LONDON
METROPOLITAN
UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

CC5051NI Databases

50% Individual Coursework

Autumn 2023

Student Name: Prabhab Khanal

London Met ID: 22068132

Assignment Submission Date: Monday, January 15, 2024

Word Count: 8640

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

"Gadget Emporium," Nepal's prominent online store spearheaded by Mr. John, is poised for significant expansion, prompting the need for a robust database system to meticulously manage customer data, order processing, product inventory, and vendor relationships. This dynamic system will streamline operations by categorizing customers, facilitating personalized services and discount management, while ensuring swift order processing, comprehensive product management, and real-time stock tracking. It will prioritize speed, scalability, and security, enabling rapid data retrieval, seamless integration of future expansions, and fortification against potential threats, thus empowering "Gadget Emporium" to navigate the fast-paced e-commerce landscape of Nepal with efficiency and agility.

To start this coursework, we first study the case details to identify the different parts like Customers, Products, Orders, and Vendors, along with their specific details. Then, we work on organizing the database in a way that removes any unnecessary repetition and ensures the information is accurate and reliable. This process, called normalization, might mean splitting tables into smaller parts and arranging them sensibly, making sure they connect properly using primary and foreign keys. After that, we put together the actual database using a plan that shows how everything is related, detailing how these parts interact with each other. This helps us build a system where we can easily handle and use all this information without any confusion or errors optimal performance and accuracy in "Gadget Emporium's" operations.

1.1. Aim and Objectives

Our goal is to create a user-friendly database system that helps Gadget Emporium run smoothly and ensures a great shopping experience for customers. By organizing electronic product details, managing customer categories and discounts, simplifying order processing, keeping track of vendors, monitoring inventory in real-time, and providing secure payment options, we aim to build a solid foundation for the online store's success.

The objective of “Gadget Emporium” are as follows:

- Drive operational excellence by automating order processing, ensuring accurate data for informed decisions, and enabling seamless scalability for future growth.
- Elevate sales through precise product information, real-time inventory management, and streamlined vendor partnerships.
- Expanding the company’s reach by increasing the number of customers served or entering new markets.
- Increasing profitability and shareholder value through strategies such as optimizing pricing, streamlining operations, or expanding the company’s offerings.
- Making a positive impact on the local community through initiatives such as supporting local business or organizations or implementing environmentally friendly practices.

1.2. Current Business Activities and Operations

1.2.1 Business Rule

The business rule for Gadget Emporium can be seen below:

- Each product is characterized by its name, description, category, price, and stock level.
- Every product belongs to a single category, while each category can contain multiple products.
- Customers are categorized as Regular (R), Staff (S), and VIP (V).
- Each customer category is entitled to a distinct discount rate (0%, 5%, 10% respectively) on product purchases.
- Customer addresses are stored to facilitate the delivery process.
- Customers are categorized as Regular (R), Staff (S), and VIP (V).
- Each customer category is entitled to a distinct discount rate (0%, 5%, 10% respectively) on product purchases.
- Customer addresses are stored to facilitate the delivery process.
- Customers are categorized as Regular (R), Staff (S), and VIP (V).
- Each customer category is entitled to a distinct discount rate (0%, 5%, 10% respectively) on product purchases.
- Customer addresses are stored to facilitate the delivery process.
- Real-time tracking of product availability to prevent overselling and maintain accurate stock levels.
- Products must have inventory details such as stock quantity and availability status.
- Integration of various payment gateways (cash on delivery, credit/debit card, e-wallet) for secure and seamless transaction processing.
- Each order is linked to one payment option.
- An invoice is issued upon order confirmation, encompassing order details, customer information, and payment specifics.

1.2.2 Business Assumption

Some business assumptions that were made are as follows:

- Each order corresponds to the generation of a single invoice, ensuring clarity and simplicity in the billing process.

- The flexibility is recognized in the customer-order relationship, allowing a customer to have no orders, one order, or multiple orders based on their engagement with the system.

- The distinction between Total_Quantity and Order_Quantity provides a nuanced understanding, where Total_Quantity reflects the overall quantity of products in an order, while Order_Quantity specifies the quantity of a specific product within the order.

- Every product is associated with a unique set of Inventory_Details, offering insights into Stock_Quantity and Availability_Status for efficient inventory management.

- The computation of Total_Amount involves the application of Discount_Rate, providing a transparent representation of the final rate post-discount.

