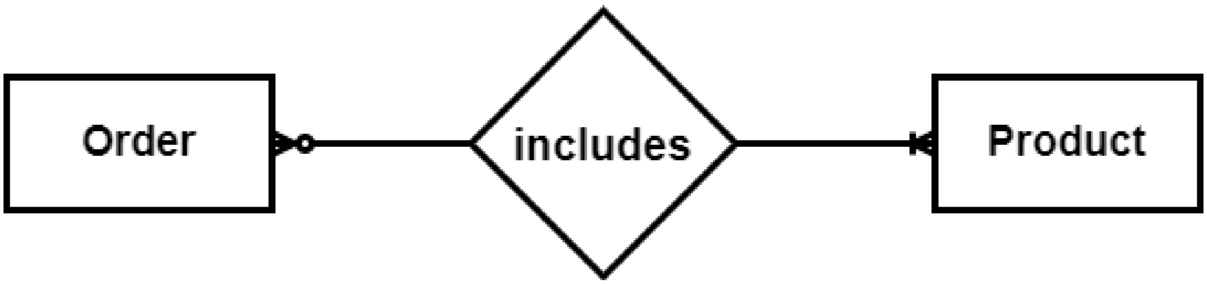


Figure 2: Optional many-to-many relationship between Order and Product

This diagram, on the other hand, shows that **Order** has an **optional many-to-many relationship** with **Product**. This is because an **order** must have **at least one product** included in it, while a **particular product** may be included in **multiple orders or none**.

2.1. Primary and Foreign Keys

The entities with their primary and foreign keys are listed in the table below:

ENTITY	PRIMARY KEY	FOREIGN KEY(S)
Customer	cust_ID	-
Order	ord_ID	cust_ID, prod_ID
Product	prod_ID	ord_ID

Table 4: Entities with their Primary and Foreign Keys before normalization

2.2. ERD (Entity Relationship Diagram)

The entities, their attributes, and the relationships between each of them are described in the initial entity-relationship diagrams as seen below:

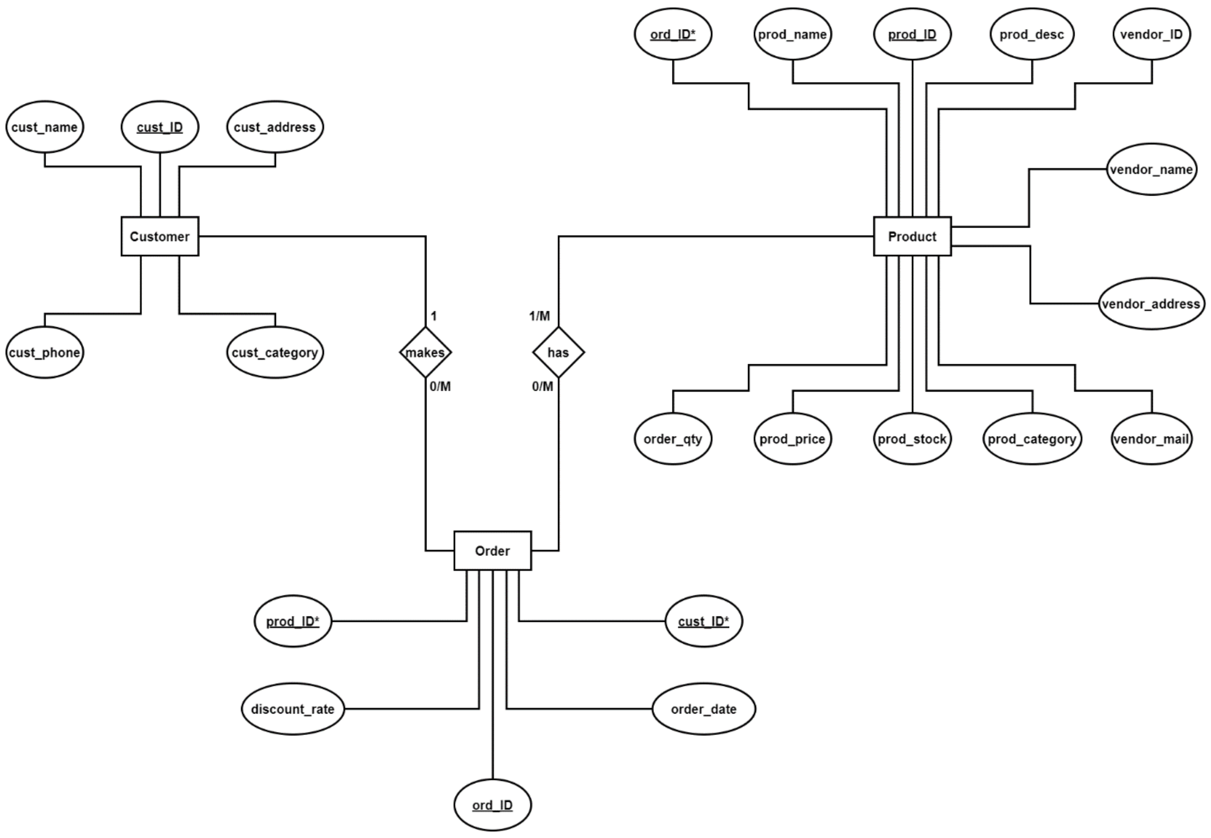


Figure 3: Initial ERD – 1

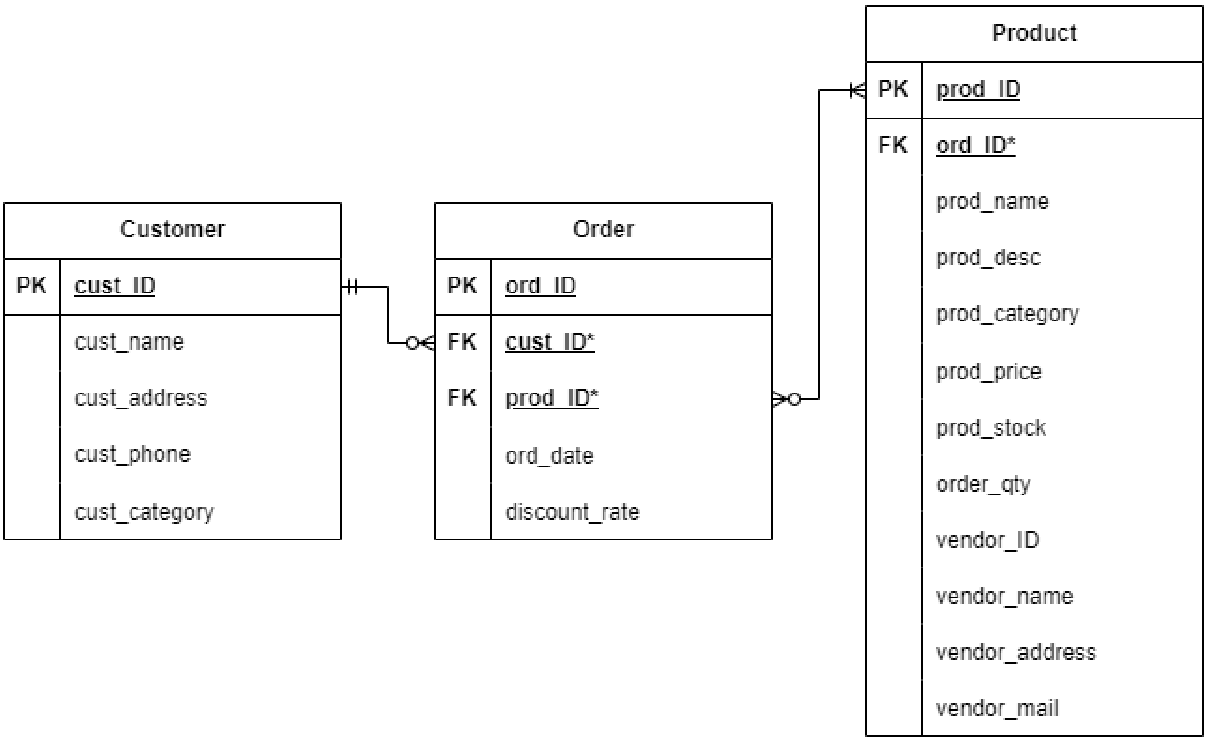


Figure 4: Initial ERD – 2

The initial ERD, represented in both formats above, shows the type of relationships that the entities have with each other. The **Customer** entity has a **one-to-many optional** relationship with the **Order** entity since a **customer** can either make multiple orders or need not have made an order yet, and an **order** must have **exactly one customer** as specified in the business rules. The **Product** entity, on the other hand, has a **many-to-many optional** relationship with the **Order** entity. This is because an **order** must have **at least one product** included, and **products** may be included in **one or multiple orders**, or none. The **primary key** of the **Order** attribute (**ord_ID**) has been included as the **foreign key** in the **Product** entity, and that of the **Product** entity (**prod_ID**) has been included as the **foreign key** in the **Order** entity. Similarly, the **Order** entity also has **cust_ID** as a **foreign key**, which is the **primary key** of the **Customer** entity.

This ERD contains **no fan traps or chasm traps**, but it does **contain a many-to-many relationship** (between the **Order** and **Product** entities) which can create data redundancy and integrity issues. This can be **resolved through normalization**.

3. Normalization

3.1. UNF (Un-Normalized Form)

Customer (cust_ID, cust_name, cust_address, cust_phone, cust_category, {ord_ID, ord_date, discount_rate, {prod_ID, prod_name, prod_desc, prod_category, prod_price, prod_stock, order_qty, vendor_ID, vendor_name, vendor_address, vendor_mail}})

Here, **cust_ID** is chosen as the **candidate key**, for which the **attributes of the Customer entity** are **repeating data**. The **attributes of the Order entity** are a **repeating group**, inside which the **attributes of the Product entity** are again a **repeating group**. This is because a **customer** can have **multiple orders**, and an **order** can include **multiple products**.

3.2. 1NF (1st Normalized Form)

Separating the repeating groups from the repeating data into separate entities, the following 1NF entities are formed:

1. **Customer-1** (cust_ID, cust_name, cust_address, cust_phone, cust_category)
2. **Order-1** (cust_ID *, ord_ID, ord_date, discount_rate)
3. **Product-1** (custID*, ord_ID*, prod_ID, prod_name, prod_desc, prod_category, prod_price, prod_stock, order_qty, vendor_ID, vendor_name, vendor_address, vendor_mail)

3.3. 2NF (2^{nd} Normalized Form)

Checking for **partial dependencies** in:

3.3.1. Customer-1

Customer-1 has only **one key attribute** and hence has **no partial dependencies**, so is **already in 2NF.** Therefore,

- **Customer-2** (<u>cust_ID</u>, cust_name, cust_address, cust_phone, cust_category)

3.3.2. Order-1

Order-1 has **two key attributes**, which may be causing partial dependencies. Using the formula **(2^n – 1)**, where **n = 2** (i.e., the number of key attributes), gives **3 possible key combinations**.

1. **ord_ID, cust_ID → discount_rate**
2. **ord_ID → ord_date**
3. cust_ID → Nothing

Here, the **discount rate** is applied to the **order** itself, but its value **depends on the customer category**, hence its dependency shown above. Similarly, the **date and time** of an order as well as its **confirmation status** depend **only on the order** itself, hence they are shown as such.

Therefore, **the 2NF entities formed from Order-1** are:

- **Order-2** (<u>ord_ID</u>, ord_date)
- **Customer-Order-2** (<u>cust_ID</u>*, <u>ord_ID</u>*, discount_rate)

3.3.3. Product-1

Product-1 has **three key attributes**, which may also be causing partial dependencies. Using the formula **(2^n – 1)** gives **7 possible key combinations**.

1. prod_ID, cust_ID, ord_ID → Nothing
2. prod_ID, cust_ID → Nothing
3. cust_ID, ord_ID → (this entity has already been created above)
4. **prod_ID, ord_ID → order_qty**
5. **prod_ID → prod_name, prod_desc, prod_category, prod_price, prod_stock, vendor_ID, vendor_name, vendor_address, vendor_mail**
6. cust_ID → Nothing
7. ord_ID → Nothing

Therefore, the **2NF entities formed from Product-1** are:

8. **Product-2** (<u>prod_ID</u>, prod_name, prod_desc, prod_category, prod_price, prod_stock, order_qty, vendor_ID, vendor_name, vendor_address, vendor_mail)

- **Customer-Order-Product-2** (<u>cust_ID</u>*, <u>ord_ID</u>*, <u>prod_ID</u>*)
- **Order-Product-2** (<u>ord_ID</u>*, <u>prod_ID</u>*, order_qty)
- **Customer-Product-2** (<u>cust_ID</u>*, <u>prod_ID</u>*)

3.3.4. All entities after 2NF

- **Customer-2** (<u>cust_ID</u>, cust_name, cust_address, cust_phone, cust_category)
- **Order-2** (<u>ord_ID</u>, ord_date)
- **Customer-Order-2** (<u>cust_ID</u>*, <u>ord_ID</u>*, discount_rate)
- **Product-2** (<u>prod_ID</u>, prod_name, prod_desc, prod_category, prod_price, prod_stock, vendor_ID, vendor_name, vendor_address, vendor_mail)
- **Customer-Order-Product-2** (<u>cust_ID</u>*, <u>ord_ID</u>*, <u>prod_ID</u>*)
- **Order-Product-2** (<u>ord_ID</u>*, <u>prod_ID</u>*, order_qty)
- **Customer-Product-2** (<u>cust_ID</u>*, <u>prod_ID</u>*)

3.4. 3NF (3rd Normalized Form)

Checking for **transitive dependencies** in:

3.4.1. Customer-2

Customer-2 (<u>cust_ID</u>, cust_name, cust_address, cust_phone, cust_category)

Here,

- <u>cust_ID</u> → cust_name, cust_address, cust_phone, cust_category → Nothing

The above assumption is made since **none of the non-key attributes have any other non-key attributes dependent on them**, which implies that **transitive dependencies do not exist**. For example, multiple customers can have identical names, but may not necessarily have the same address or phone number and may not belong to the same category. Similarly, an address alone cannot determine the name, phone number and category of a customer, and so goes for the category as well. As for phone numbers, they do have the ability to uniquely identify each customer but are volatile as customers can change their phone numbers repeatedly.

Since there are **no transitive dependencies** in **Customer-2,** it is **already in 3NF**. That is,

- **Customer-3** (<u>cust_ID</u>, cust_name, cust_address, cust_phone, cust_category)

3.4.2. Order-2

Order-2 (<u>ord_ID</u>, ord_date)

There is only **one non-key attribute** here, implying that there is **no transitive dependency** since **transitive dependencies require more than one non-key attribute** to exist. Hence, **Order-2** is **already in 3NF**, i.e.,

- **Order-3** (<u>ord_ID</u>, ord_date)

3.4.3. Product-2

Product-2 (prod_ID, prod_name, prod_desc, prod_category, prod_price, prod_stock, vendor_ID, vendor_name, vendor_address, vendor_mail)

Here,

- **prod_ID** → prod_name, prod_desc, prod_category, prod_price, prod_stock, vendor_name, vendor_address, vendor_mail → Nothing

The above assumption can be made since no non-key attribute is dependent on the other. For example, multiple products can have the same name but may have different descriptions, categories, prices, and stock levels, and may be supplied by different vendors. The same logic goes for other attributes as well since none of them are unique to a product. However,

- **prod_ID** → vendor_ID → vendor_name, vendor_address, vendor_mail

This assumption is valid since the ID of a vendor is unique to each vendor, and so can determine the name, address, and email address of the vendor. Hence, **transitive dependency** is seen in the case of **vendor_ID**, which can be resolved as follows:

- **prod_ID** → vendor_ID
- **vendor_ID** → vendor_name, vendor_address, vendor_mail

This creates **two new 3NF entities**:

- **Product-3** (prod_ID, prod_name, prod_desc, prod_category, prod_price, prod_stock, vendor_ID*)
- **Vendor-3** (vendor_ID, vendor_name, vendor_address, vendor_mail)

3.4.4. Remaining Bridge Entities

For the **remaining entities**, i.e.:

- **Customer-Order-Product-2** (<u>cust_ID</u>*, <u>ord_ID</u>*, <u>prod_ID</u>*)
- **Order-Product-2** (<u>ord_ID</u>*, <u>prod_ID</u>*, order_qty)
- **Customer-Product-2** (<u>cust_ID</u>*, <u>prod_ID</u>*)
- **Customer-Order-2** (<u>cust_ID</u>*, <u>ord_ID</u>*, discount_rate)

No transitive dependencies exist in them due to the presence of either **only one non-key attribute, or none**. Hence, they are already in 3NF.

i.e.,

- **Customer-Order-Product-3** (<u>cust_ID</u>*, <u>ord_ID</u>*, <u>prod_ID</u>*)
- **Order-Product-3** (<u>ord_ID</u>*, <u>prod_ID</u>*, order_qty)
- **Customer-Product-3** (<u>cust_ID</u>*, <u>prod_ID</u>*)
- **Customer-Order-3** (<u>cust_ID</u>*, <u>ord_ID</u>*, discount_rate)

3.4.5. All 3NF entities

- **Customer-3** (<u>cust_ID</u>, cust_name, cust_address, cust_phone, cust_category)
- **Order-3** (<u>ord_ID</u>, ord_date)
- **Product-3** (<u>prod_ID</u>, prod_name, prod_desc, prod_category, prod_price, prod_stock, <u>vendor_ID*</u>)
- **Vendor-3** (<u>vendor_ID</u>, vendor_name, vendor_address, vendor_mail)
- **Customer-Order-3** (<u>cust_ID</u>*, <u>ord_ID</u>*, discount_rate)
- **Order-Product-3** (<u>ord_ID</u>*, <u>prod_ID</u>*, order_qty)
- **Customer-Product-3** (<u>cust_ID</u>*, <u>prod_ID</u>*)
- **Customer-Order-Product-3** (<u>cust_ID</u>*, <u>ord_ID</u>*, <u>prod_ID</u>*)

3.5. Final 3NF Entities

Out of the 3NF entities obtained, the **Customer-Product-3** (cust_ID*, prod_ID*) entity can be **omitted**, since it has **no other attributes** besides the two foreign keys, and **Customer-Order-Product-3** (cust_ID*, ord_ID*, prod_ID*) already exists as the **main bridge entity.**

Hence, the entities that are required for the database schema are:

1. **Customer-3** (cust_ID, cust_name, cust_address, cust_phone, cust_category)
2. **Order-3** (ord_ID, ord_date)
3. **Product-3** (prod_ID, prod_name, prod_desc, prod_category, prod_price, prod_stock, vendor_ID*)
4. **Vendor-3** (vendor_ID, vendor_name, vendor_address, vendor_mail)
5. **Customer-Order-3** (cust_ID*, ord_ID*, discount_rate)
6. **Order-Product-3** (ord_ID*, prod_ID*, order_qty)
7. **Customer-Order-Product-3** (cust_ID*, ord_ID*, prod_ID*)

