CC5051NI Databases

50% Individual Coursework

Autumn 2023

Student Name: Sachida Paudel

London Met ID: 22068732

Assignment Submission Date:14th January, 2024

Word Count: 6384

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

Table of Contents

Table of Figures

Table Of Tables

1. Introduction

An entrepreneur and electronics enthusiast, **Mr. John** aims at launching an online platform specializes in selling of electronic devices. The online marketplace named "Gadget Emporium" is design to provide both private customers and business organization with a large section of electronic devices. The documentation details the implementation and the database structure as its execution for the store, keeping track of all the required business activity records.

1.1 Current Business Scenario and Operations

"***Gadget Emporium***" is an online e-commerce platform that specializes in selling electronic devices and accessories like headphones, laptops, camera, smart phones, tablets, and many other related items. The platform facilitates the buying and selling of electronic products. The product management team at the emporium oversees and handles the details of accessories and electronic gadgets while customers are categorized as Regular, staffs and VIP. Customers can browse and purchase one or multiple products online. The emporium collaborates with single Vendor as well as multiple Vendors who supplies product to the platform.

Customers can make online payment or use credit cards, and a cash on delivery payment option is also available. The product ordered are delivered by the designated delivery personnel. After the transition process is finalized, an invoice is created giving the summary of customers details, order details and selected payment method.

To ensure the customers receives high quality service, utilizing a third-party tool like APIs can be used to enhance inventory management. Empowering sellers to manage the orders and offering various product shipping option provides flexibility to sellers. Employing market tool to promote product sale attracts the customer. Making a user-friendly web site for the store and listing the details of the products on the website to enhance the customers shopping experience. Adding hybrid products like convertible laptop-tablet, smart phones with camera enhancing features can prove advantageous.

Efficiently fulfilling customers' orders and responding quickly to buyer's query leads to customers satisfactions. Also creating and maintaining a website creates a centralized platform for both buyers and sellers.

In the role of database designer for Mr. John e-commerce endeavor, the main objective is the implementation of durable database design for the e-commerce website. The database designed should provide seamless operation for Mr. John established “Gadget Emporium”.

1.2 Business Rule Derived from The Description of Operational Products

The business rule that must be followed while operating the business activities are as followed:

- A single customer can make multiple orders at time, yet one order belongs to only one customer.
- One order can consist of multiple products conversely a product can also be a part of multiple order placed by various customers.
- An order comprises of one invoice, similarly one invoice belongs to only one order.
- Each customer has their customer-categories and each of the customers can be categorized as regular, staffs and VIP.
- Each product is associated with a single Vendor, but a Vendor can supply multiple products.
- Each product can have one category and each category belongs to one or multiple products.
- Each order details have one payment option may it be cash on delivery or credit card or debit card or e-wallet.
- The Regular(R), Staffs(S) and VIP(V) customers categories are entitled to 0%, 5% and 10% discount respectively.
- Inventory management keeps track of real-time product availability to prevent over selling and to maintain accurate levels of stock.

1.3 Assumptions

- Product once ordered cannot be cancelled.
- Payment once made cannot be refunded.
- Address is included in the customers and Vendor attributes to facilitate the delivery process.
- Customers can track their shipping location by getting register in the emporium platform.
- Record of each customer order and payment history is maintained in the system.
- A single invoice is generated for an individual order and each order is associated with single invoice.
- Customers can buy one product, many products, or no product i.e., customer making an order is optional.
- Order_quantity and line_total both are fully dependent on product_ID and Order_ID.
- Vendor_ID is partially dependent on product_ID only.

2. ERD

The Entity relationship diagram (ERD) defines a graphical representation between the entities and the relationships between those entities in the table. The ERD are mainly made up of three main components i.e., entities, attributes, and relationships. Attributes can be defined as the identifiers of the entities. Relationships in an ERD can be either one-to-one, one-to-many and many-to-many. The ERD are mainly for designing relational database in the software engineering field (Secoda, 2023).

- One-to-one Relationship:

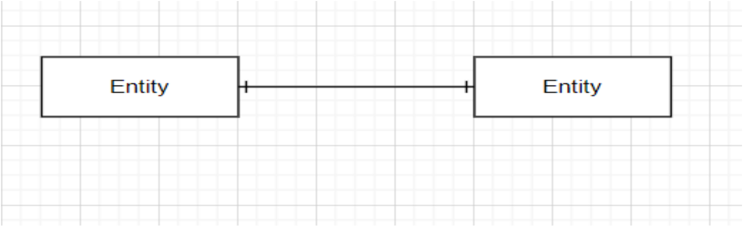


Figure 1: one-to-one relationships

- One-to-many Relationship:

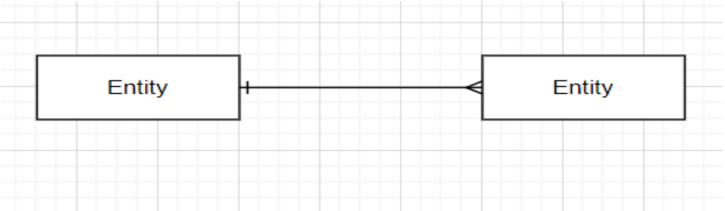


Figure 2: one-to-many relationships

- Many-to-many Relationship:

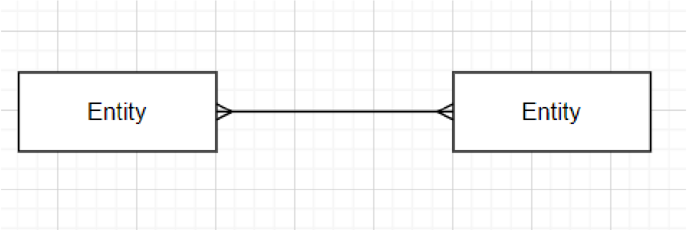


Figure 3: Many-to-many Relationships

2.1 Identifications of Entities and Attributes

Entity Is a distinctly identified real world thing, that can be unique and stands out of the crowd. It is represented in a rectangular box in an ER diagram. The characteristic of entities can be defined to as attributes. Attribute in short, can be defined as a database component, such as a table (Jain, 2023).

The entities and attributes shown below are the initial entities and attributes prior to normalization.

CUSTOMER

The entity named Customer consist of attributes like customer_ID, customer_name, customer_address, customer_phone, customer_category_ID, customer_category_type, customer_discount_rate. All the required data about the entity "CUSTOMER" are listed down in the table.

Attributes	Data Type	Constrains	Descriptions
Customer_ID	INTEGER	PRIMARY KEY, UNIQUE	It is a primary key attribute. Customer_Id holds the unique identity value of the customers.
First_name	VARCHAR2 (10)	NOT NULL	First name of the customer.
Last_name	VARCHAR2 (15)	NOT NULL	Last name of the customer.
Customer_address	VARCHAR2 (30)	NULL	The address of the customer.
Customer_phone	VARCHAR2 (15)	NOT NULL	The phone number of the customer.
Customer_category_ID	INTEGER	NOT NULL	The customer category ID of the customers.
Customer_type	VARCHAR2 (15)	NOT NULL	The type of categorized customers i.e., Regular, Staffs and VIPs
Customer_discount_rate	NUMBER (10, 2)	NOT NULL	Discount rate for the categorized customer.

Table 1: Data Dictionary for CUSTOMER entity

ORDERS

The entity Orders consists of attributes like order_ID, order_date, order_total, discount_amount, grand_total_after_discount, payment_method, payment_status, invoice_ID, invoice_number. All the required data about the entity "ORDERS" are listed down in the table.

Attributes	Data Type	Constrains	Descriptions
Order_ID	INTEGER	PRIMARY KEY, UNIQUE	It is a primary key attribute. Order_Id holds the unique identity value of the order.
Order_date	DATE	NOT NULL	The date of order.
Order_total	NUMBER (10, 2)	NOT NULL	Total amount for the order.
Discount_amount	NUMBER (10, 2)	NULL	Amount to be discounted.
Grand_total_after _discount	NUMBER (10, 2)	NOT NULL	Total amount after discount deduction.
Payment_method	VARCHAR2 (25)	NOT NULL	The method of payment used for the placed order.
Payment_status	VARCHAR2 (15)	NOT NULL	The status of the payment for the placed order.
Invoice_ID	INTEGER	NOT NULL	Unique ID for the invoice.
Invoice_number	VARCHAR2 (10)	NOT NULL	The number for the invoice issued.
Customer_ID	INTEGER	NOT NULL, FOREIGN KEY	Unique ID for customer set as Foreign key.

Table 2: Data Dictionary for ORDERS entity

PRODUCT

The entity product consists of attributes like product_ID, product_name, unit_price, product_availability, order_quantity, line_total, stock_quantity, product_category_ID, product_category_name, order_product_ID, Vendor_ID, Vendor_name, Vendor_address, Vendor_phone. All the required data about the entity "PRODUCT" are listed down in the table.

Attributes	Data Type	Constrains	Descriptions
Product_ID	INTEGER	PRIMARY KEY, UNIQUE	It is a primary key attribute. Product_ID holds the unique identity value of the product.
Product_name	VARCHAR2 (25)	NOT NULL	The name of each of the products.
unit_price	NUMBER (10, 2)	NOT NULL	The unit price of each of the products.
Product_availability	VARCHAR2 (10)	NULL	Availability status of the product.
Order_quantity	VARCHAR (10)	NOT NULL	The quantity of the product ordered.
Stock_quantity	INTEGER	NULL	The stock for the product
Line_total	NUMBER (10, 2)	NOT NULL	The multiplication of unit price and order product quantity.
Product_category_ID	INTEGER	NOT NULL	ID of the product category.
Product_category_name	VARCHAR2 (25)	NOT NULL	Name of the product category
Order_product_ID	INTEGER	NOT NULL	ID of the order and product.
Vendor_ID	INTEGER	NOT NULL	ID of the Vendors of the products.
Vendor_name	VARCHAR2 (20)	NOT NULL	Name of the product Vendor.
Vendor_address	VARCHAR2 (25)	NULL	Address of the product Vendor
Vendor_phone	VARCHAR2 (15)	NOT NULL	Phone number of the product Vendor.

Table 3: Data Dictionary for PRODUCT entity

2.2 Initial ERD

The initial ERD diagrams are the rough sketch that shows the initial relationship between attributes. The Initial ERD for the system is given below:

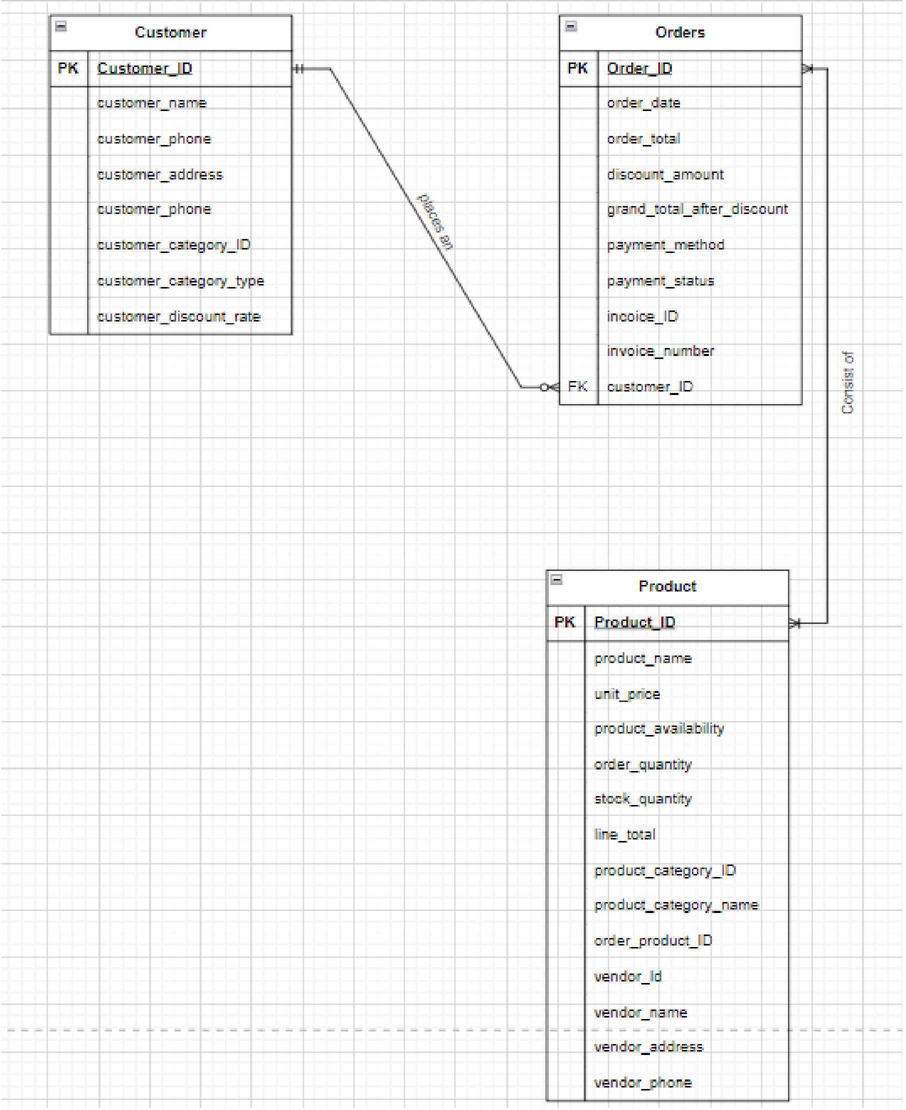


Figure 4: Initial ERD

3. Normalization

The process of breaking down of complex data table into simpler data table in an organized manner can be termed as normalization. Removing the anomalies which may further leads to data redundancy is the main reason behind performing normalization. There are mainly four steps in normalization i.e., UNF, 1NF, 2NF and 3NF (javaTpoint, 2021).

Normalization is done with the attributes of the initial ERD. Below are the attributes of the initial entities:

Customer: customer_ID, customer_name, customer_address, customer_phone, customer_category_ID, customer_type, customer_discount_rate

Orders: order_ID, order_date, order_total, discount_amount, grand_total_after_discount, payment_method, payment_status, invoice_ID, invoice_number

Products: product_ID, product_name, unit_price, product_availability, order_quantity, line_total, stock_quantity, product_category_ID, product_category_name, order_product_ID, Vendor_ID, Vendor_name, Vendor_address, Vendor_phone

Below are the following steps involved in Normalization:

3.1 Unnormalized Form (UNF)

The following steps are carried out in UNF:

- In UNF all the attributes are gathered and put into a list.
- The primary key is identified and is represented with an underline.
- The repeating group is distinguished with a curly bracket.

UNF:

ORDERS

<u>Order_ID</u>*(PK), Order_date, order_total, discount_amount, grand_total_after_discount, payment_method, payment_status, invoice_ID, invoice_number, customer_ID, customer_name, customer_phone, customer_address, customer_category_ID, customer_type, customer_discount_rate, **{**product_ID, product_name, unit_price, product_availibility, stock_quantity, order_quantity, line_total, product_category_ID, product_category_type, Vendor_ID, Vendor_name, Vendor_address, Vendor_phone, order_product_ID**}**

Explanation:

In the UNF, all the attributes in the initial ERD are listed and the repeating group and repeating values are distinguished. The repeating group are separated by curly brackets. A primary key named order_ID is selected, and the table is named “orders”.

3.2 First Normal Form (1NF)

The following steps are carried out in 1NF:

- The repeating group is moved to a new table and a new entity name is given.
- A new primary key is identified for the new entity table
- The primary key of the 'order' table is added in as foreign key in the new entity table.

1NF:

ORDERS – 1

(**order_ID***(PK), Order_date, order_total, discount_amount, grand_total_after_discount, payment_method, payment_status, invoice_ID, invoice_number, customer_ID, customer_name, customer_address, customer_phone, customer_category_ID, customer_type, customer_discount_rate)

ORDER PRODUCTS DETAILS – 1

(product_ID*(PK), product_name, unit_price, product_availibility, stock_quantity, order_quantity, line_total, product_category_ID, product_category_type, Vendor_ID, Vendor_name, Vendor_address, Vendor_phone, order_product_ID, **order_ID*(FK))**

Explanation:

In the 1NF, two table namely "orders" and "order product details" are formed. All the repeating group are placed in the table named 'order product details and a primary key product_ID is identified. order_ID, primary key of order table is added to the order product table as foreign key.

3.3 Second Normal Form (2NF)

The following are the steps carried out in the second normal form(2NF):

- The table containing composite key is chosen.
- Full Functional Dependencies (FFD) and Partial Dependencies (PD) are identified by checking the dependencies of the key value attributes with a non- key value attribute.
- The partial key and dependent key are transfer to a new table.

Description:

There is only a single key i.e., primary key in the orders table. Hence, there is no partial dependency, so orders table is already in 2NF.

For Order products details table, there are composite keys. Thus, 2NF can be carried out.

Checking the partial dependencies and full functional dependencies,

Product_ID -> product_name, unit_price, product_availibility, stock_quantity, product_category_ID, product_category_name, Vendor_ID, Vendor_name, Vendor_address, Vendor_phone.

Order_ID -> NULL

Product_ID, order_ID -> order_product_ID, order_quantity, line_total

2NF:

ORDERS – 2

(**order_ID***(PK), Order_date, order_total, discount_amount, grand_total_after_discount, payment_method, payment_status, invoice_ID, invoice_number, customer_ID, customer_name, customer_address, customer_phone, customer_category_ID, customer_type, customer_discount_rate)

ORDER PRODUCT DETAILS – 2

(Order_ID*(FK), product_ID*(FK), order_product_ID*(PK), order_quantity, line_total**)**

PRODUCT – 2

(product_ID*(PK), product_name, unit_price, product_availibility, stock_quantity, order_quantity, line_total, product_category_ID, product_category_name, Vendor_ID, Vendor_name, Vendor_address, Vendor_phone**)**

Explanation:

In 2NF, three tables i.e., Orders, Order_Product details and Product tables are formed after checking the partial functional dependencies and full functional dependencies. Each table consists of one primary key each.

3.4 Third Normal Form (3NF)

The following are the steps carried out in third normal form:

- Transitive dependencies are checked in 3NF
- Dependencies between non-key attributes is identified with each table
- The identified attributes are then moved to the new table

Transitive dependencies occur in a table when a primary key in the table determines the non-key attributes in the table and the non-key attribute determines another non-key attribute in that table.

Descriptions:

Checking transitive dependencies in **Orders – 2** table,

Order_ID (PK) gives Invoice_ID, a non-key attribute and Invoice_ID gives invoice_number, invoice_name, invoice_date, payment_method, payment_status

i.e., order_ID -> Invoice_ID -> Invoice_number, invoice_name, discount_amount, payment_method, payment_status, *grand_total_after_discount*

Thus, a transitive dependency is separated from **Orders – 2** table.

Again, checking transitive dependencies in **Orders – 2** table,

Order_ID (PK) gives customer_ID, a non-key attribute and customer_ID gives customer_name, customer_address, customer_phone, customer_category_ID, customer_type, customer_discount_rate

i.e., order_ID -> customer_ID -> customer_name, customer_address, customer_phone, customer_category_ID, customer_type, customer_discount_rate

Another transitive dependency is separated from Orders – 2 table.

In **Order Products details - 2** table, there is no transitive dependencies. Thus, no table needs to be separated.

Checking transitive dependencies in Product – 2 table,

Product_ID (PK) gives product_category_ID, a non-key attributes and product_category_ID gives product_category_name

i.e., Product_ID -> product_category_ID -> product_category_name

Thus, a transitive dependency is separated from **Product – 2** table.

Again, checking transitive dependencies in **Product – 2** table,

Product_ID (PK) gives Vendor_ID, a non-key attributes and Vendor_ID gives Vendor_name, Vendor_address, Vendor_phone

Another transitive dependency is separated from **Product – 2** table.

Again, finding transitive dependency in **Customer** table,

Customer_ID (PK) gives customer_category_Id, a non-key attribute and customer_category_ID gives customer_type, customer_discount_rate

i.e., Customer_ID -> customer_category_Id -> customer_type, customer_discount_rate

Explanations:

Here, in the 3NF a total of eight tables are formed by removing all the transitive dependencies. All the required entities and attributes for the final ERD are obtained in the 3NF form.

3NF:

The Final Tables Obtained in 3NF after normalization are:

ORDERS – 3:

(order_ID*(PK), Order_date, Order_product, order_total, **invoice_ID(FK), customer_ID(FK))**

INVOICE – 3:

(invoice_ID*, invoice_number, discount_amount, grand_total_after_discount, payment_method, payment_status**)**

CUSTOMER – 3:

(customer_ID*(PK), customer_name, customer_address, customer_phone, **customer_category_ID(FK))**

CUSTOMER CATEGORY – 3:

(**customer_category_Id*(PK),** customer_type, customer_discount_rate**)**

ORDER PRODUCT DETAILS – 3:

(Order_ID(FK), product_ID(FK), order_product_ID*(PK), order_quantity, line_total**)**

PRODUCT – 3:

(product_ID*(PK), product_name, unit_price, product_availibility, stock_quantity, order_quantity, line_total**, 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, Vendor_phone)

4. Final ERD

The final ERD is obtained by normalization of initial ERD. In the final ERD the number of tables increases and data redundancy decrease, resulting in more simple and organized tables. There is no many to many relationships in the final ERD.