- Order_Status serves as a decisive factor, indicating whether a customer has placed an order or not, streamlining the tracking of customer engagement and order fulfilment.

2. Entity Relationship Diagram (ERD)

An ERD is a visual tool showcasing different parts of a system and how they connect. It helps in planning databases by illustrating how things are related using symbols for entities, attributes, and relationships. This diagram is widely used in software and database design to make complex structures easier to understand and manage for developers and stakeholders. (Nishadha, 2022)

Importance of ERD

- ERDs provide a visual representation of data structures and relationships, simplifying complex concepts for easier comprehension.
- They serve as a universal language, facilitating effective communication between stakeholders, including developers, designers, and business analysts.
- ERDs act as a blueprint or roadmap for database construction, outlining tables, relationships, and constraints required for database implementation.
- ERDs ensure data consistency and accuracy by illustrating how different data entities are related, preventing inconsistencies or redundancies.
- They streamline the development process by guiding developers, minimizing errors, and enabling efficient implementation of databases.
- ERDs enable easy modifications and scaling of databases as business requirements change, allowing for flexibility in database structures.
- They assist in making informed decisions about database design and functionalities based on a clear visualization of data relationships.
- By visualizing relationships and data flow, ERDs help identify potential issues or gaps in the database design, allowing for pre-emptive measures to mitigate risks.