4. Final ERD (Entity Relationship Diagram)

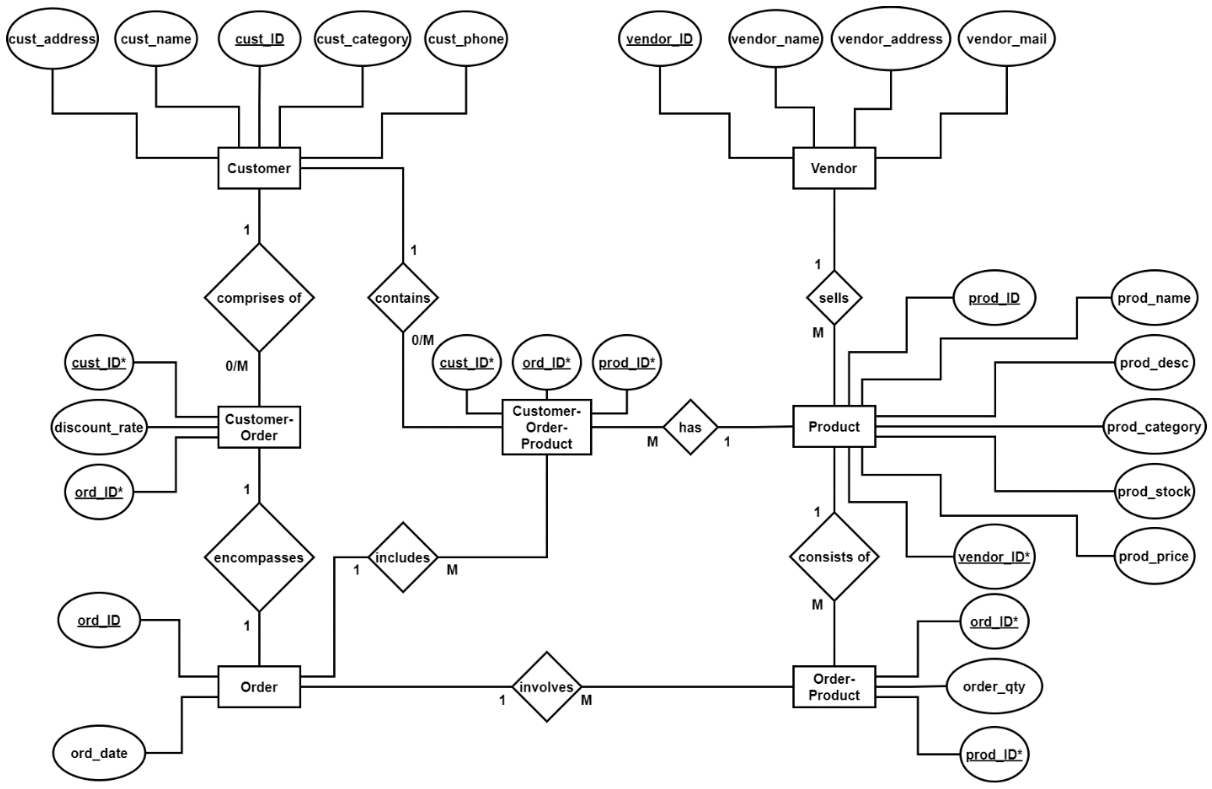


Figure 5: Final ERD – 1

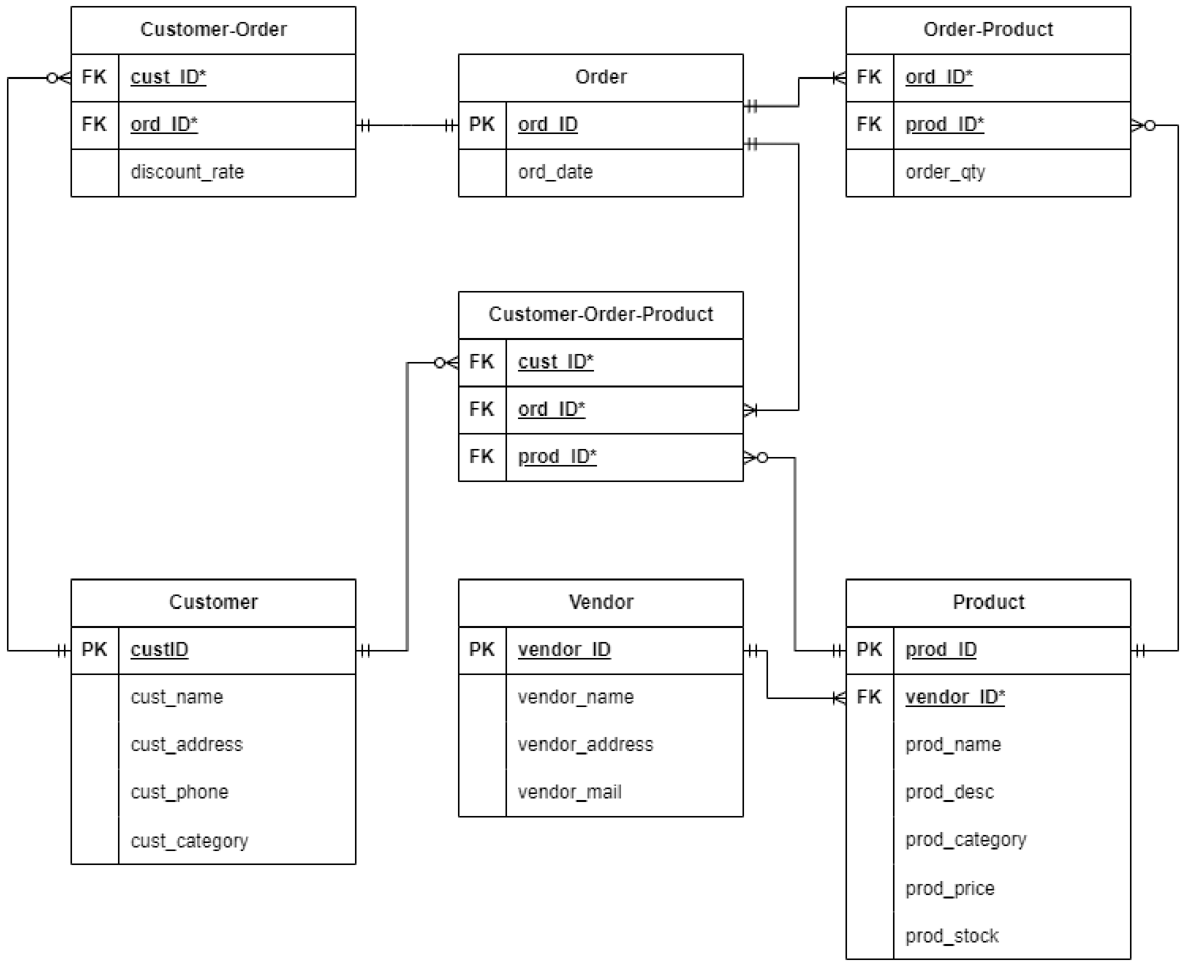


Figure 6: Final ERD – 2

In the final ERDs pictured above, it is evident that **all potential traps have been resolved** through normalization, including the **many-to-many relationship** (between the **Order** and **Product** entities).

Here, **Customer** has a **one-to-many optional relationship** with **Customer-Order**. **Order** has a **one-to-one relationship** with **Customer-Order** a **one-to-many relationship** with **Order-Product**. **Product** has a **one-to-many optional relationship** with **Order-Product** and a **many-to-one relationship** with **Vendor**. Lastly, **Order** has a **one-to-many relationship** while **Customer** and **Product** each have a **one-to-many optional relationship** with the main bridge entity **Customer-Order-Product.**

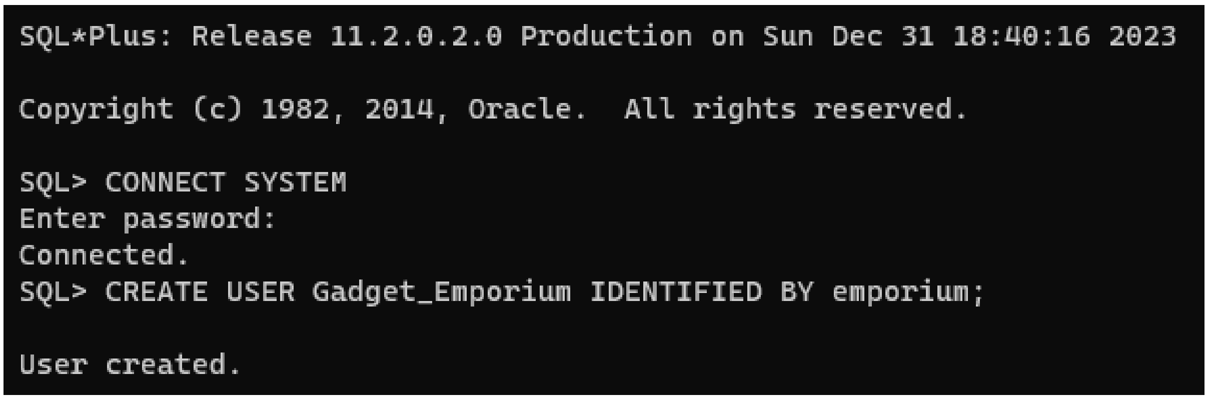
The primary and foreign keys of all entities are displayed in the following table:

ENTITIES	PRIMARY KEY	FOREIGN KEY(S)
Customer	cust_ID	-
Order	ord_ID	-
Product	prod_ID	vendor_ID
Vendor	vendor_ID	-
Customer-Order	cust_ID + ord_ID (composite)	cust_ID, ord_ID
Order-Product	ord_ID + prod_ID (composite)	ord_ID, prod_ID
Customer-Order-Product	cust_ID + ord_ID + prod_ID (composite)	cust_ID, prod_ID, ord_ID

Table 5: Entities with their Primary and Foreign Keys after normalization

5. Implementation

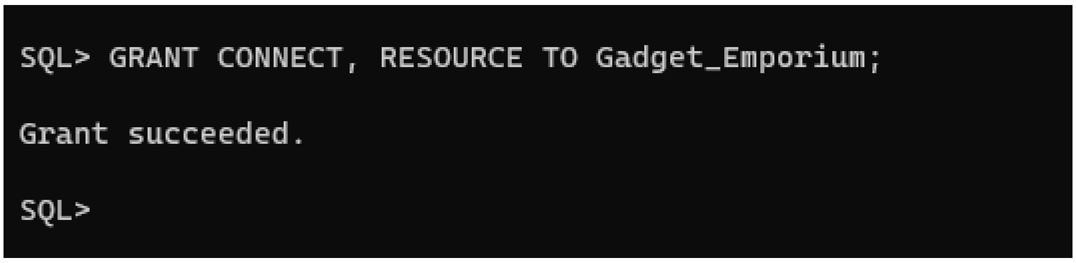
5.1. Creating the Gadget_Emporium User



```
SQL*Plus: Release 11.2.0.2.0 Production on Sun Dec 31 18:40:16 2023

Copyright (c) 1982, 2014, Oracle.  All rights reserved.

SQL> CONNECT SYSTEM
Enter password:
Connected.
SQL> CREATE USER Gadget_Emporium IDENTIFIED BY emporium;

User created.
```

Figure 7: Connecting to the System and creating the Gadget_Emporium user

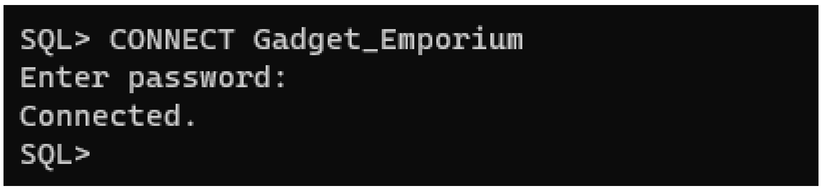
The **system** was **connected** to in the SQL server, and the **Gadget_Emporium user** was **created** which is identified by the **password** ‘emporium’.



```
SQL> GRANT CONNECT, RESOURCE TO Gadget_Emporium;

Grant succeeded.

SQL>
```

Figure 8: Granting CONNECT and RESOURCE privileges to Gadget_Emporium

The Gadget_Emporium user was then **granted** the **Connect and Resource privileges**.



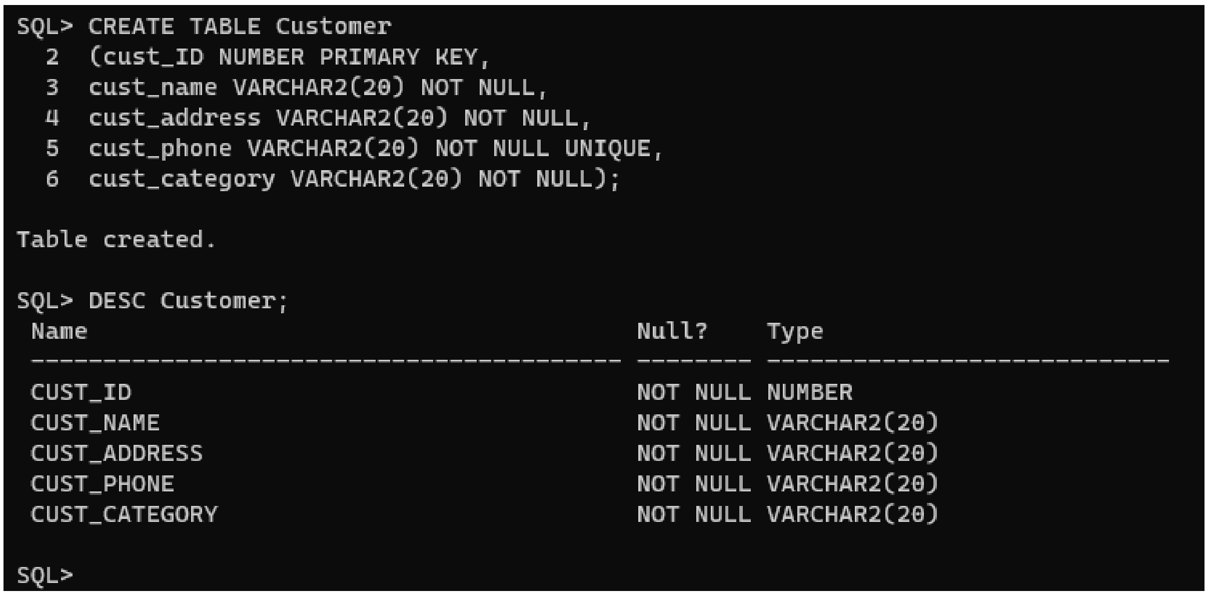
```
SQL> CONNECT Gadget_Emporium
Enter password:
Connected.
SQL>
```

Figure 9: Connecting to Gadget_Emporium

At last, the **Gadget_Emporium user** was **connected** to successfully.

5.2. Creating the Tables

Creating the Customer table

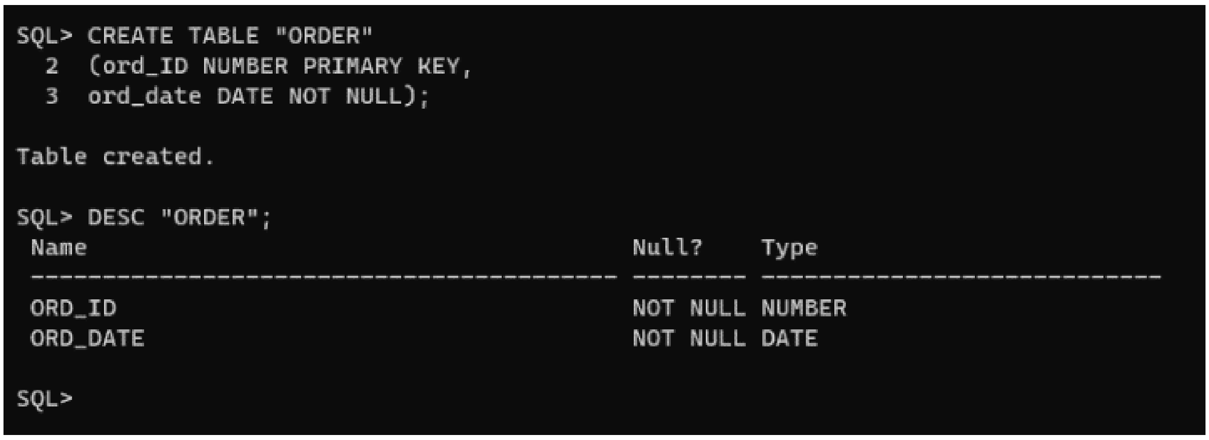


```
SQL> CREATE TABLE Customer
  2  (cust_ID NUMBER PRIMARY KEY,
  3  cust_name VARCHAR2(20) NOT NULL,
  4  cust_address VARCHAR2(20) NOT NULL,
  5  cust_phone VARCHAR2(20) NOT NULL UNIQUE,
  6  cust_category VARCHAR2(20) NOT NULL);

Table created.

SQL> DESC Customer;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_NAME                                 NOT NULL VARCHAR2(20)
 CUST_ADDRESS                              NOT NULL VARCHAR2(20)
 CUST_PHONE                                NOT NULL VARCHAR2(20)
 CUST_CATEGORY                             NOT NULL VARCHAR2(20)

SQL>
```

Figure 10: Creating and describing the Customer table

The creation of the 3NF entities was started by creating the Customer table as shown above, with the **NOT NULL** constraint used **for all values**, since a **customer cannot be without any of those attributes**. Additionally, the **UNIQUE** constraint was used in the **cust_phone** attribute since the **phone number** of a customer is **unique** to them.

Creating the Order table

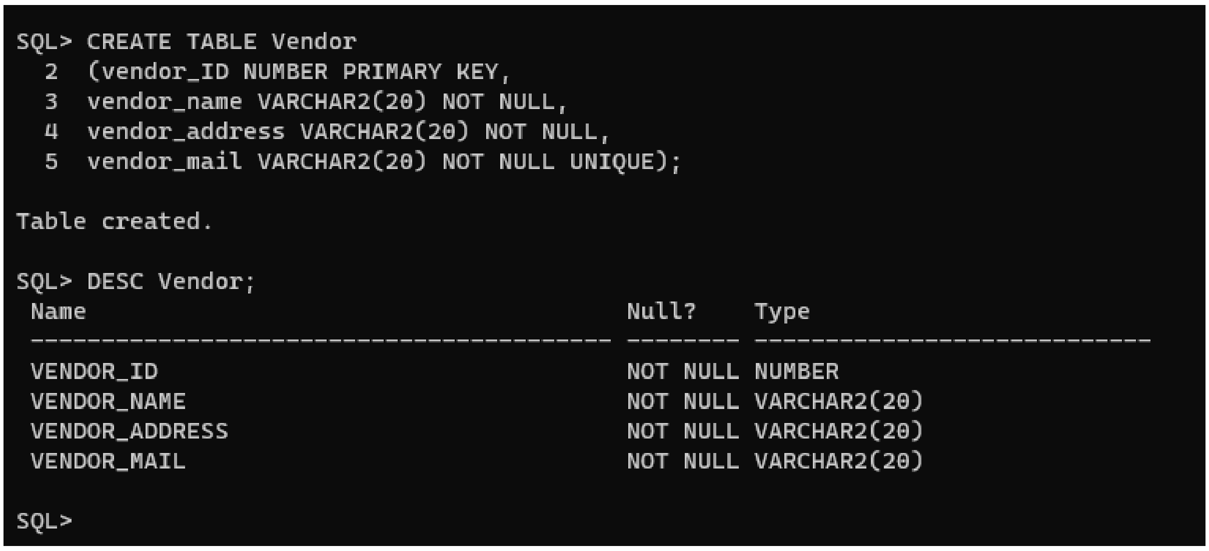


```
SQL> CREATE TABLE "ORDER"
  2  (ord_ID NUMBER PRIMARY KEY,
  3  ord_date DATE NOT NULL);

Table created.

SQL> DESC "ORDER";
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORD_ID                                    NOT NULL NUMBER
 ORD_DATE                                  NOT NULL DATE

SQL>
```

Figure 11: Creating and describing the Order table

Next, the Order table was created as seen in the picture here. The table name was written in **double quotation marks** because ‘**Order’** is a **reserved keyword in SQL**. Also, the **ord_date** attribute was specified as **NOT NULL**, since an **order must have a date** on which it was placed.