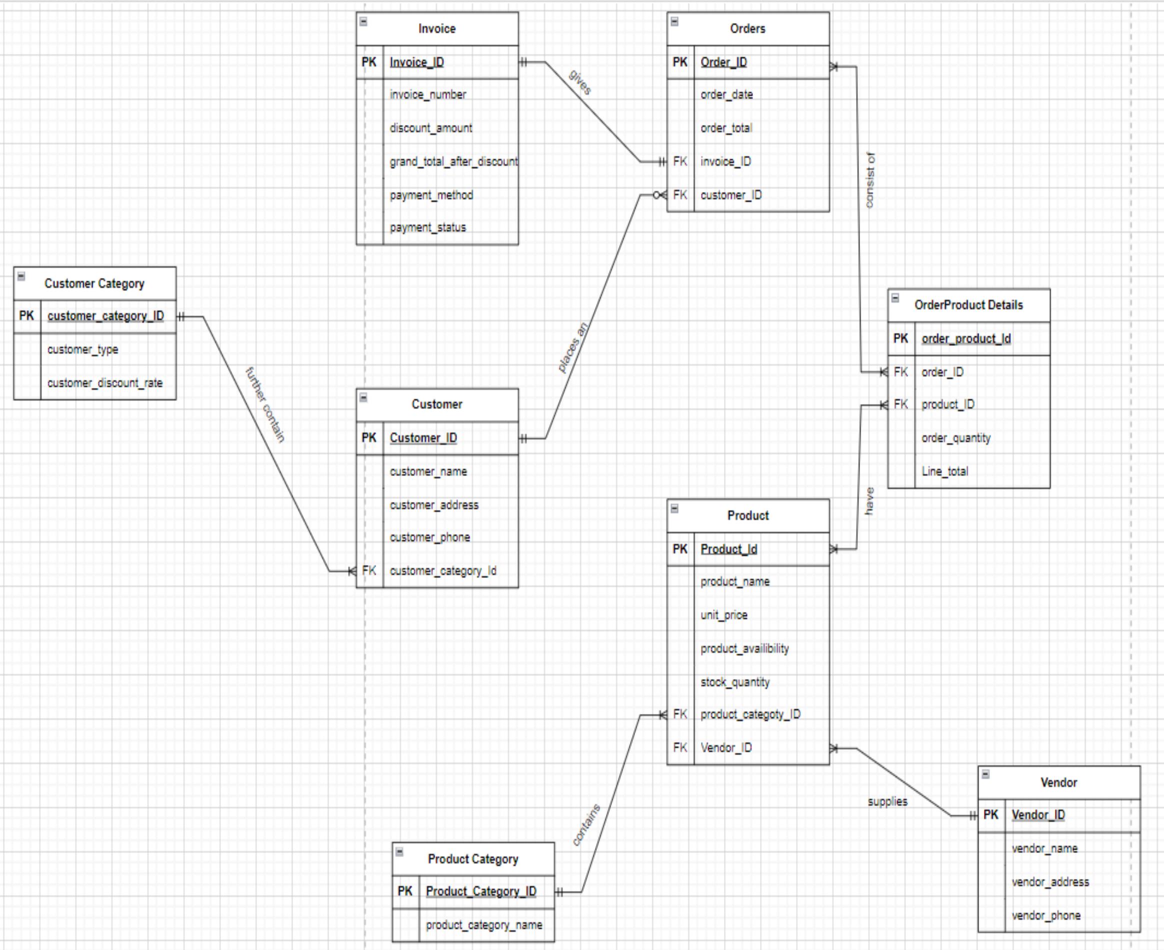
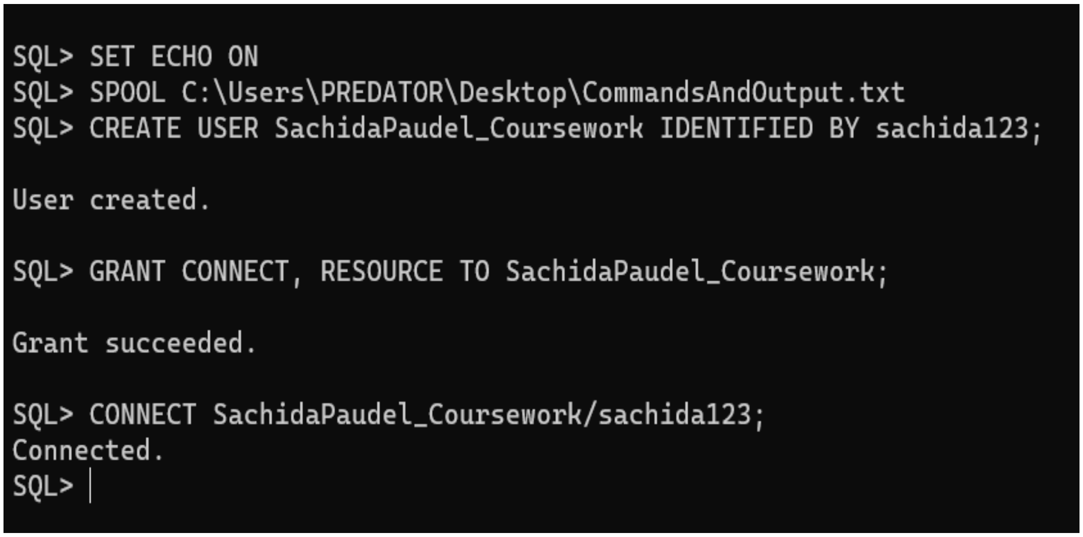


Figure 5: Final ERD

5. Implementation

While establishing and setting up a connection on the SQL plus echo is activated on and spool is employed to document the SQL plus session to the file. Subsequently, a new user named ‘SachidaPaudel_coursework’ is created. A command “grant connect, resources to SachidaPaudel_Coursework” is given to connect to the designated databases and to provide resources to the new user. Following this process, the user is connected to the database and tables are created within it.



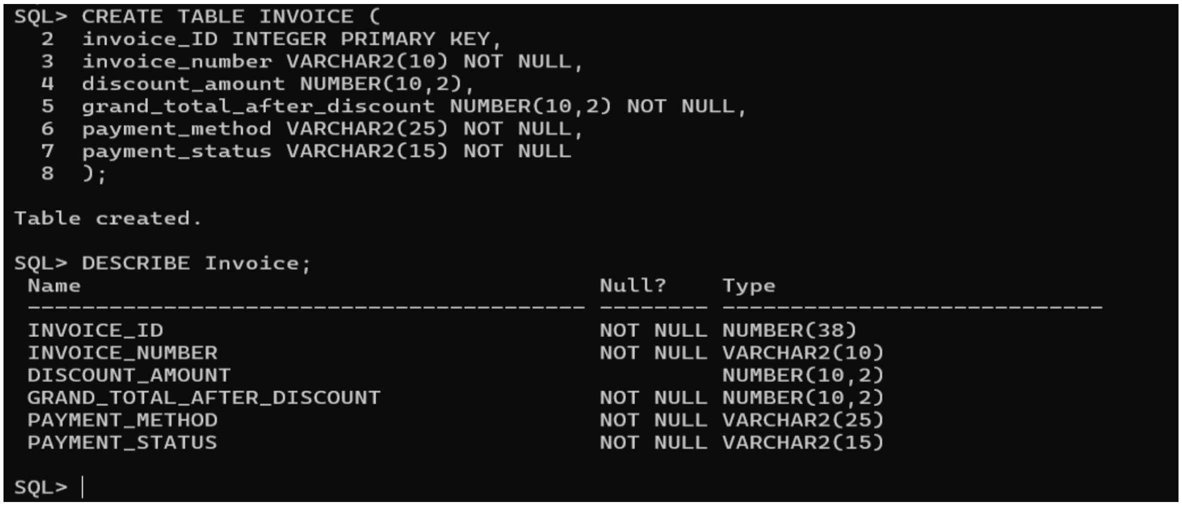
```
SQL> SET ECHO ON
SQL> SPOOL C:\Users\PREDATOR\Desktop\CommandsAndOutput.txt
SQL> CREATE USER SachidaPaudel_Coursework IDENTIFIED BY sachida123;

User created.

SQL> GRANT CONNECT, RESOURCE TO SachidaPaudel_Coursework;

Grant succeeded.

SQL> CONNECT SachidaPaudel_Coursework/sachida123;
Connected.
SQL>
```

Figure 6: Creating new user SachidaPaudel_Coursework

5.1 Creation and Explanation of Tables

5.1.1 Creating and Describing INVOICE table

A table named **INVOICE** is created with Invoice_ID as its primary key. There is no foreign key in the invoice table. Also, the table INVOICE consists of six attributes.



```
SQL> CREATE TABLE INVOICE (
  2  invoice_ID INTEGER PRIMARY KEY,
  3  invoice_number VARCHAR2(10) NOT NULL,
  4  discount_amount NUMBER(10,2),
  5  grand_total_after_discount NUMBER(10,2) NOT NULL,
  6  payment_method VARCHAR2(25) NOT NULL,
  7  payment_status VARCHAR2(15) NOT NULL
  8  );

Table created.

SQL> DESCRIBE Invoice;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 INVOICE_ID                                NOT NULL NUMBER(38)
 INVOICE_NUMBER                            NOT NULL VARCHAR2(10)
 DISCOUNT_AMOUNT                                    NUMBER(10,2)
 GRAND_TOTAL_AFTER_DISCOUNT                NOT NULL NUMBER(10,2)
 PAYMENT_METHOD                            NOT NULL VARCHAR2(25)
 PAYMENT_STATUS                            NOT NULL VARCHAR2(15)

SQL> |
```

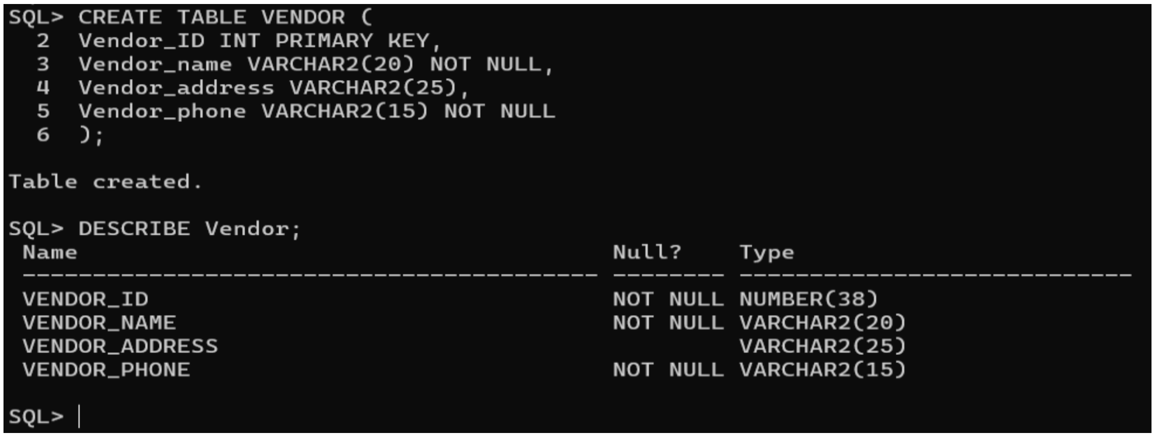
Figure 7: Creating and Describing Table "INVOICE"

Purpose	Used Commands
1. Creating a table “INVOICE”	CREATE TABLE INVOICE (invoice_ID INT PRIMARY KEY, invoice_number VARCHAR2(20) NOT NULL, discount_amount NUMBER (10), grand_total_after_discount NUMBER (10) NOT NULL, payment_method VARCHAR2(50) NOT NULL, payment_status VARCHAR2(50) NOT NULL);
2. Describing	DESCRIBE Invoice.

Table 4: Creating and Description of table "INVOICE"

5.1.2 Creating and Describing VENDOR table

A table named **VENDOR** is created with Vendor_ID as its primary key. There is no foreign key in the Vendor table. Also, the table vendor consists of four attributes.



```
SQL> CREATE TABLE VENDOR (
  2  Vendor_ID INT PRIMARY KEY,
  3  Vendor_name VARCHAR2(20) NOT NULL,
  4  Vendor_address VARCHAR2(25),
  5  Vendor_phone VARCHAR2(15) NOT NULL
  6  );

Table created.

SQL> DESCRIBE Vendor;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 VENDOR_ID                                 NOT NULL NUMBER(38)
 VENDOR_NAME                               NOT NULL VARCHAR2(20)
 VENDOR_ADDRESS                                     VARCHAR2(25)
 VENDOR_PHONE                              NOT NULL VARCHAR2(15)

SQL> |
```

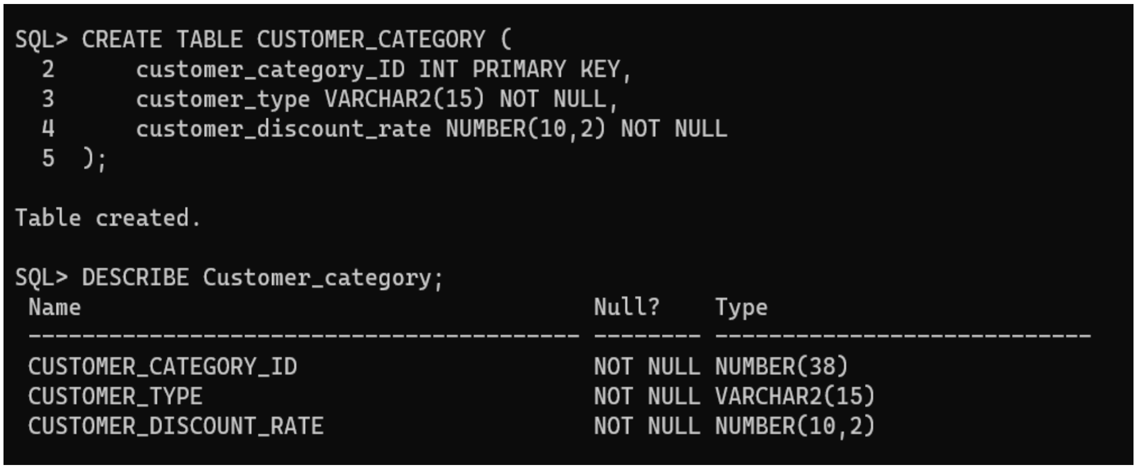
Figure 8: Creating and Describing Table "VENDOR"

Purpose	Used Commands
1. Creating a table “VENDOR”	CREATE TABLE VENDOR (Vendor_ID INT PRIMARY KEY, Vendor_name VARCHAR2(20) NOT NULL, Vendor_address VARCHAR2(25), Vendor_phone VARCHAR2(15) NOT NULL);
2. Describing	DESCRIBE Vendor.

Table 5: Creating and Description of table "VENDOR"

5.1.3 Creating and Describing CUSTOMER CATEGORY table

A table named **CUSTOMER_CATEGORY** is created with Customer_category_ID as its primary key. There is no foreign key in the Customer Category table. Also, the table customer_category consists of three attributes.



```
SQL> CREATE TABLE CUSTOMER_CATEGORY (
         customer_category_ID INT PRIMARY KEY,
         customer_type VARCHAR2(15) NOT NULL,
         customer_discount_rate NUMBER(10,2) NOT NULL
     );

Table created.

SQL> DESCRIBE Customer_category;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_CATEGORY_ID                      NOT NULL NUMBER(38)
 CUSTOMER_TYPE                             NOT NULL VARCHAR2(15)
 CUSTOMER_DISCOUNT_RATE                    NOT NULL NUMBER(10,2)
```

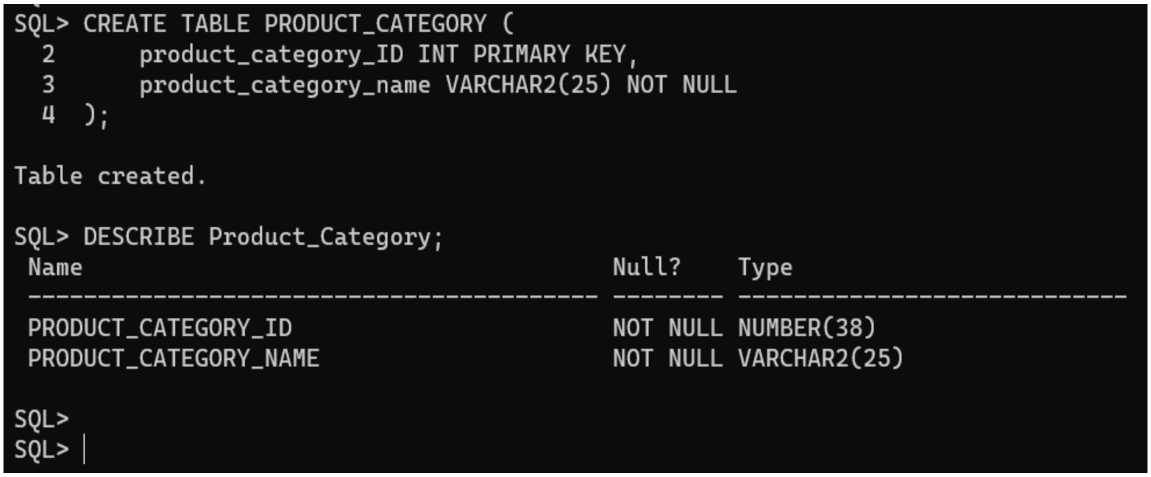
Figure 9:Creating and Describing table "Customer_Category"

Purpose	Used Commands
1. Creating a table “CUSTOMER_CATEGORY”	CREATE TABLE CUSTOMER_CATEGORY (customer_category_ID INT PRIMARY KEY, customer_type VARCHAR2(20) NOT NULL, customer_discount_rate NUMBER (10,2) NOT NULL);
2. Describing	DESCRIBE Customer_category;

Table 6: Creating and Description of table "Customer_category"

5.1.4 Creating and Describing PRODUCT CATEGORY Table

A table named **PRODUCT_CATEGORY** is created with Product_category_ID as its primary key. There is no foreign key in the Product_Category table. Also, the table Product_category consists of two attributes.



```
SQL> CREATE TABLE PRODUCT_CATEGORY (
         product_category_ID INT PRIMARY KEY,
         product_category_name VARCHAR2(25) NOT NULL
     );

Table created.

SQL> DESCRIBE Product_Category;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODUCT_CATEGORY_ID                       NOT NULL NUMBER(38)
 PRODUCT_CATEGORY_NAME                     NOT NULL VARCHAR2(25)

SQL>
SQL>
```

Figure 10: Creating and Describing Table "Product_category"

Purpose	Used Commands

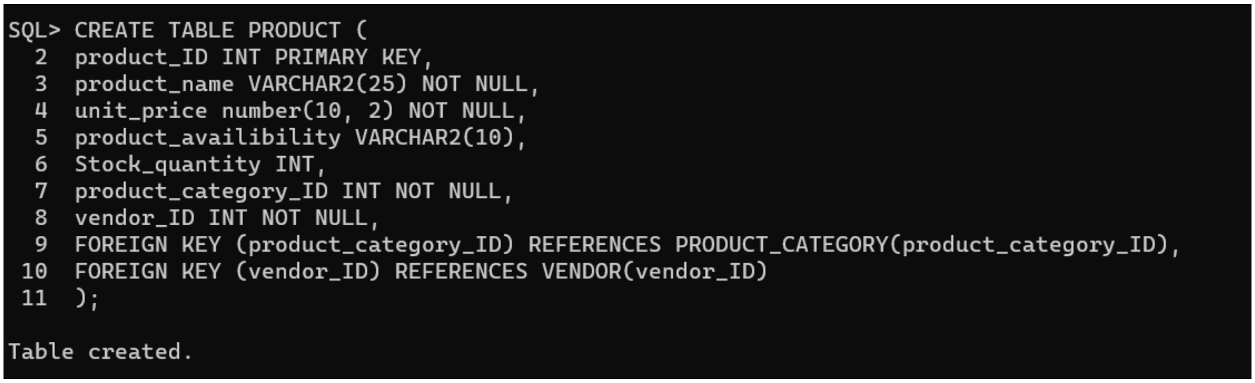
1. Creating a table “PRODUCT_CATEGORY”	CREATE TABLE PRODUCT_CATEGORY (
	product_category_ID INT PRIMARY KEY,
	product_category_name VARCHAR2(25) NOT NULL
	);

2. Describing	DESCRIBE Product_category ;

Table 7: Creating and Description of table "Product_Category"