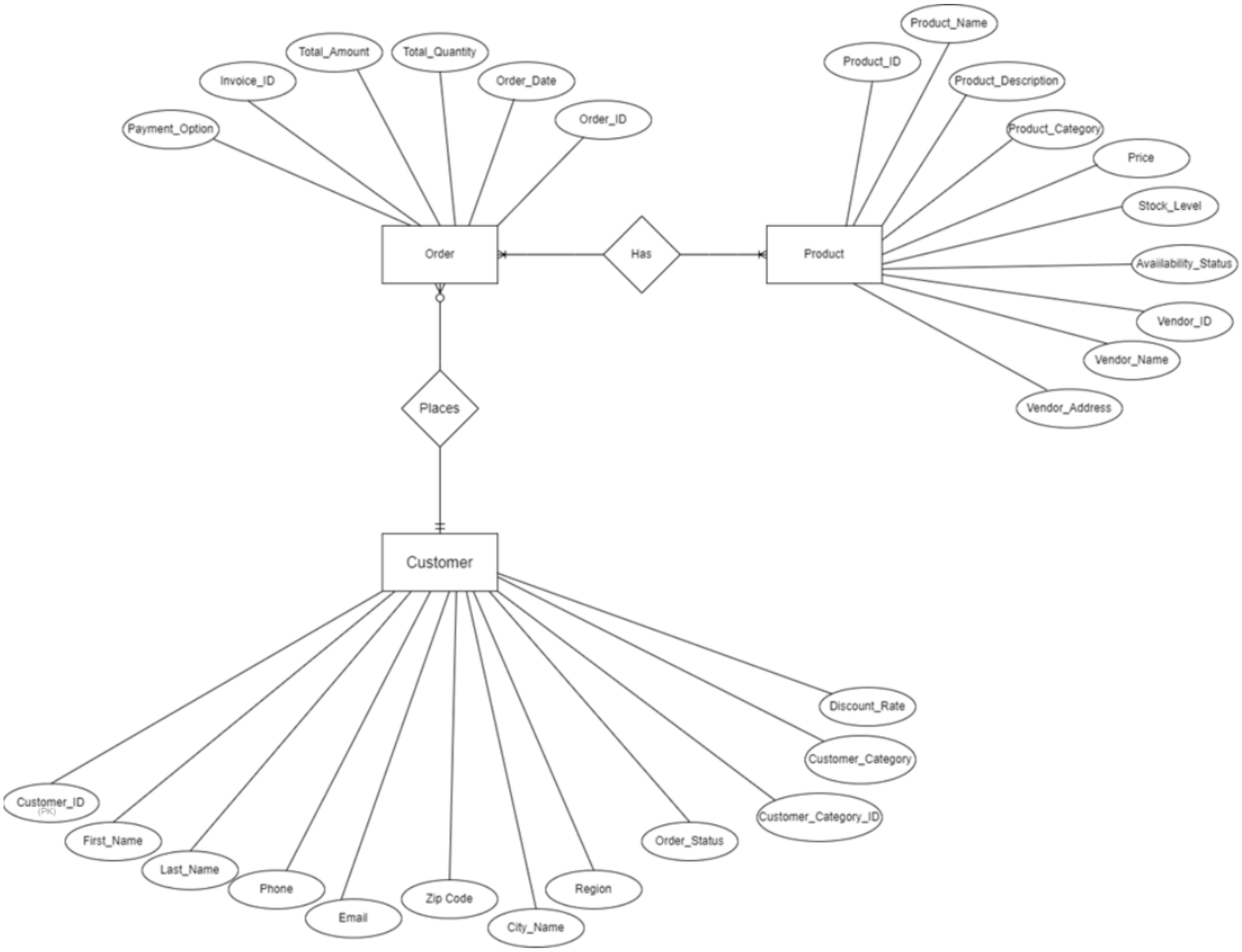


Figure 1: Initial ERD

In the relational database model, the connections among customers, orders, and products exhibit a flexible and adaptable framework. The "zero to many" association between customers and orders acknowledges the potential for a customer to have either no recorded orders or multiple orders linked to their account, allowing for scenarios where users engage with the platform without initiating purchases. Simultaneously, the "many-to-many" relationship between orders and products suggests that multiple orders can include various products, and reciprocally, a particular product might be included in multiple orders.

In my initial Entity-Relationship Diagram (ERD), the presence of partial and transitive dependencies in attributes introduces challenges that can lead to insertion, deletion, and update anomalies. Partial dependency occurs when non-prime attributes rely on only a subset of the primary key, potentially causing issues when attempting to insert data without including the entire key. Transitive dependency, on the other hand, arises when an attribute is dependent on another non-prime attribute, creating a chain of dependencies that may complicate updates and deletions.

These anomalies can hinder the database's ability to maintain data consistency and integrity. Insertion anomalies may force the inclusion of unrelated data during record creation, deletion anomalies might result in the unintentional loss of valid information, and update anomalies may lead to inconsistencies if dependencies are not properly managed. To address these challenges, it is crucial to normalize the database by restructuring tables to eliminate dependencies and reduce redundancy. Through normalization, the database design can be optimized, minimizing the likelihood of anomalies and ensuring a more robust and maintainable data model.

3. Identification of Entities and Attributes

3.1 Entities and Attributes

Entities within databases encompass distinct and identifiable elements individuals, organizations, data components, or critical components of a system. These entities, such as people, products, or events, serve as the foundational elements in databases. They are pivotal in organizing and managing data efficiently within a database system, acting as the core units that store and retrieve information effectively. (Brewer, 2024)

The entity that are present in my initial ERD are as follows:

- Customer
- Order
- Product

Entity is represented by a rectangle:
Entity Name

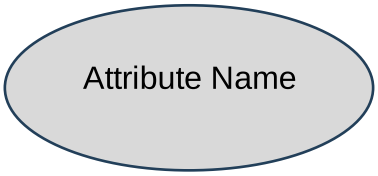
Attributes in databases are the unique qualities that define an entity, like a person's age or a product's price. Entities can have multiple attributes, with one often designated as the primary key for identification. In Entity-Relationship models, attributes are represented using elliptical shapes, encapsulating the specific details characterizing each entity.

The attributes that can be seen in my initial ERD are as follows:

Attributes of Customer: Customer_ID, First_Name, Last_Name, Zip_Code, City_Name, Region, Nearest_Landmark, Phone, Email, Order_Status, Customer_Category_ID, Customer_Category, Discount_Rate

Attributes of Order: Order_ID, Invoice_ID, Order_Date, Total_Quantity, Payment_Option, Total_Amount

Attributes of Product: Product_ID(PK), Product_Description, Product_Categories, Price, Stock_ID, Stock_Level, Availability_Status, Vendor_ID, Vendor_Name, Vendor_Status

Attributes is represented by a ellipse: 

- **Customer**

Attribute Name	Data Type	Constraints	Description
Customer_ID	Number	Primary Key	The attribute store unique customer identifier
First_Name	Varchar2(20)	Not Null	This attribute stores customer first name.
Last_Name	Varchar2(20)	Not Null	This attribute stores customer last name.
Phone	Varchar2(20)	Not Null, Unique	This attribute stores customer phone number which is unique with all.
Email	Varchar2(40)	Not Null, Unique	This attribute stores customer emails which must be unique.
Zip_Code	Number	Not Null	This attribute stores ZIP code of customer’s address.
City_Name	Varchar2(20)	Not Null	This attribute stores City_Name of customer’s address.
Region	Varchar2(20)	Not Null	This attribute stores Region of customer’s address.
Order_Status	CHAR (1)	Not Null	This attribute stores the order status if ordered by the user
Customer_Category_ID	Number	Not Null	The attribute store category of the customer.
Customer_Category	Varchar2(20)	Not Null	The category name which is assigned to the customer.
Discount_Rate	Number (5,2)	Not Null	The discount rate assigned to the customer based on their category.