Creating the Vendor table



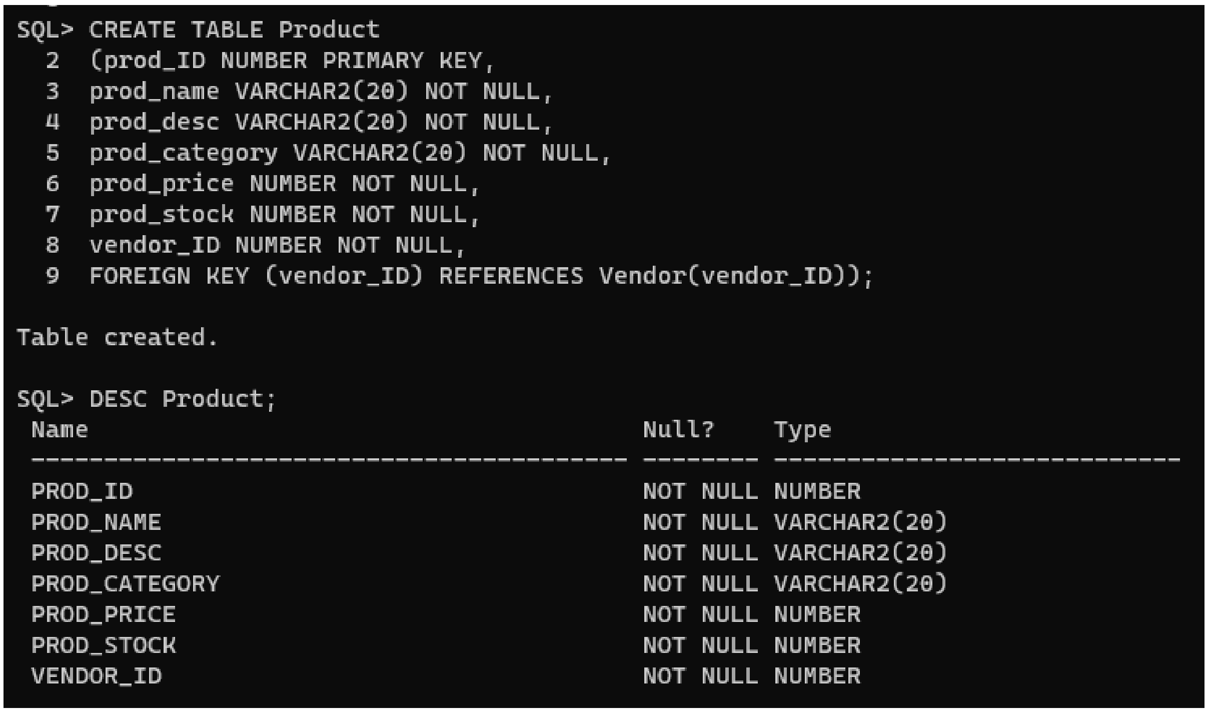
```
SQL> CREATE TABLE Vendor
  2  (vendor_ID NUMBER PRIMARY KEY,
  3  vendor_name VARCHAR2(20) NOT NULL,
  4  vendor_address VARCHAR2(20) NOT NULL,
  5  vendor_mail VARCHAR2(20) NOT NULL UNIQUE);

Table created.

SQL> DESC Vendor;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 VENDOR_ID                                 NOT NULL NUMBER
 VENDOR_NAME                               NOT NULL VARCHAR2(20)
 VENDOR_ADDRESS                            NOT NULL VARCHAR2(20)
 VENDOR_MAIL                               NOT NULL VARCHAR2(20)

SQL>
```

Figure 12: Creating and describing the Vendor table

The Vendor table was created as illustrated above, also with **NOT NULL specified for all values**, with an additional **UNIQUE** constraint for the **vendor_mail** attribute.

This is because **none of those attributes can be empty**, and an **email address** is **unique** to everyone, including vendors.

Creating the Product table

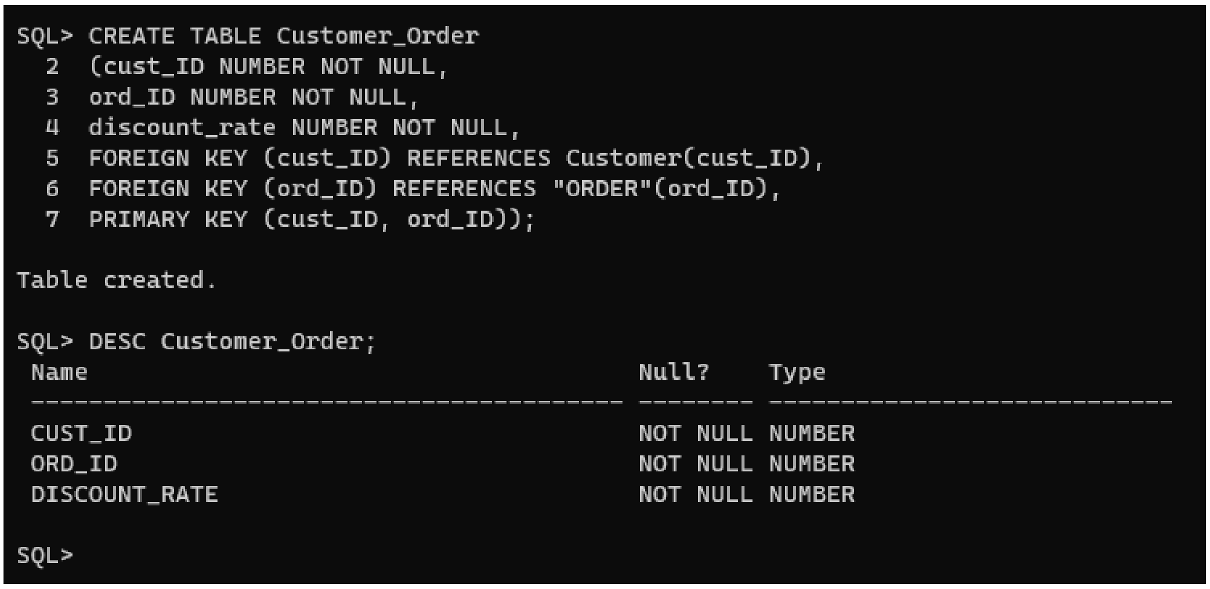


```
SQL> CREATE TABLE Product
  2  (prod_ID NUMBER PRIMARY KEY,
  3  prod_name VARCHAR2(20) NOT NULL,
  4  prod_desc VARCHAR2(20) NOT NULL,
  5  prod_category VARCHAR2(20) NOT NULL,
  6  prod_price NUMBER NOT NULL,
  7  prod_stock NUMBER NOT NULL,
  8  vendor_ID NUMBER NOT NULL,
  9  FOREIGN KEY (vendor_ID) REFERENCES Vendor(vendor_ID));

Table created.

SQL> DESC Product;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PROD_ID                                   NOT NULL NUMBER
 PROD_NAME                                 NOT NULL VARCHAR2(20)
 PROD_DESC                                 NOT NULL VARCHAR2(20)
 PROD_CATEGORY                             NOT NULL VARCHAR2(20)
 PROD_PRICE                                NOT NULL NUMBER
 PROD_STOCK                                NOT NULL NUMBER
 VENDOR_ID                                 NOT NULL NUMBER
```

Figure 13: Creating and describing the Product table

The Product table was then created as seen here, with **vendor_ID** as the **foreign key**. The **NOT NULL** constraint was used **for all attributes** here too for the same reasons as above.

Creating the Customer_Order table

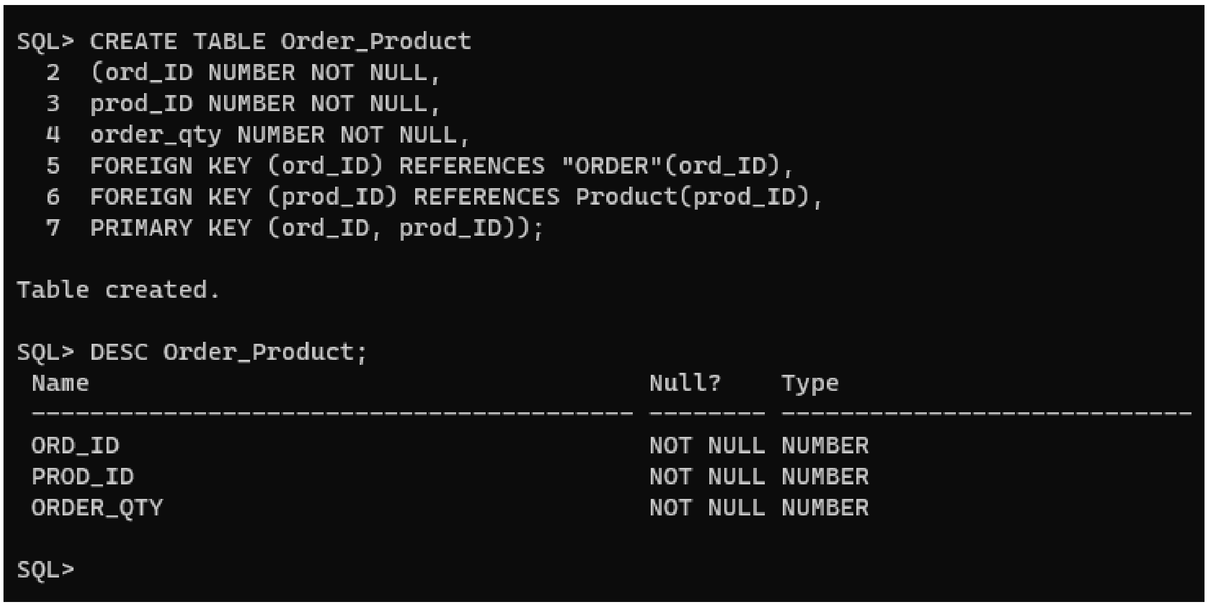


```
SQL> CREATE TABLE Customer_Order
  2  (cust_ID NUMBER NOT NULL,
  3  ord_ID NUMBER NOT NULL,
  4  discount_rate NUMBER NOT NULL,
  5  FOREIGN KEY (cust_ID) REFERENCES Customer(cust_ID),
  6  FOREIGN KEY (ord_ID) REFERENCES "ORDER"(ord_ID),
  7  PRIMARY KEY (cust_ID, ord_ID));

Table created.

SQL> DESC Customer_Order;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                   NOT NULL NUMBER
 ORD_ID                                    NOT NULL NUMBER
 DISCOUNT_RATE                             NOT NULL NUMBER

SQL>
```

Figure 14: Creating and describing the Customer_Order table

Next, the Customer_Order table was created with **customer_ID and ord_ID** as the **foreign keys**. Both attributes function as a **composite primary key** to determine the **discount_rate**. **Every attribute** was specified as **NOT NULL** as well.

Creating the Order_Product table

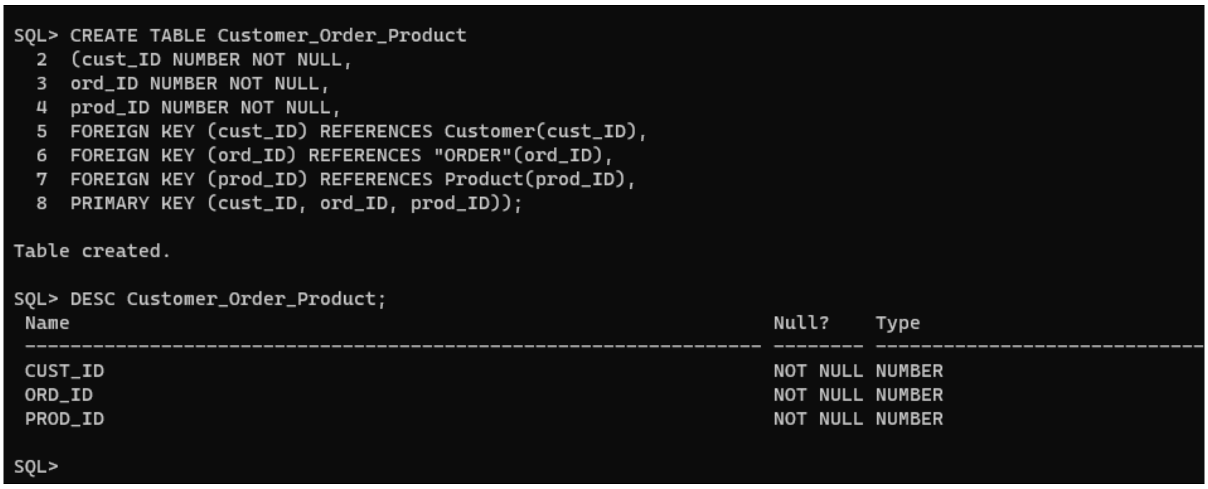


```
SQL> CREATE TABLE Order_Product
     (ord_ID NUMBER NOT NULL,
     prod_ID NUMBER NOT NULL,
     order_qty NUMBER NOT NULL,
     FOREIGN KEY (ord_ID) REFERENCES "ORDER"(ord_ID),
     FOREIGN KEY (prod_ID) REFERENCES Product(prod_ID),
     PRIMARY KEY (ord_ID, prod_ID));

Table created.

SQL> DESC Order_Product;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORD_ID                                    NOT NULL NUMBER
 PROD_ID                                   NOT NULL NUMBER
 ORDER_QTY                                 NOT NULL NUMBER

SQL>
```

Figure 15: Creating and describing the Order_Product table

The Order_Product table was then created, depicted in the picture above. The **ord_ID and prod_ID** attributes were specified as the **foreign keys**, which acted together as a **composite primary key** to find the value of **order_qty**. All attributes were specified as **NOT NULL**.

Creating the Customer_Order_Product table

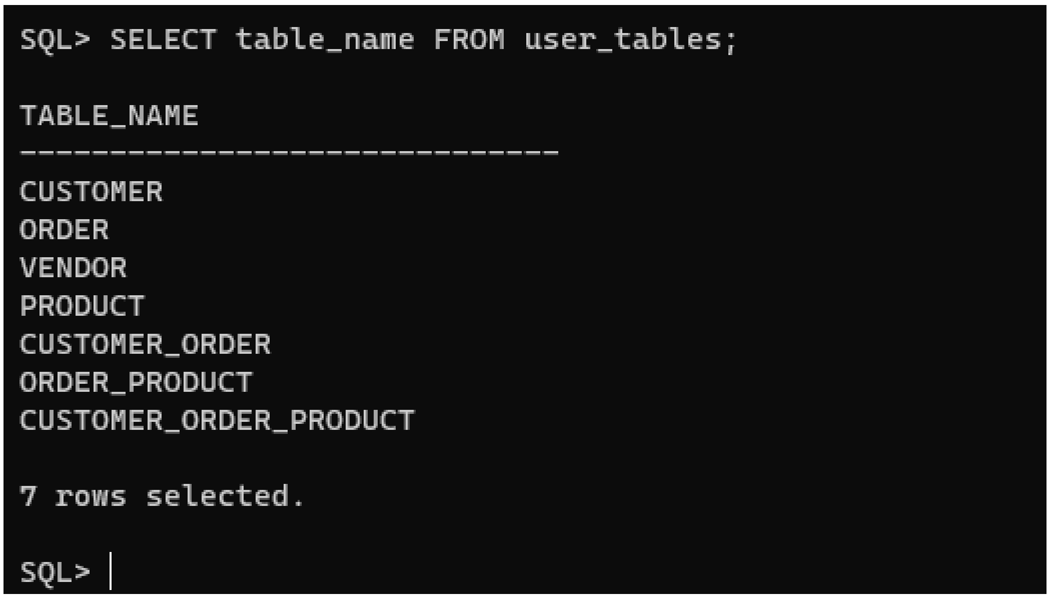


```
SQL> CREATE TABLE Customer_Order_Product
  (cust_ID NUMBER NOT NULL,
  ord_ID NUMBER NOT NULL,
  prod_ID NUMBER NOT NULL,
  FOREIGN KEY (cust_ID) REFERENCES Customer(cust_ID),
  FOREIGN KEY (ord_ID) REFERENCES "ORDER"(ord_ID),
  FOREIGN KEY (prod_ID) REFERENCES Product(prod_ID),
  PRIMARY KEY (cust_ID, ord_ID, prod_ID));

Table created.

SQL> DESC Customer_Order_Product;
 Name                                                              Null?    Type
 ----------------------------------------------------------------- -------- ----------------------------
 CUST_ID                                                           NOT NULL NUMBER
 ORD_ID                                                            NOT NULL NUMBER
 PROD_ID                                                           NOT NULL NUMBER

SQL>
```

Figure 16: Creating and describing the Customer_Order_Product table

Finally, the Customer_Order_Product table was created, with the **foreign keys cust_ID, ord_ID, and prod_ID** acting together as a **composite primary key**, with **no other attributes**. **All of them** were specified as **NOT NULL**.