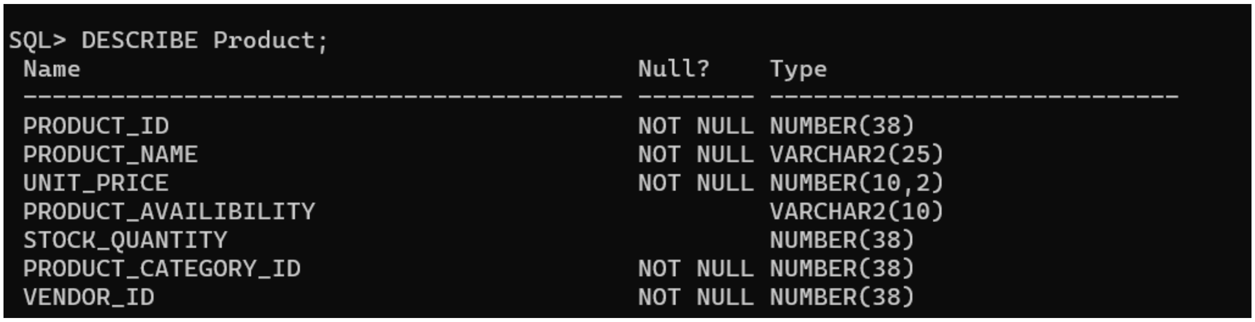
5.1.5 Creating and Describing PRODUCT Table

A table named **PRODUCT** is created with Product_ID as its primary key. There is product_category_ID and Vendor_ID as its foreign key. Also, the table Product consists of seven attributes.



```
SQL> CREATE TABLE PRODUCT (
     product_ID INT PRIMARY KEY,
     product_name VARCHAR2(25) NOT NULL,
     unit_price number(10, 2) NOT NULL,
     product_availibility VARCHAR2(10),
     Stock_quantity INT,
     product_category_ID INT NOT NULL,
     vendor_ID INT NOT NULL,
     FOREIGN KEY (product_category_ID) REFERENCES PRODUCT_CATEGORY(product_category_ID),
     FOREIGN KEY (vendor_ID) REFERENCES VENDOR(vendor_ID)
     );

Table created.
```

Figure 11: Creating table “PRODUCT”



```
SQL> DESCRIBE Product;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODUCT_ID                                NOT NULL NUMBER(38)
 PRODUCT_NAME                              NOT NULL VARCHAR2(25)
 UNIT_PRICE                                NOT NULL NUMBER(10,2)
 PRODUCT_AVAILIBILITY                               VARCHAR2(10)
 STOCK_QUANTITY                                     NUMBER(38)
 PRODUCT_CATEGORY_ID                       NOT NULL NUMBER(38)
 VENDOR_ID                                 NOT NULL NUMBER(38)
```

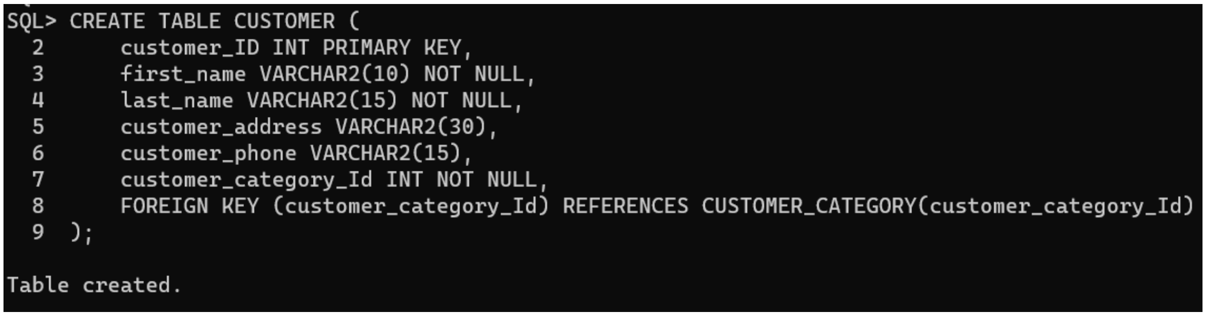
Figure 12: Describing table "Product"

Purpose	Used Commands
1. Creating a table “PRODUCT”	CREATE TABLE PRODUCT (product_ID INT PRIMARY KEY, product_name VARCHAR2(25) NOT NULL, unit_price NUMBER (10, 2) NOT NULL, product_availibility VARCHAR2(10), Stock_quantity INT NOT NULL, product_category_ID INT NOT NULL, Vendor_ID INT NOT NULL, FOREIGN KEY (product_category_ID) REFERENCES PRODUCT_CATEGORY (product_category_ID), FOREIGN KEY (Vendor_ID) REFERENCES VENDOR(Vendor_ID));
2. Describing	DESCRIBE Product;

Table 8: Creating and Description of table "Product"

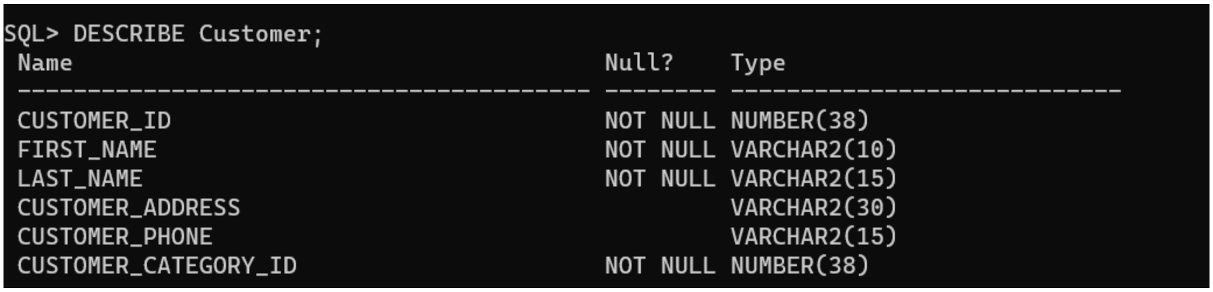
5.1.6 Creating and Describing CUSTOMER Table

A table named **CUSTOMER** is created with customer_ID as its primary key. There is customer_category_Id as its foreign key. Also, the table Customer consists of six attributes.



```
SQL> CREATE TABLE CUSTOMER (
         customer_ID INT PRIMARY KEY,
         first_name VARCHAR2(10) NOT NULL,
         last_name VARCHAR2(15) NOT NULL,
         customer_address VARCHAR2(30),
         customer_phone VARCHAR2(15),
         customer_category_Id INT NOT NULL,
         FOREIGN KEY (customer_category_Id) REFERENCES CUSTOMER_CATEGORY(customer_category_Id)
     );

Table created.
```

Figure 13: Creating table "Customer"



```
SQL> DESCRIBE Customer;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_ID                               NOT NULL NUMBER(38)
 FIRST_NAME                                NOT NULL VARCHAR2(10)
 LAST_NAME                                 NOT NULL VARCHAR2(15)
 CUSTOMER_ADDRESS                                   VARCHAR2(30)
 CUSTOMER_PHONE                                     VARCHAR2(15)
 CUSTOMER_CATEGORY_ID                      NOT NULL NUMBER(38)
```

Figure 14: Describing table "Customer"

Purpose	Used Commands

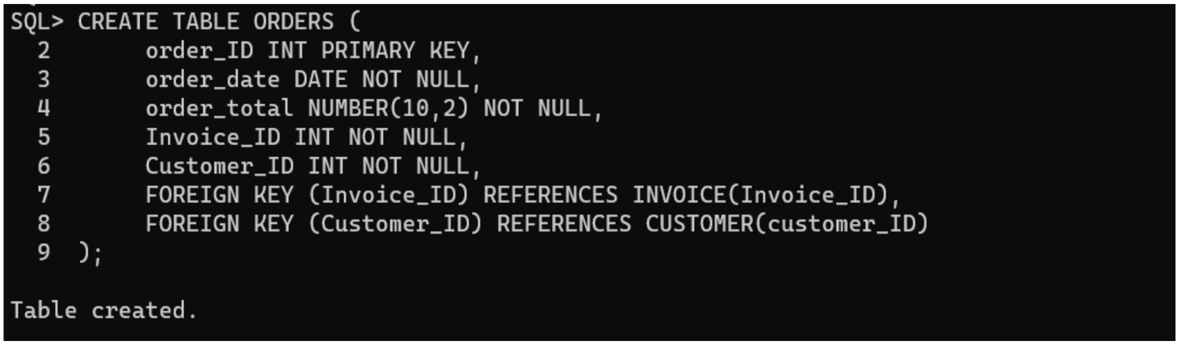
1. Creating a table “CUSTOMER”	CREATE TABLE CUSTOMER (
	    customer_ID INT PRIMARY KEY,
	    first_name VARCHAR2(50) NOT NULL,
	    last_name VARCHAR2(15) NOT NULL,
	    customer_address VARCHAR2(20),
	    customer_phone VARCHAR2(15),
	    customer_category_Id INT,
	    FOREIGN KEY (customer_category_Id)
	    REFERENCES
	    CUSTOMER_CATEGORY (customer_category_Id)
	);

2. Describing	DESCRIBE Customer;

Table 9: Creating and Description of table "Customer"

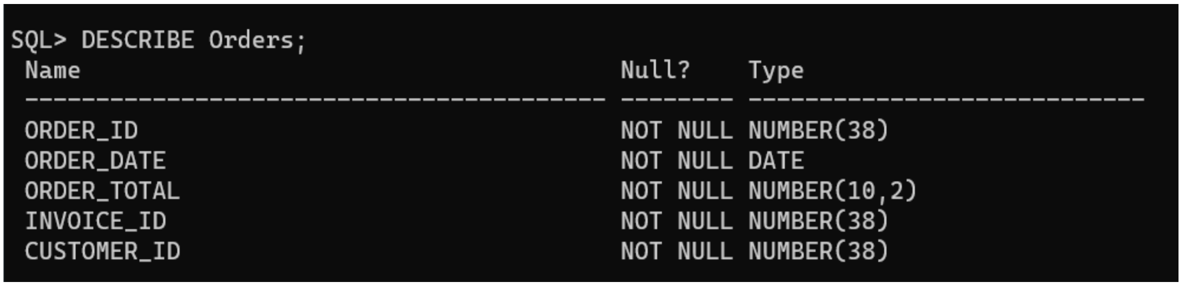
5.1.7 Creating and Describing ORDERS Table

A table named ORDERS is created with Order_ID as its primary key. There is customer _Id and invoice_ID as its foreign key. Also, the table Customer consists of five attributes.



```
SQL> CREATE TABLE ORDERS (
        order_ID INT PRIMARY KEY,
        order_date DATE NOT NULL,
        order_total NUMBER(10,2) NOT NULL,
        Invoice_ID INT NOT NULL,
        Customer_ID INT NOT NULL,
        FOREIGN KEY (Invoice_ID) REFERENCES INVOICE(Invoice_ID),
        FOREIGN KEY (Customer_ID) REFERENCES CUSTOMER(customer_ID)
   );

Table created.
```

Figure 15: Creating table "Orders"



```
SQL> DESCRIBE Orders;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORDER_ID                                  NOT NULL NUMBER(38)
 ORDER_DATE                                NOT NULL DATE
 ORDER_TOTAL                               NOT NULL NUMBER(10,2)
 INVOICE_ID                                NOT NULL NUMBER(38)
 CUSTOMER_ID                               NOT NULL NUMBER(38)
```

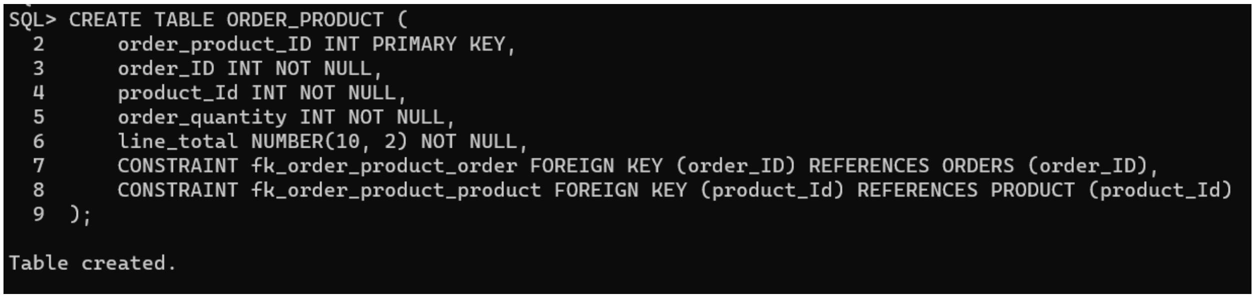
Figure 16: Describing table "Orders"

Purpose	Used Commands
1. Creating a table “ORDERS”	CREATE TABLE ORDERS (order_ID INT PRIMARY KEY, order_date DATE NOT NULL, order_total NUMBER (10, 2) NOT NULL, Invoice_ID INT, Customer_ID INT, FOREIGN KEY (Invoice_ID) REFERENCES INVOICE(Invoice_ID), FOREIGN KEY (Customer_ID) REFERENCES CUSTOMER (customer_ID));
2. Describing	DESCRIBE Orders;

Table 10: Creating and Description of table "Orders"

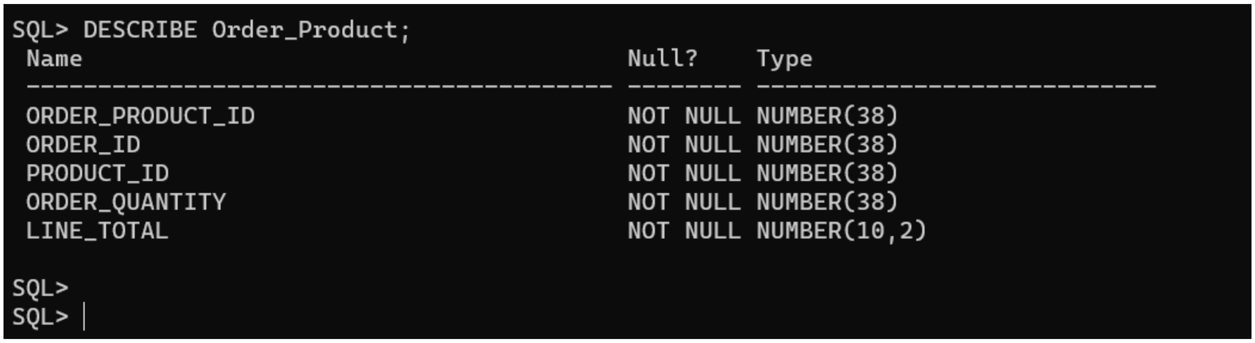
5.18 Creating and describing ORDER_PRODUCT_DETAILS Table

A table named ORDER_PRODUCT_DETAILS is created with Order_product_ID as its primary key. There is product_Id and order_ID as its foreign key. Also, the table Customer consists of five attributes.



```
SQL> CREATE TABLE ORDER_PRODUCT (
         order_product_ID INT PRIMARY KEY,
         order_ID INT NOT NULL,
         product_Id INT NOT NULL,
         order_quantity INT NOT NULL,
         line_total NUMBER(10, 2) NOT NULL,
         CONSTRAINT fk_order_product_order FOREIGN KEY (order_ID) REFERENCES ORDERS (order_ID),
         CONSTRAINT fk_order_product_product FOREIGN KEY (product_Id) REFERENCES PRODUCT (product_Id)
     );

Table created.
```

Figure 17: Creating Table "Order_Product details"



```
SQL> DESCRIBE Order_Product;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORDER_PRODUCT_ID                          NOT NULL NUMBER(38)
 ORDER_ID                                  NOT NULL NUMBER(38)
 PRODUCT_ID                                NOT NULL NUMBER(38)
 ORDER_QUANTITY                            NOT NULL NUMBER(38)
 LINE_TOTAL                                NOT NULL NUMBER(10,2)

SQL>
SQL> |
```

Figure 18: Describing table "Order_Product_Details"

Purpose	Used Commands
1. Creating a table “ORDER_PRODUCT_DETAILS”	CREATE TABLE ORDER_PRODUCT (order_product_ID INT PRIMARY KEY, order_ID INT NOT NULL, product_Id INT NOT NULL, order_quantity INT NOT NULL, line_total NUMBER (10, 2) NOT NULL, CONSTRAINT fk_order_product_order FOREIGN KEY (order_ID) REFERENCES ORDERS (order_ID), CONSTRAINT fk_order_product_product FOREIGN KEY (product_Id) REFERENCES PRODUCT (product_Id));
2. Describing	DESCRIBE Order_product;

Table 11: Creating and Description of table "Order_Product_Details"

5.2 Inserting and Displaying Tables content

5.2.1 Inserting and Displaying INVOICE Table contents

All the required data for the invoice table are entered and a total of ten data for invoice are inserted into the table.

The command to display all the **INVOICE** details in the table is:

“SELECT * FROM Invoice”.

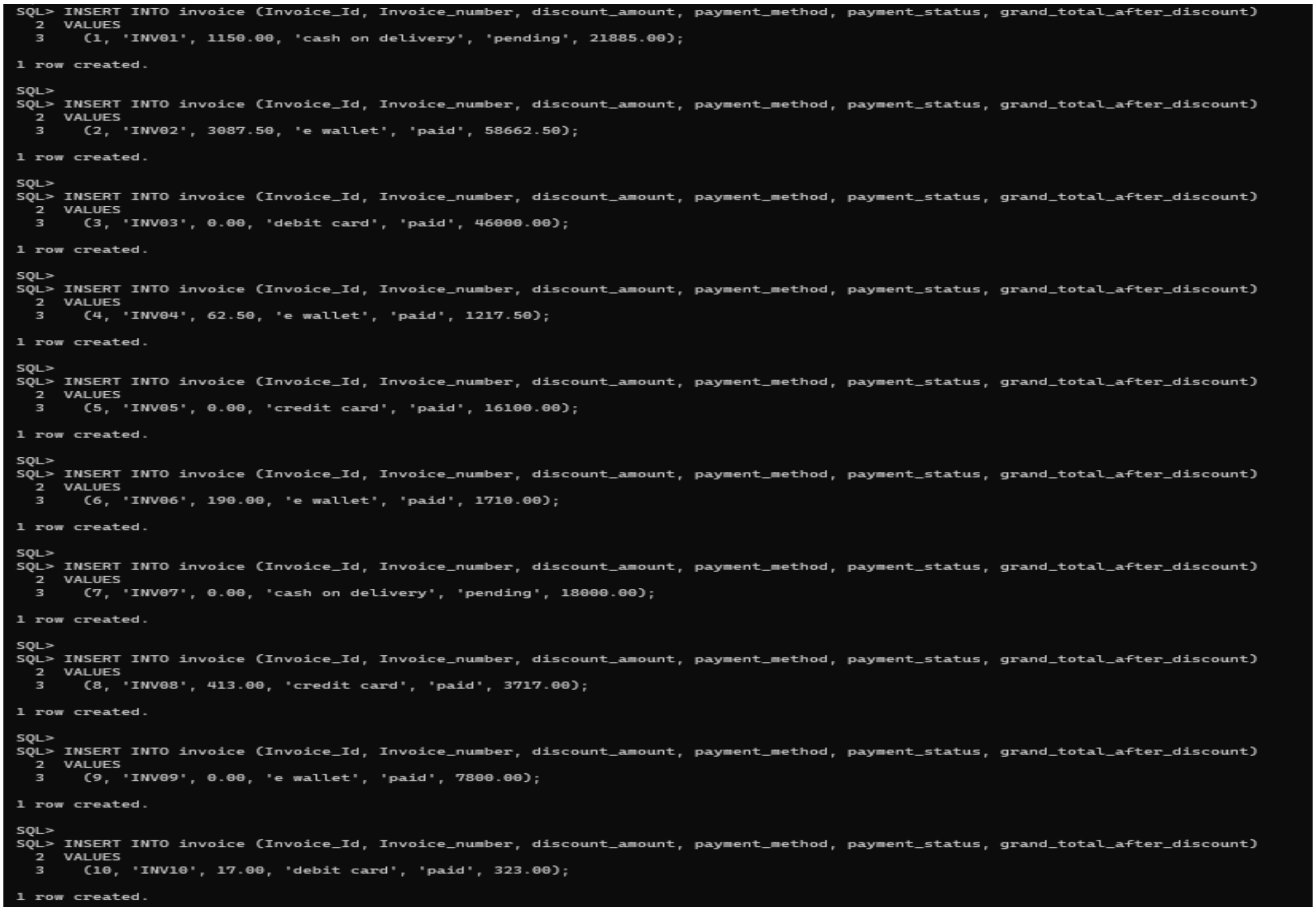
Purpose	Used Commands

1. Inserting values in INVOICE table	INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
	VALUES
	(1, 'INV01', 1150.00, 'cash on delivery', 'pending', 21885.00);
	
	INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
	VALUES
	(2, 'INV02', 3087.50, 'e wallet', 'paid', 58662.50);
	
	INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
	VALUES
	(3, 'INV03', 0.00, 'debit card', 'paid', 46000.00);
	
	INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
	VALUES
	(4, 'INV04', 62.50, 'e wallet', 'paid', 1217.50);

	INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
	VALUES
	(5, 'INV05', 0.00, 'credit card', 'paid', 16100.00);
	
	INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
	VALUES
	(6, 'INV06', 190.00, 'e wallet', 'paid', 1710.00);
	
	INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
	VALUES
	(7, 'INV07', 0.00, 'cash on delivery', 'pending', 18000.00);
	
	INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
	VALUES
	(8, 'INV08', 413.00, 'credit card', 'paid', 3717.00);
	
	INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
	VALUES (9, 'INV09', 0.00, 'e wallet', 'paid', 7800.00);
	
	INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
	VALUES (10, 'INV10', 17.00, 'debit card', 'paid', 323.00);

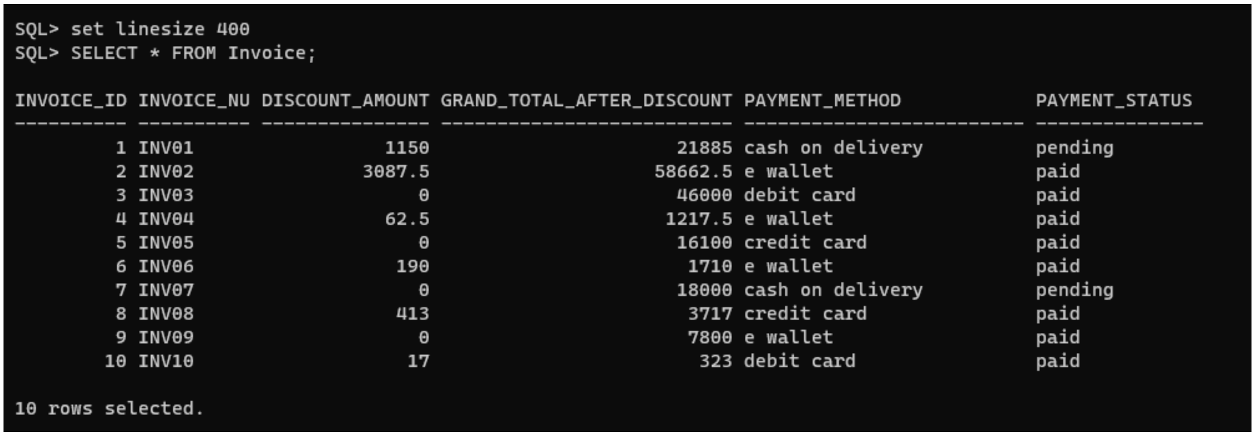
Displaying	SELECT * FROM Invoice;

Table 12: Table showing the command used in inserting values in "INVOICE" table



```
SQL> INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
  2  VALUES
  3    (1, 'INV01', 1150.00, 'cash on delivery', 'pending', 21885.00);

1 row created.

SQL>
SQL> INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
  2  VALUES
  3    (2, 'INV02', 3087.50, 'e wallet', 'paid', 58662.50);

1 row created.

SQL>
SQL> INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
  2  VALUES
  3    (3, 'INV03', 0.00, 'debit card', 'paid', 46000.00);

1 row created.

SQL>
SQL> INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
  2  VALUES
  3    (4, 'INV04', 62.50, 'e wallet', 'paid', 1217.50);

1 row created.

SQL>
SQL> INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
  2  VALUES
  3    (5, 'INV05', 0.00, 'credit card', 'paid', 16100.00);

1 row created.

SQL>
SQL> INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
  2  VALUES
  3    (6, 'INV06', 190.00, 'e wallet', 'paid', 1710.00);

1 row created.

SQL>
SQL> INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
  2  VALUES
  3    (7, 'INV07', 0.00, 'cash on delivery', 'pending', 18000.00);

1 row created.

SQL>
SQL> INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
  2  VALUES
  3    (8, 'INV08', 413.00, 'credit card', 'paid', 3717.00);

1 row created.

SQL>
SQL> INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
  2  VALUES
  3    (9, 'INV09', 0.00, 'e wallet', 'paid', 7800.00);

1 row created.

SQL>
SQL> INSERT INTO invoice (Invoice_Id, Invoice_number, discount_amount, payment_method, payment_status, grand_total_after_discount)
  2  VALUES
  3    (10, 'INV10', 17.00, 'debit card', 'paid', 323.00);

1 row created.
```

Figure 19: Inserting Values in the table "INVOICE”



```
SQL> set linesize 400
SQL> SELECT * FROM Invoice;

INVOICE_ID INVOICE_NU DISCOUNT_AMOUNT GRAND_TOTAL_AFTER_DISCOUNT PAYMENT_METHOD            PAYMENT_STATUS
---------- ---------- --------------- -------------------------- ------------------------- ---------------
         1 INV01                 1150                      21885 cash on delivery          pending
         2 INV02               3087.5                    58662.5 e wallet                  paid
         3 INV03                    0                      46000 debit card                paid
         4 INV04                 62.5                     1217.5 e wallet                  paid
         5 INV05                    0                      16100 credit card               paid
         6 INV06                  190                       1710 e wallet                  paid
         7 INV07                    0                      18000 cash on delivery          pending
         8 INV08                  413                       3717 credit card               paid
         9 INV09                    0                       7800 e wallet                  paid
        10 INV10                   17                        323 debit card                paid

10 rows selected.
```

Figure 20: Displaying Invoice Table Contents

5.2.2 Inserting and Displaying VENDOR Table contents

All the required data for the invoice table are entered and a total of seven data for vendor are inserted into the table.

The command to display all the **VENDOR** details in the table is:

“SELECT * FROM Vendor”.

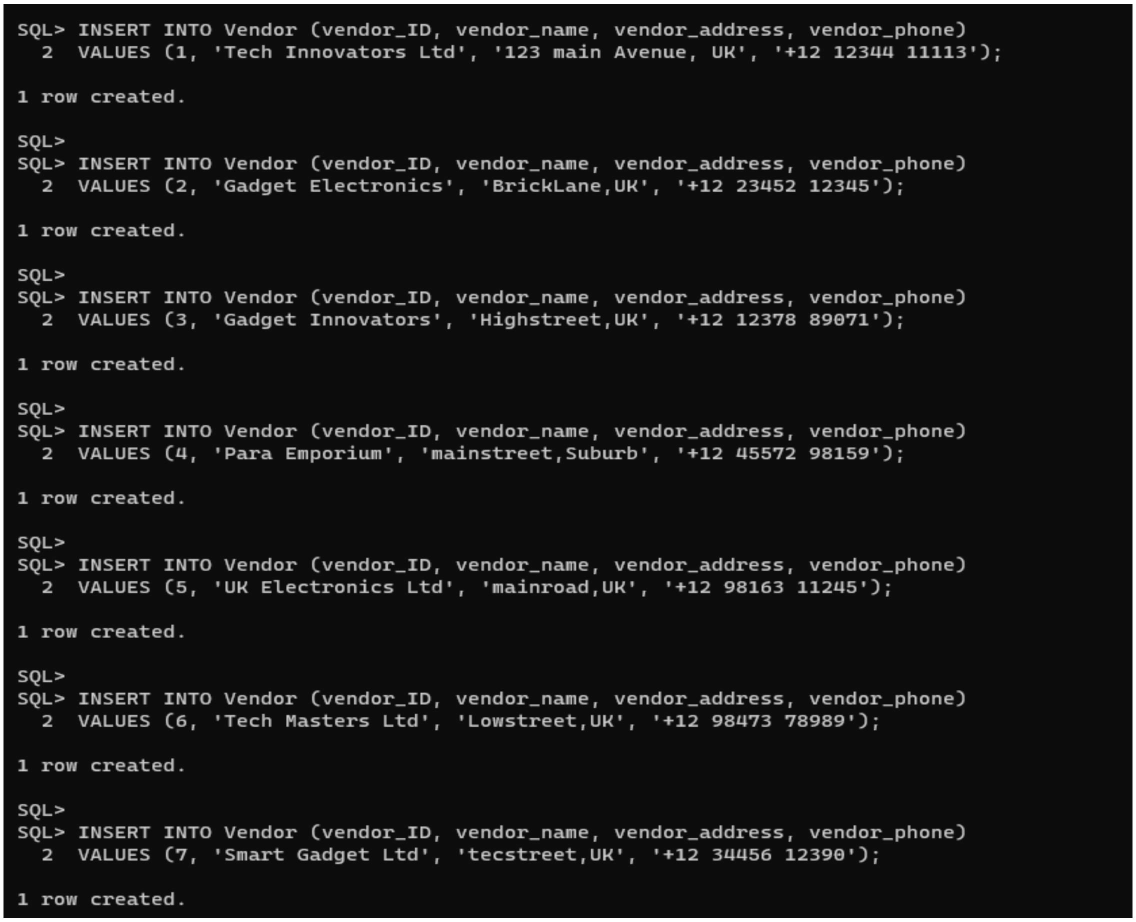
Purpose	Used Commands

1. Inserting values in VENDOR table	INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
	VALUES (1, 'Tech Innovators Ltd', '123 main Avenue, UK', '+12 12344 11113');
	
	INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
	VALUES (2, 'Gadget Electronics', 'BrickLane,UK', '+12 23452 12345');
	
	INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
	VALUES (3, 'Gadget Innovators', 'Highstreet,UK', '+12 12378 89071');
	
	INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
	VALUES (4, 'Para Emporium', 'mainstreet,Suburb', '+12 45572 98159');
	
	INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)

	VALUES (5, 'UK Electronics Ltd', 'mainroad,UK', '+12 98163 11245');
	
	INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
	VALUES (6, 'Tech Masters Ltd', 'Lowstreet,UK', '+12 98473 78989');
	
	INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
	VALUES (7, 'Smart Gadget Ltd', 'tecstreet,UK', '+12 34456 12390');

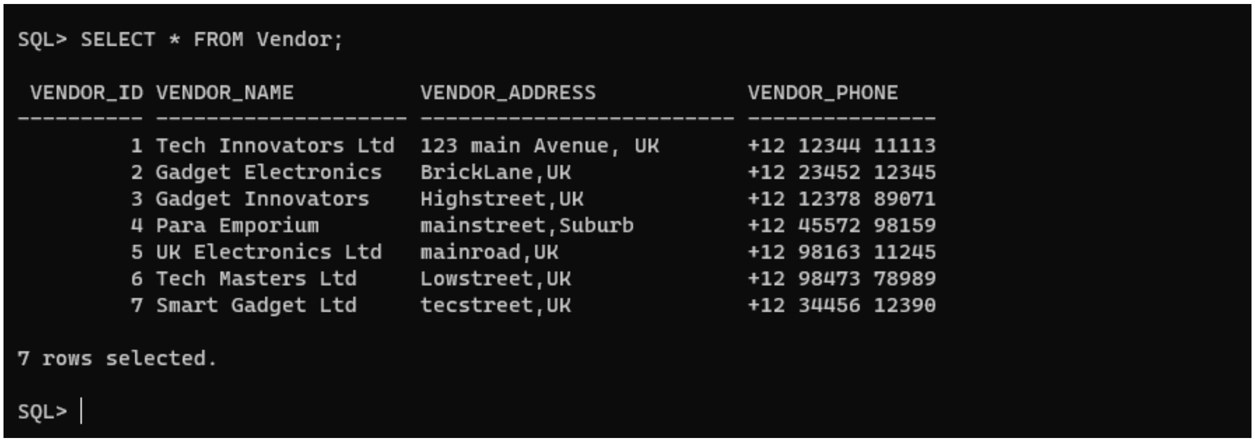
Displaying	SELECT * FROM Vendor;

Table 13: Inserting values in the table "Vendors"



```
SQL> INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
  2  VALUES (1, 'Tech Innovators Ltd', '123 main Avenue, UK', '+12 12344 11113');

1 row created.

SQL>
SQL> INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
  2  VALUES (2, 'Gadget Electronics', 'BrickLane,UK', '+12 23452 12345');

1 row created.

SQL>
SQL> INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
  2  VALUES (3, 'Gadget Innovators', 'Highstreet,UK', '+12 12378 89071');

1 row created.

SQL>
SQL> INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
  2  VALUES (4, 'Para Emporium', 'mainstreet,Suburb', '+12 45572 98159');

1 row created.

SQL>
SQL> INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
  2  VALUES (5, 'UK Electronics Ltd', 'mainroad,UK', '+12 98163 11245');

1 row created.

SQL>
SQL> INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
  2  VALUES (6, 'Tech Masters Ltd', 'Lowstreet,UK', '+12 98473 78989');

1 row created.

SQL>
SQL> INSERT INTO Vendor (vendor_ID, vendor_name, vendor_address, vendor_phone)
  2  VALUES (7, 'Smart Gadget Ltd', 'tecstreet,UK', '+12 34456 12390');

1 row created.
```

Figure 21: Inserting values in the table "Vendor"



```
SQL> SELECT * FROM Vendor;

 VENDOR_ID VENDOR_NAME          VENDOR_ADDRESS            VENDOR_PHONE
---------- -------------------- ------------------------- ---------------
         1 Tech Innovators Ltd  123 main Avenue, UK       +12 12344 11113
         2 Gadget Electronics   BrickLane,UK              +12 23452 12345
         3 Gadget Innovators    Highstreet,UK             +12 12378 89071
         4 Para Emporium        mainstreet,Suburb         +12 45572 98159
         5 UK Electronics Ltd   mainroad,UK               +12 98163 11245
         6 Tech Masters Ltd     Lowstreet,UK              +12 98473 78989
         7 Smart Gadget Ltd     tecstreet,UK              +12 34456 12390

7 rows selected.

SQL> |
```

Figure 22: Displaying Vendor table content

5.2.3 Inserting and displaying CUSTOMER_CATEGORY Table

All the required data for the Customer_category table are entered and a total of three data for customer_category is inserted into the table.

The command to display all the **CUSTOMER_CATEGORY** details in the table is:

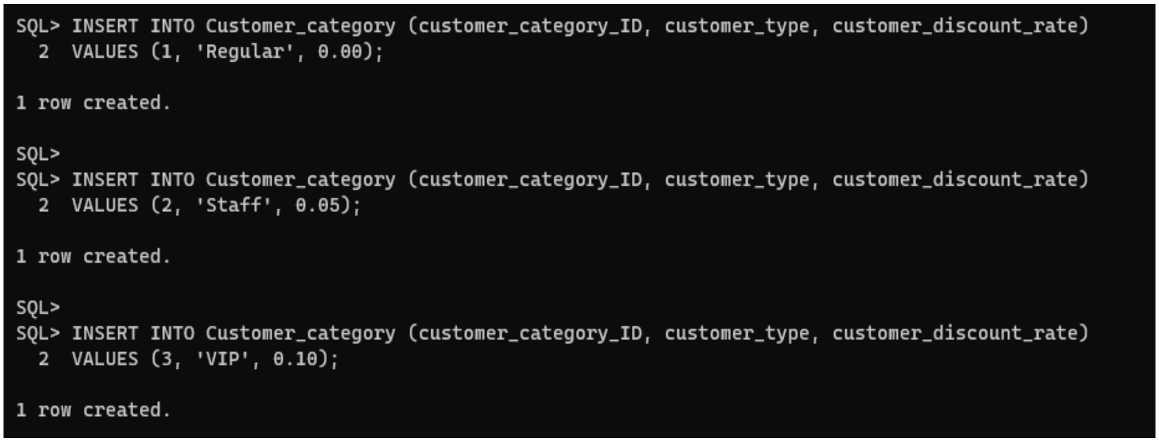
“SELECT * FROM Customer_Category”.

Purpose	Used Commands

1. Inserting values in customer_category table	INSERT INTO CUSTOMER_CATEGORY
	(customer_category_ID, customer_type,
	customer_discount_rate)
	VALUES (1, 'Regular', 0.00);
	
	INSERT INTO CUSTOMER_CATEGORY
	(customer_category_ID, customer_type,
	customer_discount_rate)
	VALUES (2, 'Staff', 0.05);
	
	INSERT INTO CUSTOMER_CATEGORY
	(customer_category_ID, customer_type,
	customer_discount_rate)
	VALUES (3, 'VIP', 0.10);

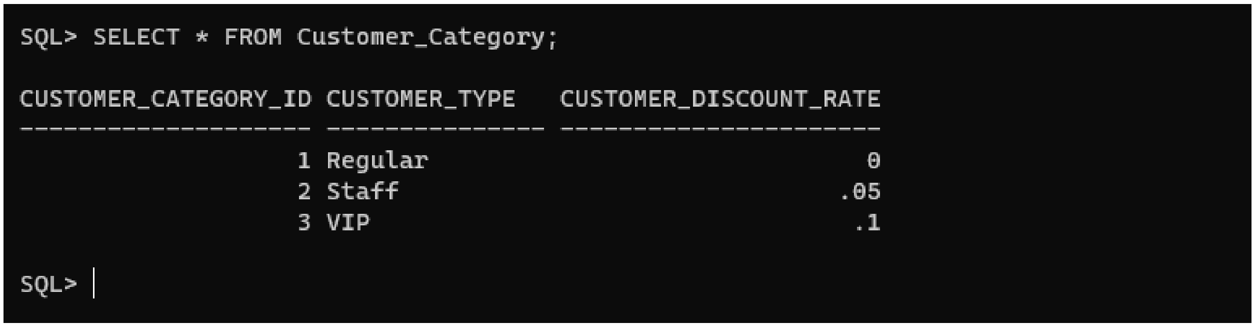
Displaying	SELECT * FROM customer_category;

Table 14: Inserting values in the table "Customer_Category"



```
SQL> INSERT INTO Customer_category (customer_category_ID, customer_type, customer_discount_rate)
  2  VALUES (1, 'Regular', 0.00);

1 row created.

SQL>
SQL> INSERT INTO Customer_category (customer_category_ID, customer_type, customer_discount_rate)
  2  VALUES (2, 'Staff', 0.05);

1 row created.

SQL>
SQL> INSERT INTO Customer_category (customer_category_ID, customer_type, customer_discount_rate)
  2  VALUES (3, 'VIP', 0.10);

1 row created.
```

Figure 23: Inserting values in the table "Customer_Category"



```
SQL> SELECT * FROM Customer_Category;

CUSTOMER_CATEGORY_ID CUSTOMER_TYPE   CUSTOMER_DISCOUNT_RATE
-------------------- --------------- ----------------------
                   1 Regular                              0
                   2 Staff                              .05
                   3 VIP                                 .1

SQL>
```

Figure 24: Displaying Customer_Category table content

5.2.4 Inserting and Displaying PRODUCT_CATEGORY Table

All the required data for the **PRODUCT_CATEGORY** table are entered and a total of eight data for Product_Category are inserted into the table.

The command to display all the Product_Category details in the table is:

“SELECT * FROM Product_category”.