Table 1: Customer Attributes

- **Order**

Attribute Name	Data Type	Constraints	Description
Order_ID	Number	Primary Key	This attribute stores the order done in the system.
Order_Date	Date	Not Null	This attribute stores the date when th e order was placed.
Total_Quantity	Number	Not Null	This attribute stores the total quantity of products or items included in the order.
Total_Amount	Number (10,2)	Not Null	This attribute stores the total monetary value of the order, including any applicable taxes or discounts.
Invoice_ID	Number	Not Null	This attribute stores the invoice generated for the order.
Payment_Option	Varchar2(20)	Not Null	This attribute stores the method used by the customer to make the payment for the order.

Table 2: Order Attributes

- Product

Attribute Name	Data Type	Constraints	Description
Product_ID	Number	Primary Key	This attribute stores the unique identifier assigned to each product in the system.
Product_Name	Varchar2(20)	Not Null	This attribute stores the name of the product.
Product_Description	Varchar2(400)	Not Null	This attribute stores the description providing additional details about the product.
Product_Category	Varchar2(20)	Not Null	This attribute stores the classification of the product to different types.
Price	Number	Not Null	This attribute stores the price of the individual product.
Stock_ID	Number	Not Null	This attribute store unique stock id for the product.
Stock_Level	Number	Not Null	This attribute stores the quantity of the product currently available in the inventory.
Availability_Status	CHAR (1)	Not Null	This attribute stores that whether the product is currently available.
Vendor_ID	Varchar2(20)	Not Null	This attribute stores the unique identifier assigned to each vendor.
Vendor_Name	Varchar2(20)	Not Null	This attribute stores the name of the company supplying the product
Vendor_Address	Varchar2(20)	Not Null	This attribute stores the location of the vendor.

Table 3: Product Attributes

4. Normalization

Normalization stands as a pivotal database design method aimed at minimizing data redundancy and eradicating issues like Insertion, Update, and Deletion Anomalies. This approach adheres to specific rules by breaking down larger tables into smaller, more organized ones and establishing relationships between them. The primary objective of Normalization in SQL is to eliminate repetitive data, ensuring logical and efficient data storage while maintaining data integrity and consistency. (Peterson, 2023)

4.1. Un-Normalized Form (UNF)

Unnormalized Form (UNF) refers to the initial stage or starting point of normalization in database design. UNF, or Unnormalized Form, lacks structured organization, leading to data redundancy. This redundancy results in insertion, update, and deletion anomalies, making it challenging to manage and maintain data accuracy.

Making a UNF from my initial ERD we can observe that a customer has multiple orders and an order have one or many products. When I start from customer, I can see that customer is the repeating data and order is a repeating group within that repeating group there is a repeating group of products.

Hence, the UNF is shown as:

Customer (Customer_ID, First_Name, Last_Name, Zip_Code, City_Name, Region, Phone, Email, Order_Status, Customer_Category_ID, Customer_Category, Discount_Rate {Order_ID, Invoice_ID, Order_Date, Total_Quantity, Payment_Option, Total_Amount {Order_Quantity, Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Stock_Level, Availability_Status, Vendor_ID, Vendor_Name, Vendor_Address}})

4.2 First Normal Form (1NF)

After the UNF is completed, we can distinguish between the repeating data and repeating groups. Then, First Normal Form (1NF) ensures that data in a table is organized with no repeating groups or multiple values in a single cell. Each column holds only single, indivisible values, minimizing redundancy and allowing efficient data organization.

To attain the First Normal Form (1NF), certain guidelines are to be followed:

- Splitting repeating groups into separate tables.
- Ensuring uniqueness and distinctness in each record.
- Assigning a primary key to each table, serving as a unique identifier for every record within.
- Providing individual, specific names for each column in a table.