Displaying all the created tables

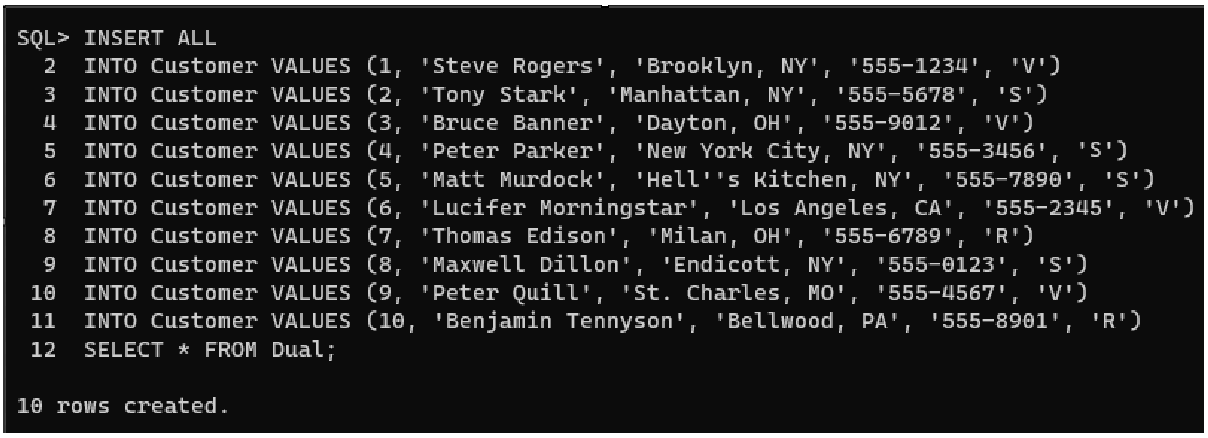


```
SQL> SELECT table_name FROM user_tables;

TABLE_NAME
------------------------------
CUSTOMER
ORDER
VENDOR
PRODUCT
CUSTOMER_ORDER
ORDER_PRODUCT
CUSTOMER_ORDER_PRODUCT

7 rows selected.

SQL>
```

Figure 17: Listing all tables created in Gadget_Emporium

5.3. Inserting Data into the Tables

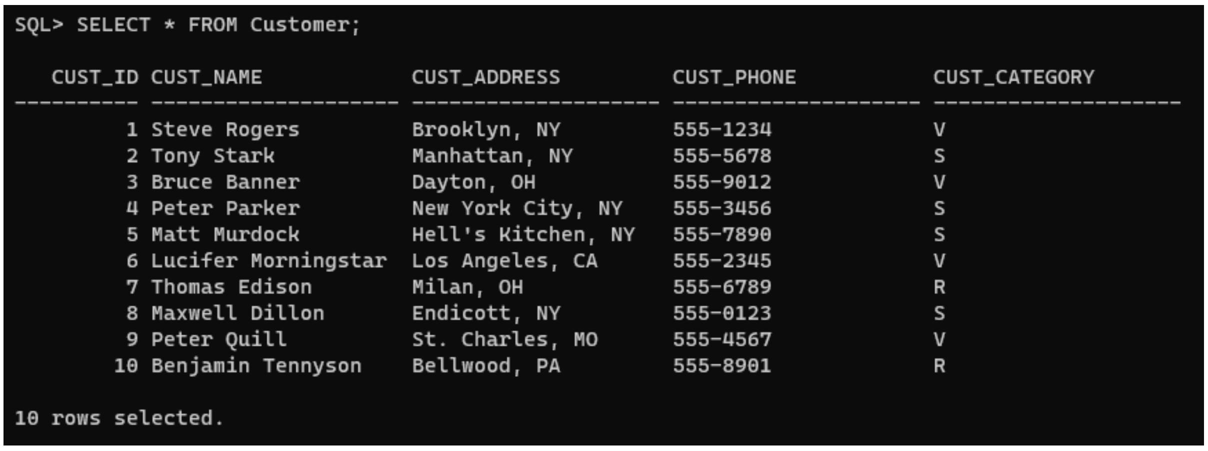
Inserting Data into the Customer table



```
SQL> INSERT ALL
INTO Customer VALUES (1, 'Steve Rogers', 'Brooklyn, NY', '555-1234', 'V')
INTO Customer VALUES (2, 'Tony Stark', 'Manhattan, NY', '555-5678', 'S')
INTO Customer VALUES (3, 'Bruce Banner', 'Dayton, OH', '555-9012', 'V')
INTO Customer VALUES (4, 'Peter Parker', 'New York City, NY', '555-3456', 'S')
INTO Customer VALUES (5, 'Matt Murdock', 'Hell''s Kitchen, NY', '555-7890', 'S')
INTO Customer VALUES (6, 'Lucifer Morningstar', 'Los Angeles, CA', '555-2345', 'V')
INTO Customer VALUES (7, 'Thomas Edison', 'Milan, OH', '555-6789', 'R')
INTO Customer VALUES (8, 'Maxwell Dillon', 'Endicott, NY', '555-0123', 'S')
INTO Customer VALUES (9, 'Peter Quill', 'St. Charles, MO', '555-4567', 'V')
INTO Customer VALUES (10, 'Benjamin Tennyson', 'Bellwood, PA', '555-8901', 'R')
SELECT * FROM Dual;

10 rows created.
```

Figure 18: Inserting values into the Customer table

In the picture above, it is seen that **10 rows of data** were inserted into the Customer table.

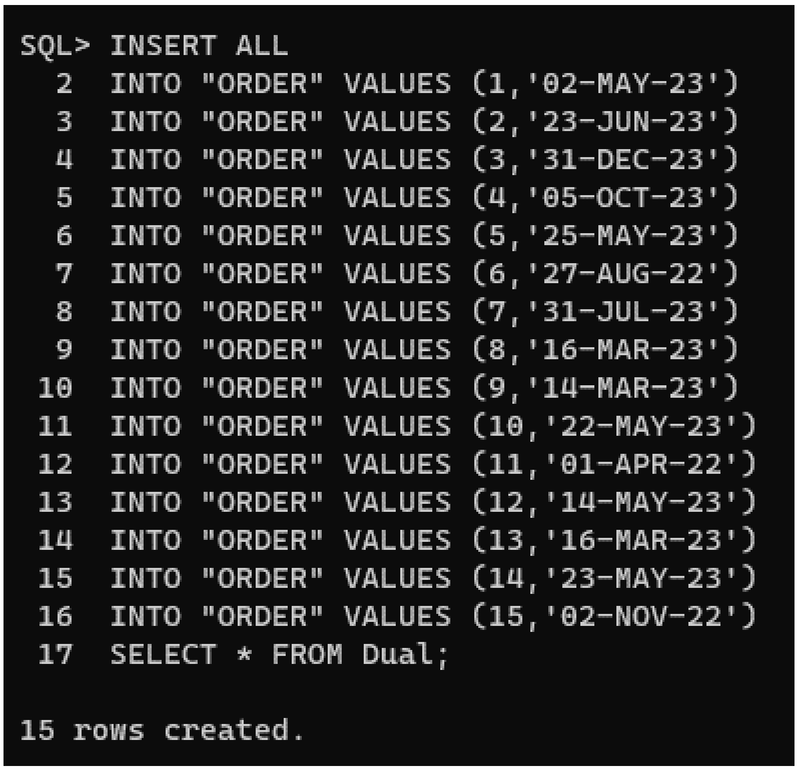


```
SQL> SELECT * FROM Customer;

   CUST_ID CUST_NAME            CUST_ADDRESS         CUST_PHONE           CUST_CATEGORY
---------- -------------------- -------------------- -------------------- --------------------
         1 Steve Rogers         Brooklyn, NY         555-1234             V
         2 Tony Stark           Manhattan, NY        555-5678             S
         3 Bruce Banner         Dayton, OH           555-9012             V
         4 Peter Parker         New York City, NY    555-3456             S
         5 Matt Murdock         Hell's Kitchen, NY   555-7890             S
         6 Lucifer Morningstar  Los Angeles, CA      555-2345             V
         7 Thomas Edison        Milan, OH            555-6789             R
         8 Maxwell Dillon       Endicott, NY         555-0123             S
         9 Peter Quill          St. Charles, MO      555-4567             V
        10 Benjamin Tennyson    Bellwood, PA         555-8901             R

10 rows selected.
```

Figure 19: Displaying the values inserted into the Customer table

The values were then displayed for verifying the insertion.

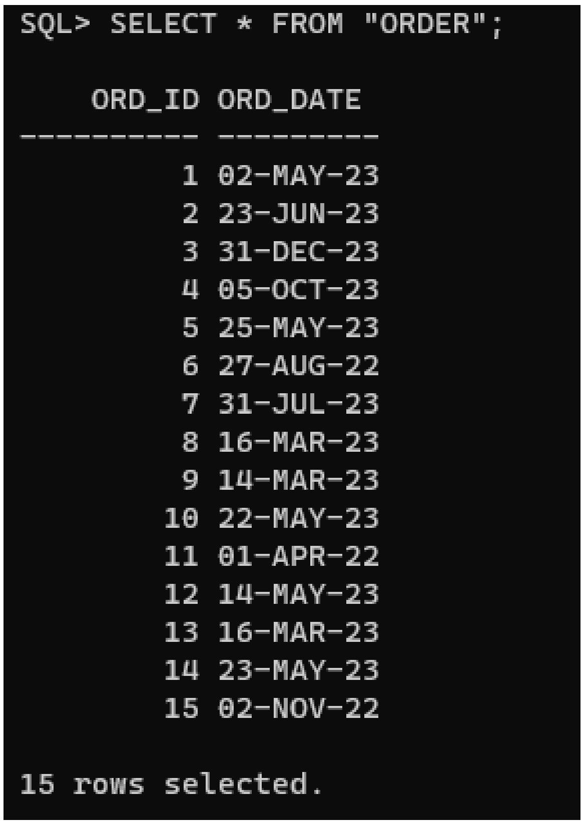
Inserting Data into the Order table



```
SQL> INSERT ALL
     INTO "ORDER" VALUES (1,'02-MAY-23')
     INTO "ORDER" VALUES (2,'23-JUN-23')
     INTO "ORDER" VALUES (3,'31-DEC-23')
     INTO "ORDER" VALUES (4,'05-OCT-23')
     INTO "ORDER" VALUES (5,'25-MAY-23')
     INTO "ORDER" VALUES (6,'27-AUG-22')
     INTO "ORDER" VALUES (7,'31-JUL-23')
     INTO "ORDER" VALUES (8,'16-MAR-23')
     INTO "ORDER" VALUES (9,'14-MAR-23')
     INTO "ORDER" VALUES (10,'22-MAY-23')
     INTO "ORDER" VALUES (11,'01-APR-22')
     INTO "ORDER" VALUES (12,'14-MAY-23')
     INTO "ORDER" VALUES (13,'16-MAR-23')
     INTO "ORDER" VALUES (14,'23-MAY-23')
     INTO "ORDER" VALUES (15,'02-NOV-22')
     SELECT * FROM Dual;

15 rows created.
```

Figure 20: Inserting values into the Order table

The picture above shows the insertion of **15 rows of data** into the Order table.

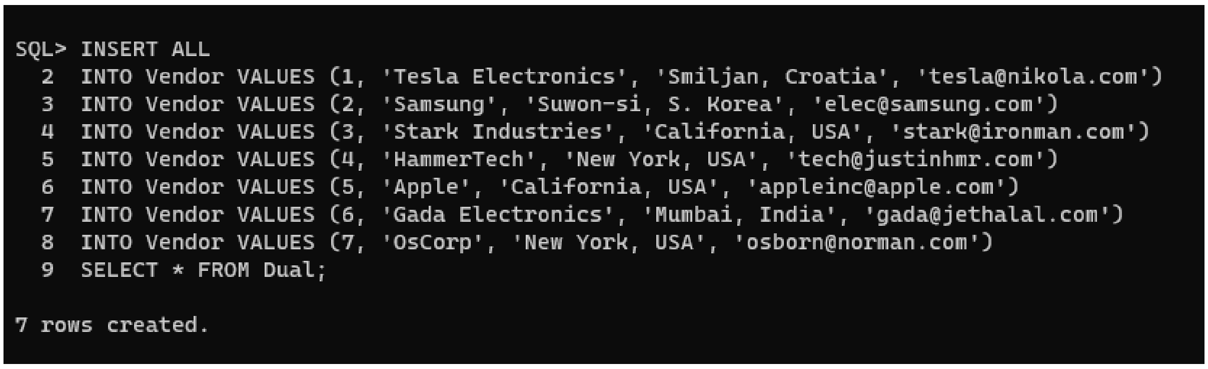


```
SQL> SELECT * FROM "ORDER";

    ORD_ID ORD_DATE
---------- ---------
         1 02-MAY-23
         2 23-JUN-23
         3 31-DEC-23
         4 05-OCT-23
         5 25-MAY-23
         6 27-AUG-22
         7 31-JUL-23
         8 16-MAR-23
         9 14-MAR-23
        10 22-MAY-23
        11 01-APR-22
        12 14-MAY-23
        13 16-MAR-23
        14 23-MAY-23
        15 02-NOV-22

15 rows selected.
```

Figure 21: Displaying the values inserted into the Order table

The inserted values were then verified by displaying all of them.

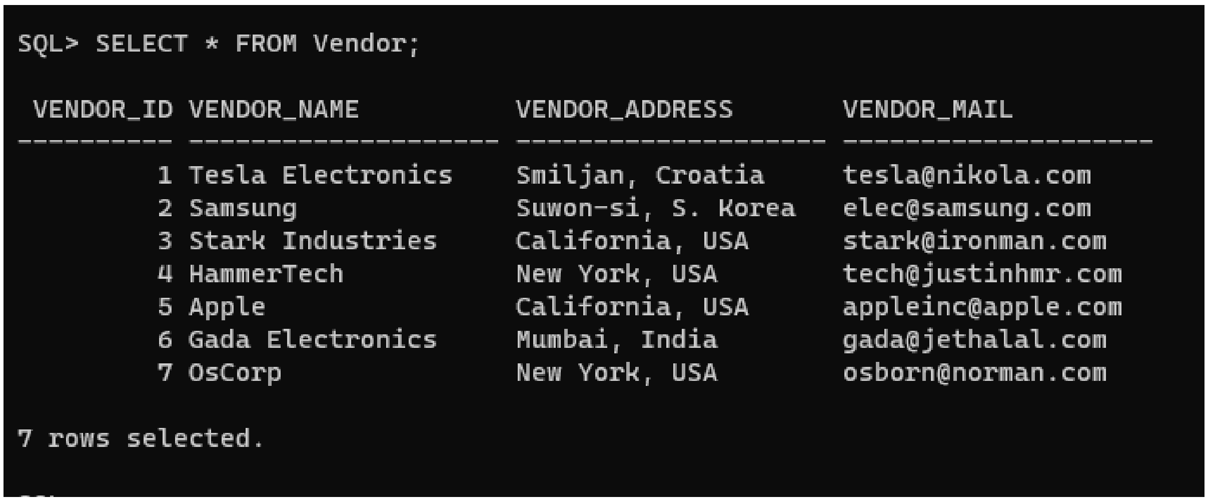
Inserting Data into the Vendor table



```
SQL> INSERT ALL
  2  INTO Vendor VALUES (1, 'Tesla Electronics', 'Smiljan, Croatia', 'tesla@nikola.com')
  3  INTO Vendor VALUES (2, 'Samsung', 'Suwon-si, S. Korea', 'elec@samsung.com')
  4  INTO Vendor VALUES (3, 'Stark Industries', 'California, USA', 'stark@ironman.com')
  5  INTO Vendor VALUES (4, 'HammerTech', 'New York, USA', 'tech@justinhmr.com')
  6  INTO Vendor VALUES (5, 'Apple', 'California, USA', 'appleinc@apple.com')
  7  INTO Vendor VALUES (6, 'Gada Electronics', 'Mumbai, India', 'gada@jethalal.com')
  8  INTO Vendor VALUES (7, 'OsCorp', 'New York, USA', 'osborn@norman.com')
  9  SELECT * FROM Dual;

7 rows created.
```

Figure 22: Inserting values into the Vendor table

Next, **7 rows of data** were inserted into the Vendor table as seen here.

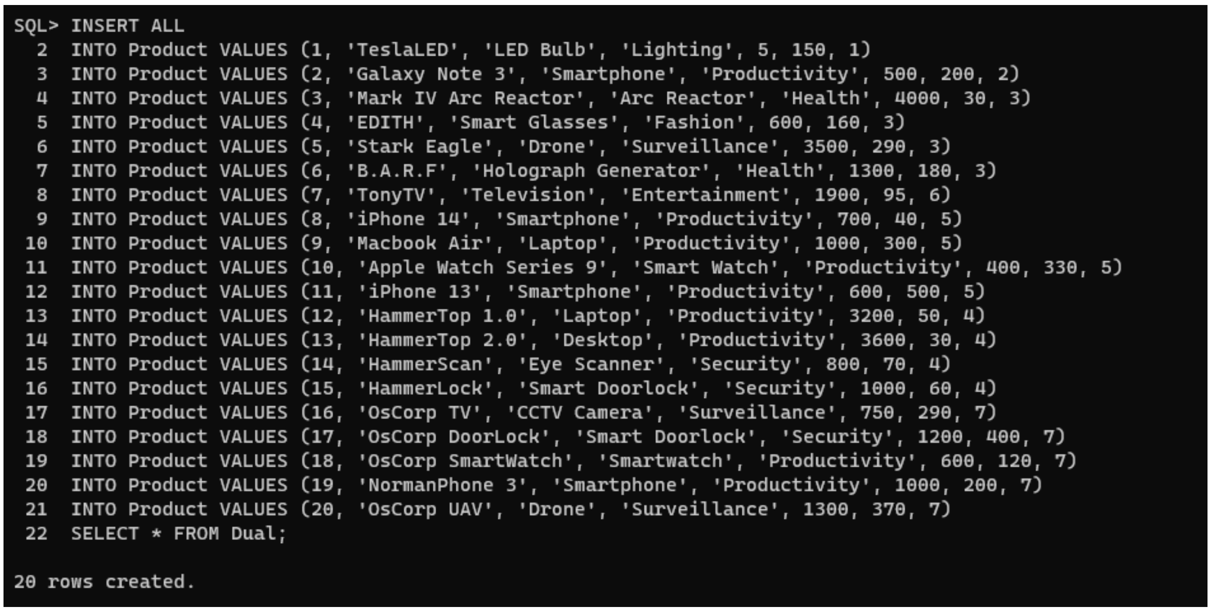


```
SQL> SELECT * FROM Vendor;

 VENDOR_ID VENDOR_NAME          VENDOR_ADDRESS       VENDOR_MAIL
---------- -------------------- -------------------- --------------------
         1 Tesla Electronics    Smiljan, Croatia     tesla@nikola.com
         2 Samsung              Suwon-si, S. Korea   elec@samsung.com
         3 Stark Industries     California, USA      stark@ironman.com
         4 HammerTech           New York, USA        tech@justinhmr.com
         5 Apple                California, USA      appleinc@apple.com
         6 Gada Electronics     Mumbai, India        gada@jethalal.com
         7 OsCorp               New York, USA        osborn@norman.com

7 rows selected.
```

Figure 23: Displaying the values inserted into the Vendor table

The values were displayed for verification as well.

