CC5051NI Databases

50% Individual Coursework

Autumn 2023

Student Name: Dikshya Sharma

London Met ID: 22067520

Assignment Submission Date: 14 January 2023

Word Count: 7243

ACKNOWLEDGEMENT

I would like to take this opportunity to express my sincere gratitude and appreciation to everyone who helped me with the completion of this coursework. Primarily, I am so grateful for my module leader Mrs. Yunisha Bajracharya, our lecturer and tutor Mr. Aadesh Tandukar and all my respected teachers for their continuous guidance and all helpful advice and assistance they provided while working on this coursework. Their expertise, feedback and constant encouragement were immensely beneficial.

Whether it was in person or online through google hangouts, they were always willing to assist. Furthermore, I am thankful to the University and Islington College for providing the opportunity to undertake this coursework and entrusting us with the use of Oracle SQL Plus, a fully managed databases service. The knowledge and skills acquired through this coursework are valuable and will surely benefit me in the days to come.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

This report unfolds the blueprint for a vibrant online marketplace - "**Gadget Emporium**" envisioned by **Mr. John**, an enthusiastic entrepreneur, and electronics enthusiast which consists of database design and implementation for launching an online store that specializes in selling electronics devices and accessories. With a focus on simplicity and elegance, Mr. John aims to establish a go to destination online marketplace for both private consumers and business organizations with a wide selection of electronic devices.

1.1. Current Business Activities and Operations

Gadget Emporium specializes in offering a wide range of electronic gadgets and accessories which includes smartphones, laptops, Monitors, Headphones, chargers, automated devices, and many more innovative accessories. It focuses on staying up to date with current technical advancements to provide customers with advanced and high-quality gadgets.

This gadget emporium believes excellence is their benchmark and hence carefully selects a range of products ensuring quality, functionality, and relevance to current market demand by delivering exceptional experiences. It is dedicated to prioritizing customer's satisfaction by providing personalized journey for every individual, secure transactions, and thoughtful customer support.

Likewise, this gadget emporium also operates as a marketplace allowing third party sellers to also list their products on the platform. With frequent deals, discounts and offers, it attracts budget conscious shoppers by providing value to the customers while also staying competitive in the market. Being an online marketplace, it provides customers from all over the world with the ease of digital buying with a user-friendly interface for browsing, purchasing, and tracking orders.

As for Mr. John , he has created used some business activities and operations to make his gadget emporium as a hassle-free solution for individual and businesses to shop for their desired gadgets through online marketplace facilitating a user-friendly efficient platform for both private consumers and business organizations.

Furthermore, Mr. John's Gadget Emporium uses following business activities and operations :

- This Gadget Emporium efficiently manages the specifications of gadgets and accessories categorizing them appropriately .
- It offers suitable discounts to the customers according to their respective categories.
- It manages the processing of customer's order.
- It maintains records of vendors supplying gadgets and accessories.
- It monitors real time products availability to ensure accurate stock levels.
- It securely manages the customer's payment ensuring seamless secured transactions.
- It generates an invoice with all necessary details including applicable discount.

1.1.1. Business Rules

While Creating a Gadget Emporium, there were certain rules to be followed which are claimed as business rules and those business rules are listed below.

- Each Product belongs to only one category.
- Each category can have one or multiple products in it.
- Each customer is categorized as Regular(R), Staff(S), and VIP(V).
- Each customer category is entitled to different discount rates on product purchases.
- Discount rates are categorized as 0 % for regular, 5% for Staff and 10% for VIP.
- Each customer's address is also stored for delivery purposes.
- An order can have multiple products.
- One type of product can be included in multiple orders placed by multiple customers.
- One product must be associated with a single vendor.
- Each Vendor can supply one or more products.
- Each order detail must have one payment option.
- Payment options include cash on delivery, credit/debit card or e wallet.
- An invoice must be generated once the customer checks out the order.
- An invoice must include the applicable discount on the order of customer with order details.
- The system tracks real-time product quantity to maintain accurate stock levels.

1.1.2. Assumptions

While creating a database system, some assumptions were made for making system design efficient and for defining scope of the system. For this Gadget Emporium database , the following assumptions were made.

- One Vendor can supply multiple products.
- A Customer may or may not have an Order.
- Once a product is ordered, it cannot be returned.
- Once a payment is made, it cannot be refunded.
- An invoice is generated for each order.
- Order data is utilized to calculate the revenue of the company for each month.
- The system lists all the customers with their order details, including the customer who has not ordered any product yet.
- **Order_Quanity** is a quantity of a product bought by a customer and included in particular order of that customer therefore to identify Order Quantity, **Order_Id** and **Product_ID** must be known.
- **Line_Total** is calculated using order quantity and unit price of a product which is dependent on both **Order_Id** and **Product_Id**.

2. Entity Relationship Diagram(ERD)

An Entity Relationship Diagram(ERD) is a graphical representation of an information system that shows the relationship between people, objects, places, concepts, or events within that system. It is a conceptual representation model of data that shows the structure of entity framework. It is a high-level logical data model that is used to construct a conceptual design for a database (Rouse, 2017).

ERD are created in various scenarios beyond conceptualizing relational databases and facilitating physical database design. Below are some of its advantages.

- To visualize database design to identify the mistakes and design flaws before executing the changes in database.
- To analyze existing databases and to discover database problems easily by visualizing entire database schema.
- To define scopes and database requirements by drawing a conceptual diagram that depicts a high-level business object of the system.
- To present a representation of physical database structure through visual representation of data.
- To make it easier for business folks to understand the system by going through graphical and conceptual data modeling.
- To make it easier to make a system by creating table and relationship and write queries.
- To assure quality and validation as it is designed early in the development process (Visual Paradigm, 2023).

ERD has some elements which are listed below.

2.1. Entities

Entities are real world objects such as person, events that has it's certain elements called attributes. It is equivalent to database tables in a relational table in a relational database. In crow's foot notation, entity is represented by a rectangle with a name in a box on top (Nalimov, 2020).

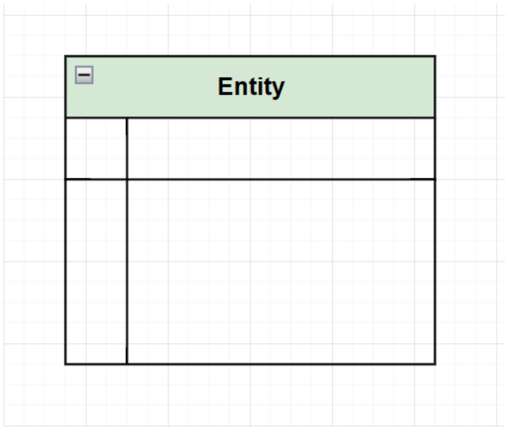


Figure 1: Entity Representation.

2.2. Attributes

Attributes are a property or a characteristic of the entity that holds it. An ERD attribute can be denoted as a primary key which defines unique key or a foreign key which can be assigned to multiple attributes or a non-key attribute. In the relational data model, a field represents an attribute. Attributes are listed underneath the entity in rest of the rectangle (Biscobing, 2019).

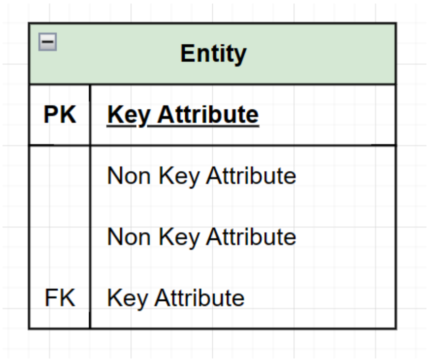


Figure 2: Attribute Representation.

2.3. Relationship

Relationships is the association that indicates how entities interact with each other which represented as straight line in crow's foot notation which has a name expressed as verb on its line to show the relationship between entities.

Relationship also consists of two indicators that are "**cardinality**" and "**modality**." The term "cardinality" describes how many times an instance of one entity can be linked to instances of another entity. The "modality" indicates how many times at most an instance of one entity can be linked to instances of the other (Nalimov, 2020).

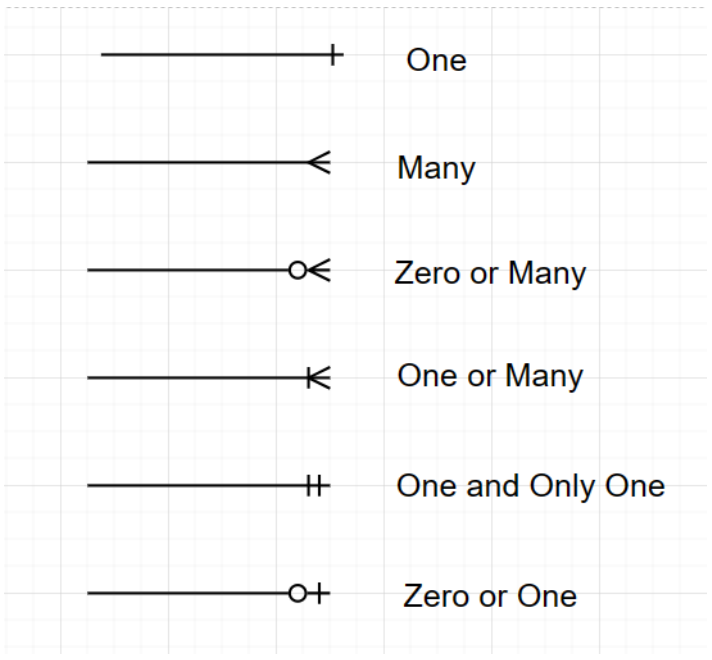


Figure 3: Relationship Representation.

3. Identification of Entities and Attributes

To optimize data consistency and minimize data redundancy in the Gadget Emporium Database, our approach involves initially creating a basic ERD. Then, we take a normalization process to enhance efficiency and reduce data duplication. This step-by-step approach helps maintain a well-structured database. Entities and Attributes involved in Initial ERD are given below.

Entities

- Orders
- Product
- Customer

Attributes that are in my Initial Entities are as follows.

Orders :- Order_Id, Order_date, Invoice_Id, Payment_Status, Payment_Type, Order-Total

Customer :- Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Contact, Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount

Product :- Product_Id, Product_Name, Stock_Quantity, Unit_Price, Order_Quantity, Line_Total, Product_Category_Id, Product_Category_Name, Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address

3.1. Orders

Attributes	Data Type	Constraints	Description
Order_Id	Number	Primary Key	This attribute stores unique order Identity.
Order_Date	Date	Not Null	This attribute stores date of order for tracking orders.
Invoice_Id	Number	Unique Not Null	This attribute stores unique identity of invoice.
Payment_Status	Varchar2(20)	Not Null	This attribute stores the status of a payment to track if the payment is done or not.
Payment_Type	Varchar2(20)	Not Null	This attribute stores type of payment whether is cash, credit or debit card or e wallet.
Order_Total	Number(10,2)	Not Null	This attribute stores total order amount.
Discount_Amount	Number(10,2)	Not Null	This attribute stores discounted amount of the order.
Grand_Total	Number(10,2)	Not_Null	This attribute stores total amount after discount.
Customer_Id	Number	Foreign Key Not Null	This attribute stores customer id and acts as Foreign Key for linking table orders and customer.

Table 1: Data Dictionary of Order.

3.2. Customer

Attributes	Data Type	Constraints	Description
Customer_Id	Number	Primary Key	This attribute stores unique customer identity.
First_Name	Varchar2(20)	Not Null	This attribute stores first name of a customer required for order and invoice.
Last_Name	Varchar2(20)	Not Null	This attribute stores last name of a customer required for order and invoice.
Customer_Address	Varchar2(50)	Not Null	This attribute stores address of a customer required for order processing.
Customer_Contact	Varchar2(20)	Unique Not Null	This attribute stores contact number of a customer.
Customer_Category_Id	Number	Unique Not Null	This attribute stores customer category id.
Customer_Category_Type	Varcha2(20)	Unique Not Null	This attribute stores customer category Type like VIP, Regular and staff.
Customer_Category_Discount	Number(10,2)	Unique Not Null	This attribute stores discount rate that is to be provided to the customer according to their categories.

Table 2: Data Dictionary of Customer.

3.3. Product

Attributes	Data Type	Constraints	Description
Product_Id	Number	Primary Key	This attribute stores unique identity of Product.
Product_Name	Varchar2(20)	Not Null	This attribute stores name of a product.
Stock_Quantity	Number	Not Null Check	This attribute store stock quantity to track real-time stock levels.
Unit_Price	Number(10,2)	Not Null	This attribute stores unit price of a product.
Order_Quantity	Number	Not Null	This attribute stores total product quantity ordered.
Line_Total	Number(10,2)	Not Null	This attribute stores line total that is, multiplies unit price by quantity.
Product_Category_Id	Number	Unique Not Null	This attribute stores product category Id.
Product_Category_Name	Varchar2(20)	Unique Not Null	This attribute stores category name of a product.
Vendor_Id	Number	Unique Not Null	This attribute stores vendor Id.
Vendor_Name	Varchar2(20)	Not Null	This attribute stores name of a vendor.
Vendor_Contact	Varchar2(50)	Unique	This attribute stores contact info of a vendor.
Vendor_Address	Varchar2(50)	N/A	This attribute stores address info of a vendor

Table 3:Data Dictionary of Product.

4. Initial Entity Relationship Diagram

The initial ERD for our Gadget Emporium is below:-

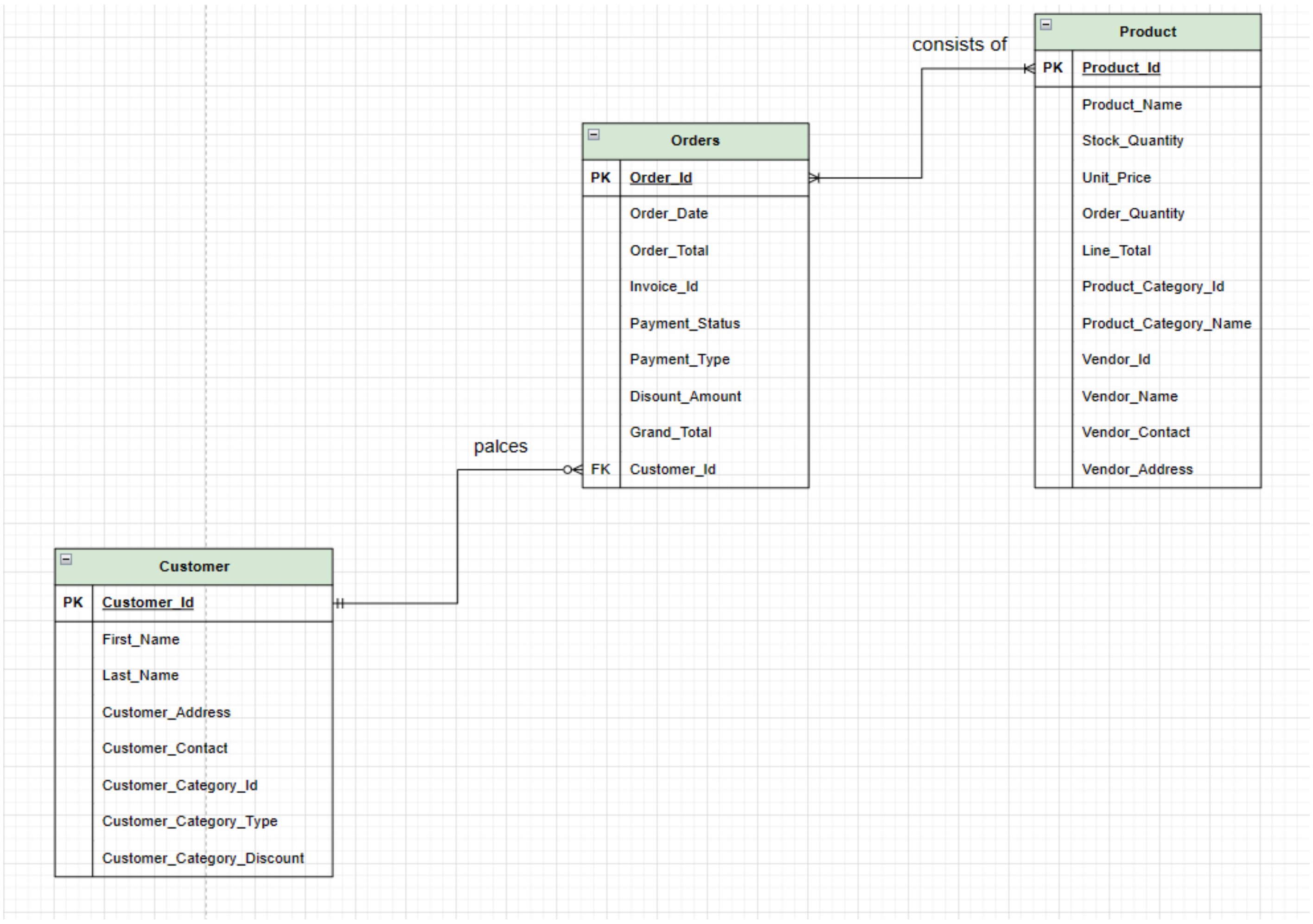


Figure 4: Initial ERD.

Here, in my Initial Entity Relationship Diagram, one customer can place one multiple order and one order is associated with only one customer. Similarly, one order consists of one or multiple products and one product can be associated with multiple orders. It consists of partial and transitive dependencies and the relation between two attributes is many-to-many which further results in insertion, deletion, and update anomalies.Likewise, while connecting the tables there is difficulty is identifying Foreign Key on table Orders and Products. Therefore, it needs to be normalized.

5. Normalization

Normalization is a process of organizing data in such a way that any redundancies and anomalies are eliminated. It is typically a refinement process after the initial exercise of identifying the data objects that must be in relational database by identifying their relationships and tables required. It is a multi-step process that stores data in a tabular form (Rouse, 2019).

Normalization degrees of relational database tables have been defined and includes:-

5.1. Un-Normalized Form(UNF)

Un-Normalized Form(UNF) is the simplest form of representing tables in a database which is also known as unnormalized relation or non-first normal form (NF^2) relation. It is characterized by redundant data and frequently includes complex data structures in a single attribute (Garge, 2022).

In this step all attributes from initial are included as unnormalized form and then followings steps are performed :-

- Identifier is given for the attributes.
- Repeating groups are kept in a curly brace.

Below is my unnormalized form of data of given scenario :-

Orders :- (Order_Id, Order_date, Order_total, Discount_Amount, Grand_Total, Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type, Customer_Id, First_Name, Last_Name, Customer_address, Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount, {Product_Id, Product_Name, Stock_Quantity, Unit_Price, Order_Quantity, Line_Total, Product_Category_Id, Product_Category_Name, Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address})

Here in my scenario from Initial ERD, I started with Orders attribute followed by Customer and Product hence the attributes of orders and products are my repeating data, and the attributes of product are my repeating groups.

The attributes are given an identifier and the repeating groups in it are separated using curly brace.

5.2. First Normal Form(1NF)

1NF is the basic normal forms in database normalization where any repeating group is separated, so that there is a single valued attribute at the intersection of each row and column of the table. In this form, following steps are included :-

- Repeating groups are removed to separate Relation(entity).
- A key attribute is assigned to the repeating group of attributes which has been separated to a new relation.
- The key attribute of non-repeating group is assigned to a new relation.

Below is the first normal form of my database:-

Orders-1 :- (**Order_Id(PK)**, Order_date, Order_total, Discount_Amount, Grand_Total, Invoice_Id, Payment_Status, Payment_Type, Customer_Id, First_Name, Last_Name, Customer_address, Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount)

Order_Product_Details-1 :- (**Order_Id*, Product_Id***, Product_Name, Stock_Quantity, Unit_Price, Order_Quantity, Line_Total, Product_Category_Id, Product_Category_Name, Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)

Here, in my 1NF , repeating groups are separated to a new relation named as **Order_Product_Details** and a key attribute is assigned and a Primary key from non-repeating group is carryforward to this relation.

5.3. Second Normal Form(2NF)

Second Normal Form is a second step of normalization where the repeating is separated on a new relation and does not contain any partial dependencies that is all non-key attributes are fully dependent on a primary key.

In this step, attributes that are fully dependent on only part of composite identifier should be removed to a separate relation. And partial functional dependencies on an identifier are avoided to eliminate data redundancy. In this form, following steps are included :-

- Functional Dependencies are identified from 1NF.
- Each determinant primary key for a new relation is made.
- Non-Keys attributes that are partially or fully dependent are separated with the key attribute on which they were dependent.

Partial Dependency

In this dependency, non-key attributes are functionally dependent on a component of a candidate key.

Fully Functional Dependency

In this dependency, non-key attributes are functionally reliant on candidate key.

Below is the process of second normal form of my database:-

- **Checking Partial Dependency For Orders Relation**

  Since there is only one key attribute, we cannot check for partial dependencies on this relation as it is already on Second Normal Form.

  Orders-1 :- (**Order_Id(PK)**, Order_date, Order_total, Discount_Amount, Grand_Total , Invoice_Id, Payment_Status, Payment_Type, Customer_Id, First_Name, Last_Name, Customer_address, Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount)

- **Checking Partial Dependency For Order_Product_Details Relation**

  On **Order_Product_Details** Relation, it is clearly visible that there is a presence of candidate key. Hence, there may be partial dependency on this relation.

  Order_Product_Details-1 :- (**Order_Id*, Product_Id***, Product_Name, Stock_Quantity, Unit_Price, Order_Quantity, Line_Total, Product_Category_Id, Product_Category_Name, Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)

  Order_Id -> XX

  There are not any non-key attributes that are partially dependent on **Order_Id**.

  Product_Id -> Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Product_Category_Name, Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address

All these non-key attributes are partially dependent on **Product_Id** attribute, so these attribute with its key must be separated to a new relation to reduce data redundancy.

Order_Id, Product_ID -> Order_Quantity, Line_Total

Assumption

- **Order_Quanity** is a quantity of a product bought by a customer and included in particular order of that customer therefore to identify it, **Order_Id** and **Product_ID** must be known.
- **Line_Total** is calculated using order quantity and unit price of a product. Hence, it is dependent on both **Order_Id** and **Product_Id**.

Final Relations in 2NF

Orders-2 :- (**Order_Id(PK)**, Order_date, Order_total, Discount_Amount, Grand_Total , Invoice_Id, Payment_Status, Payment_Type, Customer_Id, First_Name, Last_Name, Customer_address, Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount)

Order_Product_Details-2 :- (**Order_Id*, Product_Id***, Order_Quantity, Line_Total)

Products-2 :- (**Product_Id(PK)**, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Product_Category_Name, Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)

5.4. Third Normal Form(3NF)

A relation is said to be 3NF, if it satisfies the 2NF and does not have any transitive dependency.

Transitive Dependency

When a non-key attribute is dependent on other non-key attributes, it is said to be a transitive dependency. It must be removed from the relation as it creates data redundancy.

Therefore, this process is performed to element inconsistencies which includes following steps:-

- One relation for each determent is created in transitive dependency.
- Determinants Primary Keys are made in their respective relations.
- Those attributes that depend on determinant are included as non-key attribute.
- Foreign keys are assigned.

Below is the process of third normal form of my database:-

- **Checking Transitive Dependency on Orders**

Orders-2 :- (**Order_Id(PK)**, Order_date, Order_total, Discount_Amount, Grand_Total , Invoice_Id, Payment_Status, Payment_Type, Customer_Id, First_Name, Last_Name, Customer_address, Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount)

Order_Id -> Invoice_Id -> Discount_Amount, Grand_Total, Payment_Status, Payment_Type

Order_ID -> Customer_Id -> First_Name, Last_Name, Customer_address,Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount

Here, in Relation **Orders**, there exists a transitive dependency since the non-key attributes **Payment_Status**, **Payment_Type** , **Discount_Amount** and **Grand_Total** are dependent on another non-key attribute i.e., **Invoice_ID**.

So, we must eliminate transitive dependency by separating these attributes to different relation to eliminate data redundancy.

Likewise, other non-key attributes like **First_Name, Last_Name, Customer_address,Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount** are dependent on another non-key attribute i.e., **Customer_Id**.

So, we must also eliminate this transitive dependency by separating these attributes to different relation to eliminate data redundancy.

Orders-3 (**Order_Id(PK)**, Order_date, Order_total, **Invoice_Id(FK)**, **Customer_Id(FK)**)

Invoice -3 (**Invoice_Id(PK)**, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)

Customer-3 (**Customer_Id(PK)**, First_Name, Last_Name, Customer_address,Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount)

There is another transitive spotted inside **Customer** Relation, hence it must also be eliminated.