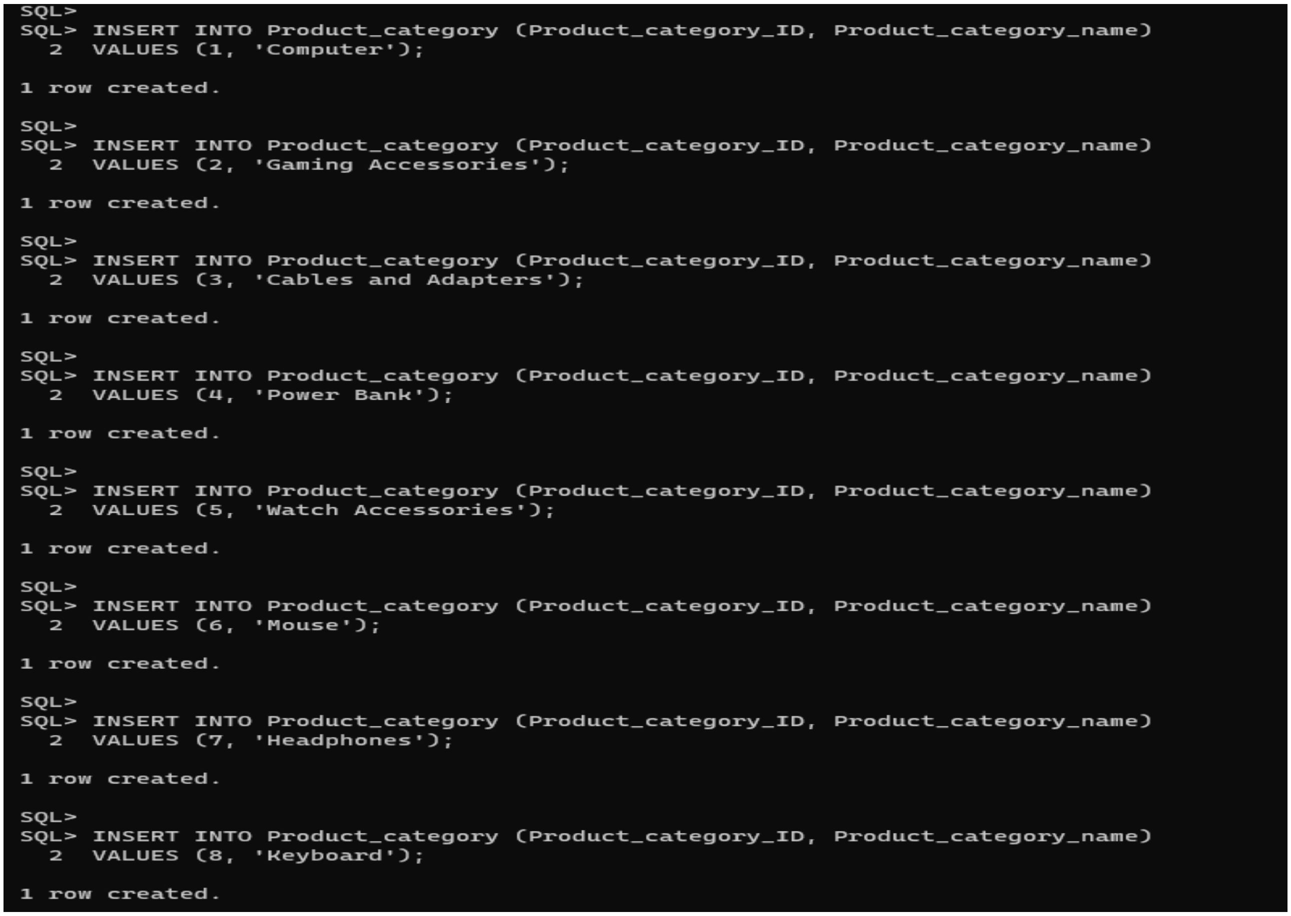
Purpose	Used Commands

1. Inserting values in Product_Category table	INSERT INTO Product_category (Product_category_ID, Product_category_name) VALUES (1, 'Computer');
	INSERT INTO Product_category (Product_category_ID, Product_category_name) VALUES (2, 'Gaming Accessories');
	INSERT INTO Product_category (Product_category_ID, Product_category_name) VALUES (3, 'Cables and Adapters');
	INSERT INTO Product_category (Product_category_ID, Product_category_name) VALUES (4, 'Power Bank');
	INSERT INTO Product_category (Product_category_ID, Product_category_name) VALUES (5, 'Watch Accessories');
	INSERT INTO Product_category (Product_category_ID, Product_category_name)

	VALUES (6, 'Mouse');
	INSERT INTO Product_category
	(Product_category_ID, Product_category_name)
	VALUES (7, 'Headphones');
	INSERT INTO Product_category
	(Product_category_ID, Product_category_name)
	VALUES (8, 'Keyboard');

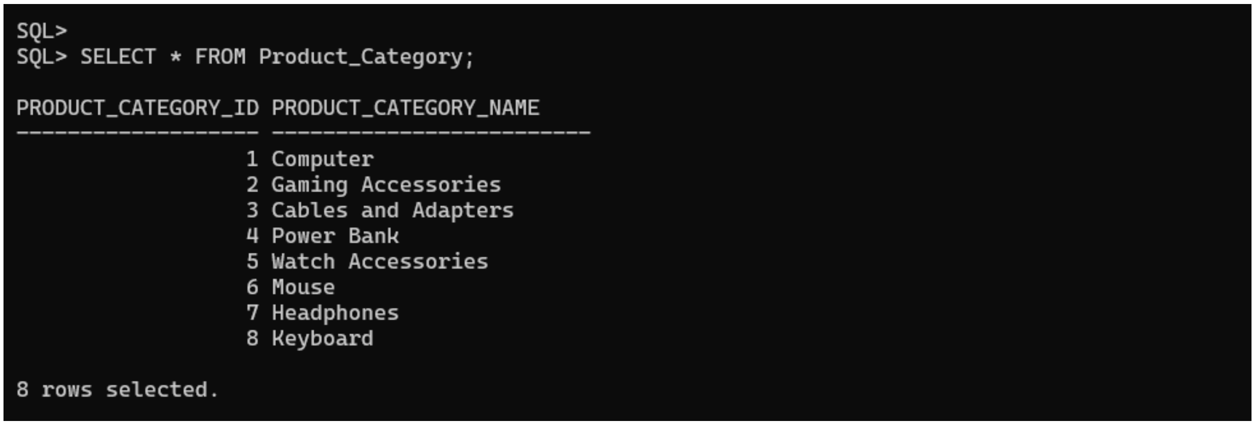
Displaying	SELECT * FROM Product_Category;

Table 15: Inserting values in the table "Product_category"



```
SQL>
SQL> INSERT INTO Product_category (Product_category_ID, Product_category_name)
  2  VALUES (1, 'Computer');

1 row created.

SQL>
SQL> INSERT INTO Product_category (Product_category_ID, Product_category_name)
  2  VALUES (2, 'Gaming Accessories');

1 row created.

SQL>
SQL> INSERT INTO Product_category (Product_category_ID, Product_category_name)
  2  VALUES (3, 'Cables and Adapters');

1 row created.

SQL>
SQL> INSERT INTO Product_category (Product_category_ID, Product_category_name)
  2  VALUES (4, 'Power Bank');

1 row created.

SQL>
SQL> INSERT INTO Product_category (Product_category_ID, Product_category_name)
  2  VALUES (5, 'Watch Accessories');

1 row created.

SQL>
SQL> INSERT INTO Product_category (Product_category_ID, Product_category_name)
  2  VALUES (6, 'Mouse');

1 row created.

SQL>
SQL> INSERT INTO Product_category (Product_category_ID, Product_category_name)
  2  VALUES (7, 'Headphones');

1 row created.

SQL>
SQL> INSERT INTO Product_category (Product_category_ID, Product_category_name)
  2  VALUES (8, 'Keyboard');

1 row created.
```

Figure 25: Inserting values in the table "Product_Category"



```
SQL>
SQL> SELECT * FROM Product_Category;

PRODUCT_CATEGORY_ID PRODUCT_CATEGORY_NAME
------------------- -------------------------
                  1 Computer
                  2 Gaming Accessories
                  3 Cables and Adapters
                  4 Power Bank
                  5 Watch Accessories
                  6 Mouse
                  7 Headphones
                  8 Keyboard

8 rows selected.
```

Figure 26: Displaying Product_Category table content

5.2.5 Inserting and Displaying PRODUCT Table

All the required data for the **PRODUCT** table are entered and a total of ten data for Product are inserted into the table.

The command to display all the PRODUCT details in the table is:

“SELECT * FROM Product”.

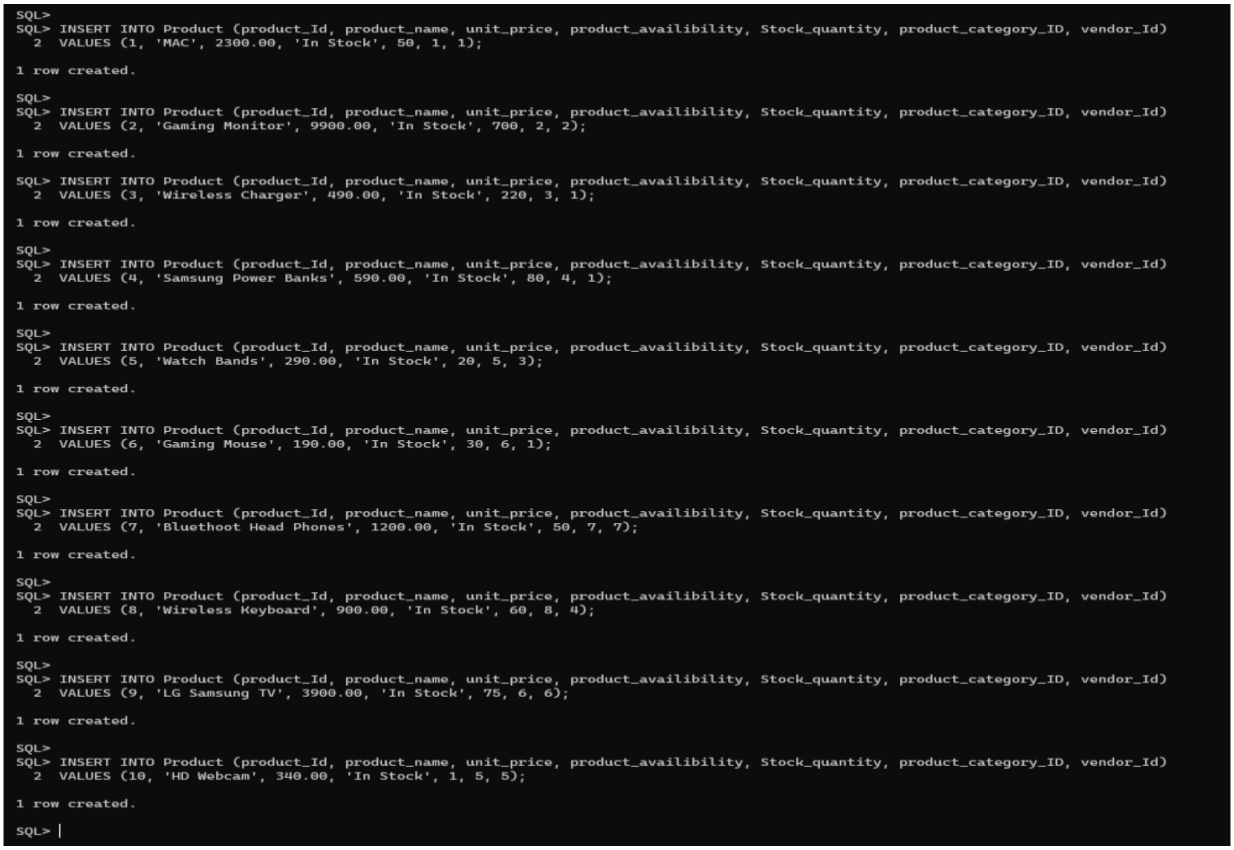
Purpose	Used Commands

1. Inserting values in Product table	INSERT INTO PRODUCT (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, Vendor_Id)
	VALUES (1, 'MAC', 2300.00, 'In Stock', 50, 1, 1);
	
	INSERT INTO PRODUCT (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, Vendor_Id)
	VALUES (2, 'Gaming Monitor', 9900.00, 'In Stock', 700, 2, 2);
	
	INSERT INTO PRODUCT (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, Vendor_Id)
	VALUES (3, 'Wireless Charger', 490.00, 'In Stock', 220, 3, 1);
	
	INSERT INTO PRODUCT (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, Vendor_Id)
	VALUES (4, 'Samsung Power Banks', 590.00, 'In Stock', 80, 4, 1);

	INSERT INTO PRODUCT (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, Vendor_Id) VALUES (5, 'Watch Bands', 290.00, 'In Stock', 20, 5, 3);
	INSERT INTO PRODUCT (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, Vendor_Id) VALUES (6, 'Gaming Mouse', 190.00, 'In Stock', 30, 6, 1);
	INSERT INTO PRODUCT (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, Vendor_Id) VALUES (7, 'Bluetooth Headphones', 1200.00, 'In Stock', 50, 7, 7);
	INSERT INTO PRODUCT (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, Vendor_Id) VALUES (8, 'Wireless Keyboard', 900.00, 'In Stock', 60, 8, 4);
	INSERT INTO PRODUCT (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, Vendor_Id) VALUES (9, 'LG Samsung TV', 3900.00, 'In Stock', 75, 6, 6);
	INSERT INTO PRODUCT (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, Vendor_Id) VALUES (10, 'HD Webcam', 340.00, 'In Stock', 1, 5, 5);

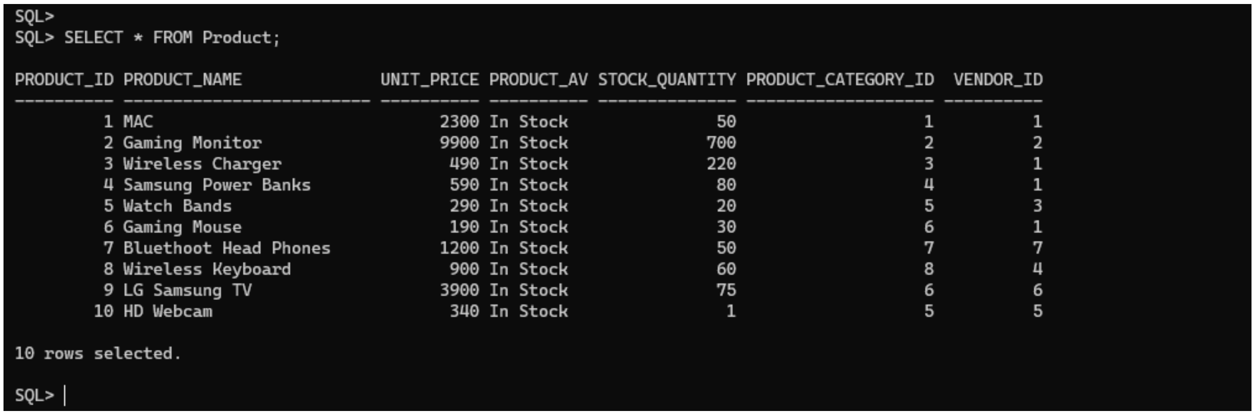
Displaying	SELECT * FROM product;

Figure 27: Inserting values in the table "Product"



```
SQL>
SQL> INSERT INTO Product (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, vendor_Id)
  2  VALUES (1, 'MAC', 2300.00, 'In Stock', 50, 1, 1);

1 row created.

SQL>
SQL> INSERT INTO Product (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, vendor_Id)
  2  VALUES (2, 'Gaming Monitor', 9900.00, 'In Stock', 700, 2, 2);

1 row created.

SQL> INSERT INTO Product (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, vendor_Id)
  2  VALUES (3, 'Wireless Charger', 490.00, 'In Stock', 220, 3, 1);

1 row created.

SQL>
SQL> INSERT INTO Product (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, vendor_Id)
  2  VALUES (4, 'Samsung Power Banks', 590.00, 'In Stock', 80, 4, 1);

1 row created.

SQL>
SQL> INSERT INTO Product (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, vendor_Id)
  2  VALUES (5, 'Watch Bands', 290.00, 'In Stock', 20, 5, 3);

1 row created.

SQL>
SQL> INSERT INTO Product (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, vendor_Id)
  2  VALUES (6, 'Gaming Mouse', 190.00, 'In Stock', 30, 6, 1);

1 row created.

SQL>
SQL> INSERT INTO Product (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, vendor_Id)
  2  VALUES (7, 'Bluethoot Head Phones', 1200.00, 'In Stock', 50, 7, 7);

1 row created.

SQL>
SQL> INSERT INTO Product (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, vendor_Id)
  2  VALUES (8, 'Wireless Keyboard', 900.00, 'In Stock', 60, 8, 4);

1 row created.

SQL>
SQL> INSERT INTO Product (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, vendor_Id)
  2  VALUES (9, 'LG Samsung TV', 3900.00, 'In Stock', 75, 6, 6);

1 row created.

SQL>
SQL> INSERT INTO Product (product_Id, product_name, unit_price, product_availibility, Stock_quantity, product_category_ID, vendor_Id)
  2  VALUES (10, 'HD Webcam', 340.00, 'In Stock', 1, 5, 5);

1 row created.

SQL> |
```

Figure 28: Inserting values in the table "Product "



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

PRODUCT_ID PRODUCT_NAME              UNIT_PRICE PRODUCT_AV STOCK_QUANTITY PRODUCT_CATEGORY_ID  VENDOR_ID
---------- ------------------------- ---------- ---------- -------------- -------------------- ----------
         1 MAC                             2300 In Stock               50                    1          1
         2 Gaming Monitor                  9900 In Stock              700                    2          2
         3 Wireless Charger                 490 In Stock              220                    3          1
         4 Samsung Power Banks              590 In Stock               80                    4          1
         5 Watch Bands                      290 In Stock               20                    5          3
         6 Gaming Mouse                     190 In Stock               30                    6          1
         7 Bluethoot Head Phones           1200 In Stock               50                    7          7
         8 Wireless Keyboard                900 In Stock               60                    8          4
         9 LG Samsung TV                   3900 In Stock               75                    6          6
        10 HD Webcam                        340 In Stock                1                    5          5

10 rows selected.

SQL> |
```

Figure 29: Displaying Product table content

5.2.6 Inserting Values and Displaying CUSTOMER table

All the required data for the customer table are entered and a total of seven data for Customers are inserted into the table.

The command to display all the CUSTOMER details in the table is:

“SELECT * FROM Customer”.

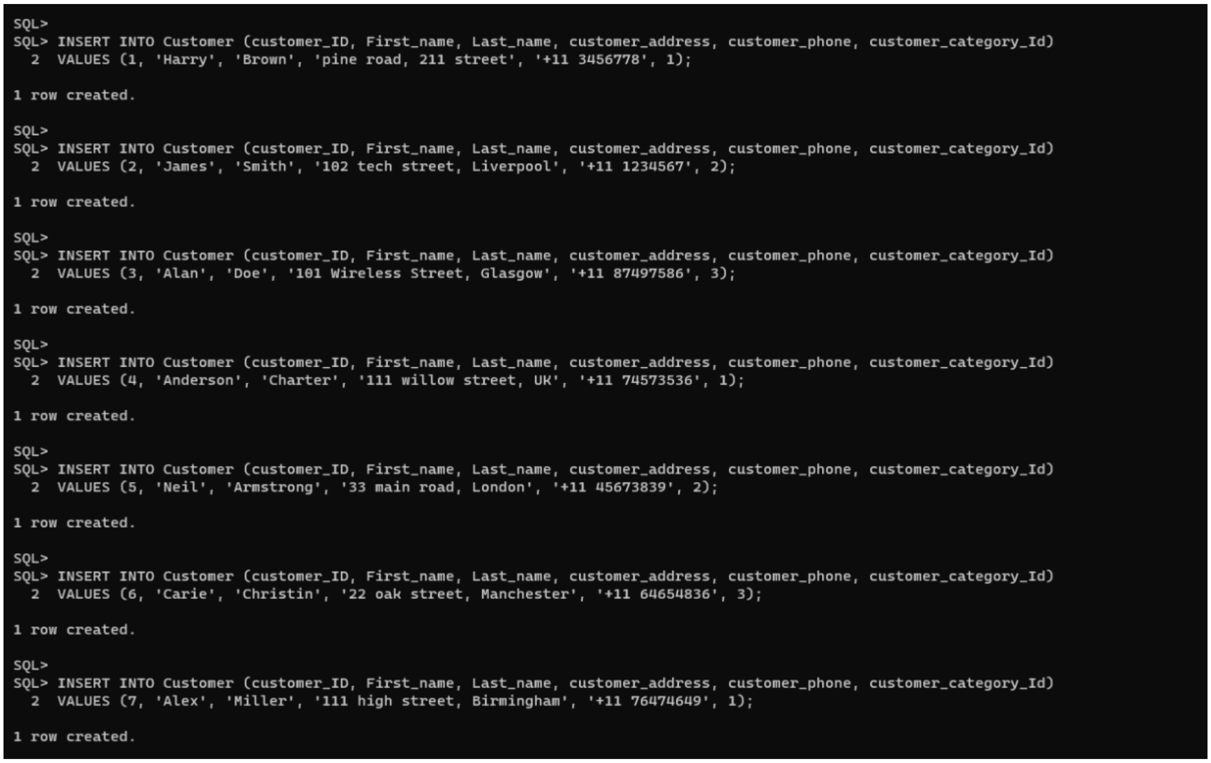
Purpose	Used Commands

1. Inserting values in Customer table	INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id) VALUES (1, 'Harry', 'Brown', 'pine road, 211 street', '+11 3456778', 1);
	INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id) VALUES (2, 'James', 'Smith', '102 tech street, Liverpool', '+11 1234567', 2);
	INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id) VALUES (3, 'Alan', 'Doe', '101 Wireless Street, Glasgow', '+11 87497586', 3);
	INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)

	VALUES (4, 'Anderson', 'Charter', '111 willow street, UK', '+11 74573536', 1);
	
	INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
	VALUES (5, 'Neil', 'Armstrong', '33 main road, London', '+11 45673839', 2);
	
	INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
	VALUES (6, 'Carie', 'Christin', '22 oak street, Manchester', '+11 64654836', 3);
	
	INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
	VALUES (7, 'Alex', 'Miller', '111 high street, Birmingham', '+11 76474649', 1);

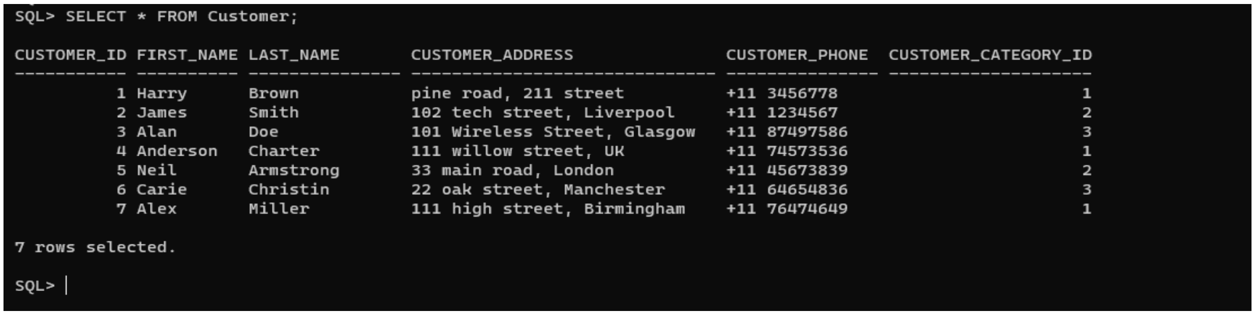
Displaying	SELECT * FROM Customer;

Table 16: Inserting values in the table "Customer"



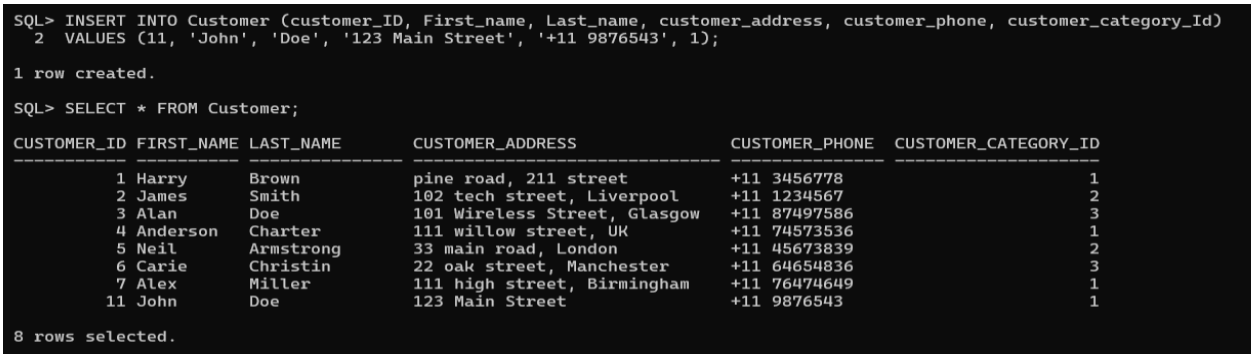
```
SQL>
SQL> INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
  2  VALUES (1, 'Harry', 'Brown', 'pine road, 211 street', '+11 3456778', 1);

1 row created.