Here, from our UNF we can separate Customer table as it has only the repeating data. Here other table that are separated from UNF are Order and Product. The 1NF can be seen here:

Customer 1 *(Customer_ID(**PK**), First_Name, Last_Name, Zip_Code, City_Name, Region, Nearest_Landmark, Phone, Email, Order_Status, Customer_Category_ID, Customer_Category, Discount_Rate)*

Order 1 (*Order_ID(**PK)**, Customer_ID(**FK**), Invoice_ID, Order_Date, Total_Quantity, Payment_Option, Total_Amount*)

Product 1 *(Product_ID(**PK**), Order_ID(**FK**), Customer_ID(**FK**), Product_Description, Product_Categories, Price, Stock_ID, Stock_Level, Availability_Status, Vendor_ID, Vendor_Name, Vendor_Status)*

4.3 Second Normal Form (2NF)

Here, we check for the partial dependency. Partial dependency in databases means that a certain piece of information in a table is dependent on only a part of the primary key, not the entire key. It's like having a connection between two things, but one of them is only related to a specific part of the other thing, not the whole thing. In the world of databases, we often want relationships to be as clear and direct as possible to keep things organized and efficient. Partial dependency can complicate these relationships, so we often try to avoid it when designing a database.

When the table are in First Normal Form, we can achieve Second Normal Form (2NF). From above table we need to check the partial dependencies.

Checking Partial Dependency on Customer1

Customer 2 *(Customer_ID(**PK**), First_Name, Last_Name, Zip_Code, City_Name, Region, Nearest_Landmark, Phone, Email, Order_Status, Customer_Category_ID, Customer_Category, Discount_Rate)*

In the customer table there is only a key but for partial dependency to occur there must be 2 keys hence partial dependency does not occur on the customer table and it remains as it is.

Checking Partial Dependency on Order1

In the Order table we can see two keys i.e. Order_ID and Customer_ID, so we must check for the partial dependency in this table. We can use the formula to know the tables that are to be created when second normal form is done i.e. 2^{n-1}. We get know that 3 tables are to be formed that can be classified as:

The key Customer_ID itself is a foreign key on the order table hence it doesn't give any value primarily.

Customer_ID → XXXX

For Order_ID it gives all the attributes of the Order table except the Customer_ID.

Order_ID → *Invoice_ID, Order_Date, Total_Quantity, Payment_Option, Total_Amount*

The attributes Customer_ID and Order_ID forms a composite key but doesn't give any attributes, but the table must be included as there are two columns and a table is formed called Customer_Order_Details.

<u>Customer_ID, Order_ID</u>*→ XXXXX

Final Table for Order After Second Normal Form:

Order 2 (*Order_ID(**PK)**, Invoice_ID, Order_Date, Total_Quantity, Payment_Option, Total_Amount*)

Customer_Order_Details 2 (*Customer_ID(**FK**), Order_ID(**FK**)*)

Here there is no table of Customer_ID but of Customer_Order_Details as in Customer_ID there consist of a single column which we tend to remove but on the other hand in Customer_Order_Details there is two column which we tend to keep.

Checking Partial Dependency on Product

In the Order table we can see three keys i.e. Product_ID, Order_ID and Customer_ID, so we must check for the partial dependency in this table. We can use the formula to know the tables that are to be created when second normal form is done i.e., 2^{n-1}. We get know that 7 tables are to be formed that can be classified as:

The Product_ID gives all the attributes from the Product1 table except the Order_Quantity, Order_ID, Customer_ID. Hence, there is a partial dependency to all the attributes excluding the ones mentioned.

Now,

Product_ID → *Product_Name, Product_Description, Product_Categories, Price, Stock_ID, Stock_Level, Availability_Status, Vendor_ID, Vendor_Name, Vendor_Status*

For Order_ID and Customer_ID they are both the foreign key on this table joining the and they don't give any attributes on the table. So, we write them as

Customer_ID → XXXX

Order_ID → XXXX

Similarly, both the keys Customer_ID and Order_ID forms a composite key which doesn't give any partial functional dependency on the products table. But a table is created called Order_Customer_Details which is formed but we need not show the table as the same relation is shown in the above second normal form of Order and normalization means reducing redundancy hence the table is not formed by us to determine uniqueness of data.

<u>**Customer_ID, Order_ID**</u>* → XXXX

Both the keys Order_ID and Product_ID forms a composite key on the product table gives Order_Quantity which shows it has a partial functional dependency forming an Order_Product_Details Table that also acts as a bridging entity for our many to many relation that is formed between Order and Product which also gives brief information about the quantity of the ordered items.