Customer_Id -> Customer_Category_Id -> Customer_Category_Type, Customer_Category_Discount

Here, non-key attributes **Customer_Category_Type** and **Customer_Category_Discount** are dependent on another non-key attribute i.e., **Customer_Category_Id**. Therefore, it must also be separated on another relation.

<u>Final Relations in 3NF for Orders</u>

Orders-3 (**Order_Id(PK)**, Order_date, Order_total, **Invoice_Id(FK)**, **Customer_Id(FK)**)

Invoice -3 (**Invoice_Id(PK)**, Discount_Amount, Grand_Total Payment_Status, Payment_Type)

Customer-3 (**Customer_Id(PK)**, First_Name, Last_Name, Customer_address,**Customer_Category_Id(FK))**

Customer_Category-3 (**Customer_Category_Id(PK),** Customer_Category_Type, Customer_Category_Discount)

- **Checking Transitive Dependency on Products**

Products-2 :- (**Product_Id(PK)**, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Product_Category_Name, Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)

Product_Id -> Product_Category_Id -> Product_Category_Name

Product_Id -> Vendor_Id -> Vendor_Name, Vendor_Contact, Vendor_Address

Here, in Relation **Products**, there exists a transitive dependency since the non-key attribute **Product_Category_Name** is dependent on another non-key attribute i.e., **Product_Category_Id**.

So, we must eliminate transitive dependency by separating these attributes to different relation to eliminate data redundancy.

Likewise, there exists a transitive dependency since the non-key attributes **Vendor_Name, Vendor_Contact** and **Vendor_Address** is dependent on another non-key attribute i.e., **Vendor_Id**. So, we must eliminate transitive dependency by separating these attributes to different relation to eliminate data redundancy.

Final Relations in 3NF for Products

Product-3 (**Product_Id(PK)**, Product_Name, Stock_Quantity, Unit_Price, **Product_Category_Id(FK)**, **Vendor_Id(FK)**)

Product_Category-3 (**Product_Category_Id(PK),** Product_Category_Name)

Vendor-3 (**Vendor_Id(PK) ,** Vendor_Name, Vendor_Address)

- **Checking Transitive Dependency on Order_Product_Details**

Order_Product_Details-2 :- (**Order_Id*, Product_Id***, Order_Quantity, Line_Total)

Order_Id, Product_Id -> Order_Quantity -> X

Order_Id, Product_Id -> Line_Total -> X

On this Relation, there is not any transitive dependency as non-key attributes are not dependent on any other non-key attributes.

<u>Final Relations in 3NF for Products</u>

Orders (**Order_Id(PK)**, Order_Date, Order_total, **Invoice_Id(FK)**, **Customer_Id(FK)**)

Invoice (**Invoice_Id(PK)**, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)

Customer (**Customer_Id(PK)**, First_Name, Last_Name, Customer_address, **Customer_Category_Id(FK))**

Customer_Category (**Customer_Category_Id(PK),** Customer_Category_Type, Customer_Category_Discount)

Product (**Product_Id(PK)**, Product_Name, Stock_Quantity, Unit_Price, **Product_Category_Id(FK)**, **Vendor_Id(FK)**)

Product_Category (**Product_Category_Id(PK),** Product_Category_Name)

Vendor-(**Vendor_Id(PK) ,** Vendor_Name, Vendor_Address)

Order_Product_Details :- (**Order_Id(PK, FK), Product_Id(PK, FK)**, Order_Quantity, Line_Total)

6. Final Entity Relationship Diagram

The final ERD created after normalization is shown below:-

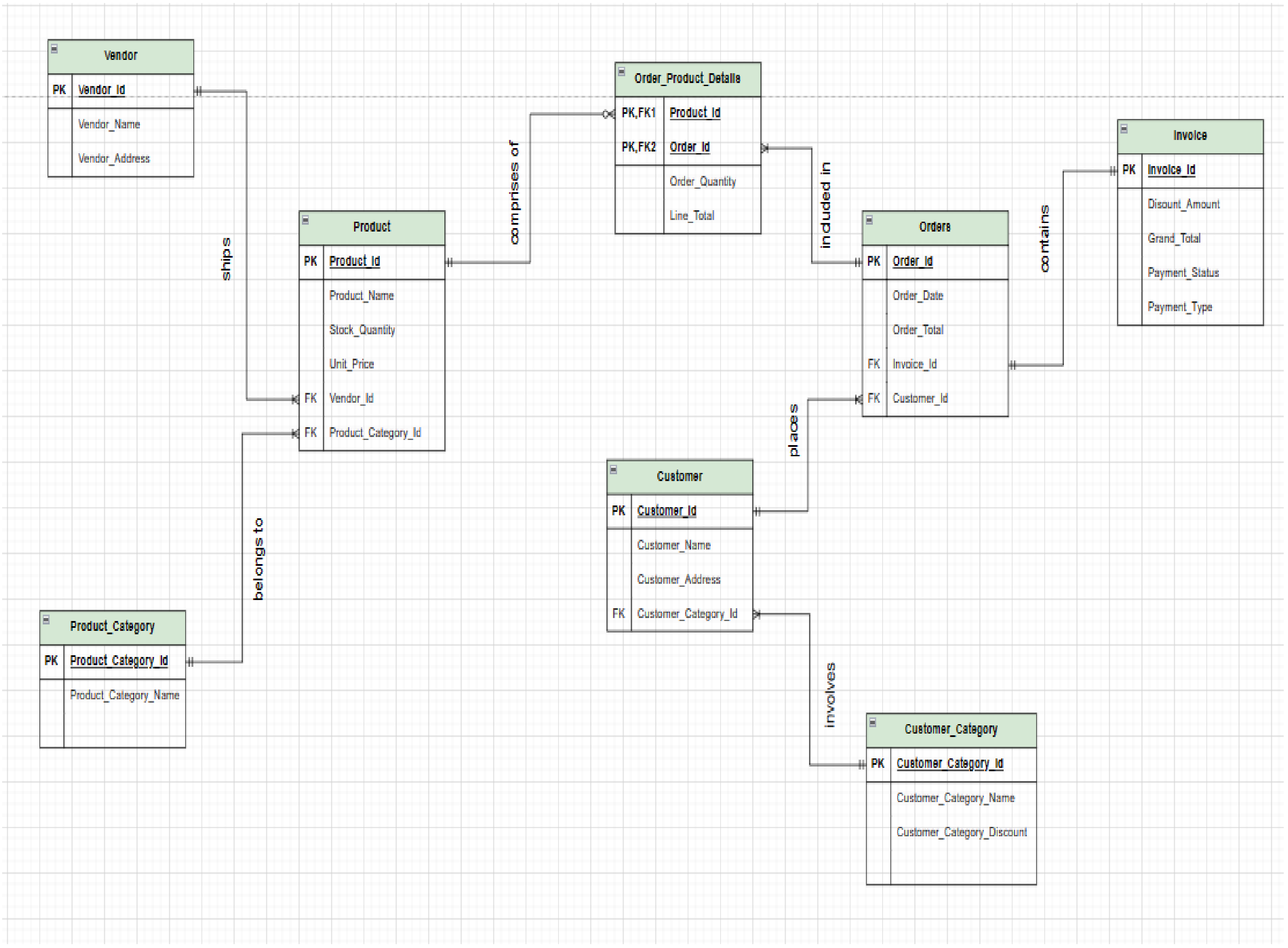


Figure 5: Final ERD.

The outcome of the normalization process is the Final ERD that is seen above. It has eight tables with distinct types of data stored in them, each having unique features. There is one bridge table that connects two tables establishing the proper connection between two tables which were initially of many – many relationships.

The tables that are formed after normalization are **Orders, Product, Order_Product_Details, Vendor, Product_Category, Customer, Customer_Category** and **Invoice.** Here, **Order_Product_Details** is a bridge table between **Orders** and **Products** that helps to maintain integrity in a database.

Similarly, in the above ERD, the connections are shown via Primary Key and Foreign Key for better understanding the tables and how they are connected. There is also a presence of cardinality and modality between the tables which is integral for maintaining data integrity and understanding and communicating about the architecture of database schema.

Likewise, logical model is utilized in the above ERD to provide a thorough structured description of the data elements and the relationships between them. It focuses on the relationships between the entities and the essential properties that link two tables providing a clear and concise visual representation of the tables.

Below is the total number of entities with their respective attributes of Final ERD:-

Orders (**Order_Id(PK)**, Order_Date, Order_total, **Invoice_Id(FK)**, **Customer_Id(FK)**)

Invoice (**Invoice_Id(PK)**, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)

Customer (**Customer_Id(PK)**, First_Name, Last_Name, Customer_address, **Customer_Category_Id(FK))**

Customer_Category (**Customer_Category_Id(PK),** Customer_Category_Type, Customer_Category_Discount)

Product (**Product_Id(PK)**, Product_Name, Stock_Quantity, Unit_Price, **Product_Category_Id(FK)**, **Vendor_Id(FK)**)

Product_Category (**Product_Category_Id(PK),** Product_Category_Name)

Vendor-(**Vendor_Id(PK) ,** Vendor_Name, Vendor_Address)

Order_Product_Details :- (**Order_Id(PK, FK), Product_Id(PK, FK)**, Order_Quantity, Line_Total)

7. Implementation

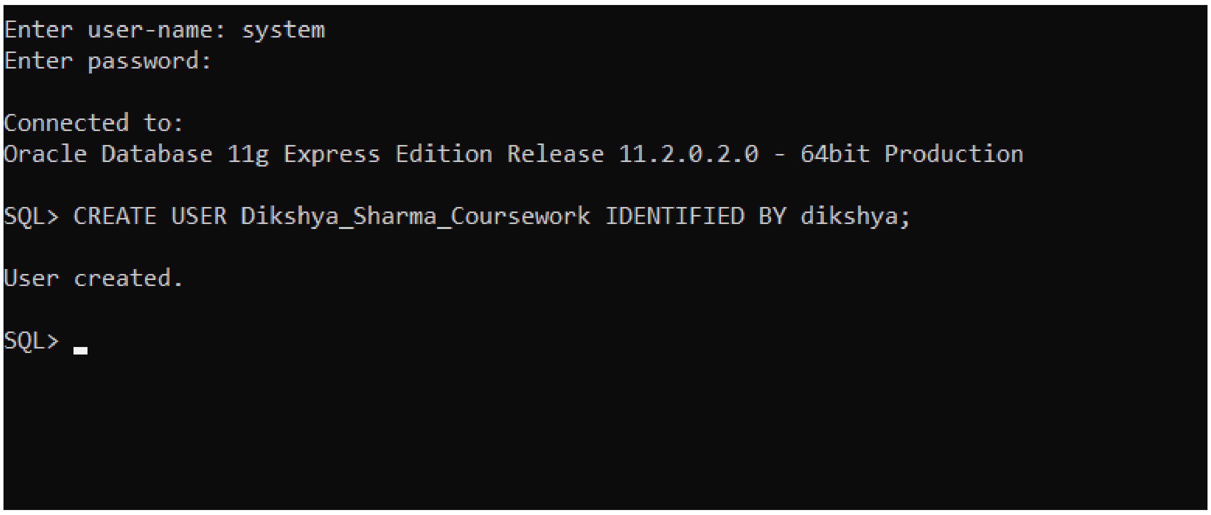
In SQL Plus, Implementation of tables involves following steps:-

7.1. Creating User

First step of database implementation is Creating a User on the system. Here, **CREATE** statement is used for creating a user **IDENTIFIED BY** statement is used for setting password for user.

Purpose	Command Implemented
Creating a new User.	**CREATE USER** Dikshya_Sharma_Coursework **IDENTIFIED BY** dikshya;

Table 4: Creating User.



```
Enter user-name: system
Enter password:

Connected to:
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production

SQL> CREATE USER Dikshya_Sharma_Coursework IDENTIFIED BY dikshya;

User created.

SQL> _
```

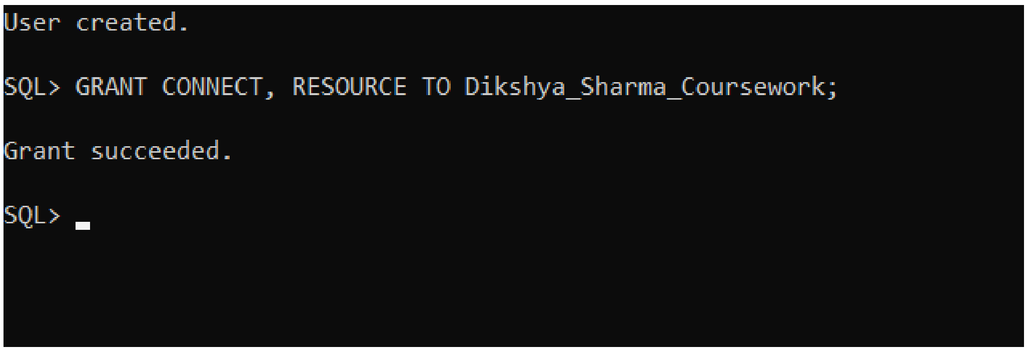
Figure 6: Creating User.

7.2. Granting Privileges to the User

Second Step includes granting privileges to a user. A user privilege is a right to execute a particular type of SQL statement or to access another user's object which are defined by Oracle. Here, GRANT statement is used for providing permission or privileges or resources to database user and objects.

Purpose	Command Implemented
Granting Privileges to User.	**GRANT CONNECT**, **RESOURCE TO** Dikshya_Sharma_Coursework;

Table 5: Granting Privileges to the User.



```
User created.

SQL> GRANT CONNECT, RESOURCE TO Dikshya_Sharma_Coursework;

Grant succeeded.

SQL>
```

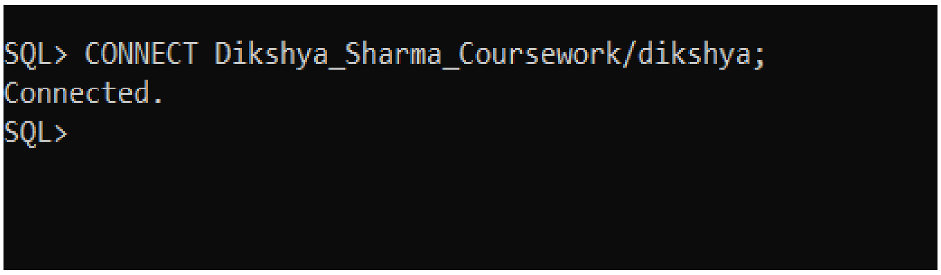
Figure 7: Granting Privileges to user.

7.3. Connecting User

After Creating User and Granting Privilege to the User, we connected user for further implementation.

Purpose	Command Implemented
Connecting User.	**CONNECT** Dikshya_Sharma_Coursework/dikshya;

Table 6: Connecting User.



```
SQL> CONNECT Dikshya_Sharma_Coursework/dikshya;
Connected.
SQL>
```

Figure 8: Connecting User.

7.4. Creation and Description of Tables

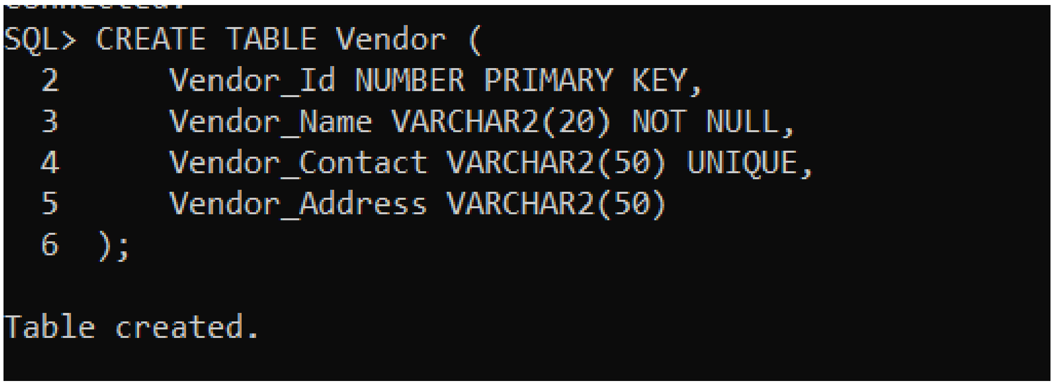
Following that, we now create a table and display it's description.

7.4.1. Vendor

Table Vendor was created with four attributes in it where Vendor_Id is the Primary Key that uniquely defines the table.

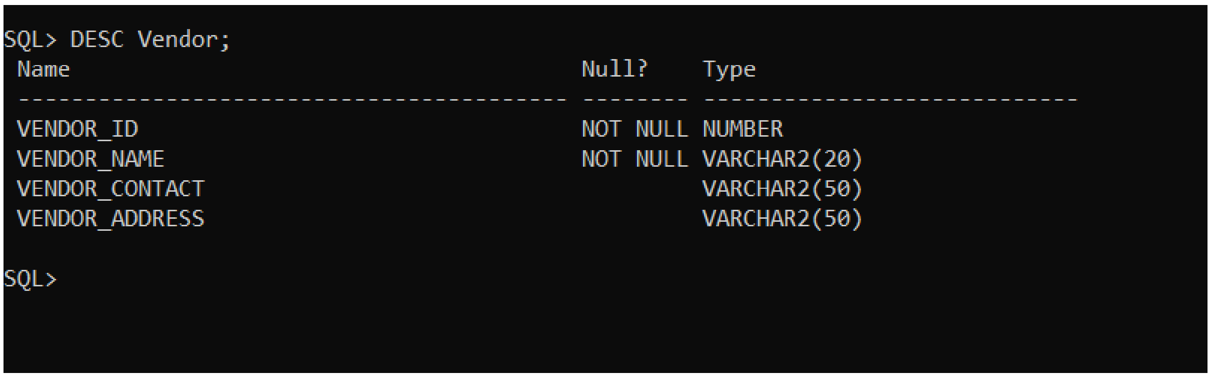
Purpose	Command Implemented
Creating a Vendor Table.	**CREATE TABLE** Vendor (Vendor_Id NUMBER PRIMARY KEY, Vendor_Name VARCHAR2(20) NOT NULL, Vendor_Contact VARCHAR2(50) UNIQUE, Vendor_Address VARCHAR2(50));
Describing Vendor Table.	**Desc** Vendor;

Table 7: Creating and Describing Vendor Table.



```
SQL> CREATE TABLE Vendor (
  2      Vendor_Id NUMBER PRIMARY KEY,
  3      Vendor_Name VARCHAR2(20) NOT NULL,
  4      Vendor_Contact VARCHAR2(50) UNIQUE,
  5      Vendor_Address VARCHAR2(50)
  6  );

Table created.
```

Figure 9: Creating Vendor Table.



```
SQL> DESC Vendor;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 VENDOR_ID                                 NOT NULL NUMBER
 VENDOR_NAME                               NOT NULL VARCHAR2(20)
 VENDOR_CONTACT                                     VARCHAR2(50)
 VENDOR_ADDRESS                                     VARCHAR2(50)

SQL>
```

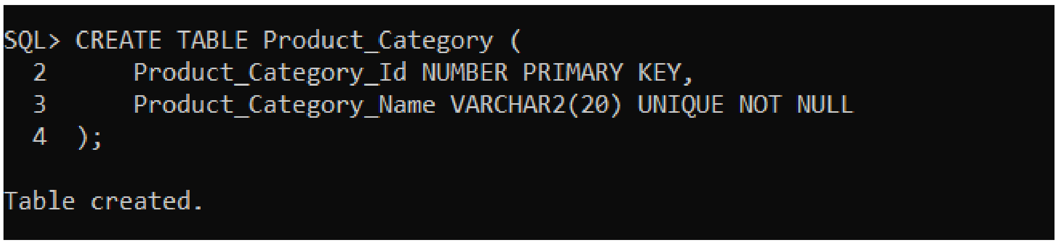
Figure 10:Describing Vendor Table.

7.4.2. Product_Category

Table Product_Category was created with two attributes in it where Product_Category_Id is the Primary Key that uniquely defines the table.

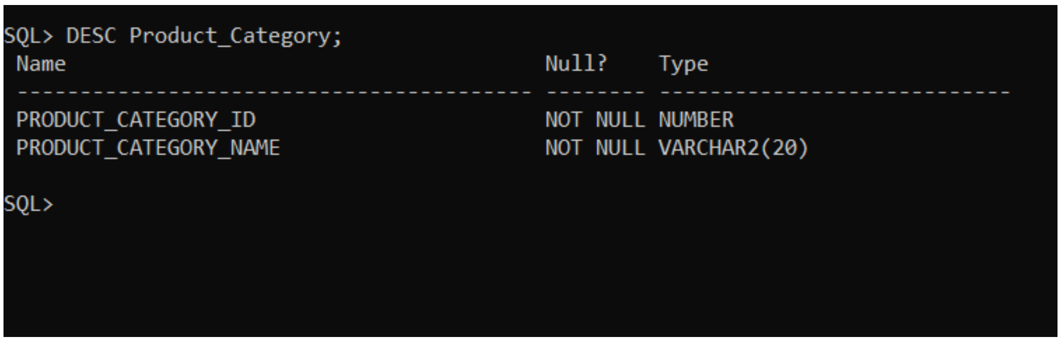
Purpose	Command Implemented
Creating a Product_Category Table.	**CREATE TABLE** Product_Category (Product_Category_Id NUMBER PRIMARY KEY, Product_Category_Name VARCHAR2(20) UNIQUE NOT NULL);
Describing Product_Category Table.	**Desc** Product_Category;

Table 8:Creating and Describing Product_Category Table.



```
SQL> CREATE TABLE Product_Category (
  2      Product_Category_Id NUMBER PRIMARY KEY,
  3      Product_Category_Name VARCHAR2(20) UNIQUE NOT NULL
  4  );

Table created.
```

Figure 11: Creating Product_Category Table.



```
SQL> DESC Product_Category;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODUCT_CATEGORY_ID                       NOT NULL NUMBER
 PRODUCT_CATEGORY_NAME                     NOT NULL VARCHAR2(20)

SQL>
```

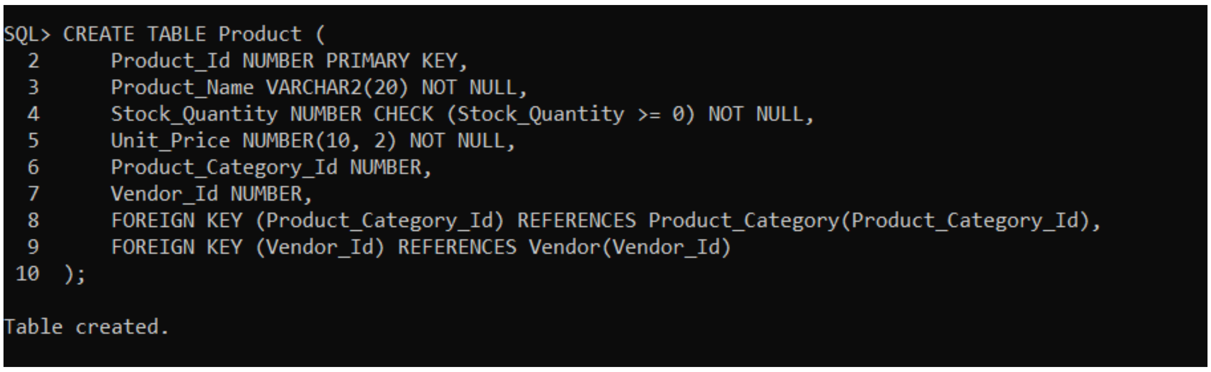
Figure 12: Describing Product_Category Table.

7.4.3. Product

Table Product was created with six attributes in it where Product_Id is the Primary Key that uniquely defines the table and Product_Category_Id and Vendor_Id is the Foreign Key that connects the table .