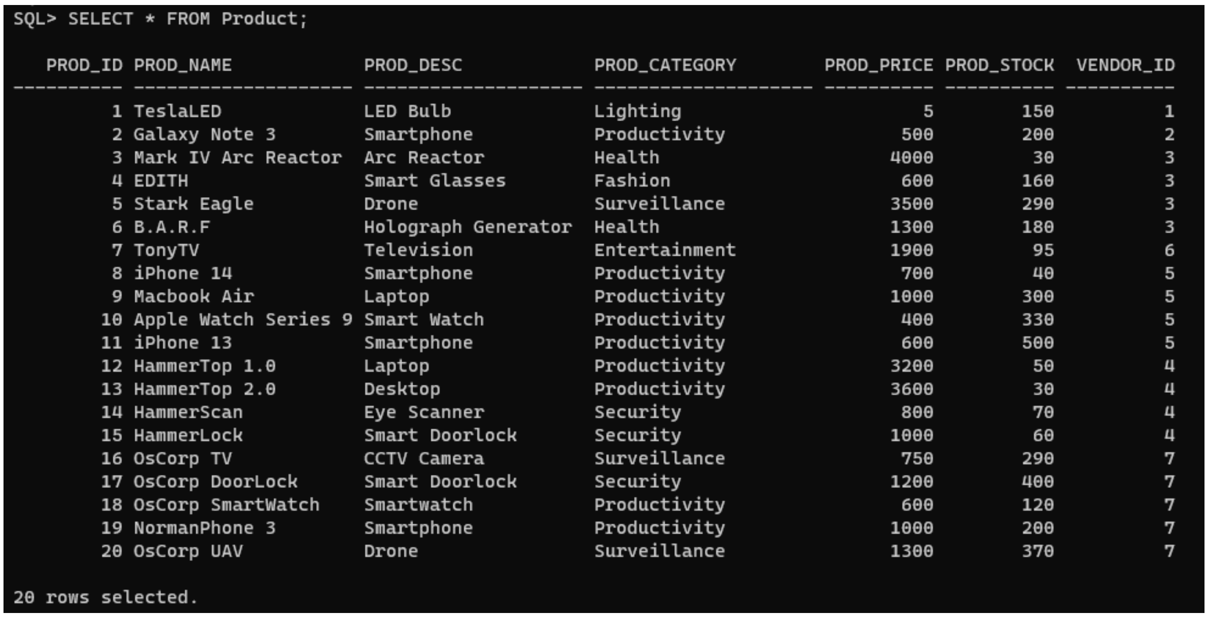
Inserting Data into the Product table



```
SQL> INSERT ALL
  INTO Product VALUES (1, 'TeslaLED', 'LED Bulb', 'Lighting', 5, 150, 1)
  INTO Product VALUES (2, 'Galaxy Note 3', 'Smartphone', 'Productivity', 500, 200, 2)
  INTO Product VALUES (3, 'Mark IV Arc Reactor', 'Arc Reactor', 'Health', 4000, 30, 3)
  INTO Product VALUES (4, 'EDITH', 'Smart Glasses', 'Fashion', 600, 160, 3)
  INTO Product VALUES (5, 'Stark Eagle', 'Drone', 'Surveillance', 3500, 290, 3)
  INTO Product VALUES (6, 'B.A.R.F', 'Holograph Generator', 'Health', 1300, 180, 3)
  INTO Product VALUES (7, 'TonyTV', 'Television', 'Entertainment', 1900, 95, 6)
  INTO Product VALUES (8, 'iPhone 14', 'Smartphone', 'Productivity', 700, 40, 5)
  INTO Product VALUES (9, 'Macbook Air', 'Laptop', 'Productivity', 1000, 300, 5)
  INTO Product VALUES (10, 'Apple Watch Series 9', 'Smart Watch', 'Productivity', 400, 330, 5)
  INTO Product VALUES (11, 'iPhone 13', 'Smartphone', 'Productivity', 600, 500, 5)
  INTO Product VALUES (12, 'HammerTop 1.0', 'Laptop', 'Productivity', 3200, 50, 4)
  INTO Product VALUES (13, 'HammerTop 2.0', 'Desktop', 'Productivity', 3600, 30, 4)
  INTO Product VALUES (14, 'HammerScan', 'Eye Scanner', 'Security', 800, 70, 4)
  INTO Product VALUES (15, 'HammerLock', 'Smart Doorlock', 'Security', 1000, 60, 4)
  INTO Product VALUES (16, 'OsCorp TV', 'CCTV Camera', 'Surveillance', 750, 290, 7)
  INTO Product VALUES (17, 'OsCorp DoorLock', 'Smart Doorlock', 'Security', 1200, 400, 7)
  INTO Product VALUES (18, 'OsCorp SmartWatch', 'Smartwatch', 'Productivity', 600, 120, 7)
  INTO Product VALUES (19, 'NormanPhone 3', 'Smartphone', 'Productivity', 1000, 200, 7)
  INTO Product VALUES (20, 'OsCorp UAV', 'Drone', 'Surveillance', 1300, 370, 7)
  SELECT * FROM Dual;

20 rows created.
```

Figure 24: Inserting values into the Product table

The insertion of **20 rows of data** in the Product table followed, as seen in the picture above.

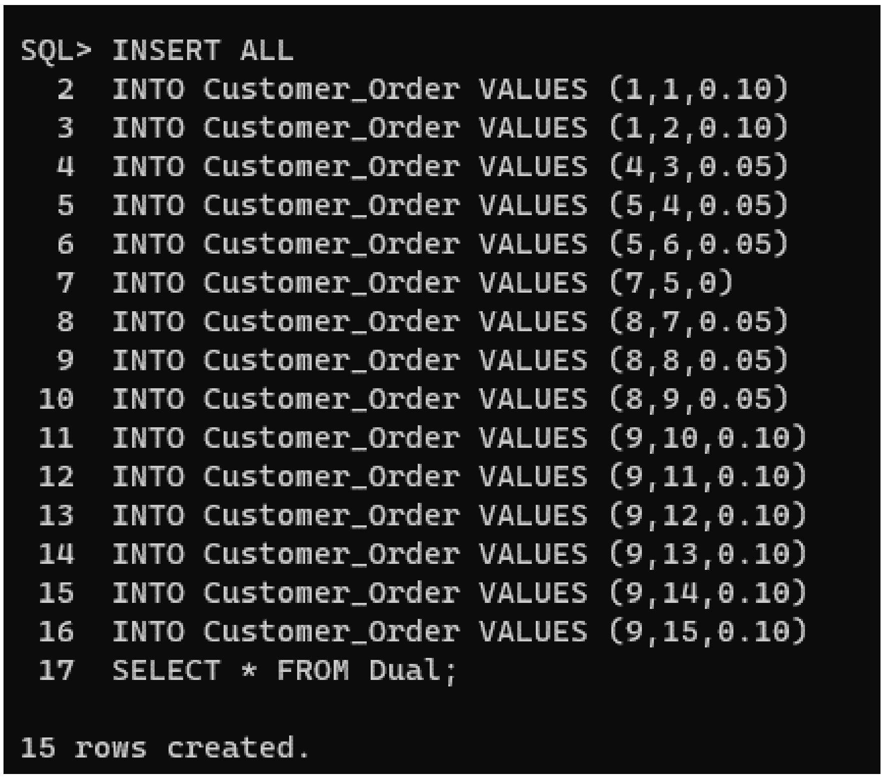


```
SQL> SELECT * FROM Product;

   PROD_ID PROD_NAME            PROD_DESC            PROD_CATEGORY        PROD_PRICE PROD_STOCK  VENDOR_ID
---------- -------------------- -------------------- -------------------- ---------- ---------- ----------
         1 TeslaLED             LED Bulb             Lighting                      5        150          1
         2 Galaxy Note 3        Smartphone           Productivity                500        200          2
         3 Mark IV Arc Reactor  Arc Reactor          Health                     4000         30          3
         4 EDITH                Smart Glasses        Fashion                     600        160          3
         5 Stark Eagle          Drone                Surveillance               3500        290          3
         6 B.A.R.F              Holograph Generator  Health                     1300        180          3
         7 TonyTV               Television           Entertainment              1900         95          6
         8 iPhone 14            Smartphone           Productivity                700         40          5
         9 Macbook Air          Laptop               Productivity               1000        300          5
        10 Apple Watch Series 9 Smart Watch          Productivity                400        330          5
        11 iPhone 13            Smartphone           Productivity                600        500          5
        12 HammerTop 1.0        Laptop               Productivity               3200         50          4
        13 HammerTop 2.0        Desktop              Productivity               3600         30          4
        14 HammerScan           Eye Scanner          Security                    800         70          4
        15 HammerLock           Smart Doorlock       Security                   1000         60          4
        16 OsCorp TV            CCTV Camera          Surveillance                750        290          7
        17 OsCorp DoorLock      Smart Doorlock       Security                   1200        400          7
        18 OsCorp SmartWatch    Smartwatch           Productivity                600        120          7
        19 NormanPhone 3        Smartphone           Productivity               1000        200          7
        20 OsCorp UAV           Drone                Surveillance               1300        370          7

20 rows selected.
```

Figure 25: Displaying the values inserted into the Product table

The inserted values were displayed for verification.

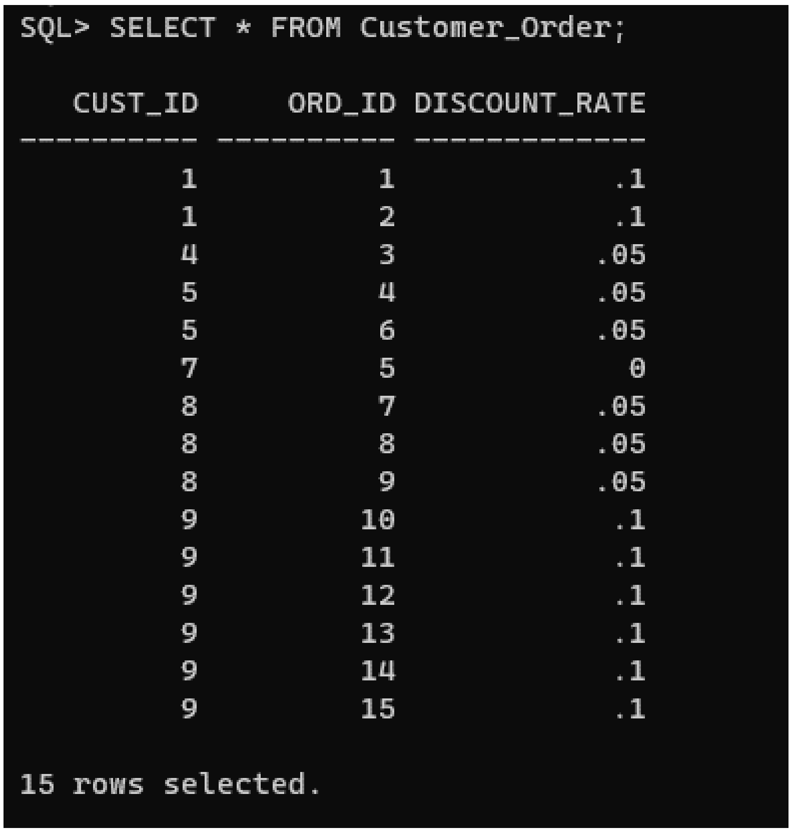
Inserting Data into the Customer_Order table



```
SQL> INSERT ALL
  2  INTO Customer_Order VALUES (1,1,0.10)
  3  INTO Customer_Order VALUES (1,2,0.10)
  4  INTO Customer_Order VALUES (4,3,0.05)
  5  INTO Customer_Order VALUES (5,4,0.05)
  6  INTO Customer_Order VALUES (5,6,0.05)
  7  INTO Customer_Order VALUES (7,5,0)
  8  INTO Customer_Order VALUES (8,7,0.05)
  9  INTO Customer_Order VALUES (8,8,0.05)
 10  INTO Customer_Order VALUES (8,9,0.05)
 11  INTO Customer_Order VALUES (9,10,0.10)
 12  INTO Customer_Order VALUES (9,11,0.10)
 13  INTO Customer_Order VALUES (9,12,0.10)
 14  INTO Customer_Order VALUES (9,13,0.10)
 15  INTO Customer_Order VALUES (9,14,0.10)
 16  INTO Customer_Order VALUES (9,15,0.10)
 17  SELECT * FROM Dual;

15 rows created.
```

Figure 26: Inserting values into the Customer_Order table

As depicted in the picture above, **15 rows of data** were inserted into the Customer_Order table.

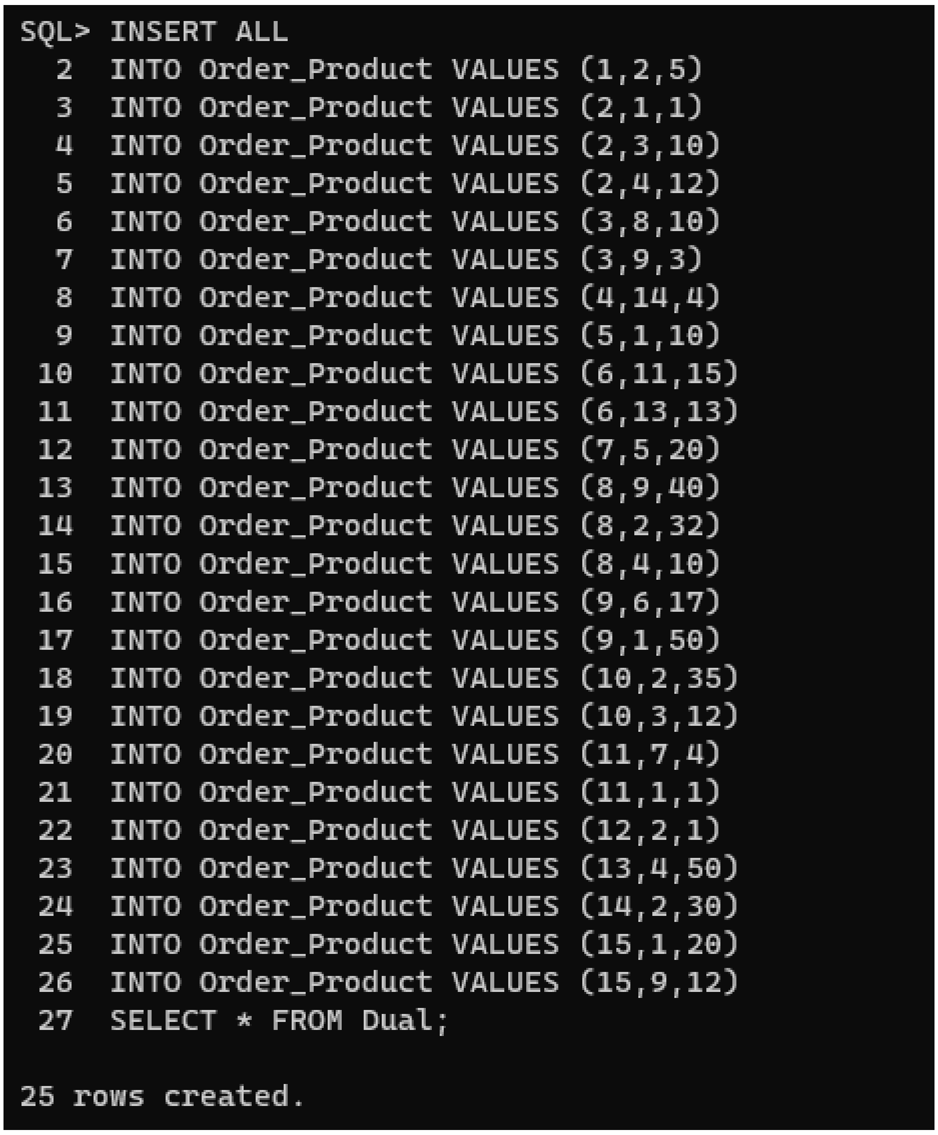


```
SQL> SELECT * FROM Customer_Order;

   CUST_ID     ORD_ID DISCOUNT_RATE
---------- ---------- -------------
         1          1            .1
         1          2            .1
         4          3           .05
         5          4           .05
         5          6           .05
         7          5             0
         8          7           .05
         8          8           .05
         8          9           .05
         9         10            .1
         9         11            .1
         9         12            .1
         9         13            .1
         9         14            .1
         9         15            .1

15 rows selected.
```

Figure 27: Displaying the values inserted into the Customer_Order table

The values were then displayed for verifying the insertion.

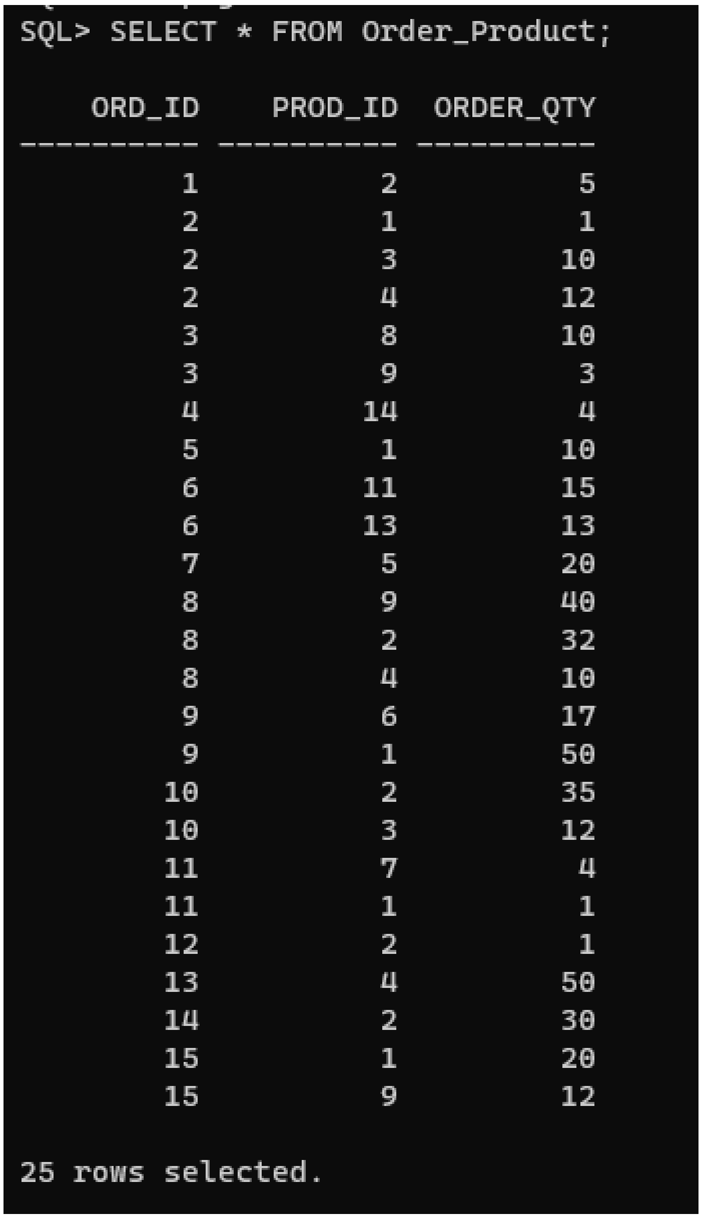
Inserting Data into the Order_Product table



```
SQL> INSERT ALL
     INTO Order_Product VALUES (1,2,5)
     INTO Order_Product VALUES (2,1,1)
     INTO Order_Product VALUES (2,3,10)
     INTO Order_Product VALUES (2,4,12)
     INTO Order_Product VALUES (3,8,10)
     INTO Order_Product VALUES (3,9,3)
     INTO Order_Product VALUES (4,14,4)
     INTO Order_Product VALUES (5,1,10)
     INTO Order_Product VALUES (6,11,15)
     INTO Order_Product VALUES (6,13,13)
     INTO Order_Product VALUES (7,5,20)
     INTO Order_Product VALUES (8,9,40)
     INTO Order_Product VALUES (8,2,32)
     INTO Order_Product VALUES (8,4,10)
     INTO Order_Product VALUES (9,6,17)
     INTO Order_Product VALUES (9,1,50)
     INTO Order_Product VALUES (10,2,35)
     INTO Order_Product VALUES (10,3,12)
     INTO Order_Product VALUES (11,7,4)
     INTO Order_Product VALUES (11,1,1)
     INTO Order_Product VALUES (12,2,1)
     INTO Order_Product VALUES (13,4,50)
     INTO Order_Product VALUES (14,2,30)
     INTO Order_Product VALUES (15,1,20)
     INTO Order_Product VALUES (15,9,12)
     SELECT * FROM Dual;

25 rows created.
```

Figure 28: Inserting values into the Order_Product table

The Order_Product table was next populated with **25 rows of data**, which can be seen in the picture above.