Now,

Order_ID, Product_ID* → Order_Quantity

From the above first normal table we see that Product_ID and Customer_ID forms a composite key may also have a partial functional dependency which in my case hasn’t occurred meaning that it does not give a partial functional dependency. Hence for this table to form it gives relation between customer and product which is generally done by order. Without order this table doesn’t give a proper explanation so we do not make it in our database to make the database more viable.

Product_ID, Customer_ID **→ XXXX*

Finally, in the case of all the three keys which forms a composite key also doesn’t give any partial functional dependencies in my case. Initially the relation between Order and Customer is defined and also between the Order and Product is also defined. Making this table also means creating data redundancy so, we do not make this table as normalization basics are followed.

Product_ID, Customer_ID, Order_ID * → XXXX

Final Table for Product after Second Normal Form

Product2 (*Product_ID, Product_Name, Product_Description, Product_Categories, Price, Stock_ID, Stock_Level, Availability_Status, Vendor_ID, Vendor_Name, Vendor_Status)*

Order_Product_Details 2 (*Product_ID, Order_ID, Order_Quantity)*

Order_Customer_Details 2 *(Order_ID, Customer_ID)*

Tables After Second Normal Form

Customer 2 *(Customer_ID(**PK**), First_Name, Last_Name, Zip_Code, City_Name, Region, Nearest_Landmark, Phone, Email, Order_Status, Customer_Category_ID, Customer_Category, Discount_Rate)*

Order 2 (*Order_ID(**PK)**, Invoice_ID, Order_Date, Total_Quantity, Payment_Option, Total_Amount***)**

Customer_Order_Details 2 (*Customer_ID(**FK**), Order_ID(**FK**)***)**

Product2 (*Product_ID, Product_Name, Product_Description, Product_Categories, Price, Stock_ID, Stock_Level, Availability_Status, Vendor_ID, Vendor_Name, Vendor_Status)*

Order_Product_Details 2 (*Product_ID, Order_ID, Order_Quantity)*

4.4. Third Normal Form (3NF)

Third Normal Form (3NF) in databases ensures that information is organized efficiently by eliminating unnecessary data dependencies. It focuses on removing indirect relationships between columns, promoting a more streamlined and maintainable structure. In essence, 3NF minimizes redundancy and helps maintain accurate and reliable data.

The third normal form removes the transitive dependency from the table created from the second normal form by checking individually.

Checking Transitive Dependency for Customer

In Customer2 table there is transitive dependency because a non-key attribute Zip_Code is giving another non key attributes which are City_Name, Region and Nearest_Landmark. To remove the following transitive dependency a new table called Address Details is formed where Zip_Code is the Primary Key of the table.

Customer_ID → Zip_Code ---- > City_Name, Region

Here, a table is obtained called Address_Details with the Zip_Code as its primary key and foreign key in Customer table.

Zip_Code → City_Name, Region

Again, a non-key attribute Customer_Category_ID is giving another non key attributes which are Customer_Category and Discount_Rate.

Customer_ID → Customer_Category_ID ----> Customer_Category, Discount_Rate

Here a table Customer_Category_Details is formed with Customer_Category_ID as the primary key which becomes foreign key in the Customer table.

All the other attributes are now only dependent to the Customer_ID.

Final Table in Third Normal Form for Customer

Customer 3 *(Customer_ID(**PK**), First_Name, Last_Name, Zip_Code(**FK**), Phone, Email, Order_Status, Customer_Category_ID(**FK**))*

Address_Details 3 *(Zip_Code(**PK**), City_Name, Region, Nearest_Landmark)*

Customer_Category_Details 3 *(Customer_Category_ID(**PK**), Customer_Category, Discount_Rate)*

Checking Transitive Dependency for Order

There is a transitive dependency on Order table because a non-key attribute Invoice_ID gives another non-key attribute Payment_Method. Now a new table Invoice_Details is formed.

Order_ID → Invoice_ID ----> Payment_Option

Here, Invoice_ID is the primary key for Invoice_Details table and foreign key for Orders table.

All other attributes are dependent only to Order_ID.