SQL>
SQL> INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
  2  VALUES (2, 'James', 'Smith', '102 tech street, Liverpool', '+11 1234567', 2);

1 row created.

SQL>
SQL> INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
  2  VALUES (3, 'Alan', 'Doe', '101 Wireless Street, Glasgow', '+11 87497586', 3);

1 row created.

SQL>
SQL> INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
  2  VALUES (4, 'Anderson', 'Charter', '111 willow street, UK', '+11 74573536', 1);

1 row created.

SQL>
SQL> INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
  2  VALUES (5, 'Neil', 'Armstrong', '33 main road, London', '+11 45673839', 2);

1 row created.

SQL>
SQL> INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
  2  VALUES (6, 'Carie', 'Christin', '22 oak street, Manchester', '+11 64654836', 3);

1 row created.

SQL>
SQL> INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
  2  VALUES (7, 'Alex', 'Miller', '111 high street, Birmingham', '+11 76474649', 1);

1 row created.
```

Figure 30: Inserting values in the table "Customer"



```
SQL> SELECT * FROM Customer;

CUSTOMER_ID FIRST_NAME LAST_NAME       CUSTOMER_ADDRESS               CUSTOMER_PHONE  CUSTOMER_CATEGORY_ID
----------- ---------- --------------- ------------------------------ --------------- --------------------
          1 Harry      Brown           pine road, 211 street          +11 3456778                        1
          2 James      Smith           102 tech street, Liverpool     +11 1234567                        2
          3 Alan       Doe             101 Wireless Street, Glasgow   +11 87497586                       3
          4 Anderson   Charter         111 willow street, UK          +11 74573536                       1
          5 Neil       Armstrong       33 main road, London           +11 45673839                       2
          6 Carie      Christin        22 oak street, Manchester      +11 64654836                       3
          7 Alex       Miller          111 high street, Birmingham    +11 76474649                       1

7 rows selected.

SQL> |
```

Figure 31: Displaying Customer table content



```
SQL> INSERT INTO Customer (customer_ID, First_name, Last_name, customer_address, customer_phone, customer_category_Id)
  2  VALUES (11, 'John', 'Doe', '123 Main Street', '+11 9876543', 1);

1 row created.

SQL> SELECT * FROM Customer;

CUSTOMER_ID FIRST_NAME LAST_NAME       CUSTOMER_ADDRESS               CUSTOMER_PHONE  CUSTOMER_CATEGORY_ID
----------- ---------- --------------- ------------------------------ --------------- --------------------
          1 Harry      Brown           pine road, 211 street          +11 3456778                        1
          2 James      Smith           102 tech street, Liverpool     +11 1234567                        2
          3 Alan       Doe             101 Wireless Street, Glasgow   +11 87497586                       3
          4 Anderson   Charter         111 willow street, UK          +11 74573536                       1
          5 Neil       Armstrong       33 main road, London           +11 45673839                       2
          6 Carie      Christin        22 oak street, Manchester      +11 64654836                       3
          7 Alex       Miller          111 high street, Birmingham    +11 76474649                       1
         11 John       Doe             123 Main Street                +11 9876543                        1

8 rows selected.
```

Figure 32: Inserting one more value to customer and displaying its details

5.2.7 Inserting Value and Displaying ORDERS Table

All the required data for the **ORDERS** table are entered and a total of ten data for orders are inserted into the table.

The command to display all the Orders in the table is:

“SELECT * FROM Orders”.

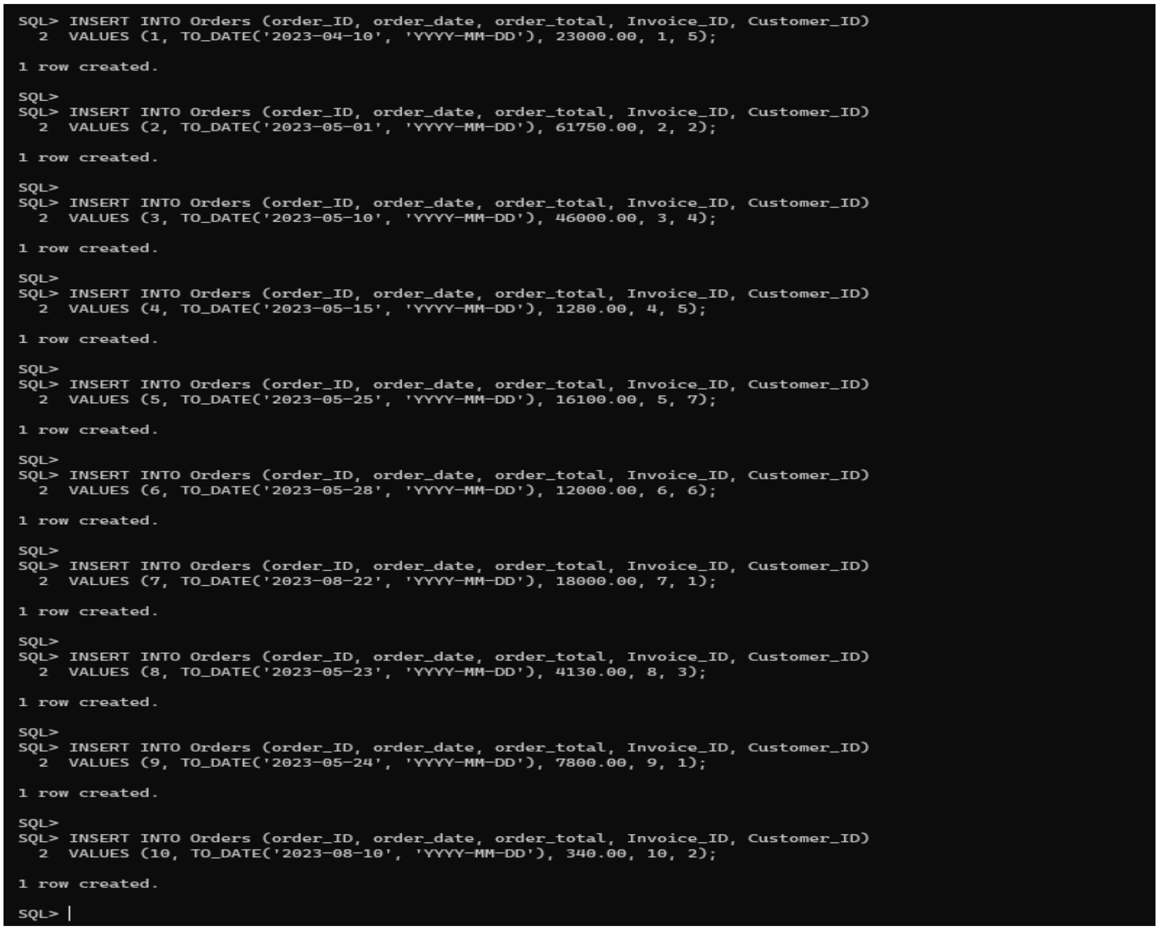
Purpose	Used Commands

1. Inserting values in Order table	INSERT INTO Orders (order_ID, order_date, order_total, Invoice_ID, Customer_ID)
	VALUES (1, TO_DATE('2023-04-10', 'YYYY-MM-DD'), 23000.00, 1, 5);
	
	INSERT INTO Orders (order_ID, order_date, order_total, Invoice_ID, Customer_ID)
	VALUES (2, TO_DATE('2023-05-01', 'YYYY-MM-DD'), 61750.00, 2, 2);
	
	INSERT INTO Orders (order_ID, order_date, order_total, Invoice_ID, Customer_ID)
	VALUES (3, TO_DATE('2023-05-10', 'YYYY-MM-DD'), 46000.00, 3, 4);
	
	INSERT INTO Orders (order_ID, order_date, order_total, Invoice_ID, Customer_ID)
	VALUES (4, TO_DATE('2023-05-15', 'YYYY-MM-DD'), 1280.00, 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'), 16100.00, 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'), 12000.00, 6, 6);
	
	INSERT INTO Orders (order_ID, order_date, order_total, Invoice_ID, Customer_ID)
	VALUES (7, TO_DATE('2023-08-22', 'YYYY-MM-DD'), 18000.00, 7, 1);
	
	INSERT INTO Orders (order_ID, order_date, order_total, Invoice_ID, Customer_ID)
	VALUES (8, TO_DATE('2023-05-23', 'YYYY-MM-DD'), 4130.00, 8, 3);
	
	INSERT INTO Orders (order_ID, order_date, order_total, Invoice_ID, Customer_ID)
	VALUES (9, TO_DATE('2023-05-24', 'YYYY-MM-DD'), 7800.00, 9, 1);
	
	INSERT INTO Orders (order_ID, order_date, order_total, Invoice_ID, Customer_ID)
	VALUES (10, TO_DATE('2023-08-10', 'YYYY-MM-DD'), 340.00, 10, 2);

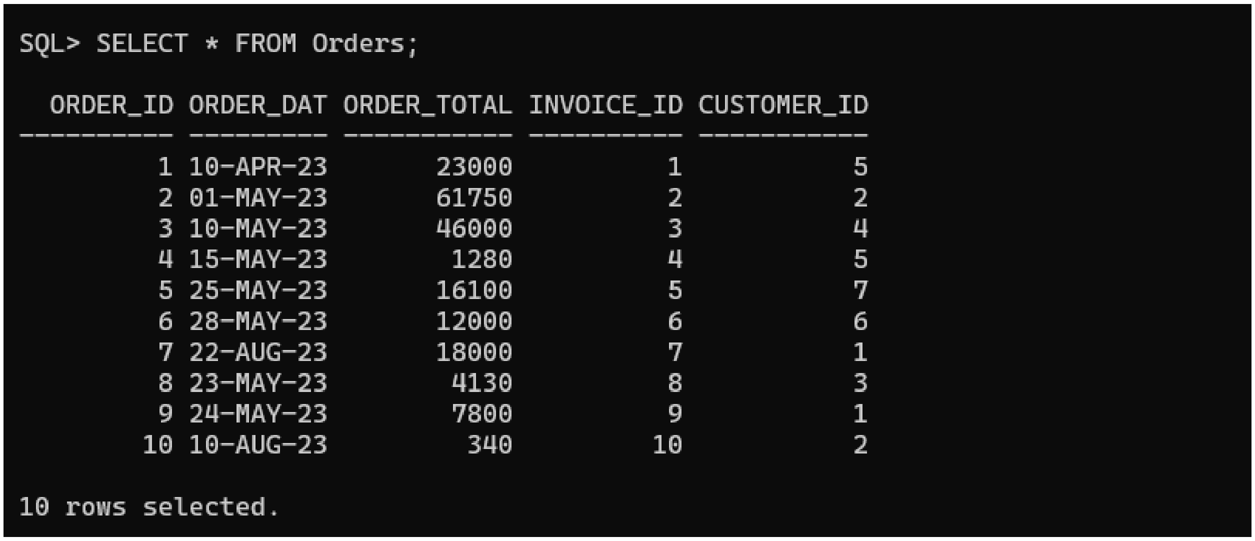
Displaying	SELECT * FROM Orders;

Figure 33: Inserting values in the table "Orders"



```
SQL> INSERT INTO Orders (order_ID, order_date, order_total, Invoice_ID, Customer_ID)
  2  VALUES (1, TO_DATE('2023-04-10', 'YYYY-MM-DD'), 23000.00, 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('2023-05-01', 'YYYY-MM-DD'), 61750.00, 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('2023-05-10', 'YYYY-MM-DD'), 46000.00, 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('2023-05-15', 'YYYY-MM-DD'), 1280.00, 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'), 16100.00, 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'), 12000.00, 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('2023-08-22', 'YYYY-MM-DD'), 18000.00, 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('2023-05-23', 'YYYY-MM-DD'), 4130.00, 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('2023-05-24', 'YYYY-MM-DD'), 7800.00, 9, 1);

1 row created.

SQL>
SQL> INSERT INTO Orders (order_ID, order_date, order_total, Invoice_ID, Customer_ID)
  2  VALUES (10, TO_DATE('2023-08-10', 'YYYY-MM-DD'), 340.00, 10, 2);

1 row created.

SQL> |
```

Figure 34: Inserting values in the table "Orders"



```
SQL> SELECT * FROM Orders;

  ORDER_ID ORDER_DAT ORDER_TOTAL INVOICE_ID CUSTOMER_ID
---------- --------- ----------- ---------- -----------
         1 10-APR-23       23000          1           5
         2 01-MAY-23       61750          2           2
         3 10-MAY-23       46000          3           4
         4 15-MAY-23        1280          4           5
         5 25-MAY-23       16100          5           7
         6 28-MAY-23       12000          6           6
         7 22-AUG-23       18000          7           1
         8 23-MAY-23        4130          8           3
         9 24-MAY-23        7800          9           1
        10 10-AUG-23         340         10           2

10 rows selected.
```

Figure 35: Displaying Orders table content

5.2.8 Inserting Value and Displaying ORDER_PRODUCT DETAILS Table

All the required data for the **ORDER_PRODUCT_DETAILS** table are entered and a total of twelve data for orders_product_details are inserted into the table.

The command to display all the ORDER_PRODUCT_DETAILS in the table is:

“SELECT * FROM Order_product”

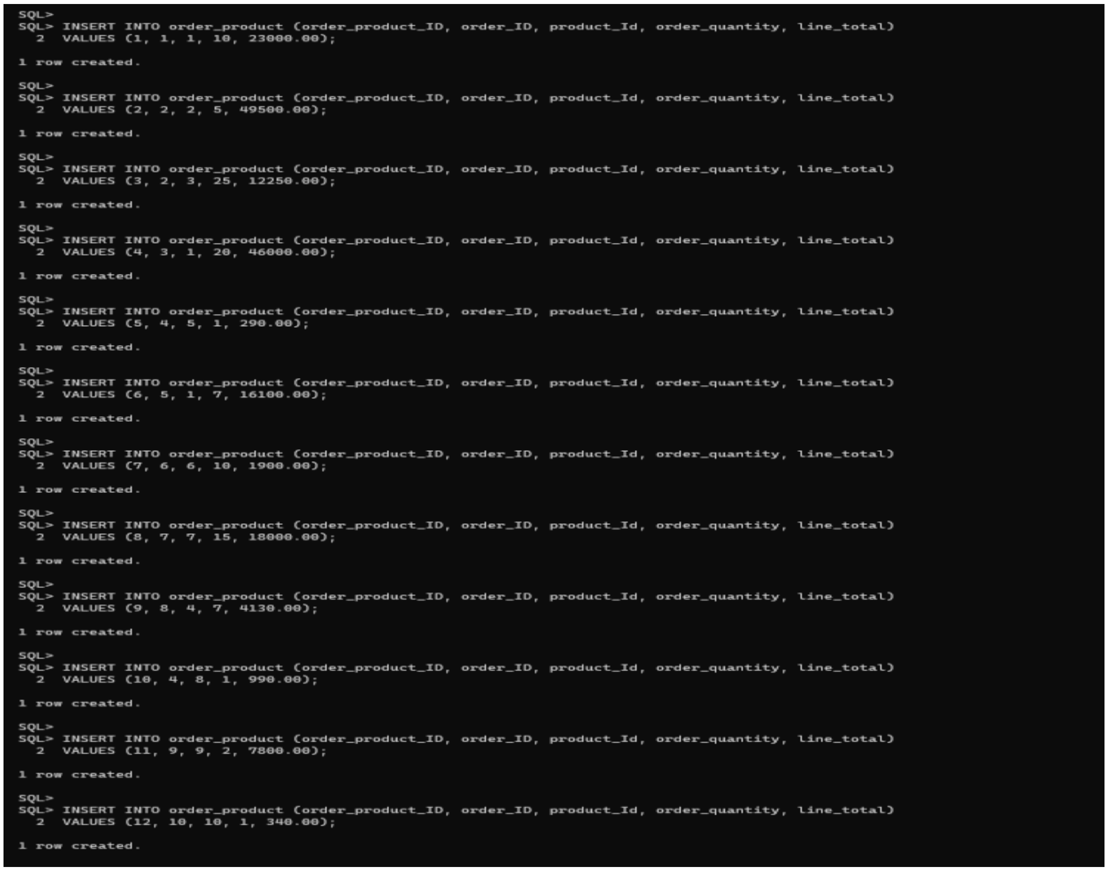
Purpose	Used Commands

1. Inserting values in Order table	INSERT INTO ORDER_PRODUCT (order_product_ID, order_ID, product_Id, order_quantity, line_total)
	VALUES (1, 1, 1, 10, 23000.00);
	
	INSERT INTO ORDER_PRODUCT (order_product_ID, order_ID, product_Id, order_quantity, line_total)
	VALUES (2, 2, 2, 5, 49500.00);
	
	INSERT INTO ORDER_PRODUCT (order_product_ID, order_ID, product_Id, order_quantity, line_total)
	VALUES (3, 2, 3, 25, 12250.00);
	
	INSERT INTO ORDER_PRODUCT (order_product_ID, order_ID, product_Id, order_quantity, line_total)
	VALUES (4, 3, 1, 20, 11800.00);
	
	INSERT INTO ORDER_PRODUCT (order_product_ID, order_ID, product_Id, order_quantity, line_total)
	VALUES (5, 4, 5, 1, 290.00);

	INSERT INTO ORDER_PRODUCT (order_product_ID,
	order_ID, product_Id, order_quantity, line_total)
	VALUES (6, 5, 1, 7, 1330.00);
	
	INSERT INTO ORDER_PRODUCT (order_product_ID,
	order_ID, product_Id, order_quantity, line_total)
	VALUES (7, 6, 6, 10, 12000.00);
	
	INSERT INTO ORDER_PRODUCT (order_product_ID,
	order_ID, product_Id, order_quantity, line_total)
	VALUES (8, 7, 7, 15, 13500.00);
	
	INSERT INTO ORDER_PRODUCT (order_product_ID,
	order_ID, product_Id, order_quantity, line_total)
	VALUES (9, 8, 4, 7, 27300.00);
	
	INSERT INTO ORDER_PRODUCT (order_product_ID,
	order_ID, product_Id, order_quantity, line_total)
	VALUES (10, 4, 8, 1, 340.00);
	
	INSERT INTO ORDER_PRODUCT (order_product_ID,
	order_ID, product_Id, order_quantity, line_total)
	VALUES (11, 9, 9, 2, 82000.00);
	
	INSERT INTO ORDER_PRODUCT (order_product_ID,
	order_ID, product_Id, order_quantity, line_total)
	VALUES (12, 10, 10, 1, 10000.00);

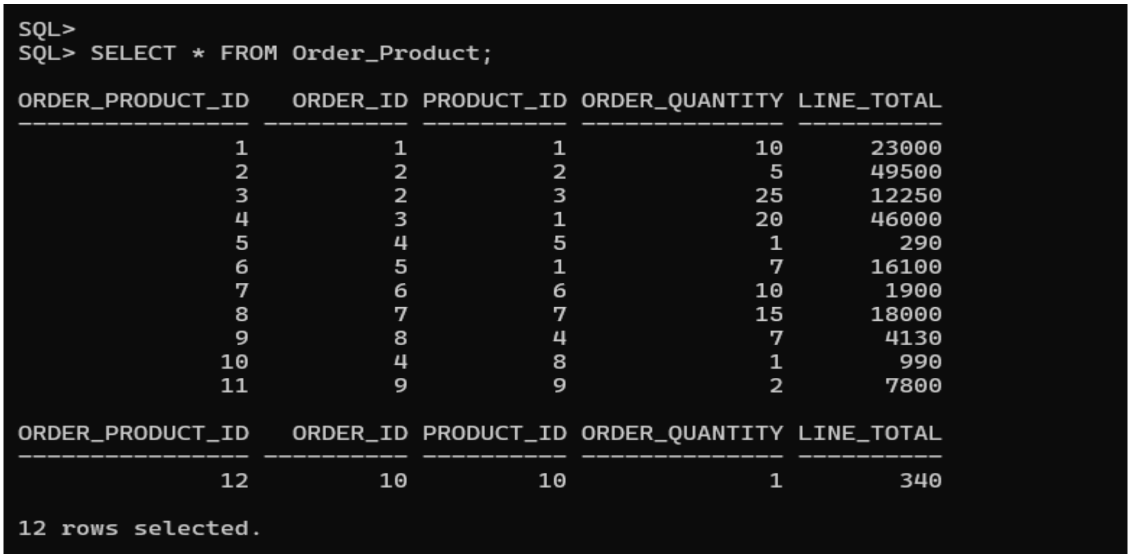
Displaying	SELECT * FROM Order_Product;

Figure 36: Inserting values in the table "Orders_Product_details"



```
SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (1, 1, 1, 10, 23000.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (2, 2, 2, 5, 49500.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (3, 2, 3, 25, 12250.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (4, 3, 1, 20, 46000.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (5, 4, 5, 1, 290.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (6, 5, 1, 7, 16100.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (7, 6, 6, 10, 1900.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (8, 7, 7, 15, 18000.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (9, 8, 4, 7, 4130.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (10, 4, 8, 1, 990.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (11, 9, 9, 2, 7800.00);

1 row created.