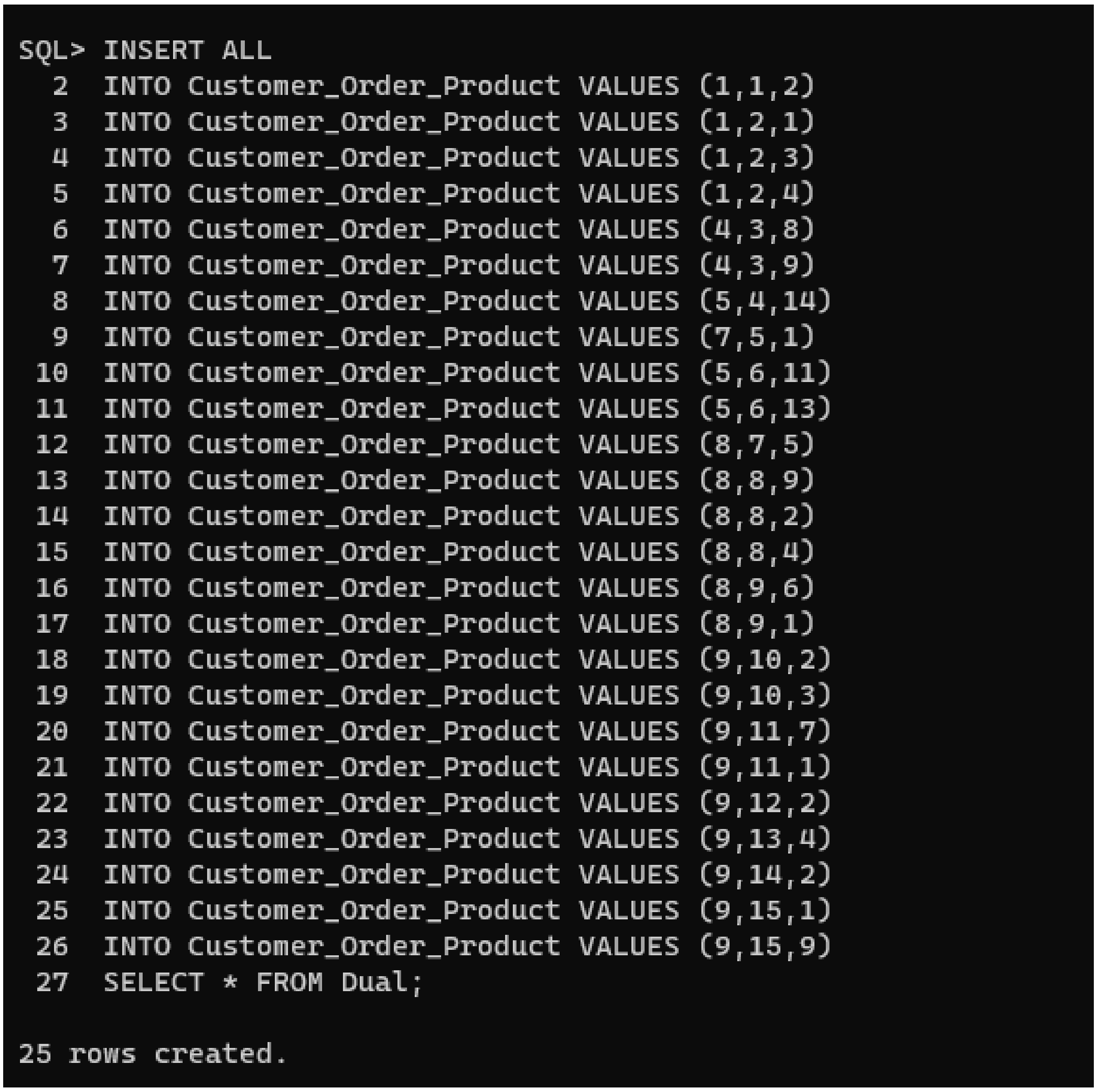


```
SQL> SELECT * FROM Order_Product;

    ORD_ID    PROD_ID  ORDER_QTY
---------- ---------- ----------
         1          2          5
         2          1          1
         2          3         10
         2          4         12
         3          8         10
         3          9          3
         4         14          4
         5          1         10
         6         11         15
         6         13         13
         7          5         20
         8          9         40
         8          2         32
         8          4         10
         9          6         17
         9          1         50
        10          2         35
        10          3         12
        11          7          4
        11          1          1
        12          2          1
        13          4         50
        14          2         30
        15          1         20
        15          9         12

25 rows selected.
```

Figure 29: Displaying the values inserted into the Order_Product table

The values were displayed to ensure that no mistakes were made during the insertion.

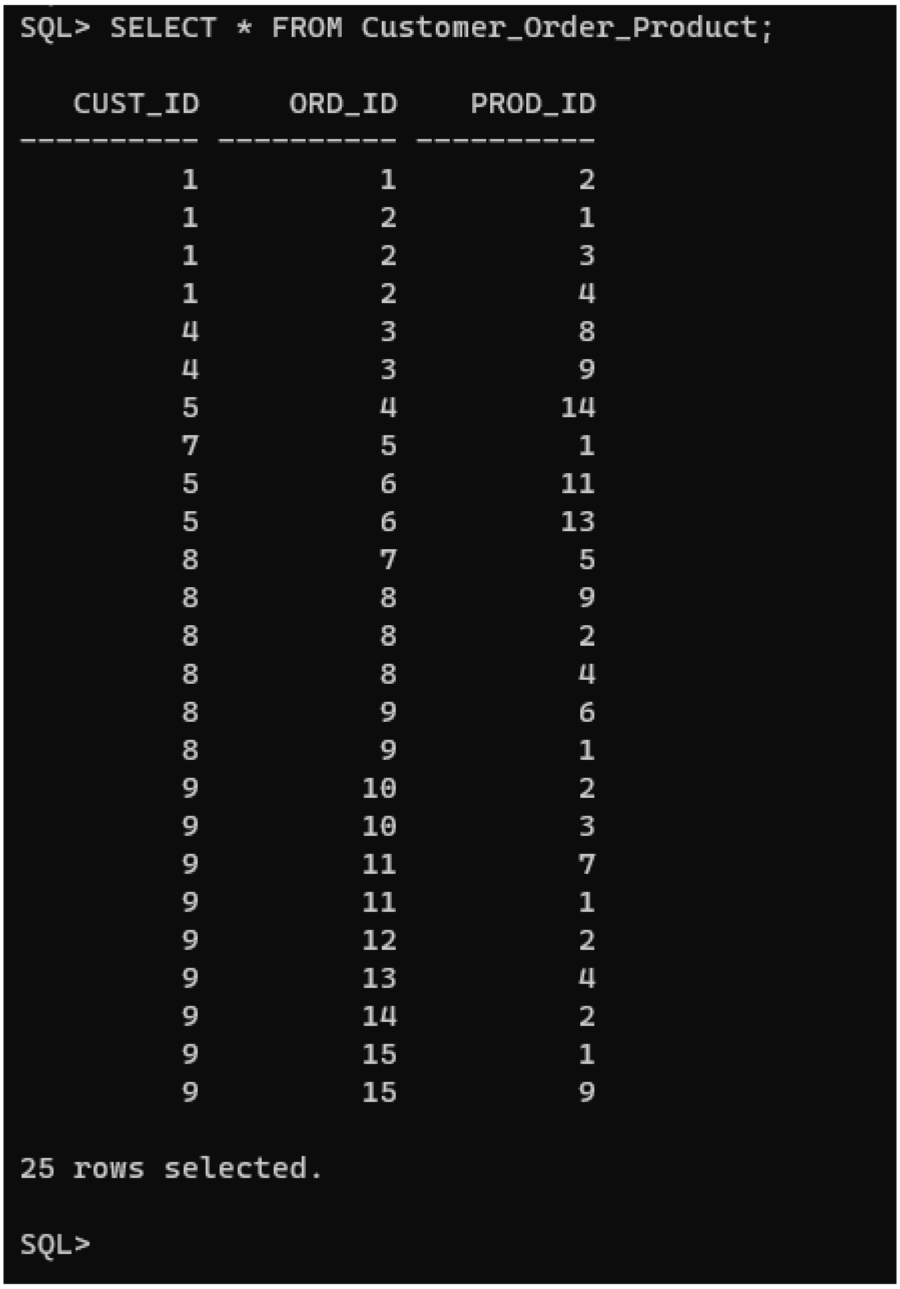
Inserting Data into the Customer_Order_Product table



```
SQL> INSERT ALL
  2  INTO Customer_Order_Product VALUES (1,1,2)
  3  INTO Customer_Order_Product VALUES (1,2,1)
  4  INTO Customer_Order_Product VALUES (1,2,3)
  5  INTO Customer_Order_Product VALUES (1,2,4)
  6  INTO Customer_Order_Product VALUES (4,3,8)
  7  INTO Customer_Order_Product VALUES (4,3,9)
  8  INTO Customer_Order_Product VALUES (5,4,14)
  9  INTO Customer_Order_Product VALUES (7,5,1)
 10  INTO Customer_Order_Product VALUES (5,6,11)
 11  INTO Customer_Order_Product VALUES (5,6,13)
 12  INTO Customer_Order_Product VALUES (8,7,5)
 13  INTO Customer_Order_Product VALUES (8,8,9)
 14  INTO Customer_Order_Product VALUES (8,8,2)
 15  INTO Customer_Order_Product VALUES (8,8,4)
 16  INTO Customer_Order_Product VALUES (8,9,6)
 17  INTO Customer_Order_Product VALUES (8,9,1)
 18  INTO Customer_Order_Product VALUES (9,10,2)
 19  INTO Customer_Order_Product VALUES (9,10,3)
 20  INTO Customer_Order_Product VALUES (9,11,7)
 21  INTO Customer_Order_Product VALUES (9,11,1)
 22  INTO Customer_Order_Product VALUES (9,12,2)
 23  INTO Customer_Order_Product VALUES (9,13,4)
 24  INTO Customer_Order_Product VALUES (9,14,2)
 25  INTO Customer_Order_Product VALUES (9,15,1)
 26  INTO Customer_Order_Product VALUES (9,15,9)
 27  SELECT * FROM Dual;

25 rows created.
```

Figure 30: Inserting values into the Customer_Order_Product table

Finally, the Customer_Order_Product table was filled with **25 rows of data**, as described in the picture above.

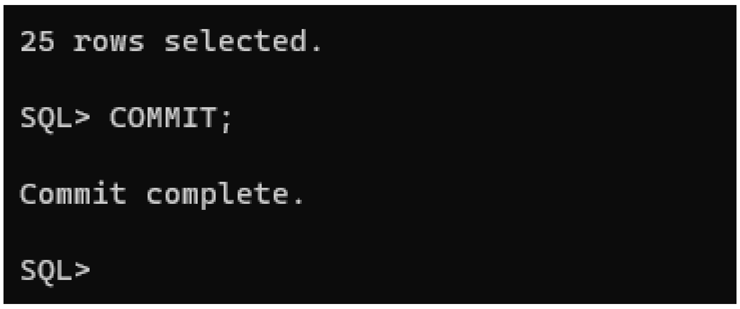


```
SQL> SELECT * FROM Customer_Order_Product;

   CUST_ID     ORD_ID    PROD_ID
---------- ---------- ----------
         1          1          2
         1          2          1
         1          2          3
         1          2          4
         4          3          8
         4          3          9
         5          4         14
         7          5          1
         5          6         11
         5          6         13
         8          7          5
         8          8          9
         8          8          2
         8          8          4
         8          9          6
         8          9          1
         9         10          2
         9         10          3
         9         11          7
         9         11          1
         9         12          2
         9         13          4
         9         14          2
         9         15          1
         9         15          9

25 rows selected.

SQL>
```

Figure 31: Displaying the values inserted into the Customer_Order_Product table

The inserted values were verified by displaying all of them.

Committing the Data Insertions



```
25 rows selected.

SQL> COMMIT;

Commit complete.

SQL>
```

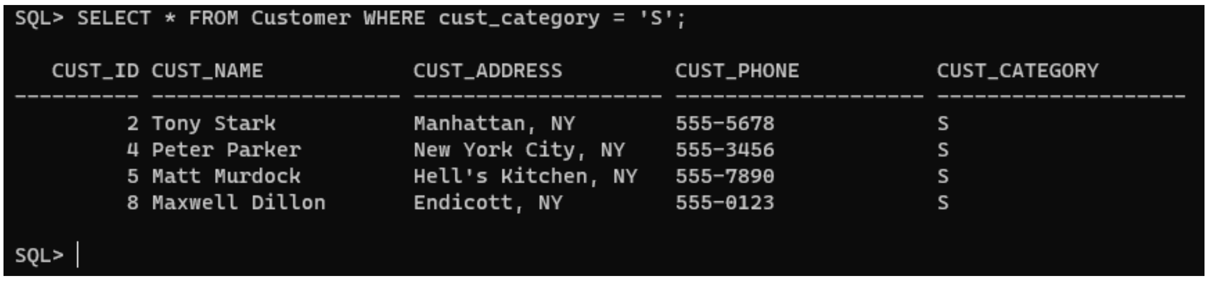
Figure 32: Committing the insertions of values into the tables

All the **data inserted** into the tables above were then **committed to,** to ensure that no records were lost in the next session.

6. Querying the Database

6.1. Information Queries

Listing all the customers that are also the staff of the company

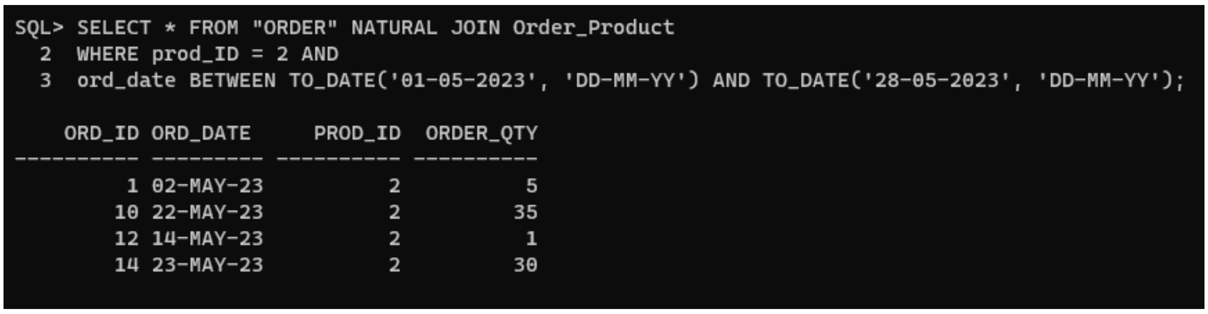


```
SQL> SELECT * FROM Customer WHERE cust_category = 'S';

   CUST_ID CUST_NAME            CUST_ADDRESS         CUST_PHONE           CUST_CATEGORY
---------- -------------------- -------------------- -------------------- --------------------
         2 Tony Stark           Manhattan, NY        555-5678             S
         4 Peter Parker         New York City, NY    555-3456             S
         5 Matt Murdock         Hell's Kitchen, NY   555-7890             S
         8 Maxwell Dillon       Endicott, NY         555-0123             S

SQL> |
```

Figure 33: Listing all customers that are also the staff of the company

It is seen here that **all attributes** from the **Customer** table were displayed where the **cust_category attribute was 'S'**.

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

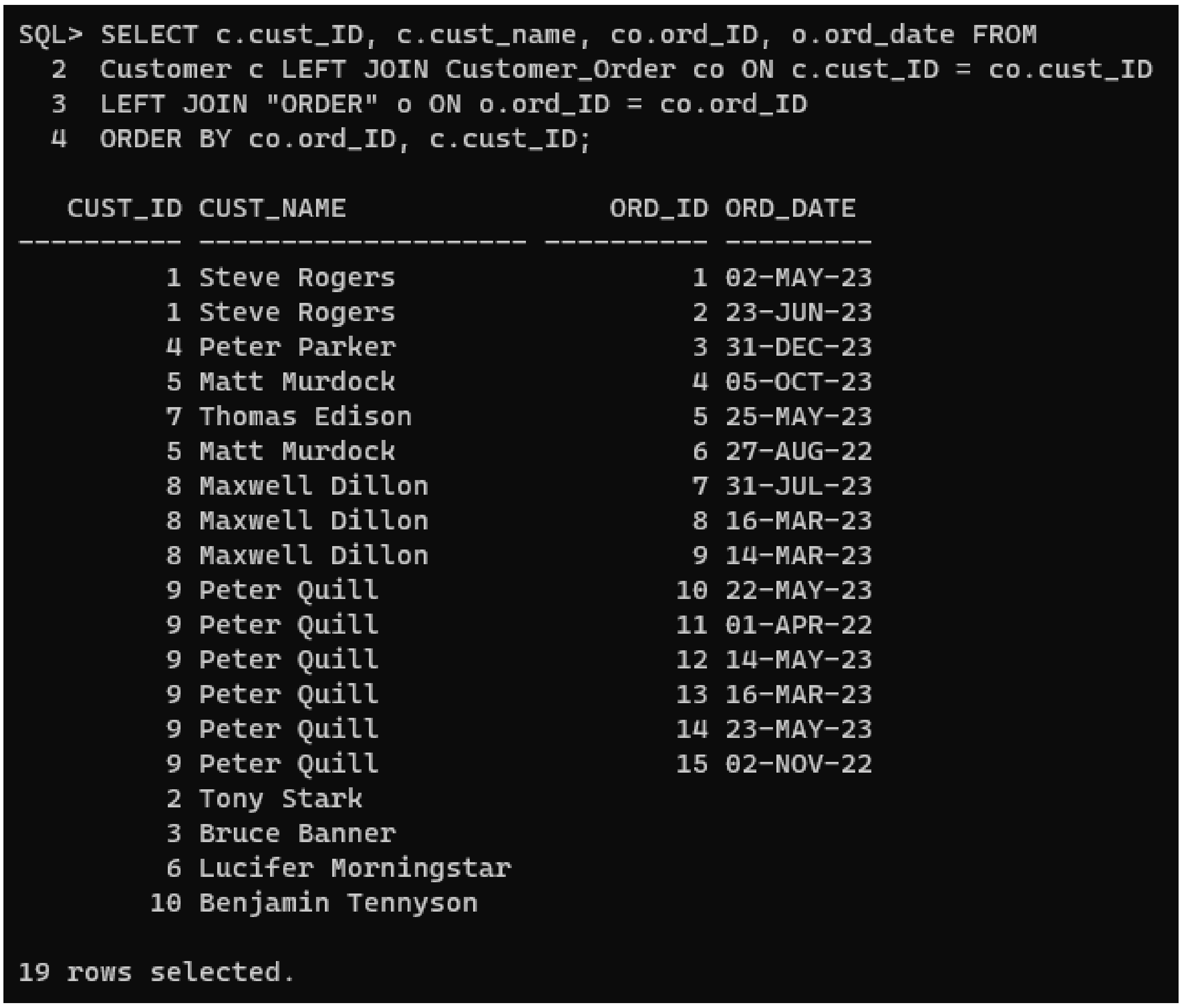


```
SQL> SELECT * FROM "ORDER" NATURAL JOIN Order_Product
  2  WHERE prod_ID = 2 AND
  3  ord_date BETWEEN TO_DATE('01-05-2023', 'DD-MM-YY') AND TO_DATE('28-05-2023', 'DD-MM-YY');

    ORD_ID ORD_DATE     PROD_ID  ORDER_QTY
---------- --------- ---------- ----------
         1 02-MAY-23          2          5
        10 22-MAY-23          2         35
        12 14-MAY-23          2          1
        14 23-MAY-23          2         30
```

Figure 34: Listing all orders made for any particular product between the dates 01-05-2023 and 28-05-2023

The picture above shows that the **Order and Order_Product** tables were **naturally joined**, and the **ord_ID, ord_date, prod_ID, and order_qty** attributes were **displayed** where the **prod_ID was 2** and the **ord_date** fell **between 01 May 2023 and 28 May 2023**.