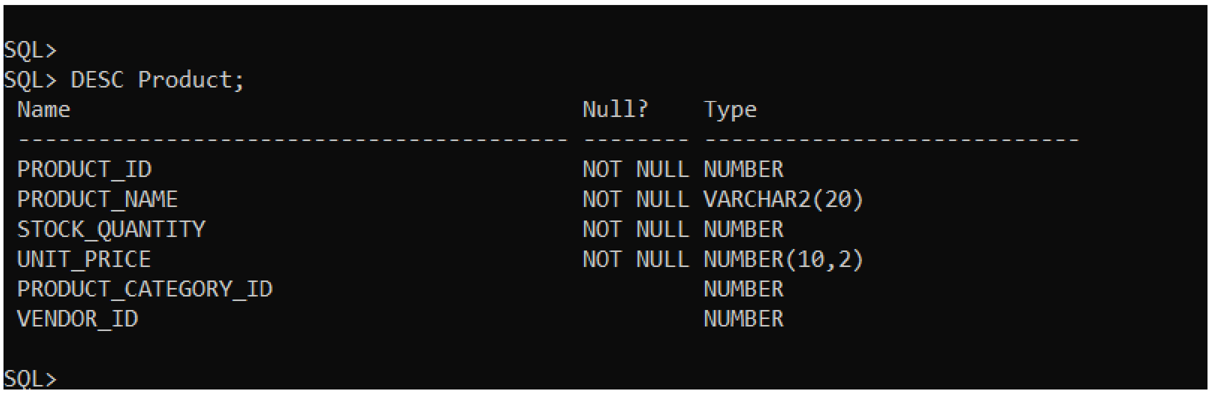
Purpose	Command Implemented
Creating a Product Table.	**CREATE TABLE** Product (Product_Id NUMBER PRIMARY KEY, Product_Name VARCHAR2(20) NOT NULL, Stock_Quantity NUMBER CHECK (Stock_Quantity >= 0) NOT NULL, Unit_Price NUMBER(10, 2) NOT NULL, Product_Category_Id NUMBER, Vendor_Id NUMBER, FOREIGN KEY (Product_Category_Id) REFERENCES Product_Category(Product_Category_Id), FOREIGN KEY (Vendor_Id) REFERENCES Vendor(Vendor_Id));
Describing Product Table.	**Desc** Product;

Table 9: Creating and Describing Product Table.



```
SQL> CREATE TABLE Product (
       Product_Id NUMBER PRIMARY KEY,
       Product_Name VARCHAR2(20) NOT NULL,
       Stock_Quantity NUMBER CHECK (Stock_Quantity >= 0) NOT NULL,
       Unit_Price NUMBER(10, 2) NOT NULL,
       Product_Category_Id NUMBER,
       Vendor_Id NUMBER,
       FOREIGN KEY (Product_Category_Id) REFERENCES Product_Category(Product_Category_Id),
       FOREIGN KEY (Vendor_Id) REFERENCES Vendor(Vendor_Id)
   );

Table created.
```

Figure 13: Creating Product Table.



```
SQL>
SQL> DESC Product;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODUCT_ID                                NOT NULL NUMBER
 PRODUCT_NAME                              NOT NULL VARCHAR2(20)
 STOCK_QUANTITY                            NOT NULL NUMBER
 UNIT_PRICE                                NOT NULL NUMBER(10,2)
 PRODUCT_CATEGORY_ID                                NUMBER
 VENDOR_ID                                          NUMBER

SQL>
```

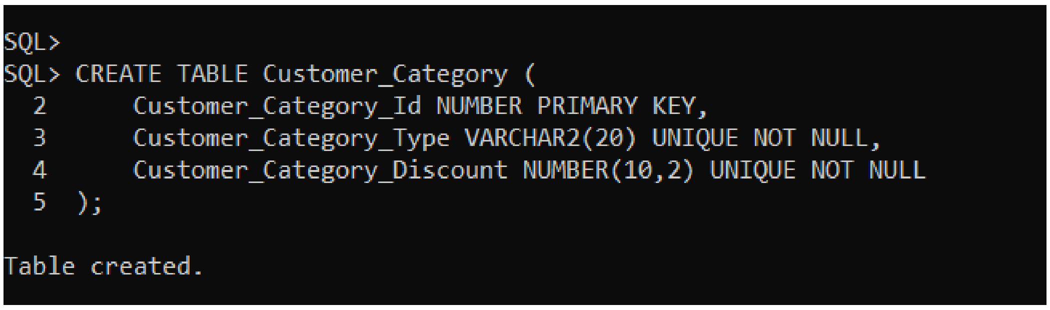
Figure 14:Describing Product Table.

7.4.4. Customer_Category

Table Customer_Category was created with three attributes in it where Customer_Category_Id is the Primary Key that uniquely defines the table.

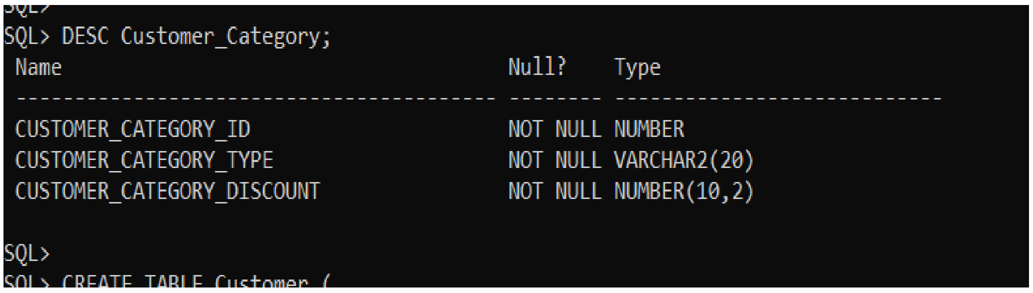
Purpose	Command Implemented
Creating a customer_Category Table.	**CREATE TABLE** Customer_Category (Customer_Category_Id NUMBER PRIMARY KEY, Customer_Category_Type VARCHAR2(20) UNIQUE NOT NULL, Customer_Category_Discount NUMBER(10,2) UNIQUE NOT NULL);
Describing Customer_Category Table.	**Desc** Customer_Category;

Table 10: Creating and Describing Customer_Category Table.



```
SQL>
SQL> CREATE TABLE Customer_Category (
  2      Customer_Category_Id NUMBER PRIMARY KEY,
  3      Customer_Category_Type VARCHAR2(20) UNIQUE NOT NULL,
  4      Customer_Category_Discount NUMBER(10,2) UNIQUE NOT NULL
  5  );

Table created.
```

Figure 15: Creating Customer_Category Table.



```
SQL>
SQL> DESC Customer_Category;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_CATEGORY_ID                      NOT NULL NUMBER
 CUSTOMER_CATEGORY_TYPE                    NOT NULL VARCHAR2(20)
 CUSTOMER_CATEGORY_DISCOUNT                NOT NULL NUMBER(10,2)

SQL>
SQL> CREATE TABLE Customer (
```

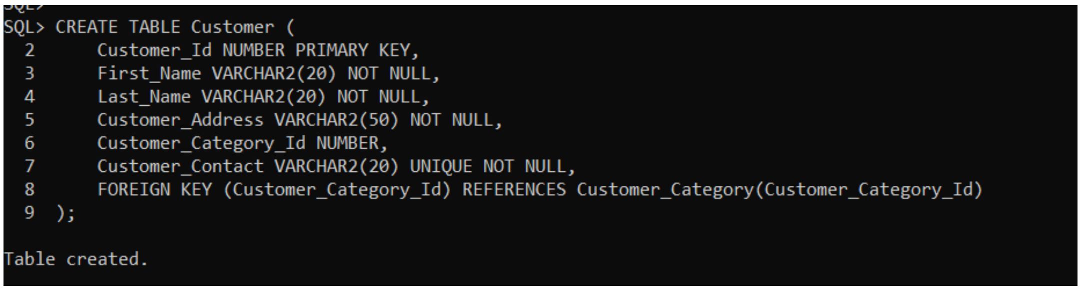
Figure 16:Describing Customer_Category Table.

7.4.5. Customer

Table Customer was created with six attributes in it where Customer_Id is the Primary Key that uniquely defines the table and Customer_Category_Id is the Foreign Key that connects the table.

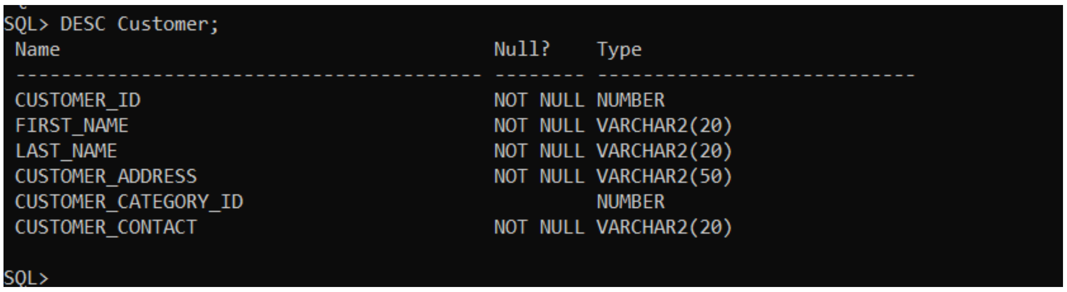
Purpose	Command Implemented
Creating a Customer Table.	**CREATE TABLE** Customer (Customer_Id NUMBER PRIMARY KEY, First_Name VARCHAR2(20) NOT NULL, Last_Name VARCHAR2(20) NOT NULL, Customer_Address VARCHAR2(50) NOT NULL, Customer_Category_Id NUMBER, Customer_Contact VARCHAR2(20) UNIQUE NOT NULL, FOREIGN KEY (Customer_Category_Id) REFERENCES Customer_Category(Customer_Category_Id));
Describing Customer Table.	**Desc** Customer;

Table 11: Creating and Describing Customer Table.



```
SQL> CREATE TABLE Customer (
      Customer_Id NUMBER PRIMARY KEY,
      First_Name VARCHAR2(20) NOT NULL,
      Last_Name VARCHAR2(20) NOT NULL,
      Customer_Address VARCHAR2(50) NOT NULL,
      Customer_Category_Id NUMBER,
      Customer_Contact VARCHAR2(20) UNIQUE NOT NULL,
      FOREIGN KEY (Customer_Category_Id) REFERENCES Customer_Category(Customer_Category_Id)
  );

Table created.
```

Figure 17: Creating Customer Table.



```
SQL> DESC Customer;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_ID                               NOT NULL NUMBER
 FIRST_NAME                                NOT NULL VARCHAR2(20)
 LAST_NAME                                 NOT NULL VARCHAR2(20)
 CUSTOMER_ADDRESS                          NOT NULL VARCHAR2(50)
 CUSTOMER_CATEGORY_ID                               NUMBER
 CUSTOMER_CONTACT                          NOT NULL VARCHAR2(20)

SQL>
```

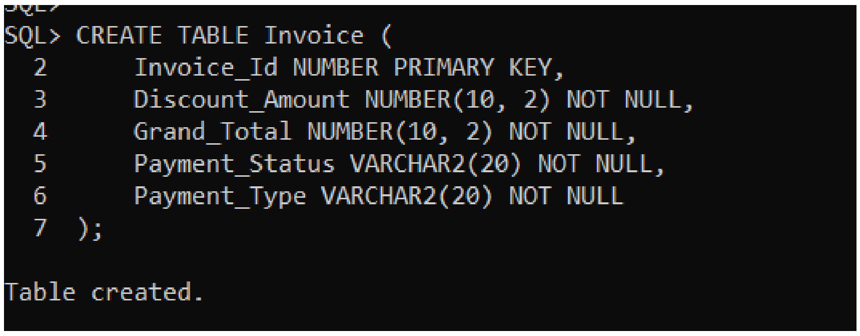
Figure 18: Describing Customer Table.

7.4.6. Invoice

Table Invoice was created with five attributes in it where Invoice_Id is the Primary Key that uniquely defines the table.

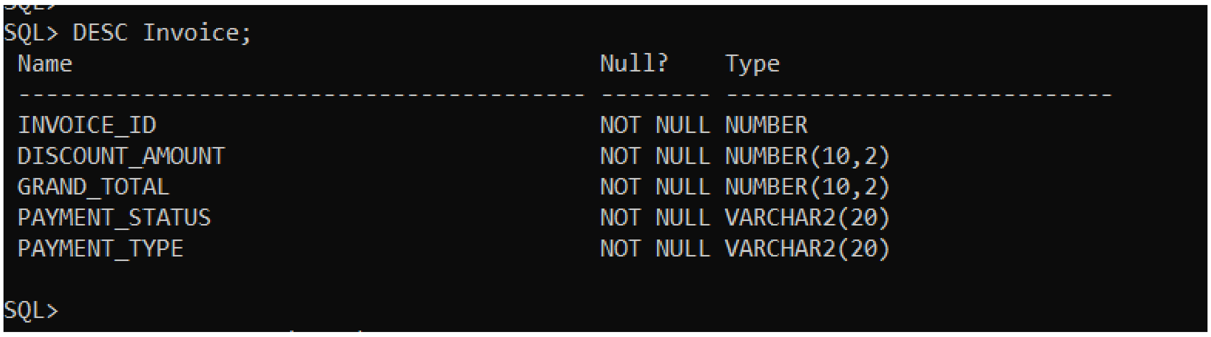
Purpose	Command Implemented
Creating an Invoice Table.	**CREATE TABLE** Invoice (Invoice_Id NUMBER PRIMARY KEY, Discount_Amount NUMBER(10, 2) NOT NULL, Grand_Total NUMBER(10, 2) NOT NULL, Payment_Status VARCHAR2(20) NOT NULL, Payment_Type VARCHAR2(20) NOT NULL);
Describing Invoice Table.	**Desc** Invoice;

Table 12: Creating and Describing Invoice Table.



```
SQL> CREATE TABLE Invoice (
  2      Invoice_Id NUMBER PRIMARY KEY,
  3      Discount_Amount NUMBER(10, 2) NOT NULL,
  4      Grand_Total NUMBER(10, 2) NOT NULL,
  5      Payment_Status VARCHAR2(20) NOT NULL,
  6      Payment_Type VARCHAR2(20) NOT NULL
  7  );

Table created.
```

Figure 19: Creating Invoice Table.



```
SQL> DESC Invoice;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 INVOICE_ID                                NOT NULL NUMBER
 DISCOUNT_AMOUNT                           NOT NULL NUMBER(10,2)
 GRAND_TOTAL                               NOT NULL NUMBER(10,2)
 PAYMENT_STATUS                            NOT NULL VARCHAR2(20)
 PAYMENT_TYPE                              NOT NULL VARCHAR2(20)

SQL>
```

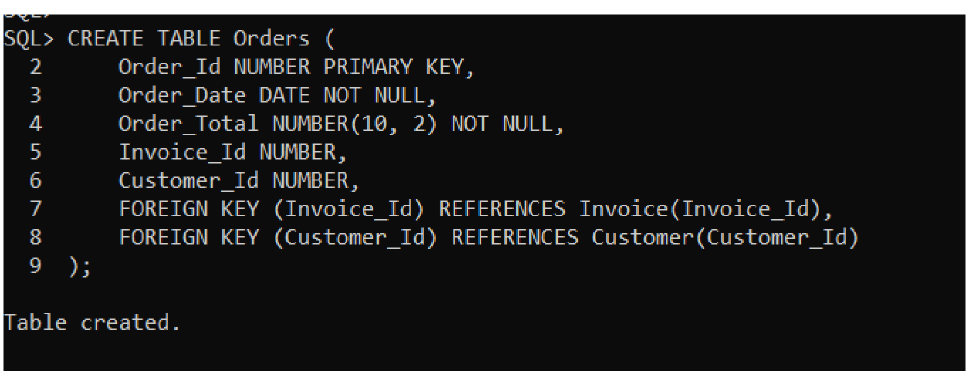
Figure 20: Describing Invoice Table.

7.4.7. Orders

Table Orders was created with five attributes in it where Order_Id is the Primary Key that uniquely defines the table and Invoice_Id and Customer_Id is the Foreign Key that connects the table.

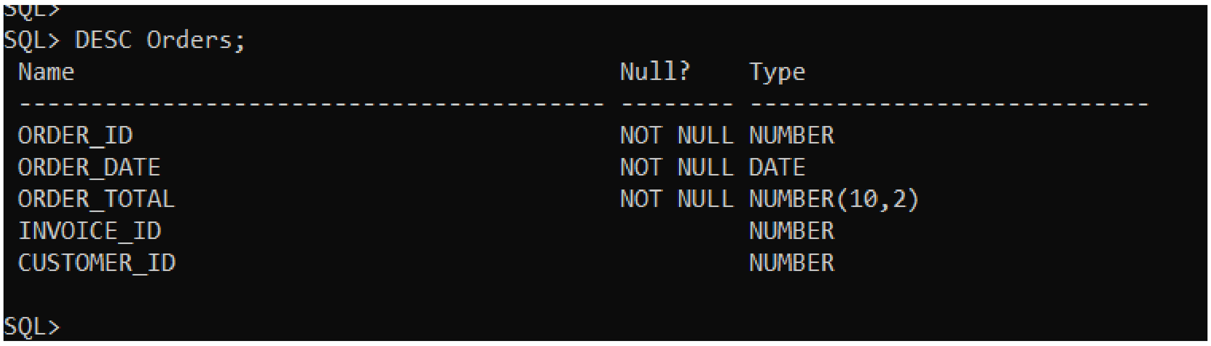
Purpose	Command Implemented
Creating an Orders Table.	**CREATE TABLE** Orders (Order_Id NUMBER PRIMARY KEY, Order_Date DATE NOT NULL, Order_Total NUMBER(10, 2) NOT NULL, Invoice_Id NUMBER, Customer_Id NUMBER, FOREIGN KEY (Invoice_Id) REFERENCES Invoice(Invoice_Id), FOREIGN KEY (Customer_Id) REFERENCES Customer(Customer_Id));
Describing Orders Table.	**Desc** Orders;

Table 13: Creating and Describing Orders Table.



```
SQL> CREATE TABLE Orders (
  2      Order_Id NUMBER PRIMARY KEY,
  3      Order_Date DATE NOT NULL,
  4      Order_Total NUMBER(10, 2) NOT NULL,
  5      Invoice_Id NUMBER,
  6      Customer_Id NUMBER,
  7      FOREIGN KEY (Invoice_Id) REFERENCES Invoice(Invoice_Id),
  8      FOREIGN KEY (Customer_Id) REFERENCES Customer(Customer_Id)
  9  );

Table created.
```

Figure 21: Creating Orders Table.



```
SQL>
SQL> DESC Orders;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORDER_ID                                  NOT NULL NUMBER
 ORDER_DATE                                NOT NULL DATE
 ORDER_TOTAL                               NOT NULL NUMBER(10,2)
 INVOICE_ID                                         NUMBER
 CUSTOMER_ID                                        NUMBER

SQL>
```

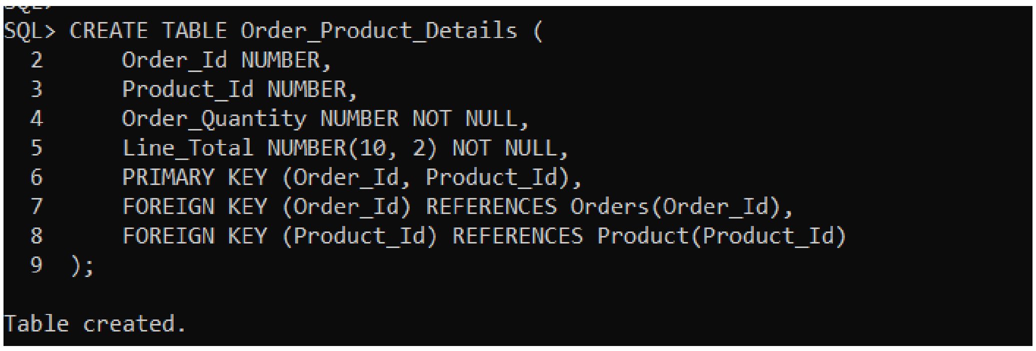
Figure 22: Describing Orders Table.

7.4.8. Order_Product_Details

Table Orders_Product_Details was created with four attributes in it where Order_Id and Product_Id are composite keys.

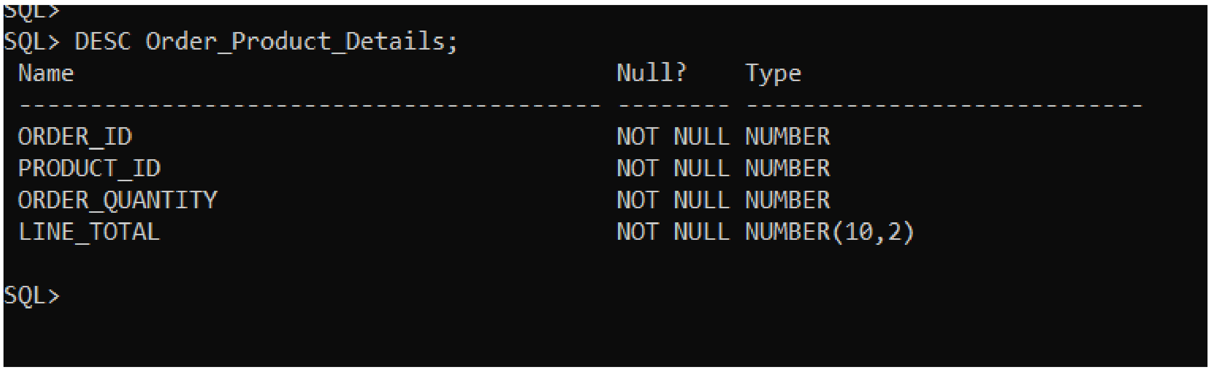
Purpose	Command Implemented
Creating an Order_Product_Details Table.	**CREATE TABLE** Order_Product_Details (Order_Id NUMBER, Product_Id NUMBER, Order_Quantity NUMBER NOT NULL, Line_Total NUMBER(10, 2) NOT NULL, PRIMARY KEY (Order_Id, Product_Id), FOREIGN KEY (Order_Id) REFERENCES Orders(Order_Id), FOREIGN KEY (Product_Id) REFERENCES Product(Product_Id));
Describing Order_Product_Details Table.	**Desc** Order_Product_Details;

Table 14:Creating and Describing Order_Product_Details Table.



```
SQL> CREATE TABLE Order_Product_Details (
  2      Order_Id NUMBER,
  3      Product_Id NUMBER,
  4      Order_Quantity NUMBER NOT NULL,
  5      Line_Total NUMBER(10, 2) NOT NULL,
  6      PRIMARY KEY (Order_Id, Product_Id),
  7      FOREIGN KEY (Order_Id) REFERENCES Orders(Order_Id),
  8      FOREIGN KEY (Product_Id) REFERENCES Product(Product_Id)
  9  );

Table created.
```

Figure 23: Creating Order_Product_Details Table.



```
SQL>
SQL> DESC Order_Product_Details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORDER_ID                                  NOT NULL NUMBER
 PRODUCT_ID                                NOT NULL NUMBER
 ORDER_QUANTITY                            NOT NULL NUMBER
 LINE_TOTAL                                NOT NULL NUMBER(10,2)

SQL>
```

Figure 24:Describing Order_Product_Details Table.

7.5. Inserting Values and Displaying Values

After creation of table, I inserted values to the tables on their respective attributes.