Final Table in Third Normal Form for Order

Order 2 (*Order_ID(**PK)**, Invoice_ID(**FK**), Order_Date, Total_Quantity, Total_Amount***)**

Invoice_Details 2 (*Invoice_ID(**PK**), Payment_Option***)**

Checking Transitive Dependency for Customer_Order_Details

There is no transitive dependency as there is only a composite key present hence no table is separated here. The table remains as it is.

Final Table in Third Normal Form for Customer_Order_Details

Customer_Order_Details 3 (*Customer_ID(***FK***), Order_ID(***FK***)***)**

Checking Transitive Dependency for Product

There is transitive dependency on Product because a non-key attribute Vendor_ID gives another non-key attributes Vendor_Name and Vendor_Address. Hence, a new table known as Vendor_Details will be formed having Vendor_ID as the primary key.

Product_ID → Vendor_ID ----> Vendor_Name, Vendor_Address

Product_ID → Stock_ID -----> Stock_Level, Availability_Status

Another non-key attribute Stock_ID gives non-key attributes Stock_Level and Availability_Status. Hence, a new table of Inventory_Details will be formed having Vendor_ID as the primary key.

All other attributes are dependent only to Product_ID.

Final Table in Third Normal Form for Order

Product 3 (*Product_ID(**PK**), Product_Name, Product_Description, Product_Categories, Price, Stock_ID(**FK**), Vendor_ID(**FK**))*

Vendor 3 (*Vendor_ID(**PK**), Vendor_Name, Vendor_Status)*

Inventory_Details 3 *(Stock_ID(**PK**), Stock_Level, Availability_Status)*

Checking Transitive Dependency for Order_Product_Details

There is no transitive dependency as there is only a composite key which is related to Order_Quantity only so, the table remains as it is.

Final Table in Third Normal Form for Customer_Order_Details

Order_Product_Details 3 (*Product_ID(***FK***), Order_ID(***FK***), Order_Quantity***)**

Final Tables in 3NF

Customer_Details 3 *(Customer_ID(**PK**), First_Name, Last_Name, Zip_Code(**FK**), Phone, Email, Order_Status, Customer_Category_ID(**FK**))*

Address_Details 3 *(Zip_Code(**PK**), City_Name, Region)*

Customer_Category_Details 3 *(Customer_Category_ID(**PK**), Customer_Category, Discount_Rate)*

Order_Details 3 (*Order_ID(**PK)**, Invoice_ID(**FK**), Order_Date, Total_Quantity, Total_Amount***)**

Invoice_Details 3 (*Invoice_ID(**PK**), Payment_Option**)***

Customer_Order_Details 3 (*Customer_ID(**FK**), Order_ID(**FK**)**)***

Product_Details 3 (*Product_ID(**PK**), Product_Name, Product_Description, Product_Categories, Price, Stock_ID(**FK**), Vendor_ID(**FK**))*

Vendor_Details 3 (*Vendor_ID(**PK**), Vendor_Name, Vendor_Status)*

Inventory_Details 3 *(Stock_ID(**PK**), Stock_Level, Availability_Status)*

Order_Product_Details 3 (*Product_ID(**FK**), Order_ID(**FK**), Order_Quantity**)***