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

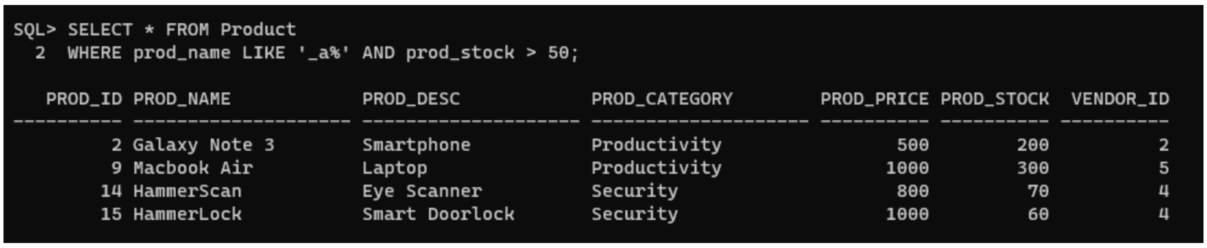


```
SQL> SELECT c.cust_ID, c.cust_name, co.ord_ID, o.ord_date FROM
  2  Customer c LEFT JOIN Customer_Order co ON c.cust_ID = co.cust_ID
  3  LEFT JOIN "ORDER" o ON o.ord_ID = co.ord_ID
  4  ORDER BY co.ord_ID, c.cust_ID;

   CUST_ID CUST_NAME                ORD_ID ORD_DATE
---------- -------------------- ---------- ---------
         1 Steve Rogers                  1 02-MAY-23
         1 Steve Rogers                  2 23-JUN-23
         4 Peter Parker                  3 31-DEC-23
         5 Matt Murdock                  4 05-OCT-23
         7 Thomas Edison                 5 25-MAY-23
         5 Matt Murdock                  6 27-AUG-22
         8 Maxwell Dillon                7 31-JUL-23
         8 Maxwell Dillon                8 16-MAR-23
         8 Maxwell Dillon                9 14-MAR-23
         9 Peter Quill                  10 22-MAY-23
         9 Peter Quill                  11 01-APR-22
         9 Peter Quill                  12 14-MAY-23
         9 Peter Quill                  13 16-MAR-23
         9 Peter Quill                  14 23-MAY-23
         9 Peter Quill                  15 02-NOV-22
         2 Tony Stark
         3 Bruce Banner
         6 Lucifer Morningstar
        10 Benjamin Tennyson

19 rows selected.
```

Figure 35: Listing all the customers with their order details and also the customers who have not ordered any products yet

As depicted in the picture above, the **Customer and Customer_Order** tables were **left joined on cust_ID** of both tables, followed by **left joining** the **Order** table on **ord_ID** of the **Order and Customer_Order** tables. Then, the **cust_ID, cust_name, ord_ID, and ord_date** attributes were **selected**, and the output was **sorted by ord_ID** and **then by cust_ID** in **ascending** order. The **left join** allowed the **customers without any orders** to be **displayed** as well.

Listing all product details that have the second letter ‘a’ in their product name and have a stock quantity more than 50

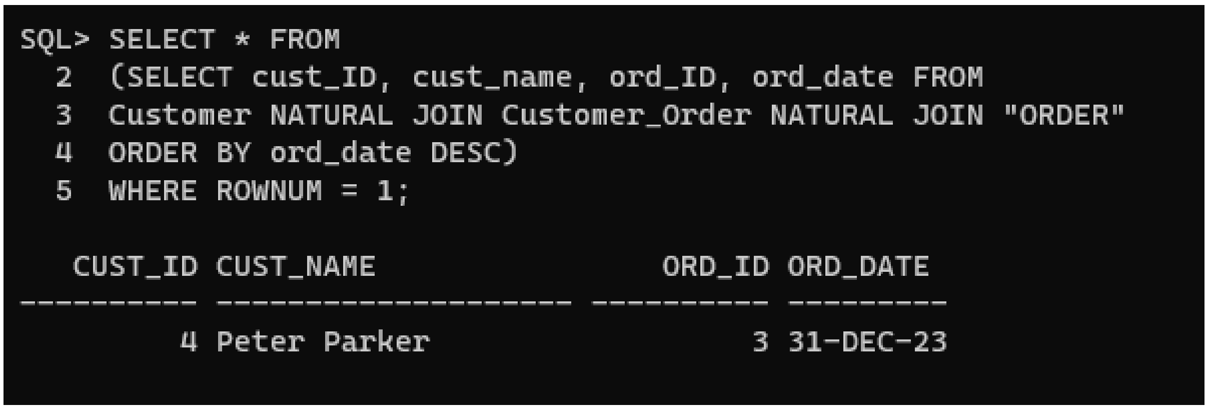


```
SQL> SELECT * FROM Product
  2  WHERE prod_name LIKE '_a%' AND prod_stock > 50;

   PROD_ID PROD_NAME            PROD_DESC            PROD_CATEGORY        PROD_PRICE PROD_STOCK  VENDOR_ID
---------- -------------------- -------------------- -------------------- ---------- ---------- ----------
         2 Galaxy Note 3        Smartphone           Productivity                500        200          2
         9 Macbook Air          Laptop               Productivity               1000        300          5
        14 HammerScan           Eye Scanner          Security                    800         70          4
        15 HammerLock           Smart Doorlock       Security                   1000         60          4
```

Figure 36: Listing all product details that have the second letter ‘a’ in their product name and have a stock quantity more than 50

The picture above shows that **all attributes** from the **Product** table were **selected** where the **prod_name** attribute had **‘a’ as the second character** while **simultaneously** having the **prod_stock** value **greater than 50**.

Finding the customer who has ordered recently



```
SQL> SELECT * FROM
  2  (SELECT cust_ID, cust_name, ord_ID, ord_date FROM
  3  Customer NATURAL JOIN Customer_Order NATURAL JOIN "ORDER"
  4  ORDER BY ord_date DESC)
  5  WHERE ROWNUM = 1;

   CUST_ID CUST_NAME                ORD_ID ORD_DATE
---------- -------------------- ---------- ---------
         4 Peter Parker                  3 31-DEC-23
```

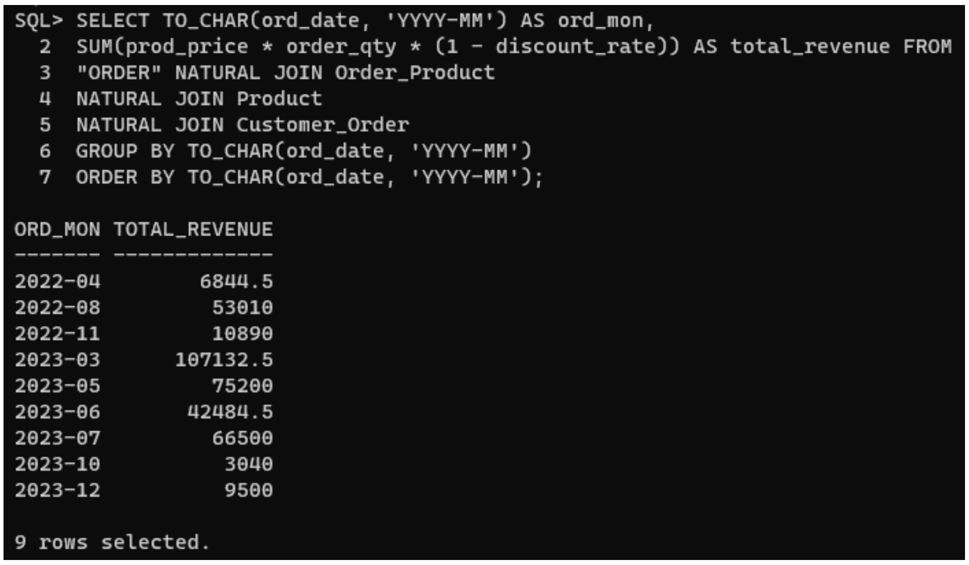
Figure 37: Finding the customer who has ordered recently

As seen above, the **Customer, Customer_Order, and Order** tables were **naturally joined**, and the **cust_ID, cust_name, ord_ID, and ord_date** attributes were selected. The result was then **sorted by ord_date** in **descending** order.

The query for the **output above** was enclosed as a **subquery**, out of which **only the top record** was then **displayed**.

6.2. Transaction queries

Showing the total revenue of the company for each month



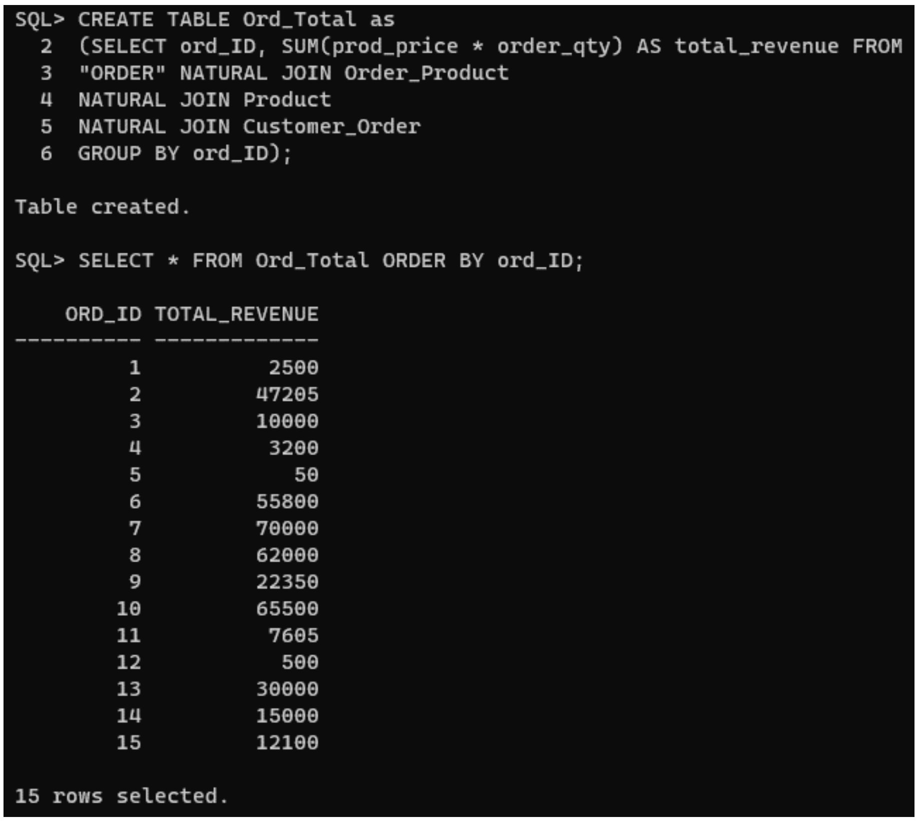
```
SQL> SELECT TO_CHAR(ord_date, 'YYYY-MM') AS ord_mon,
  2  SUM(prod_price * order_qty * (1 - discount_rate)) AS total_revenue FROM
  3  "ORDER" NATURAL JOIN Order_Product
  4  NATURAL JOIN Product
  5  NATURAL JOIN Customer_Order
  6  GROUP BY TO_CHAR(ord_date, 'YYYY-MM')
  7  ORDER BY TO_CHAR(ord_date, 'YYYY-MM');

ORD_MON TOTAL_REVENUE
------- -------------
2022-04        6844.5
2022-08         53010
2022-11         10890
2023-03      107132.5
2023-05         75200
2023-06       42484.5
2023-07         66500
2023-10          3040
2023-12          9500

9 rows selected.
```

Figure 38: Showing the total revenue of the company for each month

The picture above describes that the **Order, Order_Product, Product, and Customer_Order** tables were **naturally joined**, out of which the **ord_date (only the year and month)** and the **total_revenue** attributes were selected. The **year and month** were **extracted from ord_date** using the **TO_CHAR function**, and the **total_revenue** was calculated as the **sum of the products of the prod_price, order_qty, and (1 – discount_rate)** attributes, **grouped by** the **year-and-month** combination of the **ord_date**.

The query above resulted in the **output** of the **total revenue** of the company **per month**.

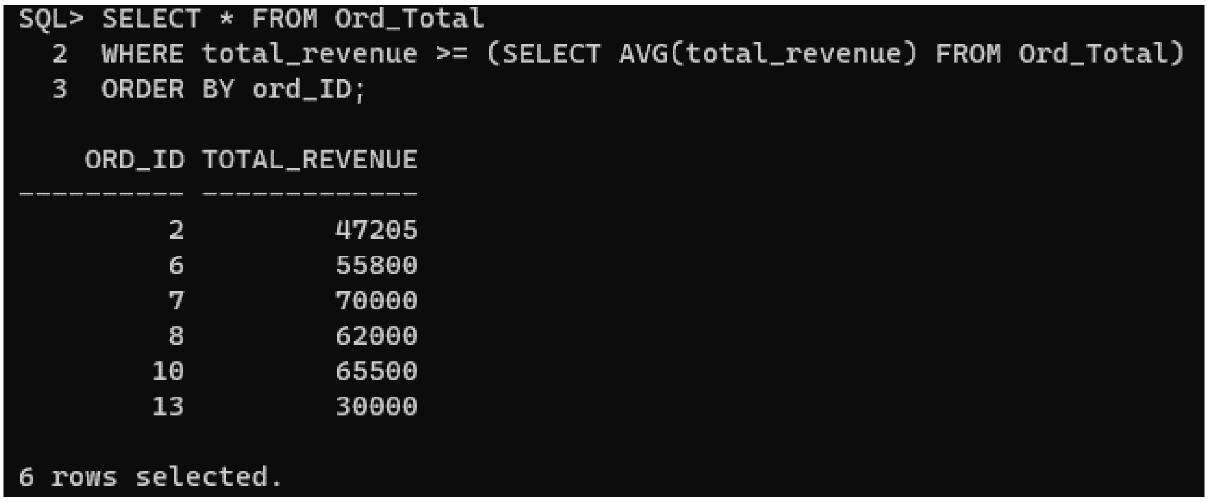
Finding the orders that are equal or higher than the average order total value



```
SQL> CREATE TABLE Ord_Total as
  (SELECT ord_ID, SUM(prod_price * order_qty) AS total_revenue FROM
  "ORDER" NATURAL JOIN Order_Product
  NATURAL JOIN Product
  NATURAL JOIN Customer_Order
  GROUP BY ord_ID);

Table created.

SQL> SELECT * FROM Ord_Total ORDER BY ord_ID;

    ORD_ID TOTAL_REVENUE
---------- -------------
         1          2500
         2         47205
         3         10000
         4          3200
         5            50
         6         55800
         7         70000
         8         62000
         9         22350
        10         65500
        11          7605
        12           500
        13         30000
        14         15000
        15         12100

15 rows selected.
```

Figure 39: Creating a temporary Ord_Total table with the order IDs and total amounts

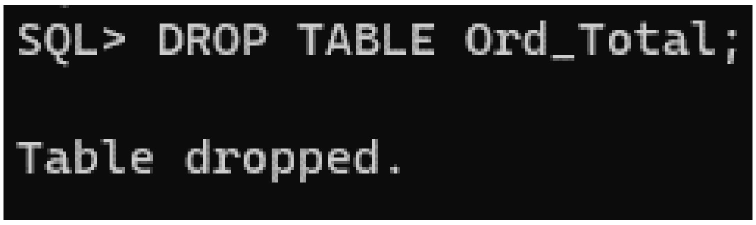
As seen in the picture above, a **temporary table** called **Ord_Total** was **created by naturally joining** the **Order, Order_Product, Product, and Customer_Order** tables, then **selecting** the **ord_ID and total_revenue** attributes, like in the previous query. The **values** in the resulting table were then **displayed**.



```
SQL> SELECT * FROM Ord_Total
  2  WHERE total_revenue >= (SELECT AVG(total_revenue) FROM Ord_Total)
  3  ORDER BY ord_ID;

    ORD_ID TOTAL_REVENUE
---------- -------------
         2         47205
         6         55800
         7         70000
         8         62000
        10         65500
        13         30000

6 rows selected.
```

Figure 40: Finding the orders that are equal or higher than the average order total value

Next, **all attributes** from the **Ord_Total** table were **selected**, and the records where **total_revenue** had values **equal to or higher than** the **average** of all its values were displayed.