7.5.1. Vendor

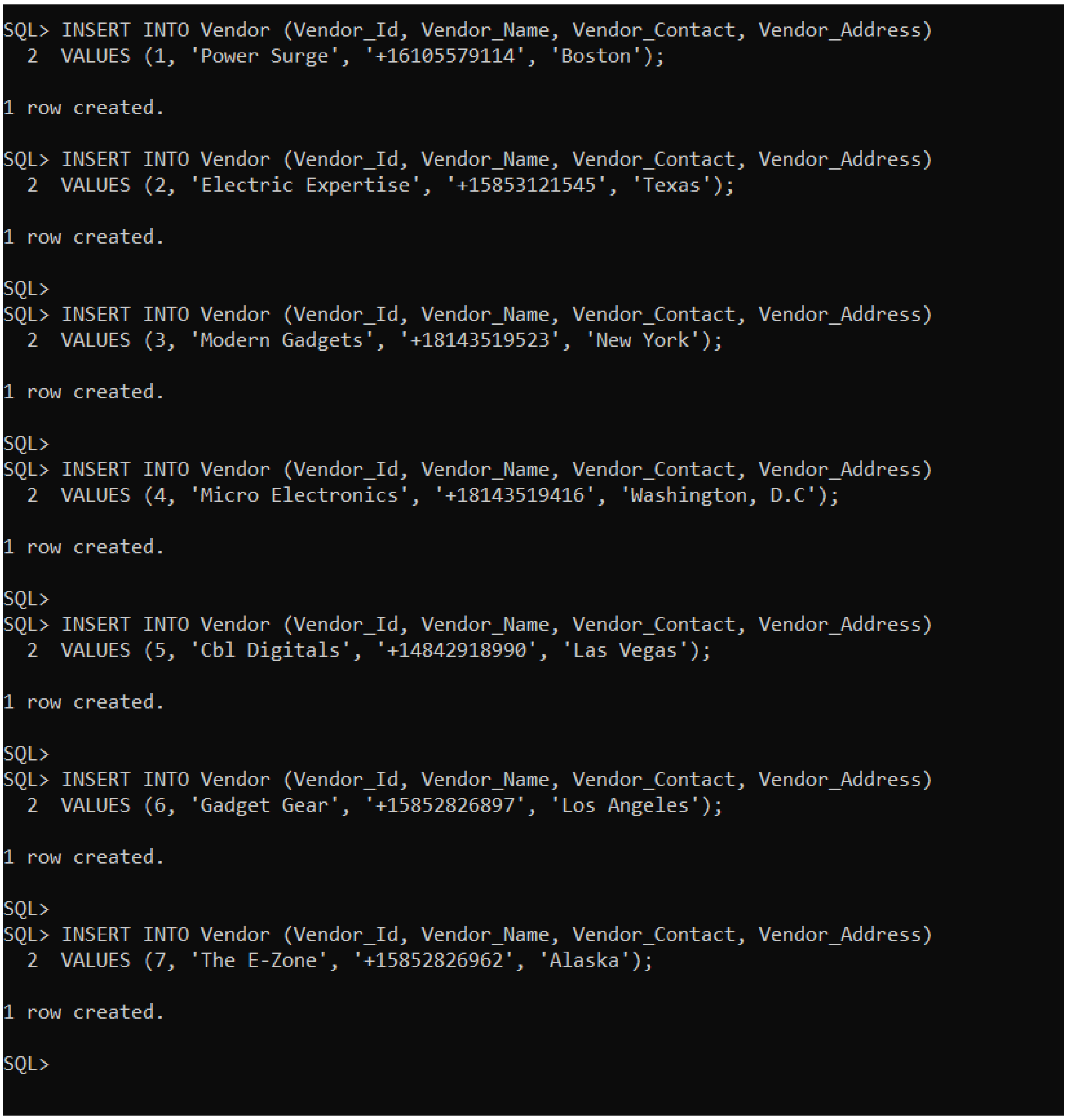
After creation of Vendor table, I inserted proper values to the table using **INSERT** statement according to it's data types and constraints.

Purpose:- Inserting Values to Vendor Table.

Command Implemented

INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address) **VALUES** (1, 'Power Surge', '+16105579114', 'Boston');
 	
INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address) **VALUES** (2, 'Electric Expertise', '+15853121545', 'Texas');
 	
INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address) **VALUES** (3, 'Modern Gadgets', '+18143519523', 'New York');
 	
INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address) **VALUES** (4, 'Micro Electronics', '+18143519416', 'Washington, D.C');
 	
INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address) **VALUES** (5, 'Cbl Digitals', '+14842918990', 'Las Vegas');
 	
INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address) **VALUES** (6, 'Gadget Gear', '+15852826897', 'Los Angeles');
 	
INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address) **VALUES** (7, 'The E-Zone', '+15852826962', 'Alaska');

Table 15: Inserting Values on Vendor Table.

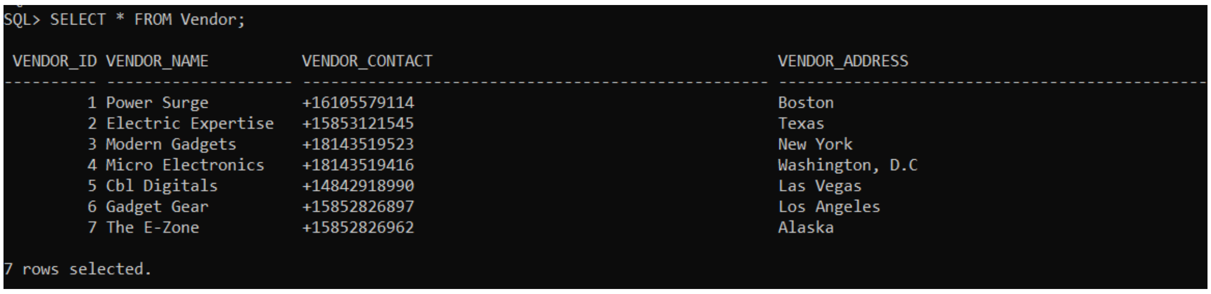


```
SQL> INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)
  2  VALUES (1, 'Power Surge', '+16105579114', 'Boston');

1 row created.

SQL> INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)
  2  VALUES (2, 'Electric Expertise', '+15853121545', 'Texas');

1 row created.

SQL>
SQL> INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)
  2  VALUES (3, 'Modern Gadgets', '+18143519523', 'New York');

1 row created.

SQL>
SQL> INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)
  2  VALUES (4, 'Micro Electronics', '+18143519416', 'Washington, D.C');

1 row created.

SQL>
SQL> INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)
  2  VALUES (5, 'Cbl Digitals', '+14842918990', 'Las Vegas');

1 row created.

SQL>
SQL> INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)
  2  VALUES (6, 'Gadget Gear', '+15852826897', 'Los Angeles');

1 row created.

SQL>
SQL> INSERT INTO Vendor (Vendor_Id, Vendor_Name, Vendor_Contact, Vendor_Address)
  2  VALUES (7, 'The E-Zone', '+15852826962', 'Alaska');

1 row created.

SQL>
```

Figure 25: : Inserting Values on Vendor Table.

Purpose :- To display all inserted values from Vendor;

Command :- SELECT * FROM Vendor;



```
SQL> SELECT * FROM Vendor;

 VENDOR_ID VENDOR_NAME          VENDOR_CONTACT                                   VENDOR_ADDRESS
---------- -------------------- ------------------------------------------------ ----------------------------------------------
         1 Power Surge          +16105579114                                     Boston
         2 Electric Expertise   +15853121545                                     Texas
         3 Modern Gadgets       +18143519523                                     New York
         4 Micro Electronics    +18143519416                                     Washington, D.C
         5 Cbl Digitals         +14842918990                                     Las Vegas
         6 Gadget Gear          +15852826897                                     Los Angeles
         7 The E-Zone           +15852826962                                     Alaska

7 rows selected.
```

Figure 26: Displaying Values from Vendor.

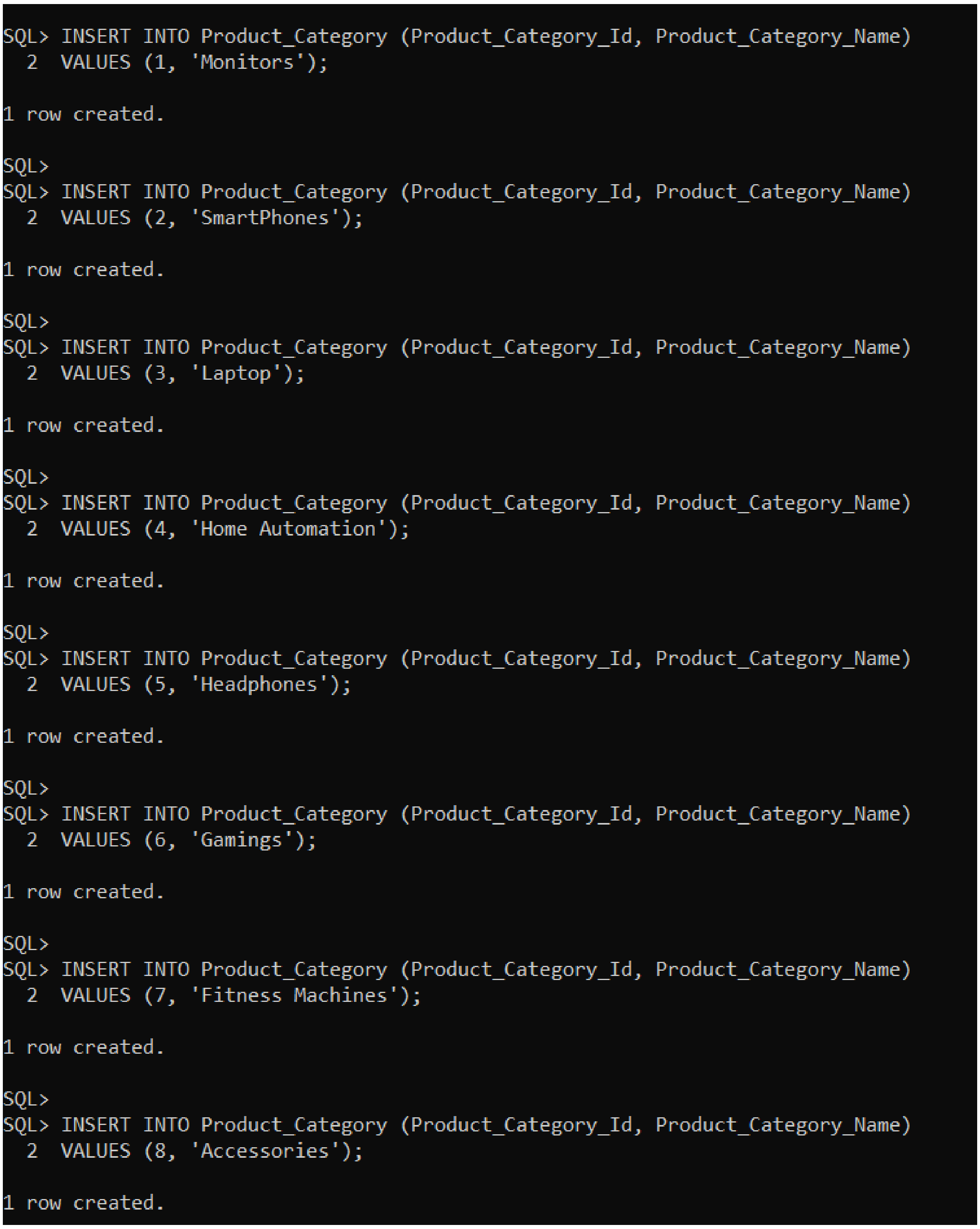
7.5.2. Product_Category

After creation of Product_Category table, I inserted proper values to the table using **INSERT** statement according to it’s data types and constraints.

Purpose :- Inserting Values to Product_Category_Table.

Command Implemented
INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name) **VALUES** (1, 'Monitors');
INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name) **VALUES** (2, 'SmartPhones');
INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name) **VALUES** (3, 'Laptop');
INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name) **VALUES** (4, 'Home Automation');
INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name) **VALUES** (5, 'Headphones');
INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name) **VALUES** (6, 'Gamings');
INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name) **VALUES** (7, 'Fitness Machines');
INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name) **VALUES** (8, 'Accessories');

Table 16: Inserting Values on Product_Category Table.

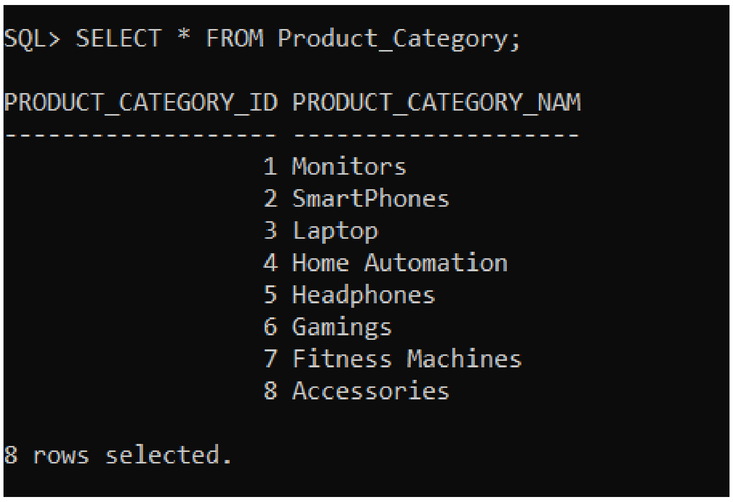


```
SQL> INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name)
  2  VALUES (1, 'Monitors');

1 row created.

SQL>
SQL> INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name)
  2  VALUES (2, 'SmartPhones');

1 row created.

SQL>
SQL> INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name)
  2  VALUES (3, 'Laptop');

1 row created.

SQL>
SQL> INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name)
  2  VALUES (4, 'Home Automation');

1 row created.

SQL>
SQL> INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name)
  2  VALUES (5, 'Headphones');

1 row created.

SQL>
SQL> INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name)
  2  VALUES (6, 'Gamings');

1 row created.

SQL>
SQL> INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name)
  2  VALUES (7, 'Fitness Machines');

1 row created.

SQL>
SQL> INSERT INTO Product_Category (Product_Category_Id, Product_Category_Name)
  2  VALUES (8, 'Accessories');

1 row created.
```

Figure 27: Inserting Values on Product_Category Table.

Purpose :- To display all inserted values from Product_Category;

Command :- SELECT * FROM Product_Category;



```
SQL> SELECT * FROM Product_Category;

PRODUCT_CATEGORY_ID PRODUCT_CATEGORY_NAM
------------------- --------------------
                  1 Monitors
                  2 SmartPhones
                  3 Laptop
                  4 Home Automation
                  5 Headphones
                  6 Gamings
                  7 Fitness Machines
                  8 Accessories

8 rows selected.
```

Figure 28: : Displaying Values from Product_Category.

7.5.3. Product

After creation of Product table, I inserted proper values to the table using **INSERT** statement according to it’s data types and constraints.

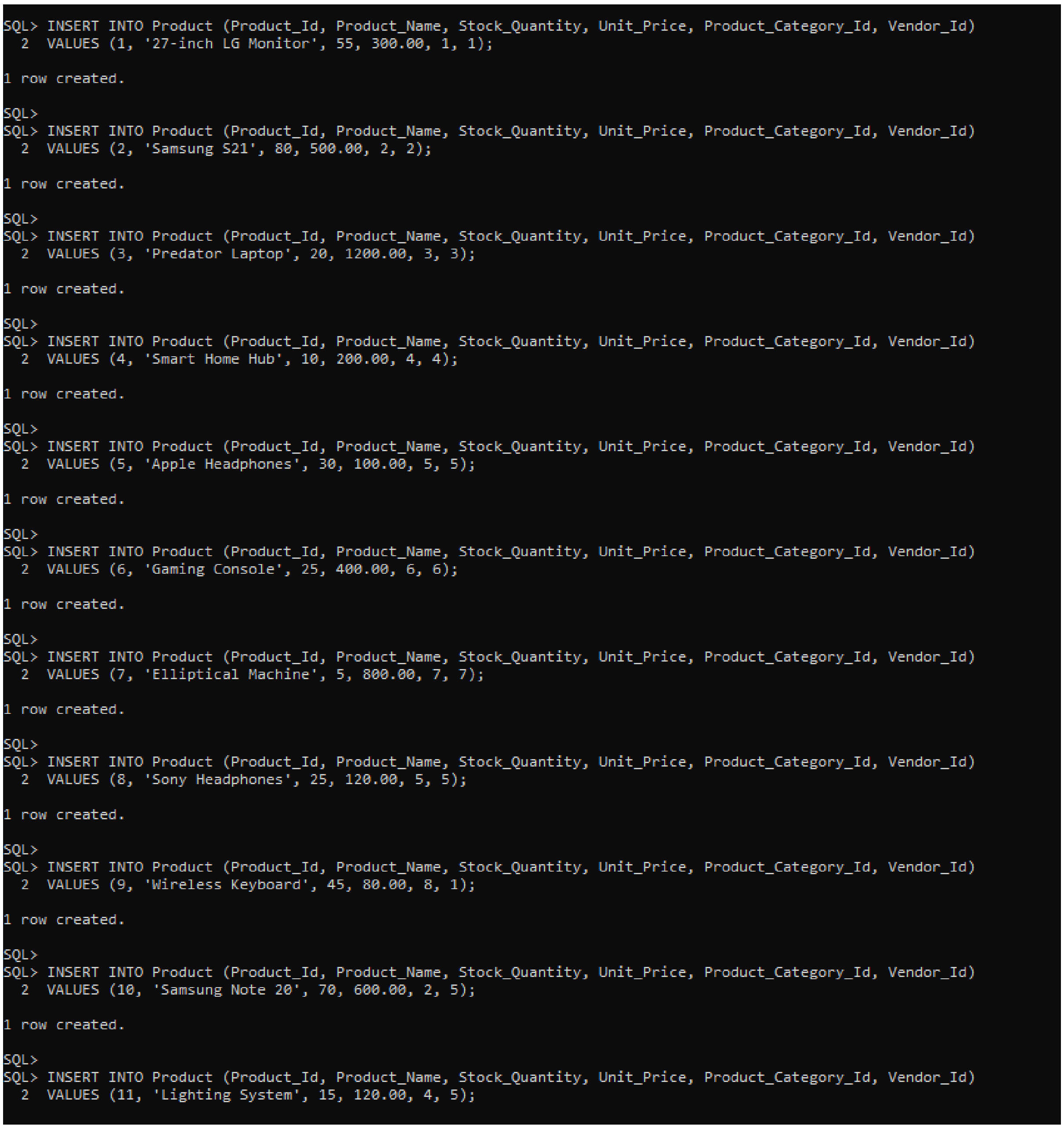
Purpose:- Inserting Values to Product Table.

Command Implemented

```
INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (1, '27-inch LG Monitor', 55, 300.00, 1, 1);

INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (2, 'Samsung S21', 80, 500.00, 2, 2);

INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (3, 'Predator Laptop', 20, 1200.00, 3, 3);

INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (4, 'Smart Home Hub', 10, 200.00, 4, 4);

INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (5, 'Apple Headphones', 30, 100.00, 5, 5);

INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (6, 'Gaming Console', 25, 400.00, 6, 6);
```

```
INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (7, 'Elliptical Machine', 5, 800.00, 7, 7);

INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (8, 'Sony Headphones', 25, 120.00, 5, 5);

INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (9, 'Wireless Keyboard', 45, 80.00, 8, 1);

INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (10, 'Samsung Note 20', 70, 600.00, 2, 5);

INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price,
Product_Category_Id, Vendor_Id)
VALUES (11, 'Lighting System', 15, 120.00, 4, 5);
```

Table 17: Inserting Values on Product Table.

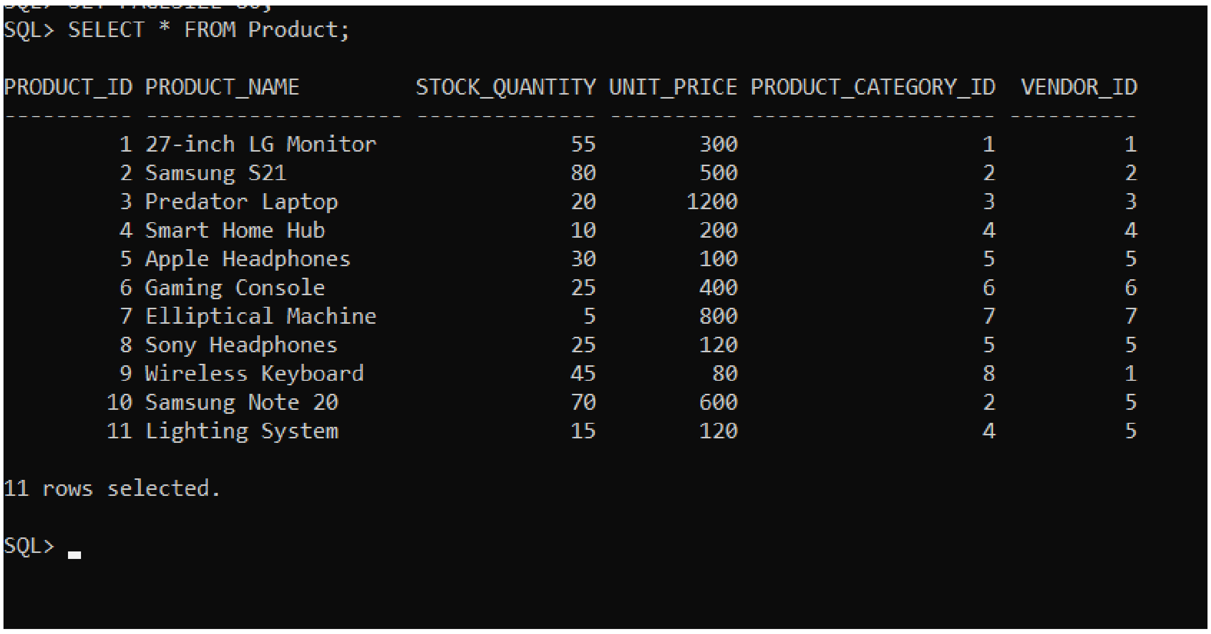


```
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (1, '27-inch LG Monitor', 55, 300.00, 1, 1);

1 row created.

SQL>
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (2, 'Samsung S21', 80, 500.00, 2, 2);

1 row created.

SQL>
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (3, 'Predator Laptop', 20, 1200.00, 3, 3);

1 row created.

SQL>
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (4, 'Smart Home Hub', 10, 200.00, 4, 4);

1 row created.

SQL>
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (5, 'Apple Headphones', 30, 100.00, 5, 5);

1 row created.

SQL>
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (6, 'Gaming Console', 25, 400.00, 6, 6);

1 row created.

SQL>
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (7, 'Elliptical Machine', 5, 800.00, 7, 7);

1 row created.

SQL>
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (8, 'Sony Headphones', 25, 120.00, 5, 5);

1 row created.

SQL>
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (9, 'Wireless Keyboard', 45, 80.00, 8, 1);

1 row created.

SQL>
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (10, 'Samsung Note 20', 70, 600.00, 2, 5);

1 row created.

SQL>
SQL> INSERT INTO Product (Product_Id, Product_Name, Stock_Quantity, Unit_Price, Product_Category_Id, Vendor_Id)
  2  VALUES (11, 'Lighting System', 15, 120.00, 4, 5);
```

Figure 29: : Inserting Values on Product Table.

Purpose :- To display all inserted values from Product;

Command :- SELECT * FROM Product;



```
SQL> SELECT * FROM Product;

PRODUCT_ID PRODUCT_NAME         STOCK_QUANTITY UNIT_PRICE PRODUCT_CATEGORY_ID  VENDOR_ID
---------- -------------------- -------------- ---------- ------------------- ----------
         1 27-inch LG Monitor               55        300                   1          1
         2 Samsung S21                      80        500                   2          2
         3 Predator Laptop                  20       1200                   3          3
         4 Smart Home Hub                   10        200                   4          4
         5 Apple Headphones                 30        100                   5          5
         6 Gaming Console                   25        400                   6          6
         7 Elliptical Machine                5        800                   7          7
         8 Sony Headphones                  25        120                   5          5
         9 Wireless Keyboard                45         80                   8          1
        10 Samsung Note 20                  70        600                   2          5
        11 Lighting System                  15        120                   4          5

11 rows selected.

SQL>
```

Figure 30: Displaying Values from Product.

7.5.4. Customer_Category

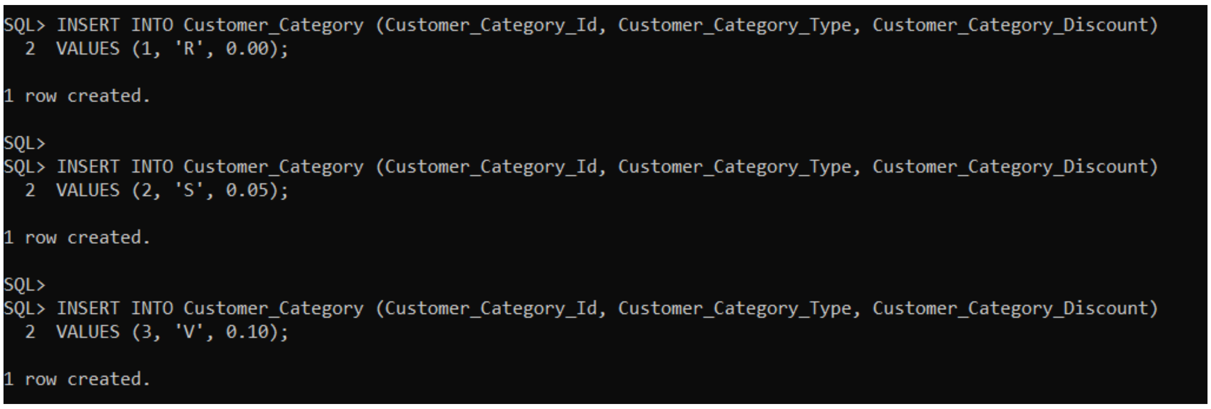
After creation of Customer_Category table, I inserted proper values to the table using **INSERT** statement according to it’s data types and constraints.