SQL>
SQL> INSERT INTO order_product (order_product_ID, order_ID, product_Id, order_quantity, line_total)
  2  VALUES (12, 10, 10, 1, 340.00);

1 row created.
```

Figure 37: Inserting values in the table "Orders_Product_Details"



```
SQL>
SQL> SELECT * FROM Order_Product;

ORDER_PRODUCT_ID   ORDER_ID PRODUCT_ID ORDER_QUANTITY LINE_TOTAL
---------------- ---------- ---------- -------------- ----------
               1          1          1             10      23000
               2          2          2              5      49500
               3          2          3             25      12250
               4          3          1             20      46000
               5          4          5              1        290
               6          5          1              7      16100
               7          6          6             10       1900
               8          7          7             15      18000
               9          8          4              7       4130
              10          4          8              1        990
              11          9          9              2       7800

ORDER_PRODUCT_ID   ORDER_ID PRODUCT_ID ORDER_QUANTITY LINE_TOTAL
---------------- ---------- ---------- -------------- ----------
              12         10         10              1        340

12 rows selected.
```

Figure 38: Displaying Order_Product details table content

5.3 Entities and Its Relation Explanations

The entities included in the database above are Customer, Vendor, Orders, Invoice, customer_categoty, product, product_category and Order_Prodcucts details. The entity customer consists of attributes like customer_ID, customer_name and Customer_category_ID, that gives the details of the customer and the discount rate provided to the customers can be calculated through it. Thus, customer and customer_category table shares one to many relationships, which means one customer can belong to one specific category, but one category can contain many customers.

The entity product consists of attributes like product_ID, stock_quantity and vendor_ID, which gives details of supplier supplying the products as well. The relation between product and vendor is one to many relationships, which means that one vendor can supply many products, but a specific product belongs to one specific vendor. Also, each product belongs to one product category and a category can contain many products.

Also, the entity orders contain of the details of orders as well as the invoice details and products details and customers details. Customer and orders share one-to-many relationship, which means that one customer can place one or many or no order and one order belongs to only one customer. Order and invoice have one-to-one relationship, which means that one order can generate only one invoice and a single invoice belongs to single order.

There is a bridge entity between Product and Orders named Order_Product Details that contains the product_ID, order_ID, order_quantity and line_total details of the emporium respectively.

6. Database Querying

6.1 Information Queries

6.1.1 List all the customers that are also staff of the company.

The query used to list the customers that are also the staff o the company is:

set linesize 400 (It is used to set the size of the line to 400)

SELECT

C.customer_ID,

C.first_name,

C.last_name,

C.customer_address,

C.customer_phone,

CC.customer_type

FROM

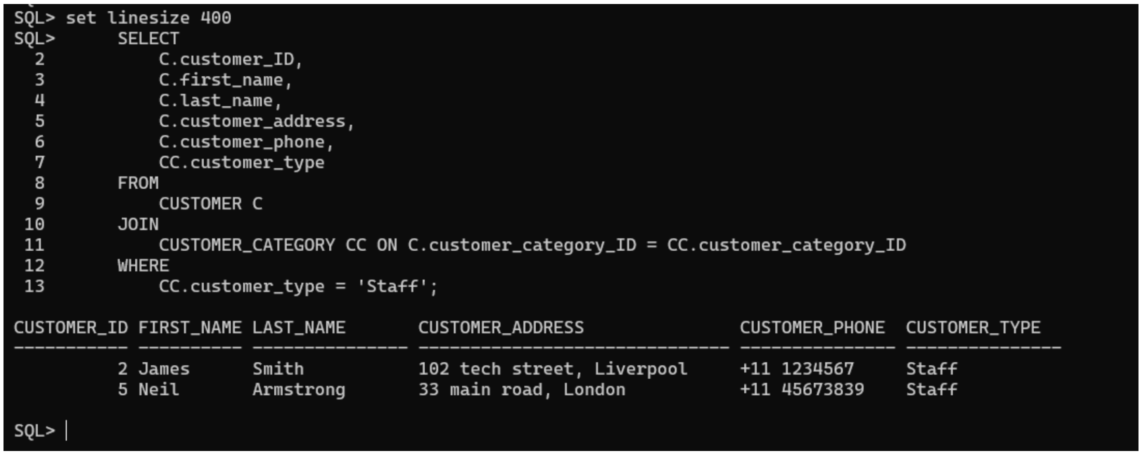
CUSTOMER C

JOIN

CUSTOMER_CATEGORY CC ON C.customer_category_ID = CC.customer_category_ID

WHERE

CC.customer_type = 'Staff';



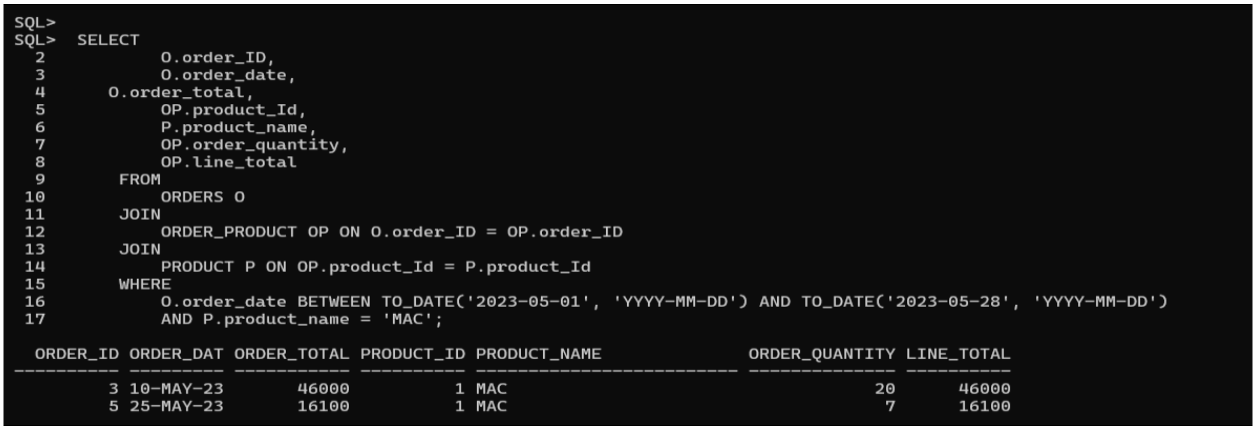
```
SQL> set linesize 400
SQL>      SELECT
              C.customer_ID,
              C.first_name,
              C.last_name,
              C.customer_address,
              C.customer_phone,
              CC.customer_type
          FROM
              CUSTOMER C
          JOIN
              CUSTOMER_CATEGORY CC ON C.customer_category_ID = CC.customer_category_ID
          WHERE
              CC.customer_type = 'Staff';

CUSTOMER_ID FIRST_NAME LAST_NAME       CUSTOMER_ADDRESS               CUSTOMER_PHONE  CUSTOMER_TYPE
----------- ---------- --------------- ------------------------------ --------------- ---------------
          2 James      Smith           102 tech street, Liverpool     +11 1234567     Staff
          5 Neil       Armstrong       33 main road, London           +11 45673839    Staff

SQL> |
```

Figure 39: List of customers that are also the staff of the company

6.1.2 List all the orders made for any particular product between the dates 01-05-2023 till 28- 05-2023.

The query used to List the products order made by product name ‘MAC’ between dates 01-05-2023 till 28-05-2023 is:

```
SELECT
        O.order_ID,
        O.order_date,
        O.order_total,
        OP.product_Id,
        P.product_name,
        OP.order_quantity,
        OP.line_total
    FROM
        ORDERS O
    JOIN
        ORDER_PRODUCT OP ON O.order_ID = OP.order_ID
    JOIN
        PRODUCT P ON OP.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')
        AND P.product_name = 'MAC';
```



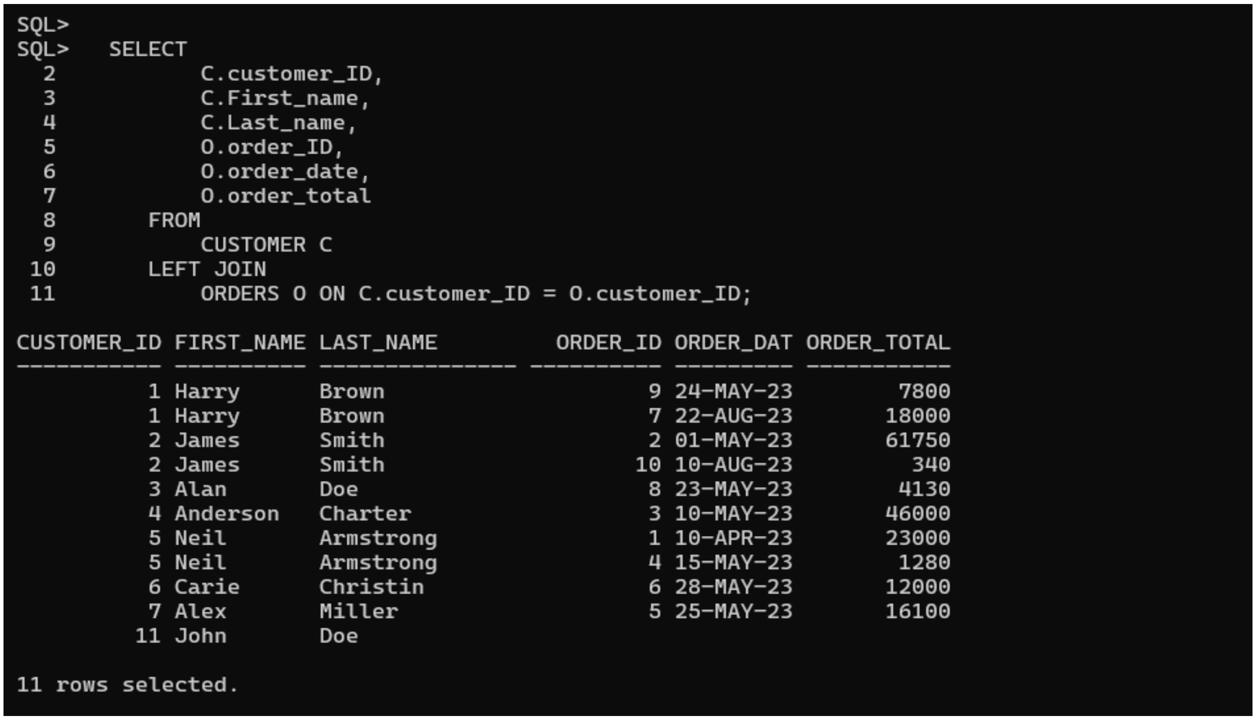
```
SQL>
SQL>  SELECT
  2           O.order_ID,
  3           O.order_date,
  4      O.order_total,
  5           OP.product_Id,
  6           P.product_name,
  7           OP.order_quantity,
  8           OP.line_total
  9       FROM
 10           ORDERS O
 11       JOIN
 12           ORDER_PRODUCT OP ON O.order_ID = OP.order_ID
 13       JOIN
 14           PRODUCT P ON OP.product_Id = P.product_Id
 15       WHERE
 16           O.order_date BETWEEN TO_DATE('2023-05-01', 'YYYY-MM-DD') AND TO_DATE('2023-05-28', 'YYYY-MM-DD')
 17           AND P.product_name = 'MAC';

  ORDER_ID ORDER_DAT ORDER_TOTAL PRODUCT_ID PRODUCT_NAME               ORDER_QUANTITY LINE_TOTAL
---------- --------- ----------- ---------- -------------------------- -------------- ----------
         3 10-MAY-23       46000          1 MAC                                    20      46000
         5 25-MAY-23       16100          1 MAC                                     7      16100
```

Figure 40: List of products order made by product named 'MAC' between 01-05-2023 till 28-05-2023

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

The query that is used to list the customers with order details and also the customers who have not ordered any products yet is:

```
SELECT
   C.customer_ID,
   C.First_name,
   C.Last_name,
   O.order_ID,
   O.order_date,
   O.order_total
FROM
   CUSTOMER C
LEFT JOIN
   ORDERS O ON C.customer_ID = O.customer_ID;
```



```
SQL>
SQL>   SELECT
              C.customer_ID,
              C.First_name,
              C.Last_name,
              O.order_ID,
              O.order_date,
              O.order_total
          FROM
              CUSTOMER C
          LEFT JOIN
              ORDERS O ON C.customer_ID = O.customer_ID;

CUSTOMER_ID FIRST_NAME LAST_NAME         ORDER_ID ORDER_DAT ORDER_TOTAL
----------- ---------- --------------- ---------- --------- -----------
          1 Harry      Brown                    9 24-MAY-23        7800
          1 Harry      Brown                    7 22-AUG-23       18000
          2 James      Smith                    2 01-MAY-23       61750
          2 James      Smith                   10 10-AUG-23         340
          3 Alan       Doe                      8 23-MAY-23        4130
          4 Anderson   Charter                  3 10-MAY-23       46000
          5 Neil       Armstrong                1 10-APR-23       23000
          5 Neil       Armstrong                4 15-MAY-23        1280
          6 Carie      Christin                 6 28-MAY-23       12000
          7 Alex       Miller                   5 25-MAY-23       16100
         11 John       Doe

11 rows selected.
```

Figure 41: Customers lists with order details also customer name 'John Doe' have not ordered any product yet'

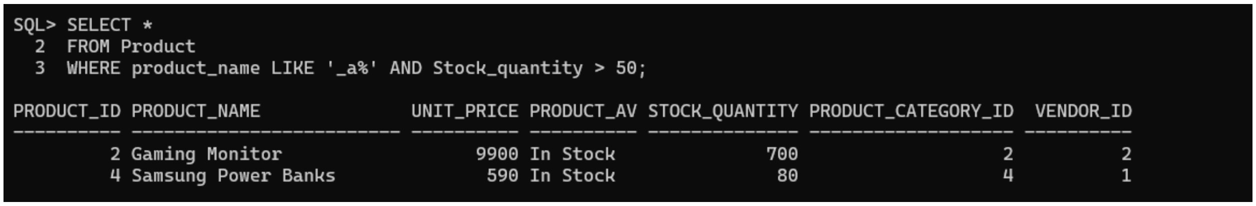
6.1.4 List all product details that have the second letter ‘a’ in their product name and have a stock quantity more than 50.

The query that is used to list the product product details with second letter ‘a’ in their name and stock quantity of more than 50 is:

SELECT *

FROM Product

WHERE product_name LIKE '_a%' AND Stock_quantity > 50;



```
SQL> SELECT *
     FROM Product
     WHERE product_name LIKE '_a%' AND Stock_quantity > 50;

PRODUCT_ID PRODUCT_NAME               UNIT_PRICE PRODUCT_AV STOCK_QUANTITY PRODUCT_CATEGORY_ID  VENDOR_ID
---------- ------------------------- ---------- ---------- -------------- ------------------- ----------
         2 Gaming Monitor                  9900 In Stock              700                   2          2
         4 Samsung Power Banks              590 In Stock               80                   4          1
```

Figure 42: List of products that have 'a' in the second letter and stock quantity of more than 50.

6.1.5 Find out the customer who has ordered recently.

The query used to find the customer detail who has ordered recently is:

SELECT customer_ID, First_name, Last_name, order_date

FROM (

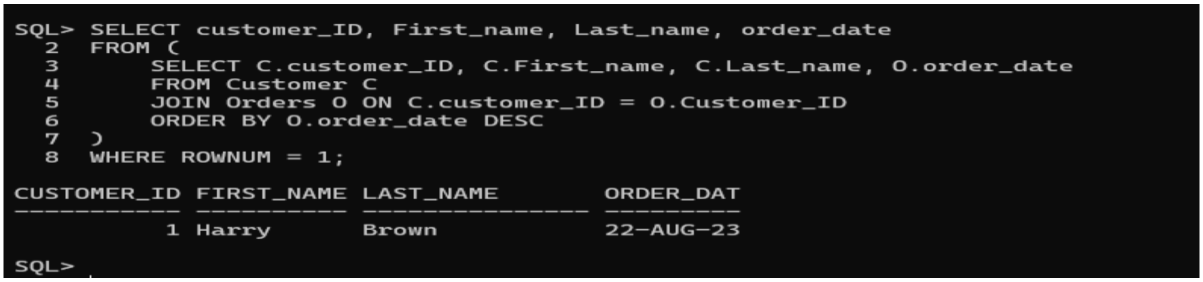
SELECT C.customer_ID, C.First_name, C.Last_name, O.order_date

FROM Customer C

JOIN Orders O ON C.customer_ID = O.Customer_ID

ORDER BY O.order_date DESC)

WHERE ROWNUM = 1;



```
SQL> SELECT customer_ID, First_name, Last_name, order_date
     FROM (
         SELECT C.customer_ID, C.First_name, C.Last_name, O.order_date
         FROM Customer C
         JOIN Orders O ON C.customer_ID = O.Customer_ID
         ORDER BY O.order_date DESC
     )
     WHERE ROWNUM = 1;

CUSTOMER_ID FIRST_NAME LAST_NAME        ORDER_DAT
----------- ---------- ---------------- ---------
          1 Harry      Brown            22-AUG-23