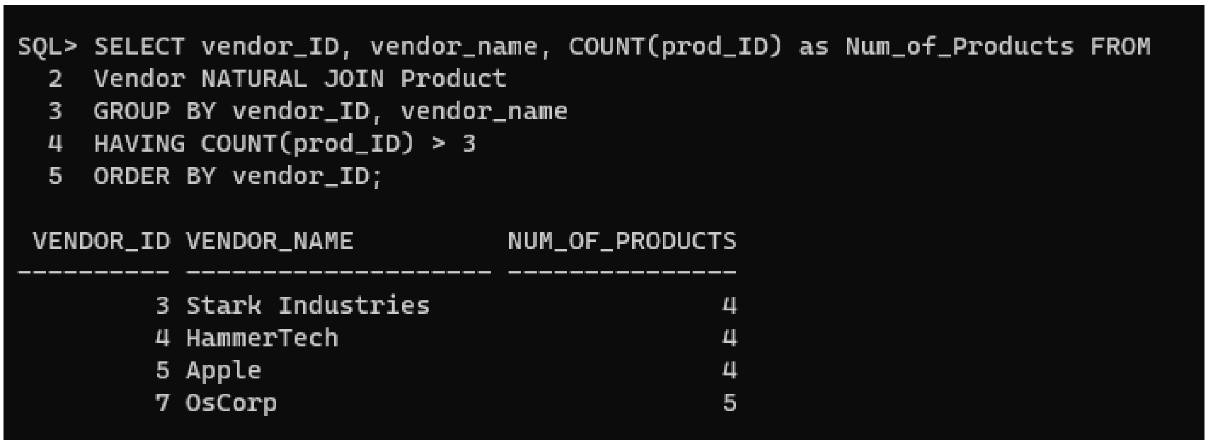


```
SQL> DROP TABLE Ord_Total;

Table dropped.
```

Figure 41: Dropping the Ord_Total table

The **Ord_Total** table was then **dropped**, as there was **no further use** of it.

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

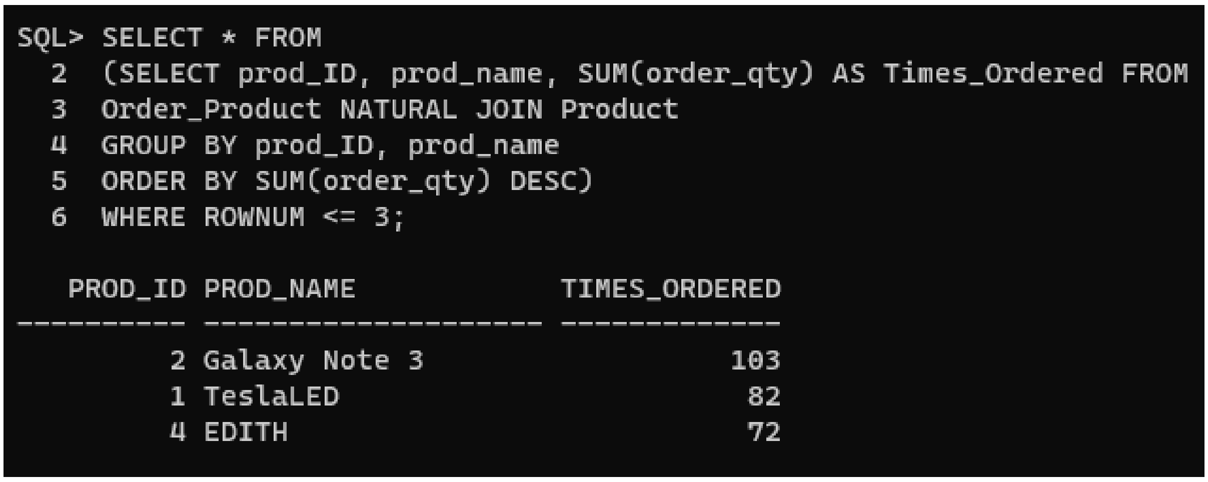


```
SQL> SELECT vendor_ID, vendor_name, COUNT(prod_ID) as Num_of_Products FROM
  2  Vendor NATURAL JOIN Product
  3  GROUP BY vendor_ID, vendor_name
  4  HAVING COUNT(prod_ID) > 3
  5  ORDER BY vendor_ID;

 VENDOR_ID VENDOR_NAME          NUM_OF_PRODUCTS
---------- -------------------- ---------------
         3 Stark Industries                   4
         4 HammerTech                         4
         5 Apple                              4
         7 OsCorp                             5
```

Figure 42: Listing the details of vendors who have supplied more than 3 products to the company

As seen in the picture above, the **Vendor and Product** tables were **naturally joined**, from which the **vendor_ID, vendor_name, and COUNT (prod_ID)** attributes were **selected**. The **COUNT (prod_ID)** was **grouped by vendor_ID and vendor_name**, which returned the **number of** different **prod_ID values** associated with **each vendor_ID**. Then, all the **records** having **COUNT (prod_ID) greater than 3** were returned, **sorted by vendor_ID** in ascending order.

Showing the top 3 product details that have been ordered the most



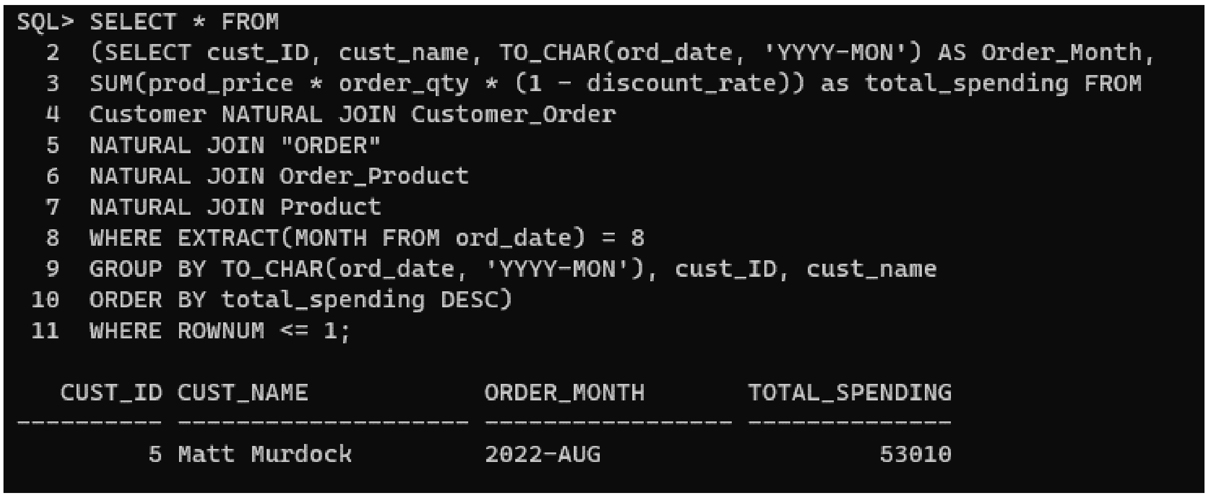
```
SQL> SELECT * FROM
  2  (SELECT prod_ID, prod_name, SUM(order_qty) AS Times_Ordered FROM
  3  Order_Product NATURAL JOIN Product
  4  GROUP BY prod_ID, prod_name
  5  ORDER BY SUM(order_qty) DESC)
  6  WHERE ROWNUM <= 3;

   PROD_ID PROD_NAME            TIMES_ORDERED
---------- -------------------- -------------
         2 Galaxy Note 3                  103
         1 TeslaLED                        82
         4 EDITH                           72
```

Figure 43: Showing the top 3 product details that have been ordered the most

The picture above shows that the **Order_Product and Product** tables were **naturally joined**, followed by **selecting** the **prod_ID, prod_name, and SUM (order_qty)** attributes. The **SUM (order_qty** was **grouped by prod_ID and prod_name**, which returned the **total quantities** of each product that had been ordered. The result was then **sorted** by the **SUM (order_qty)** in **descending** order.

The query for the **output from above** was enclosed as a **subquery**, out of which **only the top 3 rows** were then **displayed**.

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



```
SQL> SELECT * FROM
  (SELECT cust_ID, cust_name, TO_CHAR(ord_date, 'YYYY-MON') AS Order_Month,
  SUM(prod_price * order_qty * (1 - discount_rate)) as total_spending FROM
  Customer NATURAL JOIN Customer_Order
  NATURAL JOIN "ORDER"
  NATURAL JOIN Order_Product
  NATURAL JOIN Product
  WHERE EXTRACT(MONTH FROM ord_date) = 8
  GROUP BY TO_CHAR(ord_date, 'YYYY-MON'), cust_ID, cust_name
  ORDER BY total_spending DESC)
  WHERE ROWNUM <= 1;

   CUST_ID CUST_NAME            ORDER_MONTH       TOTAL_SPENDING
---------- -------------------- ----------------- --------------
         5 Matt Murdock         2022-AUG                   53010
```

Figure 44: Finding the customer who has ordered the most in August with his/her total spending on that month

It is clear from the picture above that the **Customer, Customer_Order, Order, Order_Product, and Product** tables were **naturally joined**, from which the **cust_ID, cust_name, and ord_date (year and month only)** attributes were **selected**, along with the **total_spending** attribute calculated as the **sum of the products of the prod_price, order_qty and (1 – discount_rate)** attributes **grouped by ord_date (year-and-month combination), cust_ID, and cust_name**. Then, all the **records** where the **month** in the **ord_date** attribute was **AUG (August)** were returned using the **EXTRACT function**. The records were then **sorted** by **total_spending** in **descending** order.

The query for the entire **result from above** was enclosed as a **subquery**, out of which **only the topmost row** was then **returned**.

7. Critical Evaluation

7.1. Critical Evaluation of the Module

The completion of the module CC5051NI Databases has proven to be an excellent way of learning the basic concepts of databases, including ERDs, database schemas, SQL functions, and many more. It was understood that data is the most valuable currency in the digital world, and efficient storage of data that can be accessed quickly was of utmost importance for any type of business. The most important concept in the development of a database schema was normalization, which turned out to be a critical factor in deciding whether a schema was valid for a DBMS software and ensuring that no data redundancies/anomalies existed after verifying it. Normalization was a tough concept to understand in the beginning but became significantly easier once the core of the concept clicked together. Also, normalization is not defined by strict rules but rather has some room where logical thinking and reasoning is required.

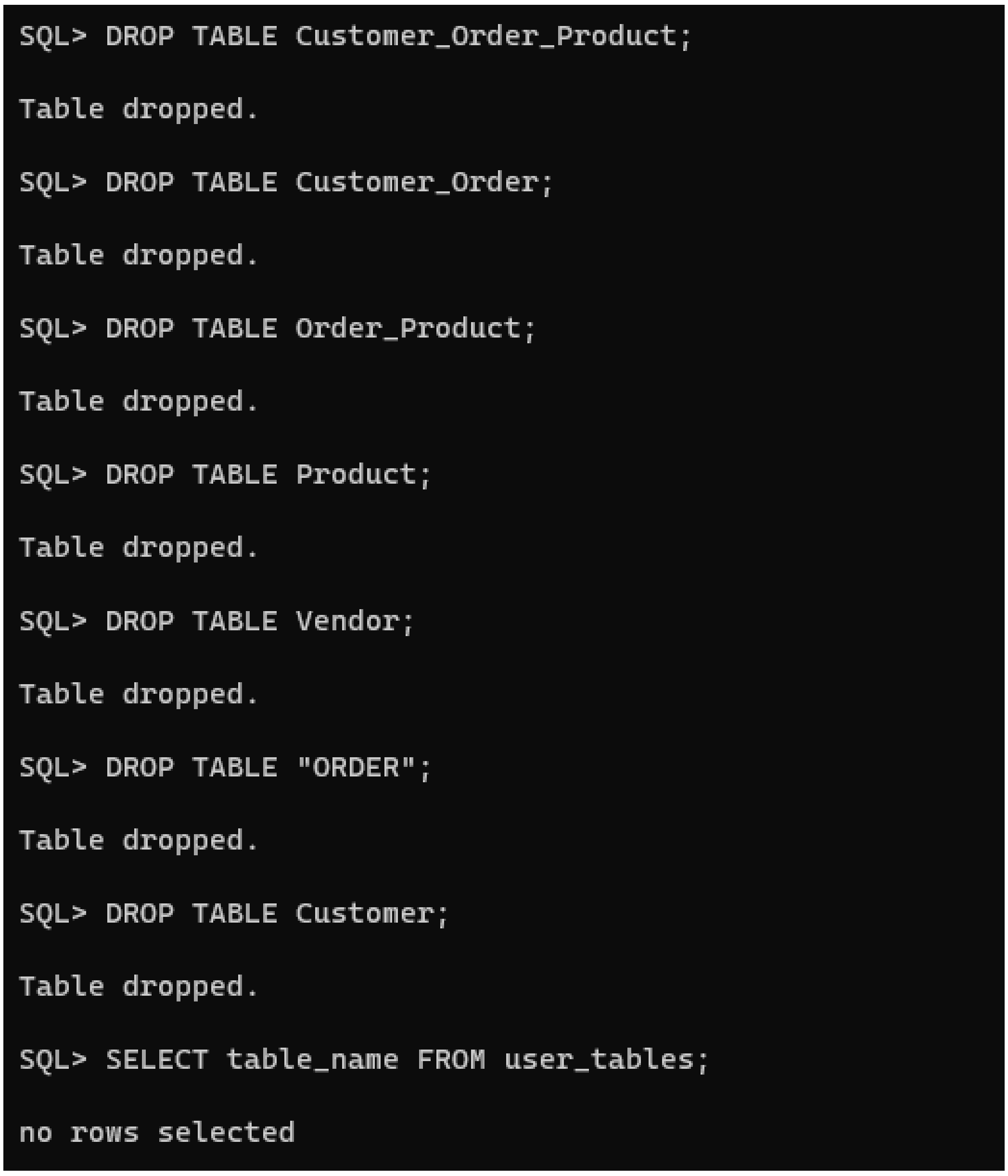
7.2. Critical Evaluation of the Coursework

The coursework for this module has served as one of the best ways of getting started with practical experience of using a DBMS and creating and managing a database. The coursework requirements demanded the development of a database that recorded the details of customers, orders, products, and vendors involved in an electronics retail store that recently launched an e-commerce platform to make use of the growing trend of online marketing. Logical thinking and reasoning were done to decide the entities and attributes that would comprise the database schema, and the records that would be inserted to produce the outputs according to the information and transaction queries mentioned in the coursework.

Overall, developing the initial ERDs were simple and straightforward, but involved multiple reviews with the module leader and tutors. Normalizing the attributes for the proposed schema followed by the designing of the final ERDs was an intensive task which required several repetitions to get right. Creating each table, populating them, and screenshotting the outputs of each query as well as describing them turned out was a tedious task as well. The spool file, server-side flat file, and dump file creations were straightforward as well.

8. Drop Queries and Dump File Creation

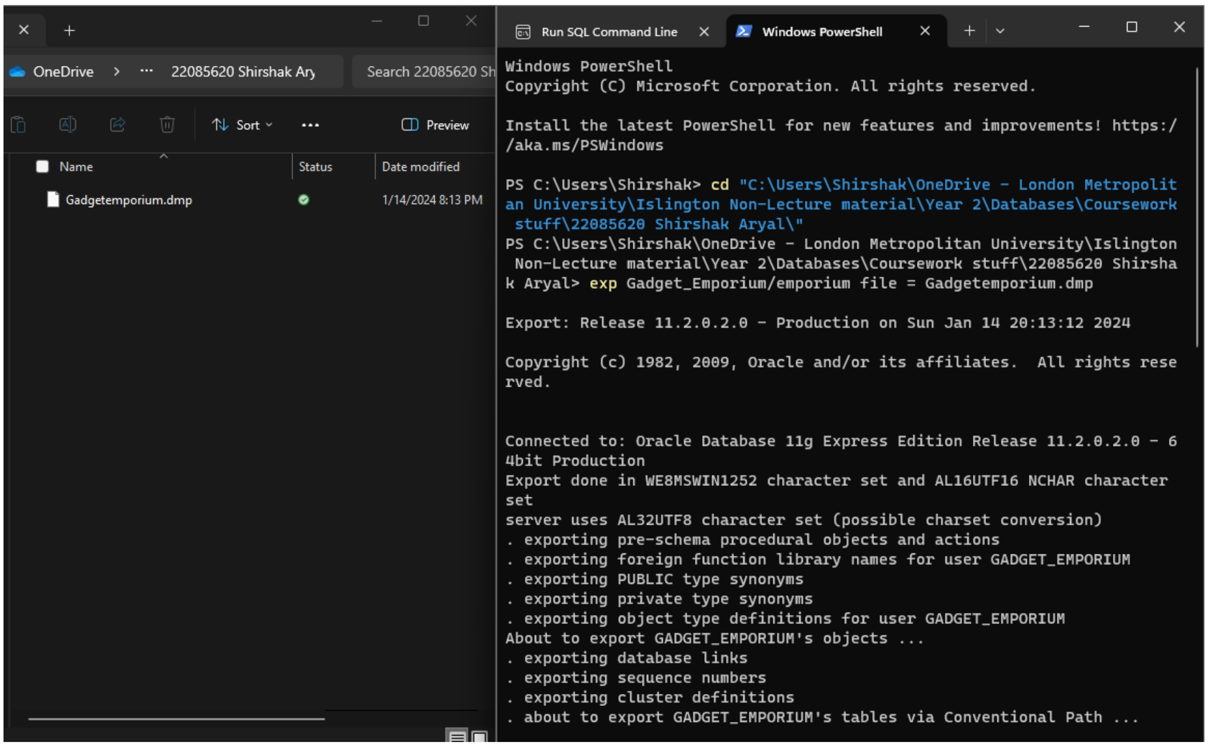
8.1. Drop Queries



```
SQL> DROP TABLE Customer_Order_Product;

Table dropped.

SQL> DROP TABLE Customer_Order;

Table dropped.

SQL> DROP TABLE Order_Product;

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> SELECT table_name FROM user_tables;

no rows selected
```

Figure 45: Dropping all tables in order

8.2. Dump File Creation



Name	Status	Date modified
Gadgetemporium.dmp	✓	1/14/2024 8:13 PM

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https:/
/aka.ms/PSWindows

PS C:\Users\Shirshak> cd "C:\Users\Shirshak\OneDrive - London Metropolit
an University\Islington Non-Lecture material\Year 2\Databases\Coursework
 stuff\22085620 Shirshak Aryal\"
PS C:\Users\Shirshak\OneDrive - London Metropolitan University\Islington
 Non-Lecture material\Year 2\Databases\Coursework stuff\22085620 Shirsha
k Aryal> exp Gadget_Emporium/emporium file = Gadgetemporium.dmp

Export: Release 11.2.0.2.0 - Production on Sun Jan 14 20:13:12 2024

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights rese
rved.


Connected to: Oracle Database 11g Express Edition Release 11.2.0.2.0 - 6
4bit Production
Export done in WE8MSWIN1252 character set and AL16UTF16 NCHAR character
set
server uses AL32UTF8 character set (possible charset conversion)
. exporting pre-schema procedural objects and actions
. exporting foreign function library names for user GADGET_EMPORIUM
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions for user GADGET_EMPORIUM
About to export GADGET_EMPORIUM's objects ...
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export GADGET_EMPORIUM's tables via Conventional Path ...
```

Figure 46: Creating a dump file of the coursework