Purpose:- Inserting Values to Customer_Category Table.

Command Implemented

INSERT INTO Customer_Category (Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount) **VALUES** (1, 'R', 0.00);
 	
INSERT INTO Customer_Category (Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount) **VALUES** (2, 'S', 0.05);
 	
INSERT INTO Customer_Category (Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount) **VALUES** (3, 'V', 0.10);

Table 18: Inserting Values on Customer_CategoryTable.

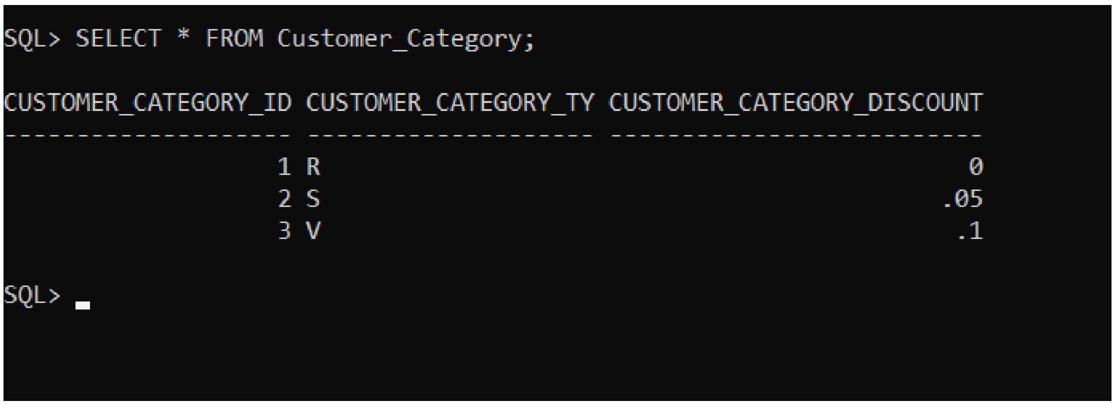


```
SQL> INSERT INTO Customer_Category (Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount)
  2  VALUES (1, 'R', 0.00);

1 row created.

SQL>
SQL> INSERT INTO Customer_Category (Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount)
  2  VALUES (2, 'S', 0.05);

1 row created.

SQL>
SQL> INSERT INTO Customer_Category (Customer_Category_Id, Customer_Category_Type, Customer_Category_Discount)
  2  VALUES (3, 'V', 0.10);

1 row created.
```

Figure 31: : Inserting Values on Customer_Category Table.

Purpose :- To display all inserted values from Customer_Category;

Command :- SELECT * FROM Customer_Category;



```
SQL> SELECT * FROM Customer_Category;

CUSTOMER_CATEGORY_ID CUSTOMER_CATEGORY_TY CUSTOMER_CATEGORY_DISCOUNT
-------------------- -------------------- --------------------------
                   1 R                                             0
                   2 S                                           .05
                   3 V                                            .1

SQL>
```

Figure 32: Displaying Values from Customer_Category.

7.5.5. Customer

After creation of Customer table, I inserted proper values into the table using **INSERT** statement according to it's data types and constraints.

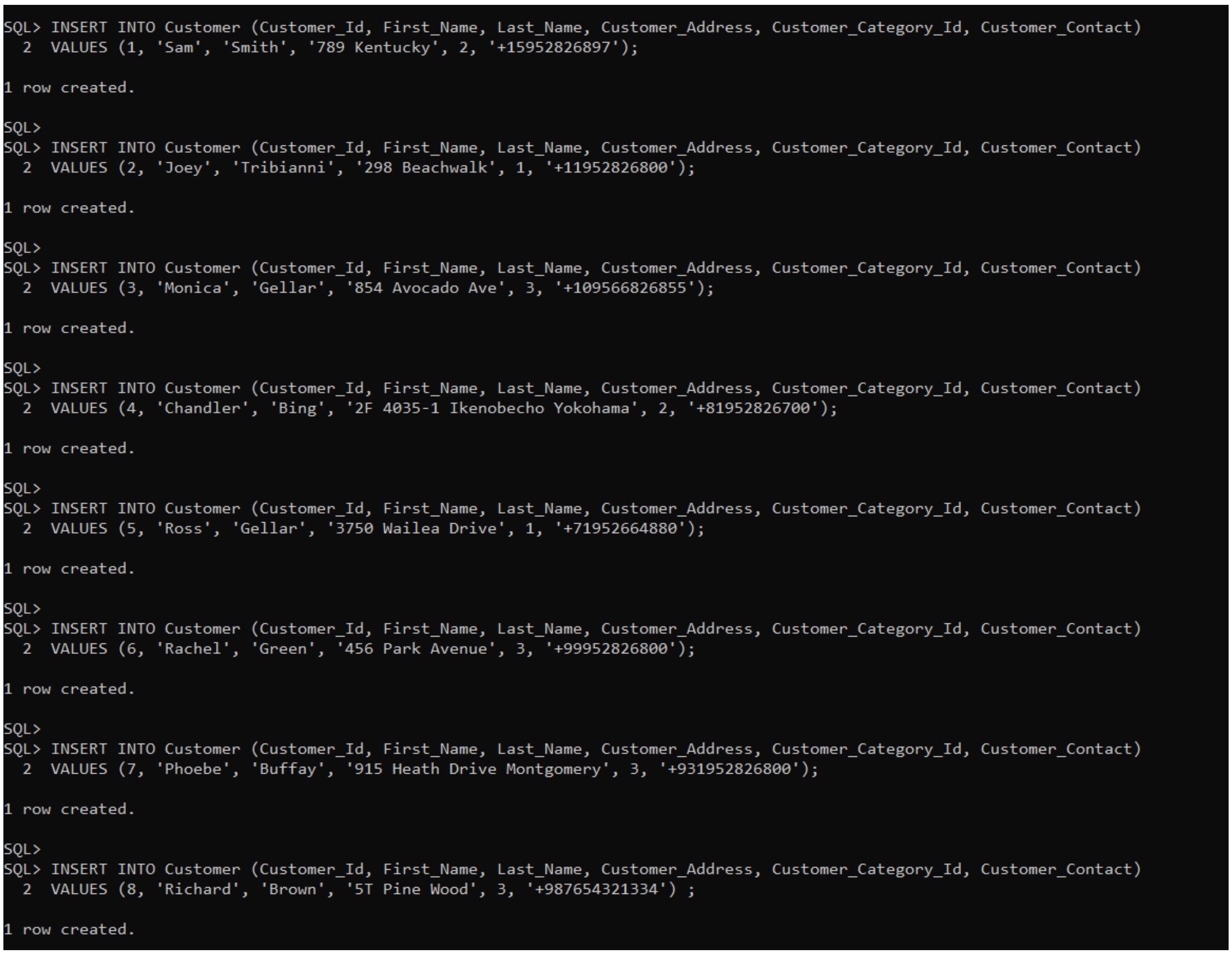
Purpose:- Inserting Values to Customer Table.

Command Implemented

INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact) **VALUES** (1, 'Sam', 'Smith', '789 Kentucky', 2, '+15952826897');
 	
 INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact) **VALUES** (2, 'Joey', 'Tribianni', '298 Beachwalk', 1, '+11952826800');
 	
 INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact) **VALUES** (3, 'Monica', 'Gellar', '854 Avocado Ave', 3, '+109566826855');
 	
 INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact) **VALUES** (4, 'Chandler', 'Bing', '2F 4035-1 Ikenobecho Yokohama', 2, '+81952826700');
 	
 INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact) **VALUES** (5, 'Ross', 'Gellar', '3750 Wailea Drive', 1, '+71952664880');
 	
 INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact) **VALUES** (6, 'Rachel', 'Green', '456 Park Avenue', 3, '+99952826800');

INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact)
VALUES (7, 'Phoebe', 'Buffay', '915 Heath Drive Montgomery', 3, '+931952826800');
INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact)
VALUES (8, 'Richard', 'Brown', '5T Pine Wood', 3, '+987654321334') ;

Table 19: : Inserting Values on CustomerTable.

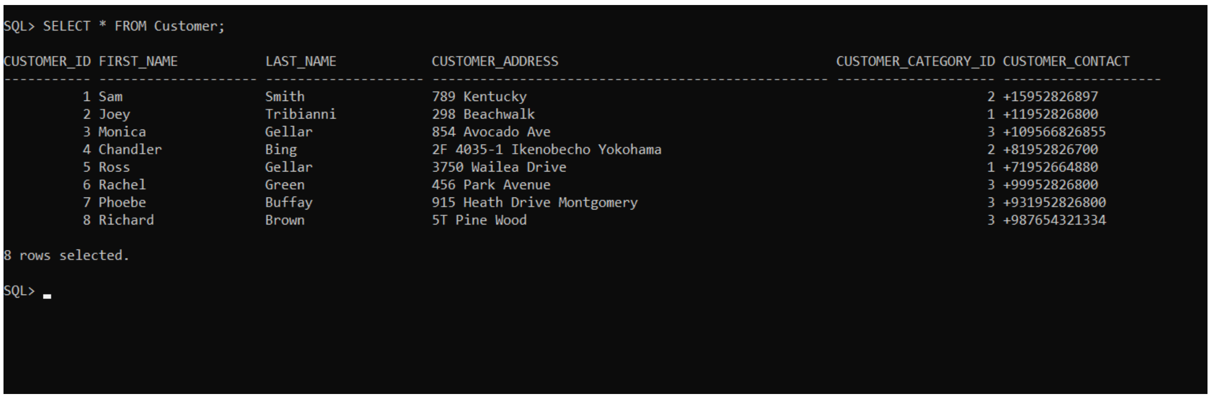


```
SQL> INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact)
  2  VALUES (1, 'Sam', 'Smith', '789 Kentucky', 2, '+15952826897');

1 row created.

SQL>
SQL> INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact)
  2  VALUES (2, 'Joey', 'Tribianni', '298 Beachwalk', 1, '+11952826800');

1 row created.

SQL>
SQL> INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact)
  2  VALUES (3, 'Monica', 'Gellar', '854 Avocado Ave', 3, '+109566826855');

1 row created.

SQL>
SQL> INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact)
  2  VALUES (4, 'Chandler', 'Bing', '2F 4035-1 Ikenobecho Yokohama', 2, '+81952826700');

1 row created.

SQL>
SQL> INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact)
  2  VALUES (5, 'Ross', 'Gellar', '3750 Wailea Drive', 1, '+71952664880');

1 row created.

SQL>
SQL> INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact)
  2  VALUES (6, 'Rachel', 'Green', '456 Park Avenue', 3, '+99952826800');

1 row created.

SQL>
SQL> INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact)
  2  VALUES (7, 'Phoebe', 'Buffay', '915 Heath Drive Montgomery', 3, '+931952826800');

1 row created.

SQL>
SQL> INSERT INTO Customer (Customer_Id, First_Name, Last_Name, Customer_Address, Customer_Category_Id, Customer_Contact)
  2  VALUES (8, 'Richard', 'Brown', '5T Pine Wood', 3, '+987654321334') ;

1 row created.
```

Figure 33: : Inserting Values on Customer Table.

Purpose :- To display all inserted values from Customer;

Command :- SELECT * FROM Customer;



```
SQL> SELECT * FROM Customer;

CUSTOMER_ID FIRST_NAME           LAST_NAME            CUSTOMER_ADDRESS                                   CUSTOMER_CATEGORY_ID CUSTOMER_CONTACT
----------- -------------------- -------------------- -------------------------------------------------- -------------------- --------------------
          1 Sam                  Smith                789 Kentucky                                                          2 +15952826897
          2 Joey                 Tribianni            298 Beachwalk                                                         1 +11952826800
          3 Monica               Gellar               854 Avocado Ave                                                       3 +109566826855
          4 Chandler             Bing                 2F 4035-1 Ikenobecho Yokohama                                         2 +81952826700
          5 Ross                 Gellar               3750 Wailea Drive                                                     1 +71952664880
          6 Rachel               Green                456 Park Avenue                                                       3 +99952826800
          7 Phoebe               Buffay               915 Heath Drive Montgomery                                            3 +931952826800
          8 Richard              Brown                5T Pine Wood                                                          3 +987654321334

8 rows selected.

SQL>
```

Figure 34: Displaying Values from Customer.

7.5.6. Invoice

After creation of Invoice table, I inserted proper values to the table using **INSERT** statement according to it's data types and constraints.

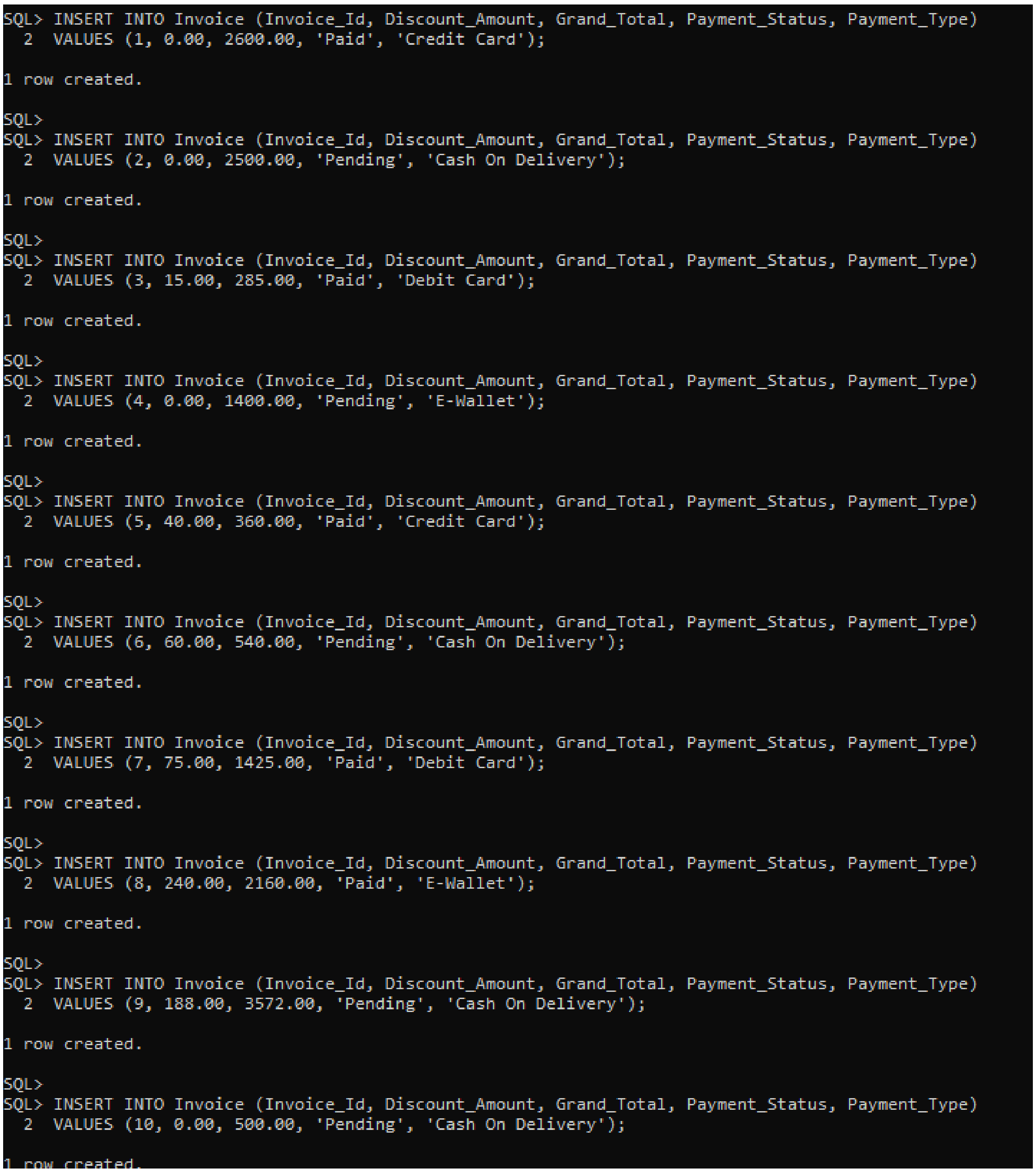
Purpose:- Inserting Values to Invoice Table.

Command Implemented

INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
VALUES (1, 0.00, 2600.00, 'Paid', 'Credit Card');
 	
INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
VALUES (2, 0.00, 2500.00, 'Pending', 'Cash On Delivery');
 	
INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
VALUES (3, 15.00, 285.00, 'Paid', 'Debit Card');
 	
INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
VALUES (4, 0.00, 1400.00, 'Pending', 'E-Wallet');
 	
INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
VALUES (5, 40.00, 360.00, 'Paid', 'Credit Card');
 	
INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
VALUES (6, 60.00, 540.00, 'Pending', 'Cash On Delivery');

```
INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status,
Payment_Type)
VALUES (7, 75.00, 1425.00, 'Paid', 'Debit Card');

INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status,
Payment_Type)
VALUES (8, 240.00, 2160.00, 'Paid', 'E-Wallet');

INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status,
Payment_Type)
VALUES (9, 188.00, 3572.00, 'Pending', 'Cash On Delivery');

INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status,
Payment_Type)
VALUES (10, 0.00, 500.00, 'Pending', 'Cash On Delivery');
```

Table 20: Inserting Values on Invoice Table.

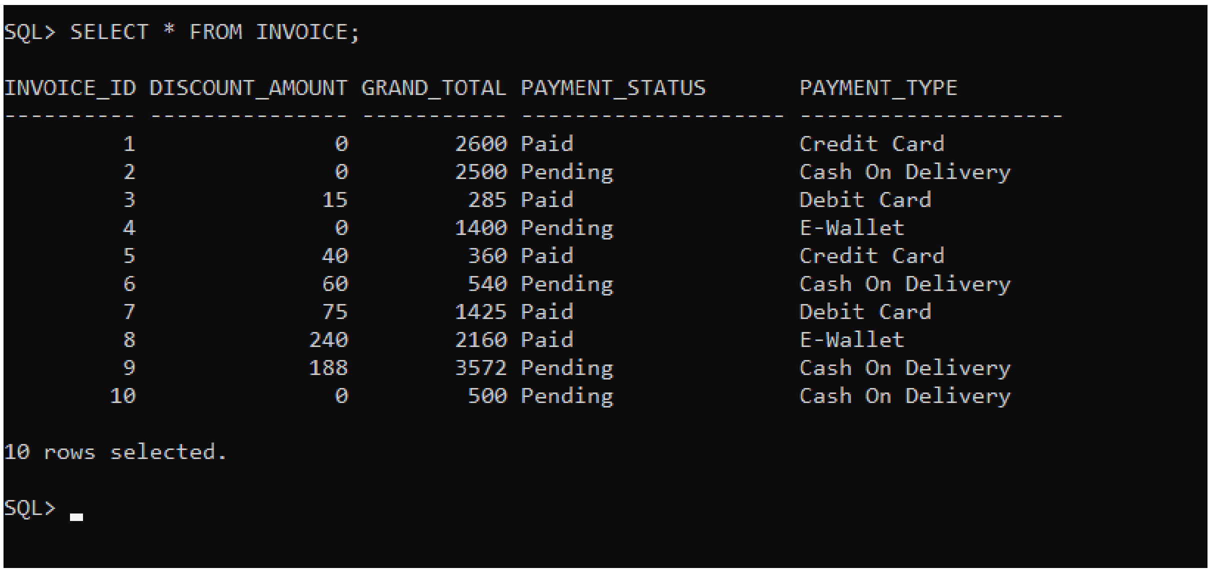


```
SQL> INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
  2  VALUES (1, 0.00, 2600.00, 'Paid', 'Credit Card');

1 row created.

SQL>
SQL> INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
  2  VALUES (2, 0.00, 2500.00, 'Pending', 'Cash On Delivery');

1 row created.

SQL>
SQL> INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
  2  VALUES (3, 15.00, 285.00, 'Paid', 'Debit Card');

1 row created.

SQL>
SQL> INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
  2  VALUES (4, 0.00, 1400.00, 'Pending', 'E-Wallet');

1 row created.

SQL>
SQL> INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
  2  VALUES (5, 40.00, 360.00, 'Paid', 'Credit Card');

1 row created.

SQL>
SQL> INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
  2  VALUES (6, 60.00, 540.00, 'Pending', 'Cash On Delivery');

1 row created.

SQL>
SQL> INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
  2  VALUES (7, 75.00, 1425.00, 'Paid', 'Debit Card');

1 row created.

SQL>
SQL> INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
  2  VALUES (8, 240.00, 2160.00, 'Paid', 'E-Wallet');

1 row created.

SQL>
SQL> INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
  2  VALUES (9, 188.00, 3572.00, 'Pending', 'Cash On Delivery');

1 row created.

SQL>
SQL> INSERT INTO Invoice (Invoice_Id, Discount_Amount, Grand_Total, Payment_Status, Payment_Type)
  2  VALUES (10, 0.00, 500.00, 'Pending', 'Cash On Delivery');

1 row created.
```

Figure 35: : Inserting Values on Invoice Table.

Purpose :- To display all inserted values from Invoice;

Command :- SELECT * FROM Invoice;



```
SQL> SELECT * FROM INVOICE;

INVOICE_ID DISCOUNT_AMOUNT GRAND_TOTAL PAYMENT_STATUS       PAYMENT_TYPE
---------- --------------- ----------- -------------------- --------------------
         1               0        2600 Paid                 Credit Card
         2               0        2500 Pending              Cash On Delivery
         3              15         285 Paid                 Debit Card
         4               0        1400 Pending              E-Wallet
         5              40         360 Paid                 Credit Card
         6              60         540 Pending              Cash On Delivery
         7              75        1425 Paid                 Debit Card
         8             240        2160 Paid                 E-Wallet
         9             188        3572 Pending              Cash On Delivery
        10               0         500 Pending              Cash On Delivery

10 rows selected.

SQL>
```

Figure 36: Displaying Values from Invoice.

7.5.7. Orders

After creation of Orders table, I inserted proper values to the table using **INSERT** statement according to it’s data types and constraints.

Purpose:- Inserting Values to Orders Table.