SQL>
```

Figure 43: Customer who have ordered recently

6.2 Transaction Queries

6.2.1 Show the total revenue of the company for each month

The query used to find the revenue of the company for each month is:

SELECT EXTRACT (MONTH FROM order_date) AS month,

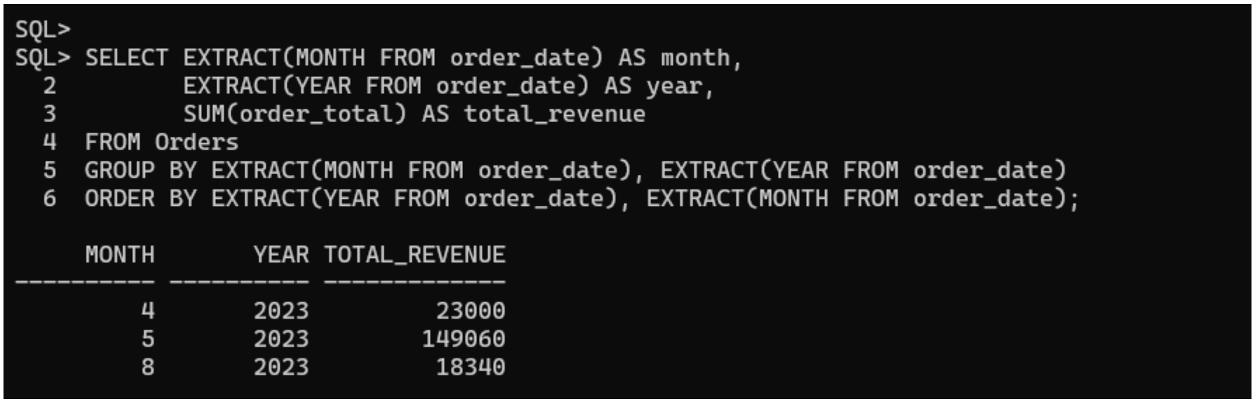
EXTRACT (YEAR FROM order_date) AS year,

SUM (order_total) AS total_revenue

FROM Orders

GROUP BY EXTRACT(MONTH FROM order_date), EXTRACT(YEAR FROM order_date)

ORDER BY EXTRACT(YEAR FROM order_date), EXTRACT(MONTH FROM order_date);



```
SQL>
SQL> SELECT EXTRACT(MONTH FROM order_date) AS month,
            EXTRACT(YEAR FROM order_date) AS year,
            SUM(order_total) AS total_revenue
     FROM Orders
     GROUP BY EXTRACT(MONTH FROM order_date), EXTRACT(YEAR FROM order_date)
     ORDER BY EXTRACT(YEAR FROM order_date), EXTRACT(MONTH FROM order_date);

     MONTH       YEAR TOTAL_REVENUE
---------- ---------- -------------
         4       2023         23000
         5       2023        149060
         8       2023         18340
```

Figure 44: Total revenue of the company for each month

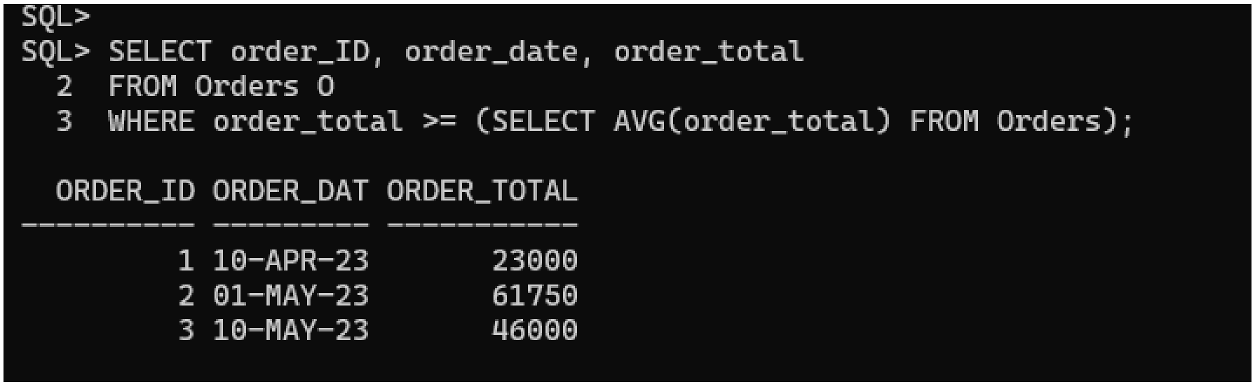
6.2.2 Find those orders that are equal or higher than the average order total value

The query that is used to find out those orders that are equal or higher than the average order is:

SELECT order_ID, order_date, order_total

FROM Orders O

WHERE order_total >= (SELECT AVG(order_total) FROM Orders);



```
SQL>
SQL> SELECT order_ID, order_date, order_total
  2  FROM Orders O
  3  WHERE order_total >= (SELECT AVG(order_total) FROM Orders);

  ORDER_ID ORDER_DAT ORDER_TOTAL
---------- --------- -----------
         1 10-APR-23       23000
         2 01-MAY-23       61750
         3 10-MAY-23       46000
```

Figure 45: Orders that are equal or higher than the average order

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

The query that is used to find the details of vendor who have supplied more than three products is:

SELECT vendor_ID, vendor_name, vendor_address, vendor_phone

FROM Vendor V

WHERE EXISTS (

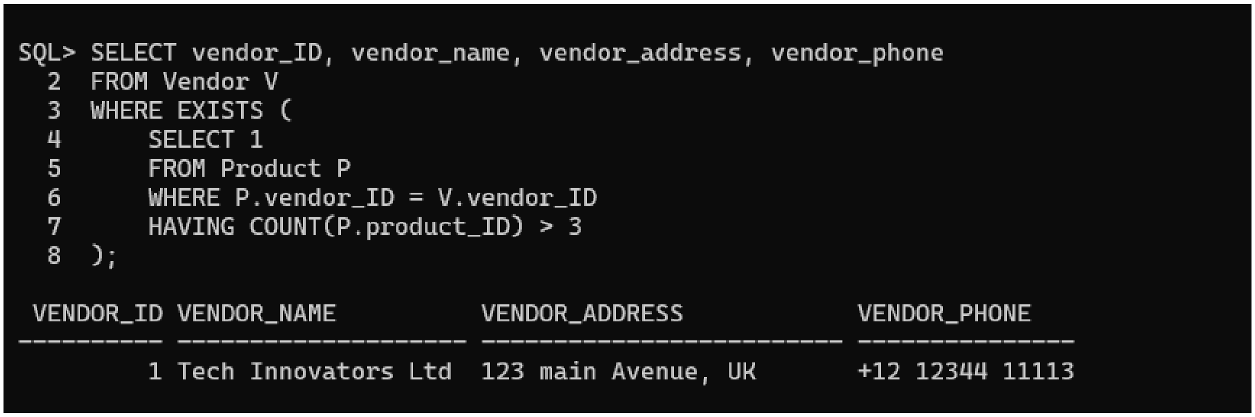
SELECT 1

FROM Product P

WHERE P.vendor_ID = V.vendor_ID

HAVING COUNT(P.product_ID) > 3

);



```
SQL> SELECT vendor_ID, vendor_name, vendor_address, vendor_phone
     FROM Vendor V
     WHERE EXISTS (
         SELECT 1
         FROM Product P
         WHERE P.vendor_ID = V.vendor_ID
         HAVING COUNT(P.product_ID) > 3
     );

 VENDOR_ID VENDOR_NAME          VENDOR_ADDRESS            VENDOR_PHONE
---------- -------------------- ------------------------- ---------------
         1 Tech Innovators Ltd  123 main Avenue, UK       +12 12344 11113
```

Figure 46: List of detail of vendor that have supplied more than one product

6.2.4 Show the top 3 product details that have been ordered the most.

The query that is used to show the list of top 3 products details that have been ordered the most is:

SELECT * FROM (

SELECT P.product_ID, P.product_name, P.unit_price, P.stock_quantity, P.vendor_ID, COUNT(OP.order_product_ID) AS count_Orders

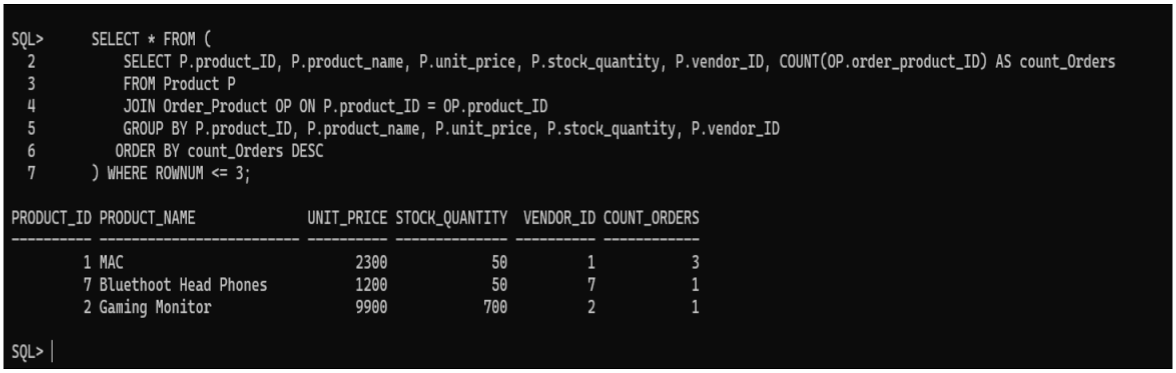
FROM Product P

JOIN Order_Product OP ON P.product_ID = OP.product_ID

GROUP BY P.product_ID, P.product_name, P.unit_price, P.stock_quantity, P.vendor_ID

ORDER BY count_Orders DESC

```
) WHERE ROWNUM <= 3;
```



```
SQL>     SELECT * FROM (
             SELECT P.product_ID, P.product_name, P.unit_price, P.stock_quantity, P.vendor_ID, COUNT(OP.order_product_ID) AS count_Orders
             FROM Product P
             JOIN Order_Product OP ON P.product_ID = OP.product_ID
             GROUP BY P.product_ID, P.product_name, P.unit_price, P.stock_quantity, P.vendor_ID
            ORDER BY count_Orders DESC
         ) WHERE ROWNUM <= 3;

PRODUCT_ID PRODUCT_NAME              UNIT_PRICE STOCK_QUANTITY  VENDOR_ID COUNT_ORDERS
---------- ------------------------- ---------- -------------- ---------- ------------
         1 MAC                             2300             50          1            3
         7 Bluethoot Head Phones           1200             50          7            1
         2 Gaming Monitor                  9900            700          2            1

SQL> |
```

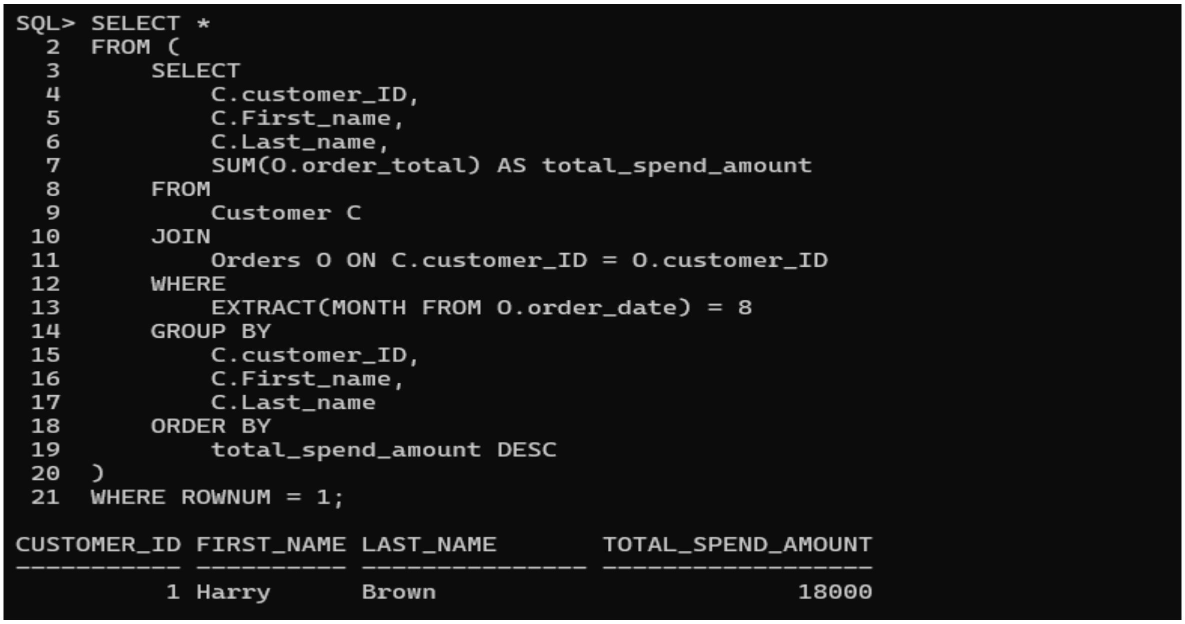
Figure 47: List of top 3 product details that have been ordered the most

6.2.5 Find out the customer who has ordered the most in August with his/her total spending on that month.

The query that is used to find out the customer that have ordered the most in the month of August with his/her total spending on that month is:

```
SELECT *
 FROM (
SELECT
   C.customer_ID,
   C.First_name,
   C.Last_name,
   SUM(O.order_total) AS total_spend_amount
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
    ORDER BY
        total_spend_amount DESC
)
WHERE ROWNUM = 1;
```

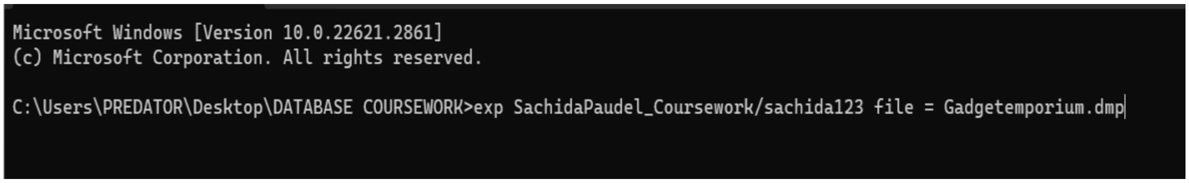


```
SQL> SELECT *
     FROM (
         SELECT
             C.customer_ID,
             C.First_name,
             C.Last_name,
             SUM(O.order_total) AS total_spend_amount
         FROM
             Customer C
         JOIN
             Orders O ON C.customer_ID = O.customer_ID
         WHERE
             EXTRACT(MONTH FROM O.order_date) = 8
         GROUP BY
             C.customer_ID,
             C.First_name,
             C.Last_name
         ORDER BY
             total_spend_amount DESC
     )
     WHERE ROWNUM = 1;

CUSTOMER_ID FIRST_NAME LAST_NAME       TOTAL_SPEND_AMOUNT
----------- ---------- --------------- ------------------
          1 Harry      Brown                        18000
```

Figure 48: Customer who has ordered the most in August with his total spending on that month

7. Database Dump file creation and Dropping Tables

7.1 Creation of Dump File



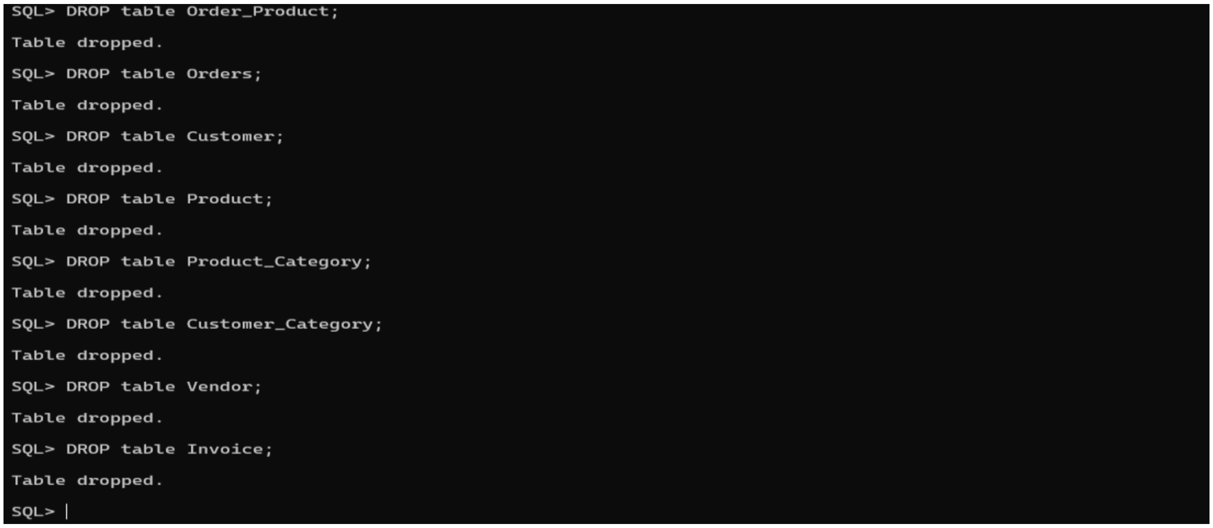
```
Microsoft Windows [Version 10.0.22621.2861]
(c) Microsoft Corporation. All rights reserved.

C:\Users\PREDATOR\Desktop\DATABASE COURSEWORK>exp SachidaPaudel_Coursework/sachida123 file = Gadgetemporium.dmp
```

Figure 49: Creation of Dump File

7.2 Dropping Tables

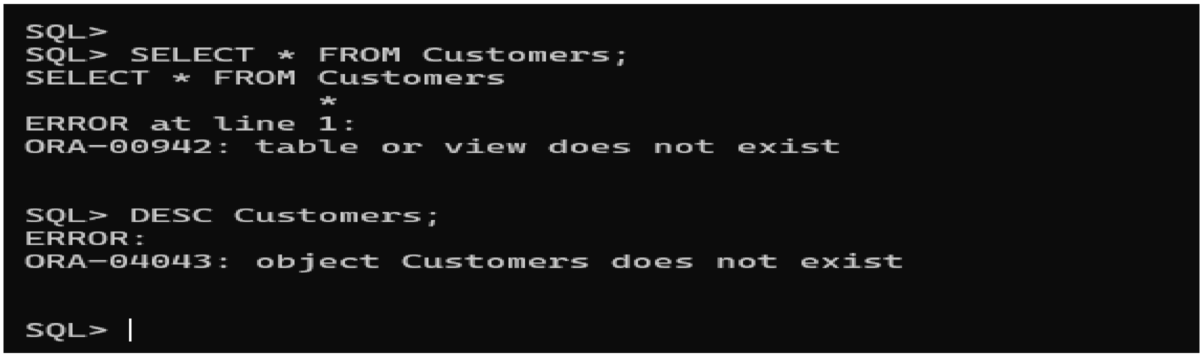
The query to drop tables is:

“DROP table table_name”



```
SQL> DROP table Order_Product;
Table dropped.
SQL> DROP table Orders;
Table dropped.
SQL> DROP table Customer;
Table dropped.
SQL> DROP table Product;
Table dropped.
SQL> DROP table Product_Category;
Table dropped.
SQL> DROP table Customer_Category;
Table dropped.
SQL> DROP table Vendor;
Table dropped.
SQL> DROP table Invoice;
Table dropped.
SQL> |
```

Figure 50: Dropping the tables



```
SQL>
SQL> SELECT * FROM Customers;
SELECT * FROM Customers
              *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> DESC Customers;
ERROR:
ORA-04043: object Customers does not exist


SQL> |
```

Figure 51: Table dropped

8. Critical Evaluation

8.1 Critical Evaluation of Module

The module named “Database” is a semester long module for students studying in the third semester. Database can be defined as organized data collection that helps to easily access, update, modify and delete the data. Studying database within this module will help us to equip in creating and utilizing table. Gaining database knowledge will help individual aspiring to purpose career in IT related industry.

Database plays a crucial role in industries related to IT and various other health sector. Studying database concept is extremely advantageous as it establishes a robust groundwork for efficiently handling database in the related sector. The versatility of database extends to various other purposes, often beyond our imaginations. Additionally, database is also essential in programming, helping programmers to execute specific functionality in their programs.

8.2 Critical Assessment of Coursework

The Database module occupied a total of 50% for this coursework. As per the requirement of the coursework, we are required to create a full database design for **Mr. John**, and enthusiast, established '**Gadget Emporium'**, that includes details of customers who buys the products, orders made, discount provided to the customers as per the category and product details.

A database that fulfills all the requirements of the coursework is created and data are inserted into it using the tool **SQL plus**. The database consists of eight tables in total including one bridge entity between the table Product and Orders. Firstly, tables are created and then values are inserted into the table. Numerous queries are performed to determine and validate the data in the database table and to check the relationship between the tables. Normalization is performed before inserting data. The entities and attributes obtained in the 3NF forms the final ERD table and data are inserted into the table after the final ERD is created.

Hence, the database I created and implemented satisfies the coursework criteria for creating a database design for **Mr. John** established '**Gadget Emporium'**.

9. References

9.1 Bibliography

ebay, 2024. *Grow your business with eBay.* [Online]
Available at: https://www.ebay.com/sl/tof/business-selling
[Accessed 5 january 2024].

Jain, D., 2023. *Codingninjas.* [Online] Available at:
https://www.codingninjas.com/studio/library/entity-and-attributes
[Accessed 3 January 2024].

javaTpoint, 2021. *Normalization.* [Online]
Available at: https://www.javatpoint.com/dbms-normalization
[Accessed 3 january 2024].

Secoda, 2023. *What is an Entity Relationship Diagram?.* [Online]
Available at: https://www.secoda.co/glossary/entity-relationship-diagram
[Accessed 3 January 2024].