Command Implemented
INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id) **VALUES** (1, TO_DATE('2023-05-05', 'YYYY-MM-DD'), 2600.00, 1, 5);
INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id) **VALUES** (2, TO_DATE('2023-08-10', 'YYYY-MM-DD'), 2500.00, 2, 2);
INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id) **VALUES** (3, TO_DATE('2023-05-15', 'YYYY-MM-DD'), 300.00, 3, 4);
INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id) **VALUES** (4, TO_DATE('2023-07-20', 'YYYY-MM-DD'), 1400.00, 4, 5);
INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id) **VALUES** (5, TO_DATE('2023-05-25', 'YYYY-MM-DD'), 400.00, 5, 7);
INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id) **VALUES** (6, TO_DATE('2023-05-28', 'YYYY-MM-DD'), 600.00, 6, 6);
INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id) **VALUES** (7, TO_DATE('2023-05-30', 'YYYY-MM-DD'), 1500.00, 7, 1);
INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id) **VALUES** (8, TO_DATE('2023-06-05', 'YYYY-MM-DD'), 2400.00, 8, 3);

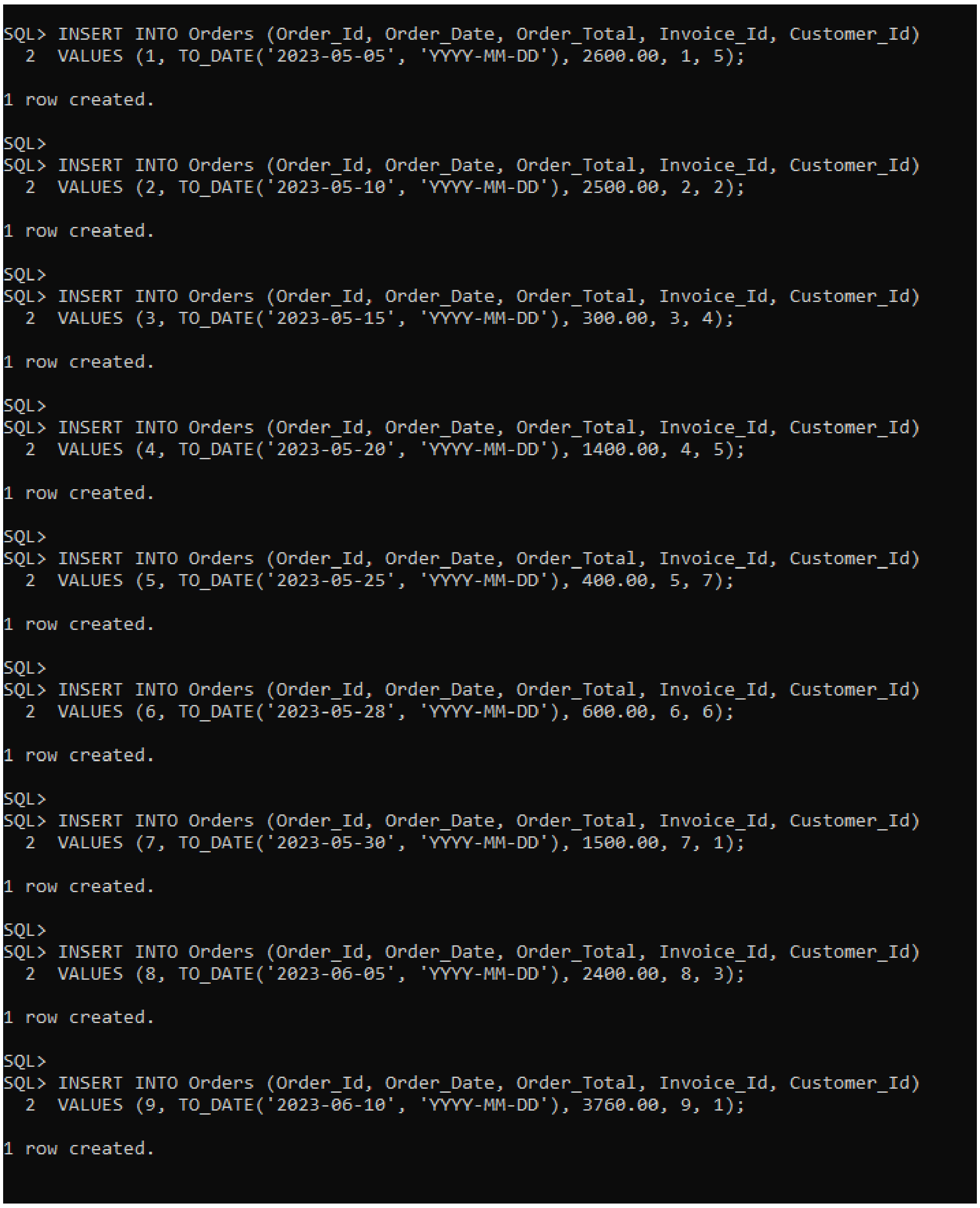
INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)

VALUES (9, TO_DATE('2023-06-10', 'YYYY-MM-DD'), 3760.00, 9, 1);

INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)

VALUES (10, TO_DATE('2023-08-15', 'YYYY-MM-DD'), 500.00, 10, 2);

Table 21: Inserting Values on Orders Table.

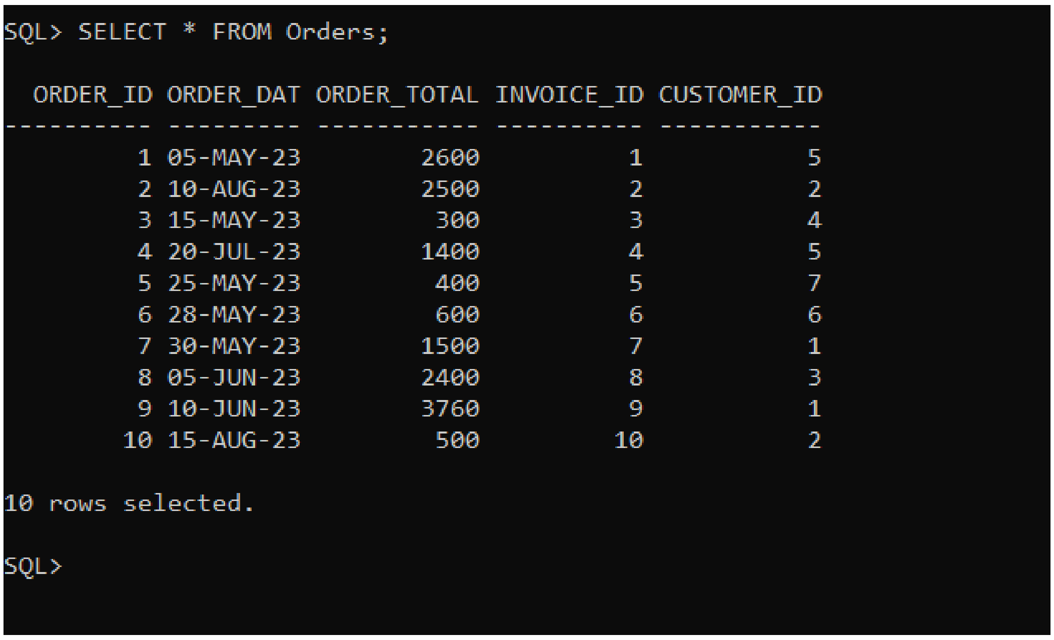


```
SQL> INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)
  2  VALUES (1, TO_DATE('2023-05-05', 'YYYY-MM-DD'), 2600.00, 1, 5);

1 row created.

SQL>
SQL> INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)
  2  VALUES (2, TO_DATE('2023-05-10', 'YYYY-MM-DD'), 2500.00, 2, 2);

1 row created.

SQL>
SQL> INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)
  2  VALUES (3, TO_DATE('2023-05-15', 'YYYY-MM-DD'), 300.00, 3, 4);

1 row created.

SQL>
SQL> INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)
  2  VALUES (4, TO_DATE('2023-05-20', 'YYYY-MM-DD'), 1400.00, 4, 5);

1 row created.

SQL>
SQL> INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)
  2  VALUES (5, TO_DATE('2023-05-25', 'YYYY-MM-DD'), 400.00, 5, 7);

1 row created.

SQL>
SQL> INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)
  2  VALUES (6, TO_DATE('2023-05-28', 'YYYY-MM-DD'), 600.00, 6, 6);

1 row created.

SQL>
SQL> INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)
  2  VALUES (7, TO_DATE('2023-05-30', 'YYYY-MM-DD'), 1500.00, 7, 1);

1 row created.

SQL>
SQL> INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)
  2  VALUES (8, TO_DATE('2023-06-05', 'YYYY-MM-DD'), 2400.00, 8, 3);

1 row created.

SQL>
SQL> INSERT INTO Orders (Order_Id, Order_Date, Order_Total, Invoice_Id, Customer_Id)
  2  VALUES (9, TO_DATE('2023-06-10', 'YYYY-MM-DD'), 3760.00, 9, 1);

1 row created.
```

Table 22: Inserting Values on Orders Table.

Purpose :- To display all inserted values from Orders;

Command :- SELECT * FROM Orders;



```
SQL> SELECT * FROM Orders;

  ORDER_ID ORDER_DAT ORDER_TOTAL INVOICE_ID CUSTOMER_ID
---------- --------- ----------- ---------- -----------
         1 05-MAY-23        2600          1           5
         2 10-AUG-23        2500          2           2
         3 15-MAY-23         300          3           4
         4 20-JUL-23        1400          4           5
         5 25-MAY-23         400          5           7
         6 28-MAY-23         600          6           6
         7 30-MAY-23        1500          7           1
         8 05-JUN-23        2400          8           3
         9 10-JUN-23        3760          9           1
        10 15-AUG-23         500         10           2

10 rows selected.

SQL>
```

Figure 37: Displaying Values from Orders.

7.5.8. Order_Product_Details

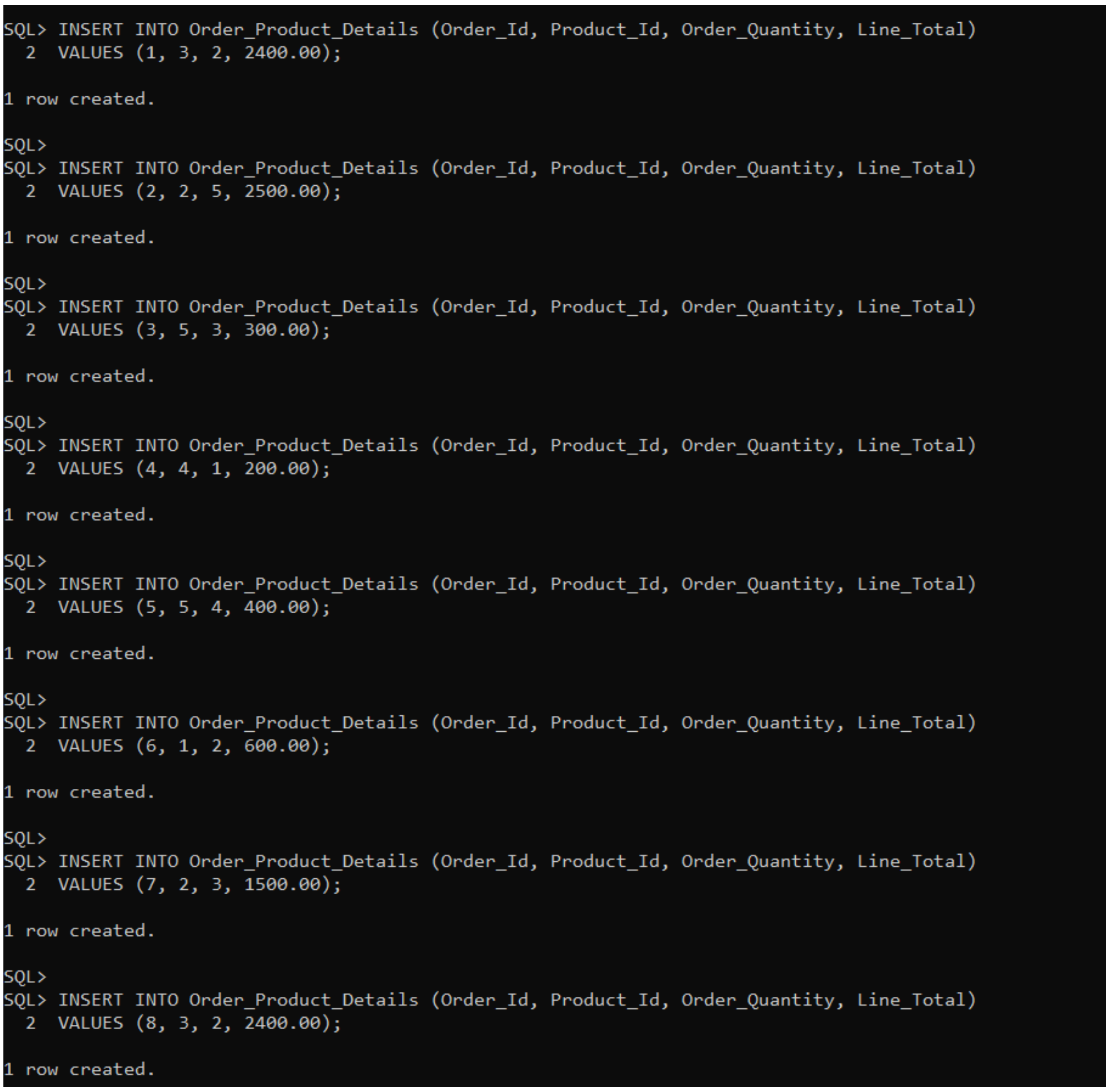
After creation of Order_Product_Details table, I inserted proper values to the table using **INSERT** statement according to it's data types and constraints.

Purpose:- Inserting Values to Order_Product_Details Table.

Command Implemented
INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total) **VALUES** (1, 3, 2, 2400.00);
INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total) **VALUES** (2, 2, 5, 2500.00);
INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total) **VALUES** (3, 5, 3, 300.00);
INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total) **VALUES** (4, 4, 1, 200.00);
INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total) **VALUES** (5, 5, 4, 400.00);
INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total) **VALUES** (6, 1, 2, 600.00);
INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total) **VALUES** (7, 2, 3, 1500.00);
INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total) **VALUES** (8, 3, 2, 2400.00);

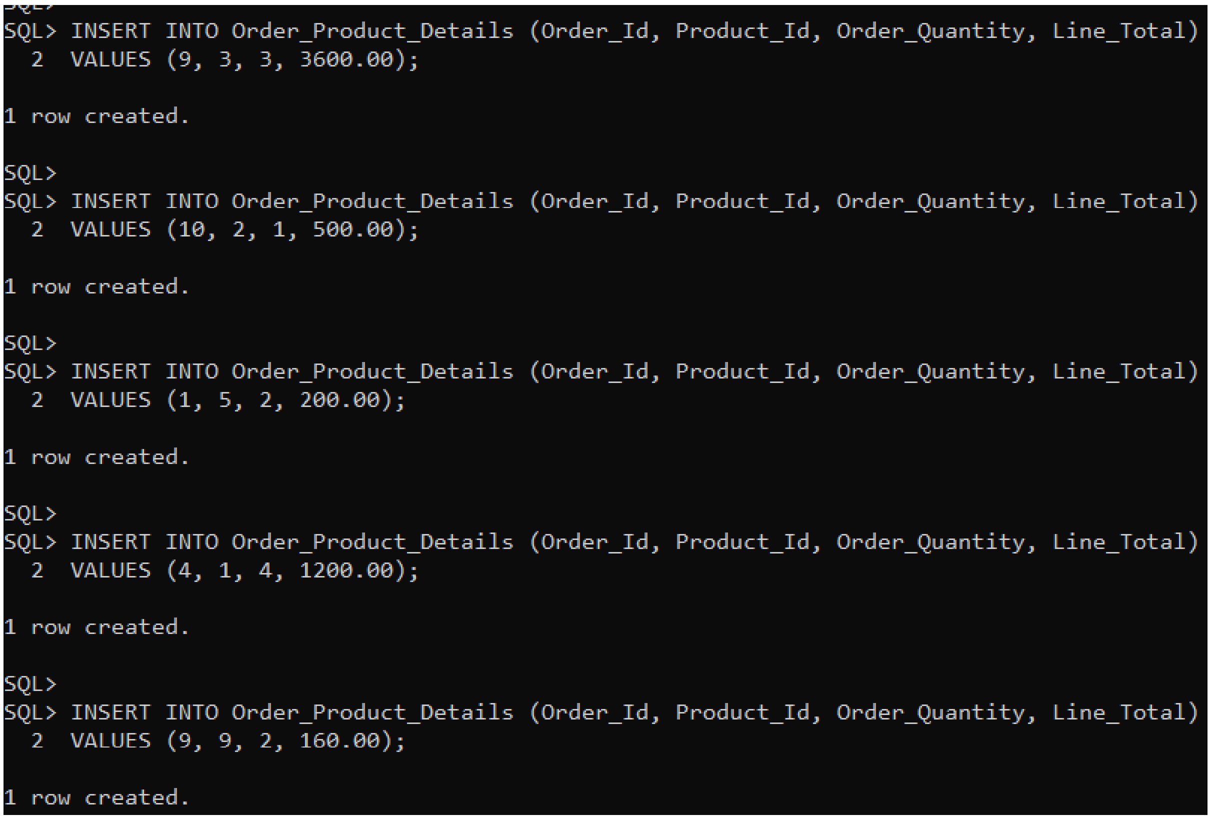
```
INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
VALUES (9, 3, 3, 3600.00);

INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
VALUES (10, 2, 1, 500.00);

INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
VALUES (1, 5, 2, 200.00);

INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
VALUES (4, 1, 4, 1200.00);

INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
VALUES (9, 9, 2, 160.00);
```

Table 23: Inserting Values on Order_Product_Details Table.



```
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (1, 3, 2, 2400.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (2, 2, 5, 2500.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (3, 5, 3, 300.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (4, 4, 1, 200.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (5, 5, 4, 400.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (6, 1, 2, 600.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (7, 2, 3, 1500.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (8, 3, 2, 2400.00);

1 row created.
```

Figure 38: Inserting Values on Order_Product_Details Table 1.0.

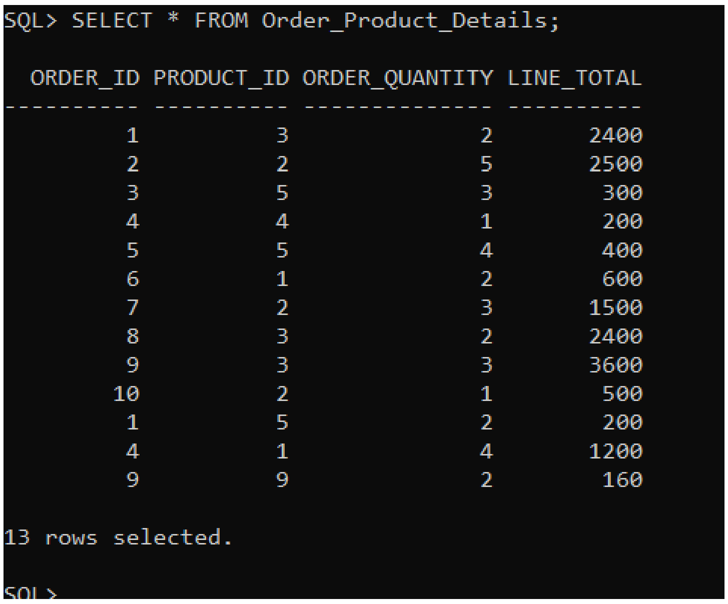


```
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (9, 3, 3, 3600.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (10, 2, 1, 500.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (1, 5, 2, 200.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (4, 1, 4, 1200.00);

1 row created.

SQL>
SQL> INSERT INTO Order_Product_Details (Order_Id, Product_Id, Order_Quantity, Line_Total)
  2  VALUES (9, 9, 2, 160.00);

1 row created.
```

Figure 39: Inserting Values on Order_Product_Details Table. 2.0

Purpose :- To display all inserted values from Order_Product_Details;

Command :- SELECT * FROM Order_Product_Details;



```
SQL> SELECT * FROM Order_Product_Details;

  ORDER_ID PRODUCT_ID ORDER_QUANTITY LINE_TOTAL
---------- ---------- -------------- ----------
         1          3              2       2400
         2          2              5       2500
         3          5              3        300
         4          4              1        200
         5          5              4        400
         6          1              2        600
         7          2              3       1500
         8          3              2       2400
         9          3              3       3600
        10          2              1        500
         1          5              2        200
         4          1              4       1200
         9          9              2        160

13 rows selected.

SQL>
```

Figure 40: Displaying Values from Order_Product_Details.

8. Database Querying

8.1. Informational Queries

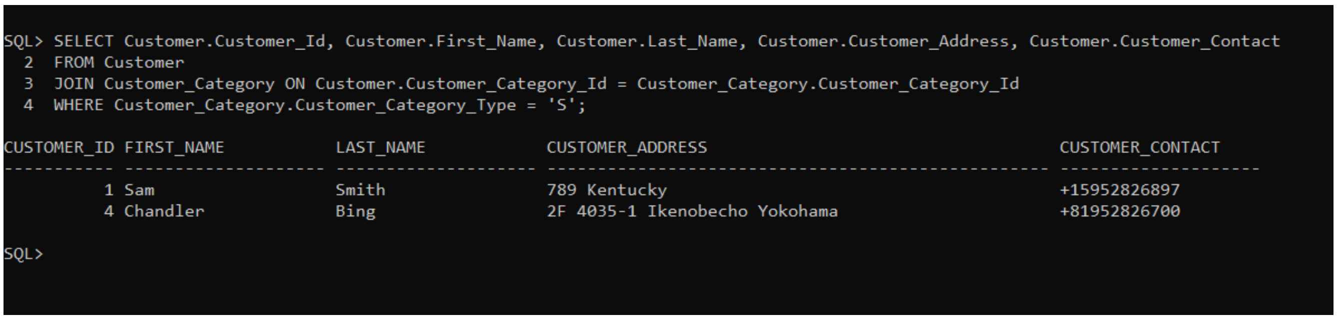
1.

Purpose: To List all the customers that are also staff of the company

Query Implemented

SELECT Customer.Customer_Id, Customer.First_Name, Customer.Last_Name, Customer.Customer_Address, Customer.Customer_Contact
FROM Customer
JOIN Customer_Category ON Customer.Customer_Category_Id = Customer_Category.Customer_Category_Id
WHERE Customer_Category.Customer_Category_Type = 'S';

Table 24: Listing all the customers that are also staff of the company.



```
SQL> SELECT Customer.Customer_Id, Customer.First_Name, Customer.Last_Name, Customer.Customer_Address, Customer.Customer_Contact
  2  FROM Customer
  3  JOIN Customer_Category ON Customer.Customer_Category_Id = Customer_Category.Customer_Category_Id
  4  WHERE Customer_Category.Customer_Category_Type = 'S';

CUSTOMER_ID FIRST_NAME           LAST_NAME            CUSTOMER_ADDRESS                                   CUSTOMER_CONTACT
----------- -------------------- -------------------- -------------------------------------------------- --------------------
          1 Sam                  Smith                789 Kentucky                                       +15952826897
          4 Chandler             Bing                 2F 4035-1 Ikenobecho Yokohama                      +81952826700

SQL>
```

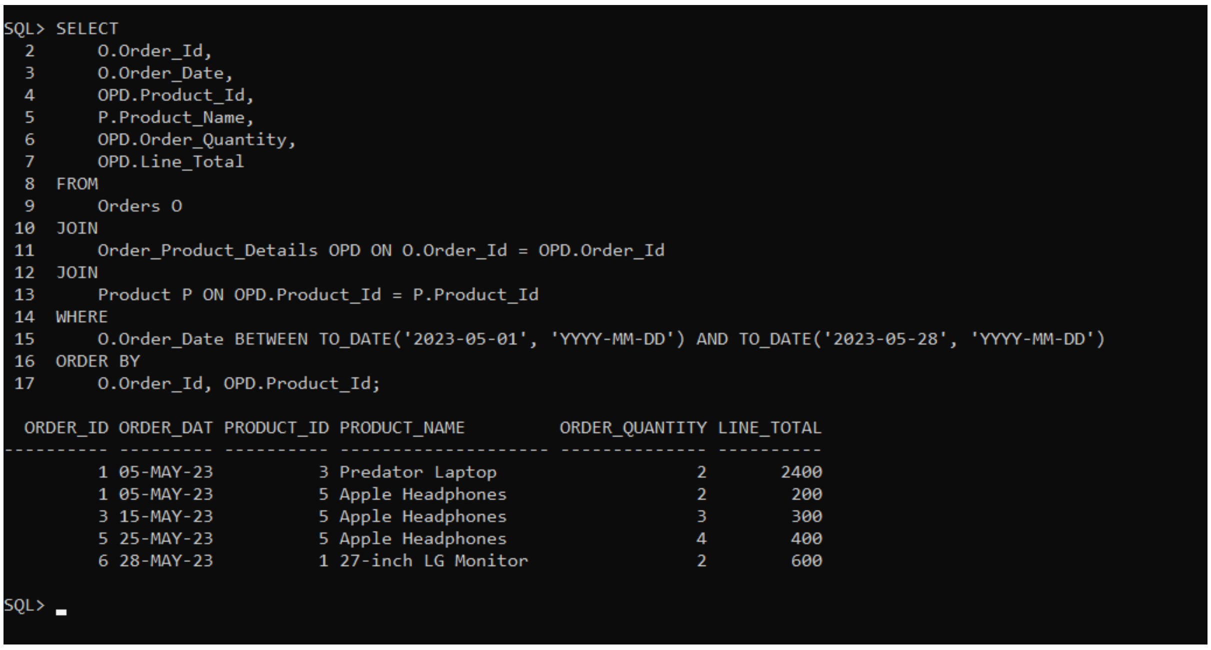
Figure 41: Listing all the customers that are also staff of the company.

2.

Purpose: To list all the orders made for any particular product between the dates 01-05-2023 till 28- 05-2023.

Query Implemented
SELECT O.Order_Id, O.Order_Date, OPD.Product_Id, P.Product_Name, OPD.Order_Quantity, OPD.Line_Total **FROM** Orders O **JOIN** Order_Product_Details OPD ON O.Order_Id = OPD.Order_Id **JOIN** Product P ON OPD.Product_Id = P.Product_Id **WHERE** O.Order_Date BETWEEN TO_DATE('2023-05-01', 'YYYY-MM-DD') AND TO_DATE('2023-05-28', 'YYYY-MM-DD') **ORDER BY** O.Order_Id, OPD.Product_Id;

Table 25: listing all the orders made for any particular product between the dates 01-05-2023 till 28- 05-2023



```
SQL> SELECT
        O.Order_Id,
        O.Order_Date,
        OPD.Product_Id,
        P.Product_Name,
        OPD.Order_Quantity,
        OPD.Line_Total
     FROM
        Orders O
     JOIN
        Order_Product_Details OPD ON O.Order_Id = OPD.Order_Id
     JOIN
        Product P ON OPD.Product_Id = P.Product_Id
     WHERE
        O.Order_Date BETWEEN TO_DATE('2023-05-01', 'YYYY-MM-DD') AND TO_DATE('2023-05-28', 'YYYY-MM-DD')
     ORDER BY
        O.Order_Id, OPD.Product_Id;

  ORDER_ID ORDER_DAT PRODUCT_ID PRODUCT_NAME         ORDER_QUANTITY LINE_TOTAL
---------- --------- ---------- -------------------- -------------- ----------
         1 05-MAY-23          3 Predator Laptop                   2       2400
         1 05-MAY-23          5 Apple Headphones                  2        200
         3 15-MAY-23          5 Apple Headphones                  3        300
         5 25-MAY-23          5 Apple Headphones                  4        400
         6 28-MAY-23          1 27-inch LG Monitor                2        600

SQL>
```

Figure 42: listing all the orders made for any particular product between the dates 01-05-2023 till 28- 05-2023

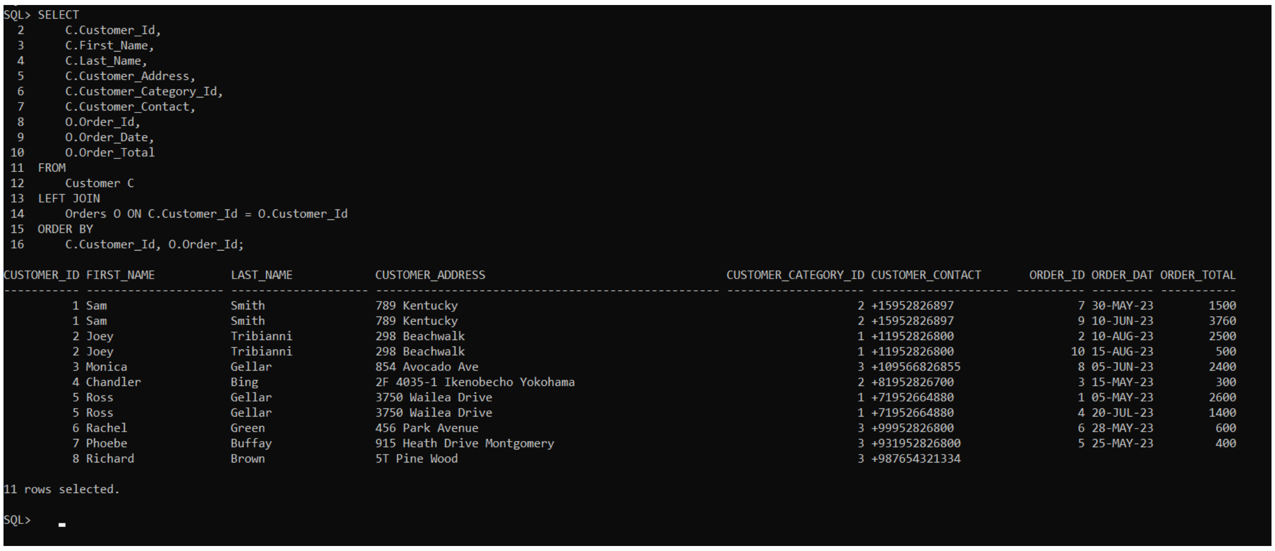
3.

Purpose: To list all the customers with their order details and also the customers who have not ordered any products yet.

Query Implemented
SELECT C.Customer_Id, C.First_Name, C.Last_Name, C.Customer_Address, C.Customer_Category_Id, C.Customer_Contact, O.Order_Id, O.Order_Date, O.Order_Total **FROM** Customer C **LEFT JOIN** Orders O ON C.Customer_Id = O.Customer_Id **ORDER BY** C.Customer_Id, O.Order_Id;

```
SELECT
        C.Customer_Id,
        C.First_Name,
        C.Last_Name,
        C.Customer_Address,
        C.Customer_Category_Id,
        C.Customer_Contact,
        O.Order_Id,
        O.Order_Date,
        O.Order_Total
FROM
        Customer C
LEFT JOIN
        Orders O ON C.Customer_Id = O.Customer_Id
ORDER BY
                C.Customer_Id, O.Order_Id;
```

Table 26: 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.First_Name,
        C.Last_Name,
        C.Customer_Address,
        C.Customer_Category_Id,
        C.Customer_Contact,
        O.Order_Id,
        O.Order_Date,
        O.Order_Total
    FROM
        Customer C
    LEFT JOIN
        Orders O ON C.Customer_Id = O.Customer_Id
    ORDER BY
        C.Customer_Id, O.Order_Id;

CUSTOMER_ID FIRST_NAME           LAST_NAME            CUSTOMER_ADDRESS                                   CUSTOMER_CATEGORY_ID CUSTOMER_CONTACT       ORDER_ID ORDER_DAT ORDER_TOTAL
---------- -------------------- -------------------- -------------------------------------------------- -------------------- -------------------- ---------- --------- ----------
         1 Sam                  Smith                789 Kentucky                                                          2 +15952826897                  7 30-MAY-23       1500
         1 Sam                  Smith                789 Kentucky                                                          2 +15952826897                  9 10-JUN-23       3760
         2 Joey                 Tribianni            298 Beachwalk                                                         1 +11952826800                  2 10-AUG-23       2500
         2 Joey                 Tribianni            298 Beachwalk                                                         1 +11952826800                 10 15-AUG-23        500
         3 Monica               Gellar               854 Avocado Ave                                                       3 +109566826855                 8 05-JUN-23       2400
         4 Chandler             Bing                 2F 4035-1 Ikenobecho Yokohama                                         2 +81952826700                  3 15-MAY-23        300
         5 Ross                 Gellar               3750 Wailea Drive                                                     1 +71952664880                  1 05-MAY-23       2600
         5 Ross                 Gellar               3750 Wailea Drive                                                     1 +71952664880                  4 20-JUL-23       1400
         6 Rachel               Green                456 Park Avenue                                                       3 +99952826800                  6 28-MAY-23        600
         7 Phoebe               Buffay               915 Heath Drive Montgomery                                            3 +931952826800                 5 25-MAY-23        400
         8 Richard              Brown                5T Pine Wood                                                          3 +987654321334

11 rows selected.

SQL>
```

Figure 43: listing all the customers with their order details and also the customers who have not ordered any products yet.

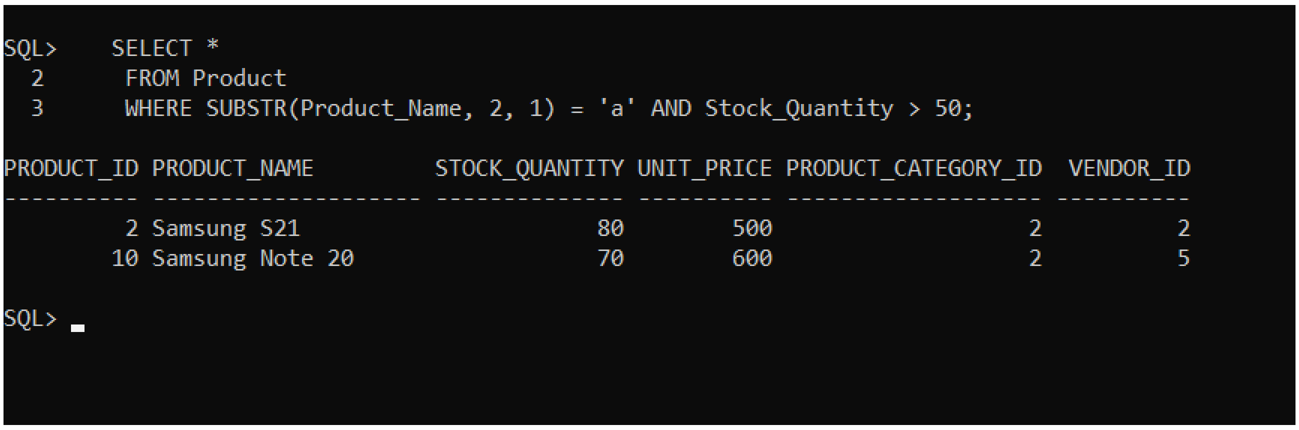
4.

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

Query Implemented

SELECT * **FROM** Product
WHERE SUBSTR(Product_Name, 2, 1) = 'a' **AND** Stock_Quantity > 50;

Table 27: listing all product details that have the second letter ‘a’ in their product name and have a stock quantity more than 50.



```
SQL>    SELECT *
  2      FROM Product
  3      WHERE SUBSTR(Product_Name, 2, 1) = 'a' AND Stock_Quantity > 50;

PRODUCT_ID PRODUCT_NAME         STOCK_QUANTITY UNIT_PRICE PRODUCT_CATEGORY_ID  VENDOR_ID
---------- -------------------- -------------- ---------- ------------------- ----------
         2 Samsung S21                      80        500                   2          2
        10 Samsung Note 20                  70        600                   2          5

SQL> _
```

Figure 44: listing all product details that have the second letter ‘a’ in their product name and have a stock quantity more than 50.

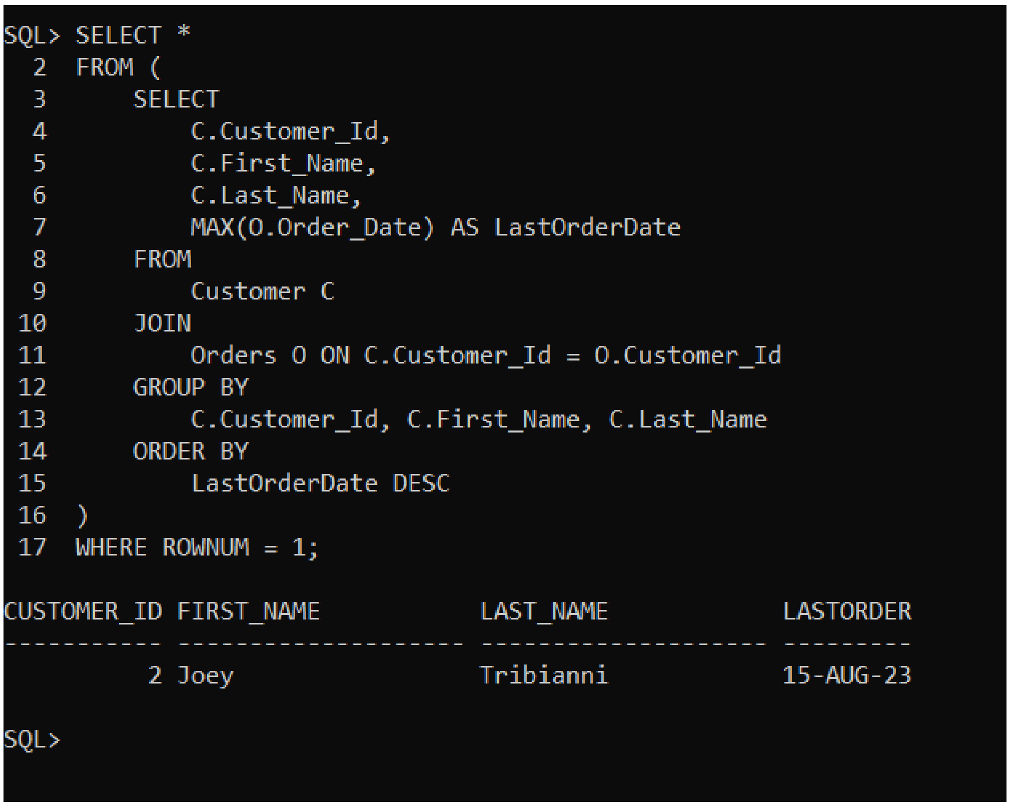
5.

Purpose: To find out the customer who has ordered recently.

Query Implemented

```
SELECT * FROM (
    SELECT
        C.Customer_Id,
        C.First_Name,
        C.Last_Name,
        MAX(O.Order_Date) AS LastOrderDate
    FROM
        Customer C
    JOIN
        Orders O ON C.Customer_Id = O.Customer_Id
    GROUP BY
        C.Customer_Id, C.First_Name, C.Last_Name
    ORDER BY
        LastOrderDate DESC
              )
          WHERE ROWNUM = 1;
```

Table 28:Finding out the customer who has ordered recently.



```
SQL> SELECT *
FROM (
    SELECT
        C.Customer_Id,
        C.First_Name,
        C.Last_Name,
        MAX(O.Order_Date) AS LastOrderDate
    FROM
        Customer C
    JOIN
        Orders O ON C.Customer_Id = O.Customer_Id
    GROUP BY
        C.Customer_Id, C.First_Name, C.Last_Name
    ORDER BY
        LastOrderDate DESC
)
WHERE ROWNUM = 1;

CUSTOMER_ID FIRST_NAME           LAST_NAME            LASTORDER
----------- -------------------- -------------------- ---------
          2 Joey                 Tribianni            15-AUG-23

SQL>
```

Figure 45: finding out the customer who has ordered recently.

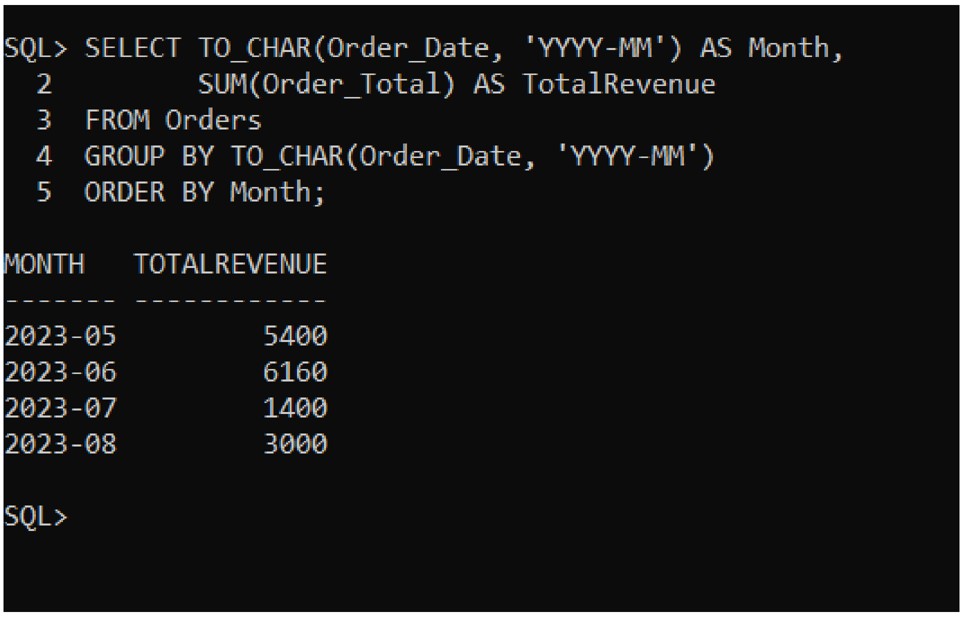
8.2. Transactional Queries

1.

Purpose: To show the total revenue of the company for each month.

Query Implemented
SELECT TO_CHAR(Order_Date, 'YYYY-MM') AS Month, **SUM**(Order_Total) AS TotalRevenue **FROM** Orders **GROUP BY** TO_CHAR(Order_Date, 'YYYY-MM') **ORDER BY** Month;

Table 29: Showing the total revenue of the company for each month.



```
SQL> SELECT TO_CHAR(Order_Date, 'YYYY-MM') AS Month,
  2         SUM(Order_Total) AS TotalRevenue
  3  FROM Orders
  4  GROUP BY TO_CHAR(Order_Date, 'YYYY-MM')
  5  ORDER BY Month;

MONTH   TOTALREVENUE
------- ------------
2023-05         5400
2023-06         6160
2023-07         1400
2023-08         3000

SQL>
```

Figure 46: Showing the total revenue of the company for each month.

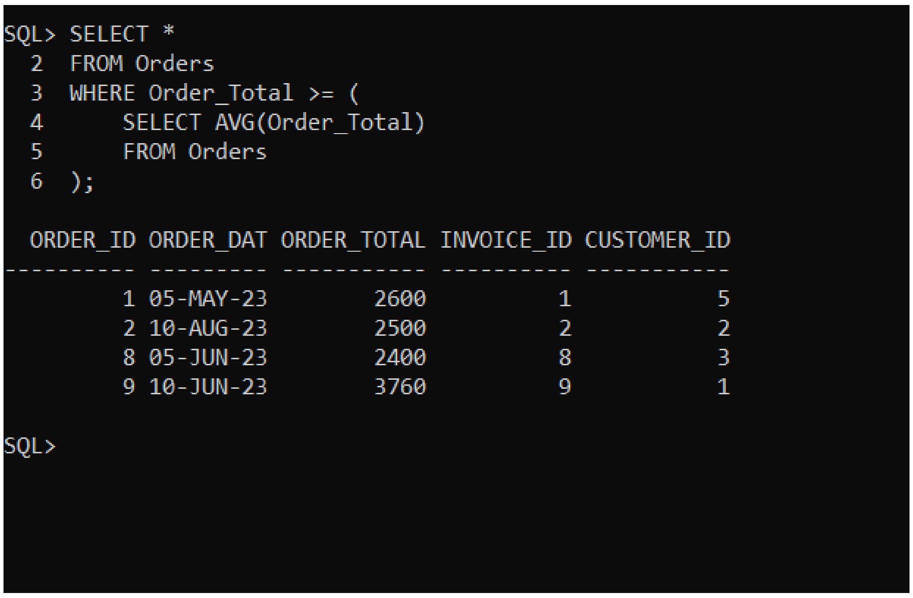
2.

Purpose:- To Find those orders that are equal or higher than the average order total value.

Query Implemented

SELECT *
FROM Orders
WHERE Order_Total >= (
SELECT AVG(Order_Total)
FROM Orders
);

Table 30: Finding those orders that are equal or higher than the average order total value.



```
SQL> SELECT *
     FROM Orders
     WHERE Order_Total >= (
         SELECT AVG(Order_Total)
         FROM Orders
     );

  ORDER_ID ORDER_DAT ORDER_TOTAL INVOICE_ID CUSTOMER_ID
---------- --------- ----------- ---------- -----------
         1 05-MAY-23        2600          1           5
         2 10-AUG-23        2500          2           2
         8 05-JUN-23        2400          8           3
         9 10-JUN-23        3760          9           1

SQL>
```

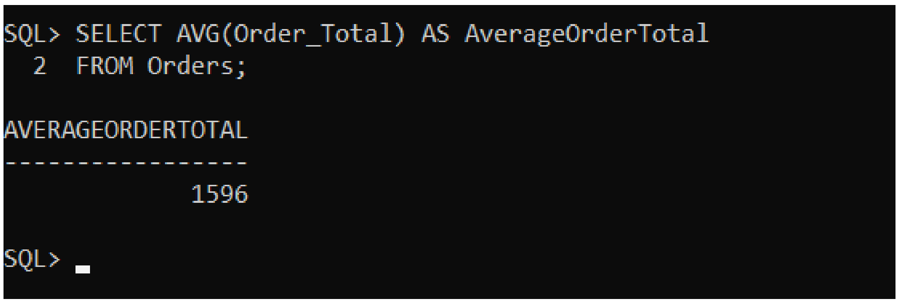
Figure 47: Finding those orders that are equal or higher than the average order total value.

Note: Calculating Just Average, to make sure the above query and result is correct.

Query Implemented

SELECT AVG(Order_Total) **AS** AverageOrderTotal
FROM Orders;

Table 31: Calculating Average of Order Total.



```
SQL> SELECT AVG(Order_Total) AS AverageOrderTotal
  2  FROM Orders;

AVERAGEORDERTOTAL
-----------------
             1596

SQL>
```

Figure 48: Finding those orders that are equal or higher than the average order total value.

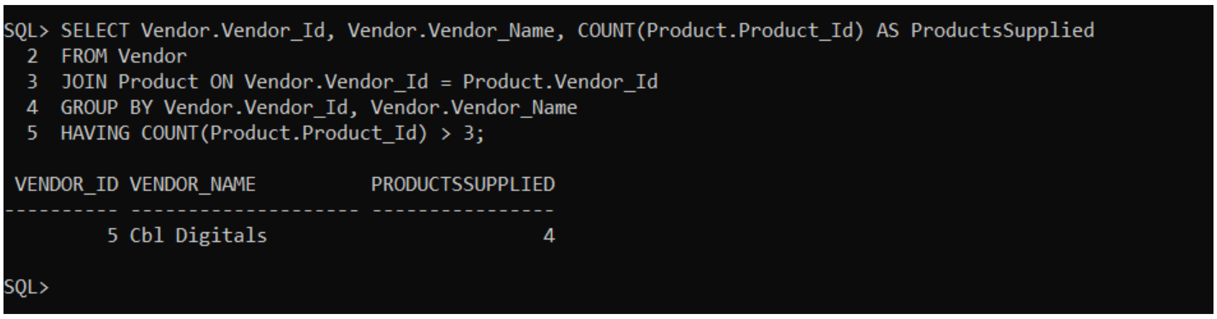
3.

Purpose:- To list the details of vendors who have supplied more than 3 products to the company.

Query Implemented

SELECT Vendor.Vendor_Id, Vendor.Vendor_Name, **COUNT**(Product.Product_Id) AS ProductsSupplied
FROM Vendor
JOIN Product **ON** Vendor.Vendor_Id = Product.Vendor_Id
GROUP BY Vendor.Vendor_Id, Vendor.Vendor_Name
HAVING COUNT(Product.Product_Id) > 3;

Table 32: listing the details of vendors who have supplied more than 3 products to the company.



```
SQL> SELECT Vendor.Vendor_Id, Vendor.Vendor_Name, COUNT(Product.Product_Id) AS ProductsSupplied
  2  FROM Vendor
  3  JOIN Product ON Vendor.Vendor_Id = Product.Vendor_Id
  4  GROUP BY Vendor.Vendor_Id, Vendor.Vendor_Name
  5  HAVING COUNT(Product.Product_Id) > 3;

 VENDOR_ID VENDOR_NAME          PRODUCTSSUPPLIED
---------- -------------------- ----------------
         5 Cbl Digitals                        4

SQL>
```

Figure 49: listing the details of vendors who have supplied more than 3 products to the company.

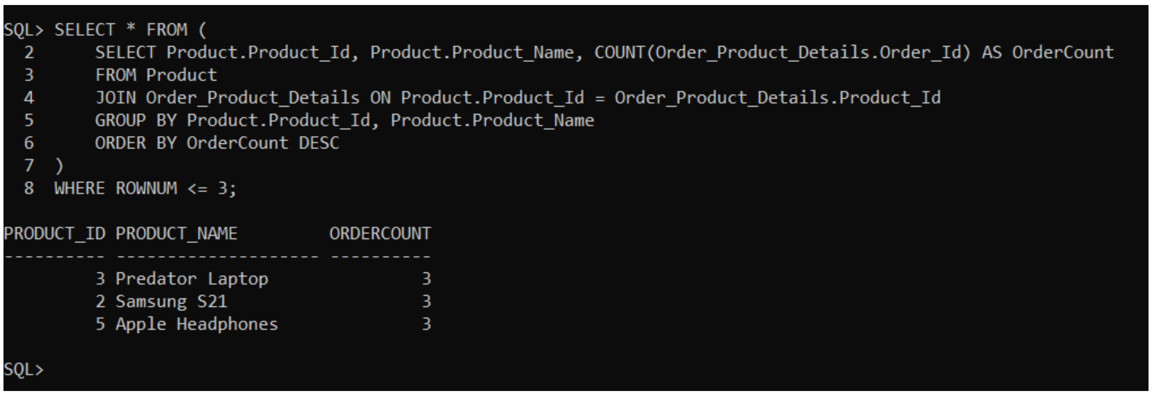
4.

Purpose:- To show the top 3 product details that have been ordered the most.

Query Implemented

SELECT * FROM (
 SELECT Product.Product_Id, Product.Product_Name, **COUNT**(Order_Product_Details.Order_Id) AS OrderCount
 FROM Product
 JOIN Order_Product_Details ON Product.Product_Id = Order_Product_Details.Product_Id
 GROUP BY Product.Product_Id, Product.Product_Name
 ORDER BY OrderCount DESC
)
 WHERE ROWNUM <= 3;

Table 33: Showing the top 3 product details that have been ordered the most.



```
SQL> SELECT * FROM (
        SELECT Product.Product_Id, Product.Product_Name, COUNT(Order_Product_Details.Order_Id) AS OrderCount
        FROM Product
        JOIN Order_Product_Details ON Product.Product_Id = Order_Product_Details.Product_Id
        GROUP BY Product.Product_Id, Product.Product_Name
        ORDER BY OrderCount DESC
     )
     WHERE ROWNUM <= 3;

PRODUCT_ID PRODUCT_NAME         ORDERCOUNT
---------- -------------------- ----------
         3 Predator Laptop               3
         2 Samsung S21                   3
         5 Apple Headphones              3

SQL>
```

Figure 50: Showing the top 3 product details that have been ordered.

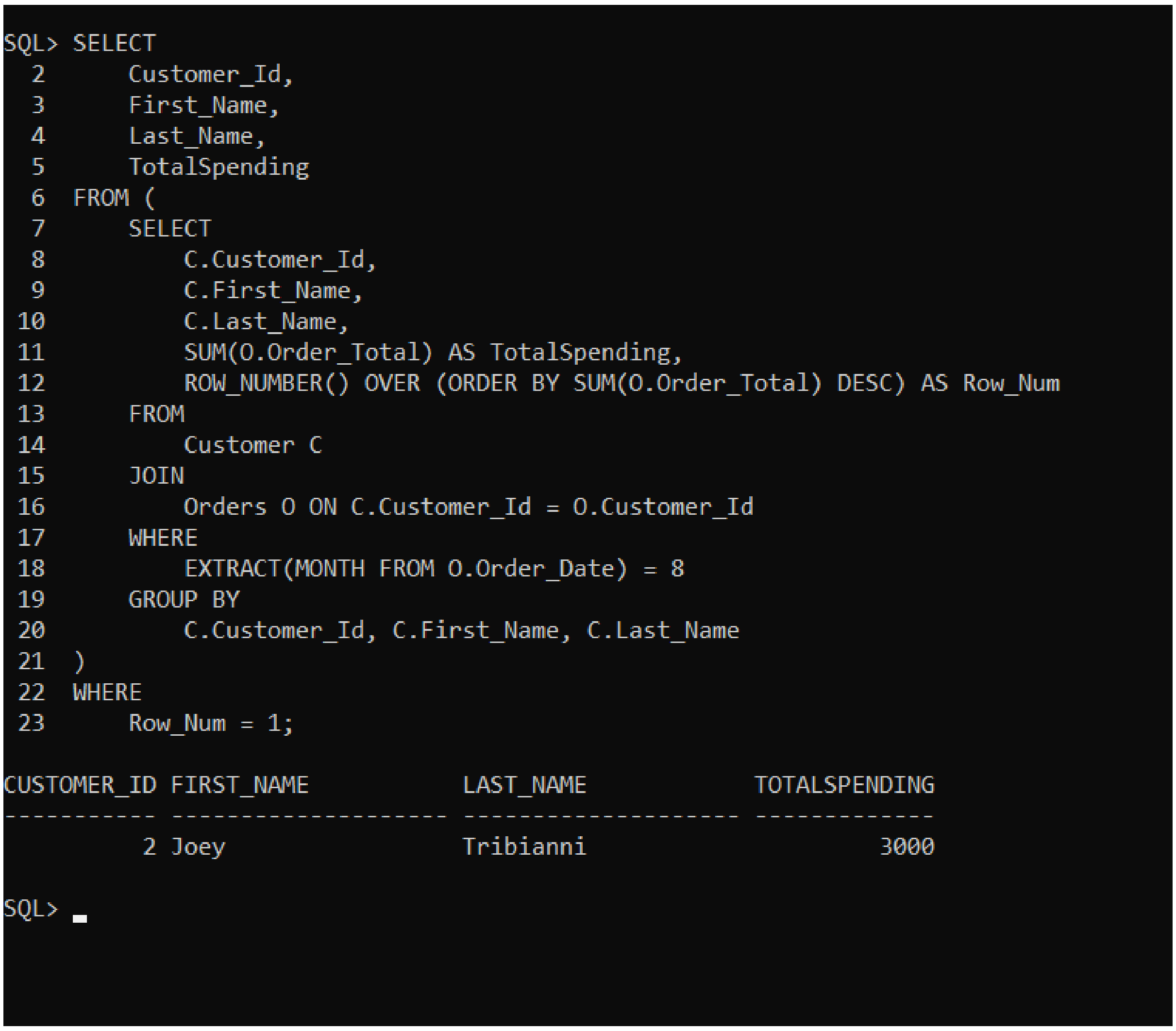
5.

Purpose:-To find out the customer who has ordered the most in August with his/her total spending on that month.

Query Implemented
SELECT Customer_Id, First_Name, Last_Name, TotalSpending **FROM** (**SELECT** C.Customer_Id, C.First_Name, C.Last_Name, **SUM**(O.Order_Total) AS TotalSpending, **ROW_NUMBER() OVER** (ORDER BY SUM(O.Order_Total) DESC) AS Row_Num **FROM** Customer C **JOIN** Orders O ON C.Customer_Id = O.Customer_Id **WHERE** **EXTRACT**(MONTH FROM O.Order_Date) = 8 **GROUP BY** C.Customer_Id, C.First_Name, C.Last_Name) **WHERE** **Row_Num** = 1;

```
SELECT
   Customer_Id,
   First_Name,
   Last_Name,
   TotalSpending
FROM (
   SELECT
      C.Customer_Id,
      C.First_Name,
      C.Last_Name,
      SUM(O.Order_Total) AS TotalSpending,
      ROW_NUMBER() OVER (ORDER BY SUM(O.Order_Total)    DESC)
AS Row_Num
   FROM
      Customer C
   JOIN
      Orders O ON C.Customer_Id = O.Customer_Id
   WHERE
      EXTRACT(MONTH FROM O.Order_Date) = 8
   GROUP BY
      C.Customer_Id, C.First_Name, C.Last_Name
     )
WHERE
   Row_Num = 1;
```

Table 34: finding out the customer who has ordered the most in August with his/her total spending on that month



```
SQL> SELECT
         Customer_Id,
         First_Name,
         Last_Name,
         TotalSpending
     FROM (
         SELECT
             C.Customer_Id,
             C.First_Name,
             C.Last_Name,
             SUM(O.Order_Total) AS TotalSpending,
             ROW_NUMBER() OVER (ORDER BY SUM(O.Order_Total) DESC) AS Row_Num
         FROM
             Customer C
         JOIN
             Orders O ON C.Customer_Id = O.Customer_Id
         WHERE
             EXTRACT(MONTH FROM O.Order_Date) = 8
         GROUP BY
             C.Customer_Id, C.First_Name, C.Last_Name
     )
     WHERE
         Row_Num = 1;

CUSTOMER_ID FIRST_NAME           LAST_NAME            TOTALSPENDING
----------- -------------------- -------------------- -------------
          2 Joey                 Tribianni                     3000

SQL>
```

Figure 51: finding out the customer who has ordered the most in August with his/her total spending on that month.

9. File Creation

9.1. Creating dump file

Step 1:- A folder where dump file was to be created was opened and "cmd" was typed to open command prompt on that location.

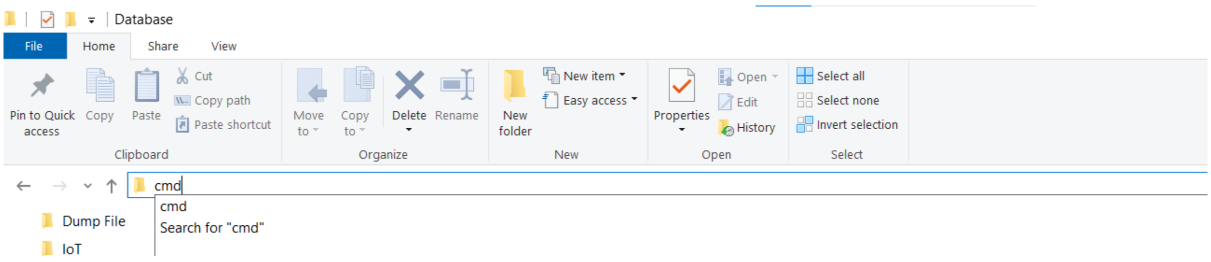
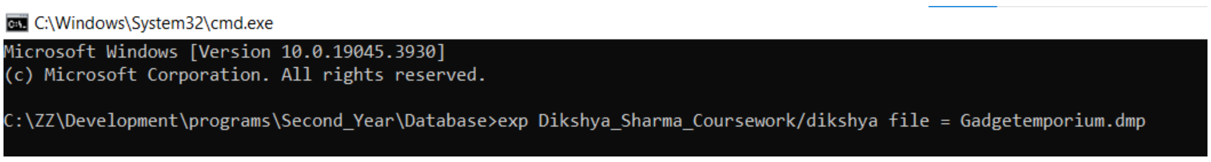


Figure 52: Step 1 of Dump File Creation.

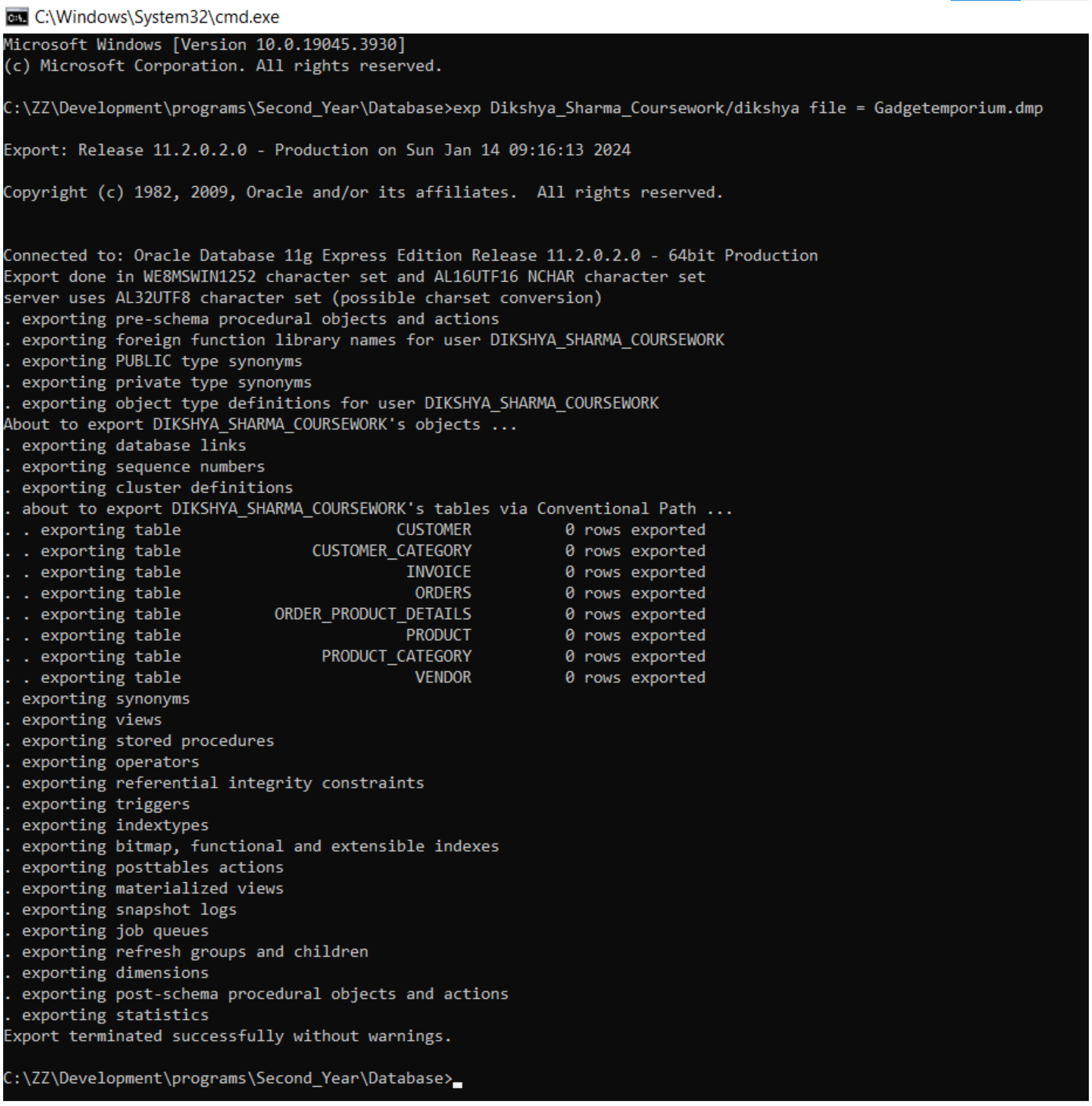
Step 2:- Following that, on command prompt "exp Dikshya_Sharma_Coursework/dikshya file = Gadgetemporium.dmp" was typed.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\ZZ\Development\programs\Second_Year\Database>exp Dikshya_Sharma_Coursework/dikshya file = Gadgetemporium.dmp
```

Figure 53: Step 2 of Dump File Creation.

Step 3:- After pressing enter, dump file was successfully created.

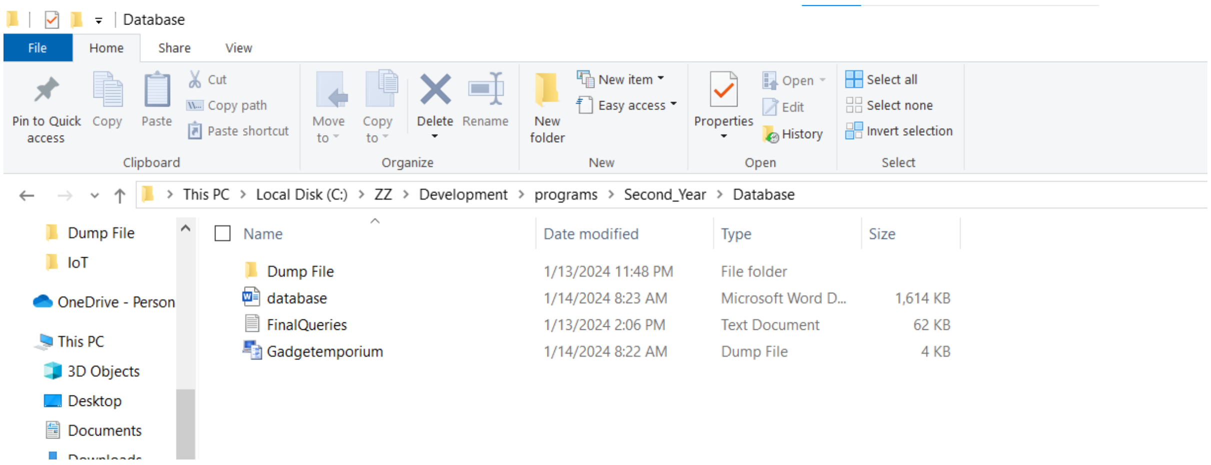


C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\ZZ\Development\programs\Second_Year\Database>exp Dikshya_Sharma_Coursework/dikshya file = Gadgetemporium.dmp

Export: Release 11.2.0.2.0 - Production on Sun Jan 14 09:16:13 2024

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


Connected to: Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit 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 DIKSHYA_SHARMA_COURSEWORK
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions for user DIKSHYA_SHARMA_COURSEWORK
About to export DIKSHYA_SHARMA_COURSEWORK's objects ...
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export DIKSHYA_SHARMA_COURSEWORK's tables via Conventional Path ...
. . exporting table                       CUSTOMER          0 rows exported
. . exporting table              CUSTOMER_CATEGORY          0 rows exported
. . exporting table                        INVOICE          0 rows exported
. . exporting table                         ORDERS          0 rows exported
. . exporting table          ORDER_PRODUCT_DETAILS          0 rows exported
. . exporting table                        PRODUCT          0 rows exported
. . exporting table               PRODUCT_CATEGORY          0 rows exported
. . exporting table                         VENDOR          0 rows exported
. exporting synonyms
. exporting views
. exporting stored procedures
. exporting operators
. exporting referential integrity constraints
. exporting triggers
. exporting indextypes
. exporting bitmap, functional and extensible indexes
. exporting posttables actions
. exporting materialized views
. exporting snapshot logs
. exporting job queues
. exporting refresh groups and children
. exporting dimensions
. exporting post-schema procedural objects and actions
. exporting statistics
Export terminated successfully without warnings.

C:\ZZ\Development\programs\Second_Year\Database>
```

Figure 54: Step 3 of dump file Creation.

Dump File on it's provided location.



Name	Date modified	Type	Size
Dump File	1/13/2024 11:48 PM	File folder	
database	1/14/2024 8:23 AM	Microsoft Word D...	1,614 KB
FinalQueries	1/13/2024 2:06 PM	Text Document	62 KB
Gadgetemporium	1/14/2024 8:22 AM	Dump File	4 KB

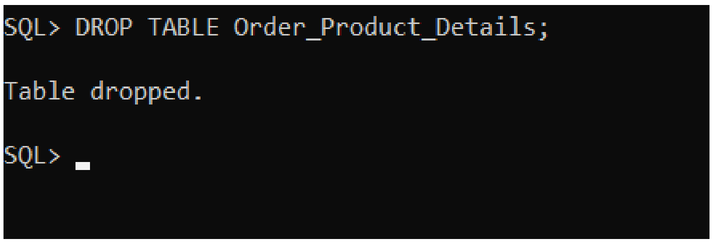
Figure 55: Dump File

10. DROP TABLES

Dropping tables on orders.

1. Table **Order_Product_Details.**

   Command:- **DROP TABLE** Order_Product_Details;

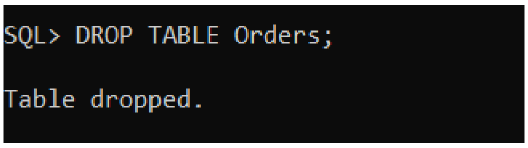


```
SQL> DROP TABLE Order_Product_Details;

Table dropped.

SQL> _
```

Figure 56: Dropping table Order_Product_Details.

2. Table **Orders.**

   Command:- **DROP TABLE** Orders;

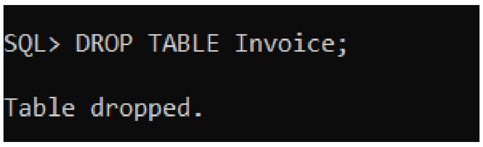


```
SQL> DROP TABLE Orders;

Table dropped.
```

Figure 57: Dropping table Orders;

3. Table **Invoice.**

   Command:- **DROP TABLE** Invoice;

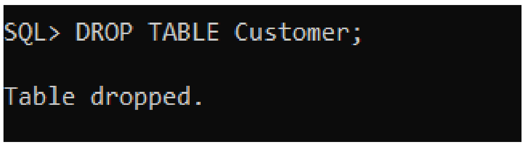


```
SQL> DROP TABLE Invoice;

Table dropped.
```

Figure 58: Dropping table Invoice;

4. Table **Customer**.

   Command:- **DROP TABLE** Customer;

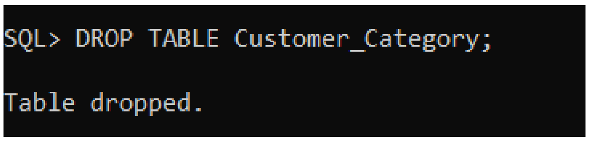


```
SQL> DROP TABLE Customer;

Table dropped.
```

Figure 59: Dropping Table Customer.

5. Table **Customer_Category**.

   Command:- **DROP TABLE** Customer_Category;

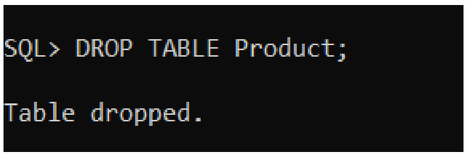


```
SQL> DROP TABLE Customer_Category;

Table dropped.
```

Figure 60: Dropping table Customer_Category.

6. Table **Product**.

   Command:- **DROP TABLE** Product;

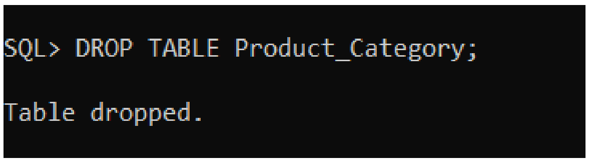


```
SQL> DROP TABLE Product;

Table dropped.
```

Figure 61:Dropping table Product.

7. Table **Product_Category**.

   Command:- **DROP TABLE** Product_Category;

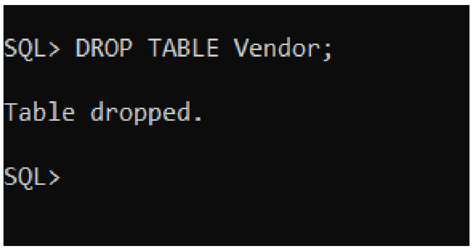


```
SQL> DROP TABLE Product_Category;

Table dropped.
```

Figure 62: Dropping table Product_Category.

8. Table **Vendor**;

   Command:- **DROP TABLE** Vendor;



```
SQL> DROP TABLE Vendor;

Table dropped.

SQL>
```

Figure 63: Dropping Table Vendor.

11. Critical evaluation

The ‘CC5051NI – Databases’ module proves to be a crucial component in the curriculum of Level 5, offering a well-rounded understanding of database systems by seamlessly blending theory with real-world experiences. As excellent module leader and tutors guide it, it covers essential topics such as crafting ERD, normalizations techniques, and the practical application of relational algebra and queries which collectively form a solid foundation inspiring students with crucial skills set that extends beyond theoretical concepts. These practical designs skills are invaluable as they enable students to create efficient and effective database structures and to maintain data integrity, addressing the evolving needs of industries. The broad applicability of this module includes IT organizations, health and commerce which prepares students for the diverse challenges that they may encounter in the near future. This module is not only limited to itself but relates to other modules as well. As it serves as a foundational blueprint for interconnected modules like Software Engineering where we encountered a scenario where we were supposed to create an ERD and thus, we utilized the knowledge and understanding of Databases module to successfully complete it.

This coursework was an example of developing a database management system for Mr. John to create a Gadget Emporium following his requirements and I undoubtedly considered this coursework as a captivating journey with many challenges serving as a source of knowledge. This coursework revolved around the practical implementation of concepts , involving the use of Oracle-based SQL program to bring the database to life. Managing queries and ensuring the seamless functioning of the system demanded real effort, yet the experience was remarkably enjoyable. Looking ahead, I recognize this coursework as a significant reference point for future works and the experience I got here was not merely an academic exercise but a practical journey that has left a lasting impact on my understanding and application of database management system.

12. References

Biscobing, J., 2019. *What is Entity Relationship Diagram (ERD)?.* [Online] Available at: https://www.techtarget.com/searchdatamanagement/definition/entity-relationship-diagram-ERD
[Accessed 5 January 2024].

Garge, S., 2022. *Unnormalized form - Alchetron, The Free Social Encyclopedia.* [Online]
Available at: https://alchetron.com/Unnormalized-form
[Accessed 9 January 2024].

Nalimov, C., 2020. *Crow's foot notation in entity-relationship diagrams.* [Online] Available at: https://www.gleek.io/blog/crows-foot-notation
[Accessed 5 January 2024].

Rouse, M., 2017. *What is an Entity-Relationship Diagram (ERD)? - Definition from Techopedia.* [Online]
Available at: https://www.techopedia.com/definition/1200/entity-relationship-diagram-erd
[Accessed 5 January 2024].

Rouse, M., 2019. *What is Database Normalization?.* [Online] Available at: https://www.techtarget.com/searchdatamanagement/definition/normalization
[Accessed 9 January 2024].

Visual Paradigm, 2023. *What is Entity Relationship Diagram (ERD)?.* [Online] Available at: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/
[Accessed 5 January 2024].