5. Final Entity Relationship Diagram(ERD)

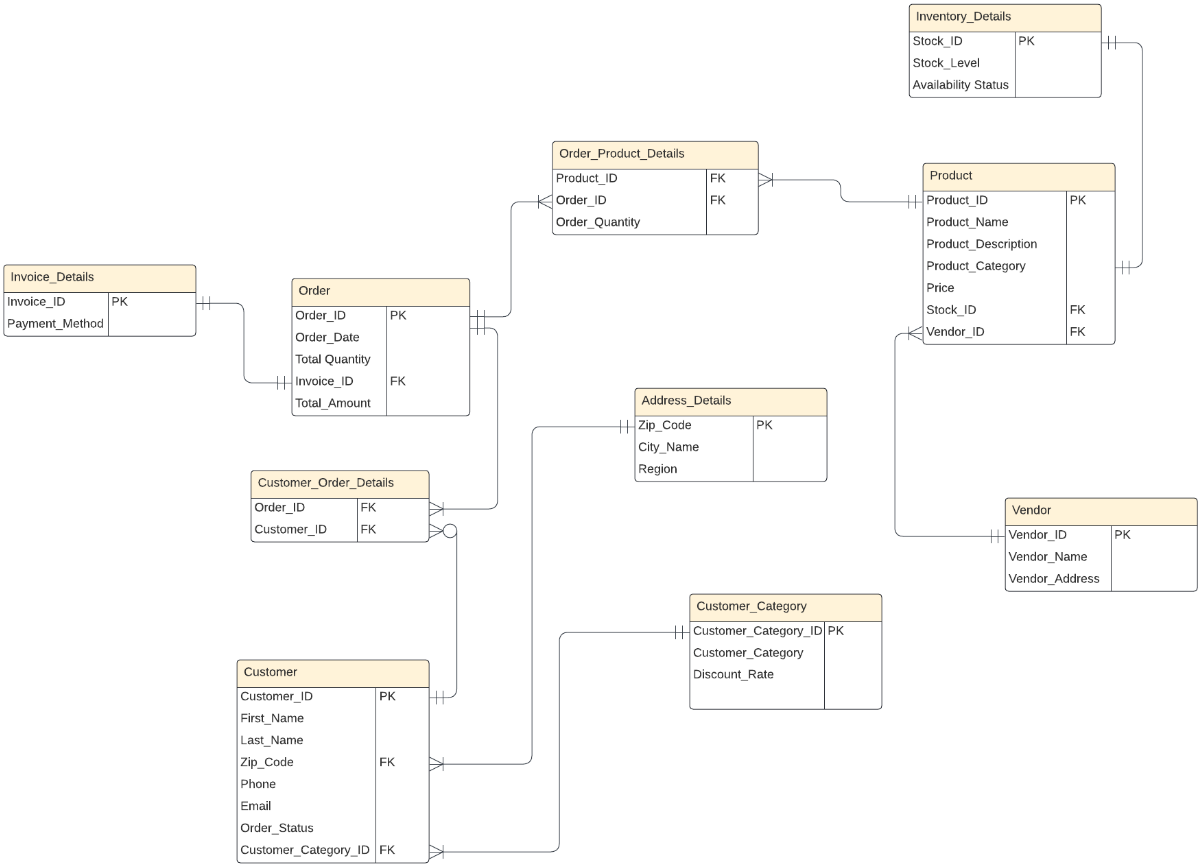


Figure 2: Final ERD

The presented Entity-Relationship Diagram (ERD) not only offers a comprehensive representation of a business model but also reveals a systematic application of normalization principles. The incorporation of distinct entities for customer details, addresses, customer categories, orders, invoices, products, vendors, inventory, and the intermediary table "Customer_Order_Details" underscores a dedicated commitment to normalization standards. Normalization, a process aimed at minimizing data redundancy and dependencies, is evident in this ERD through the strategic separation of information into individual entities and the establishment of well-defined relationships. Notable instances include the "Address_Details" and "Customer_Category_Details" entities, emphasizing the elimination of partial dependencies associated with addresses and customer categories. Moreover, the presence of linking tables like "Customer_Order_Details" and "Order_Product_Details" illustrates an effective approach to managing many-to-many relationships, contributing to a normalized and proficient structure for the relational database. In summary, the ERD showcases a deliberate normalization strategy, fostering data integrity and enhancing the efficiency of database operations.

The database comprises several interconnected entities representing different aspects of the system. The core entity, Customer_Details, holds unique customer identifiers (Customer_ID) and is associated with address information through the Zip_Code foreign key, linking to the Address_Details entity. Additionally, the customer's category and applicable discount rate are captured through the foreign key Customer_Category_ID, establishing a connection to the Customer_Category_Details entity. Order_Details provide information on customer transactions, with a foreign key (Invoice_ID) linking to the associated invoice details in the Invoice_Details entity. The Customer_Order_Details entity acts as a bridge between customers and their corresponding orders, linking Customer_ID to Customer_Details and Order_ID to Order_Details. The Product_Details entity encompasses product-related information, including vendor details linked via the Vendor_ID foreign key, and stock and availability details through the Stock_ID foreign key, connecting to the Inventory_Details entity. The Order_Product_Details entity establishes relationships between ordered products and specific orders, connecting Product_Details and Order_Details through foreign keys. Overall, these entities and their interconnections form a comprehensive database schema, facilitating organized storage and retrieval of information related to customers, orders, products, vendors, and inventory. When visually represented in an Entity-Relationship Diagram (ERD), this structure provides a clear overview of the relationships between different components within the database.

6. Implementation

Creating And Granting User:

SQL> conn system

Enter password:

Connected.

SQL> create user course_work identified by 12345;

User created.

SQL> grant connect,resource to course_work;

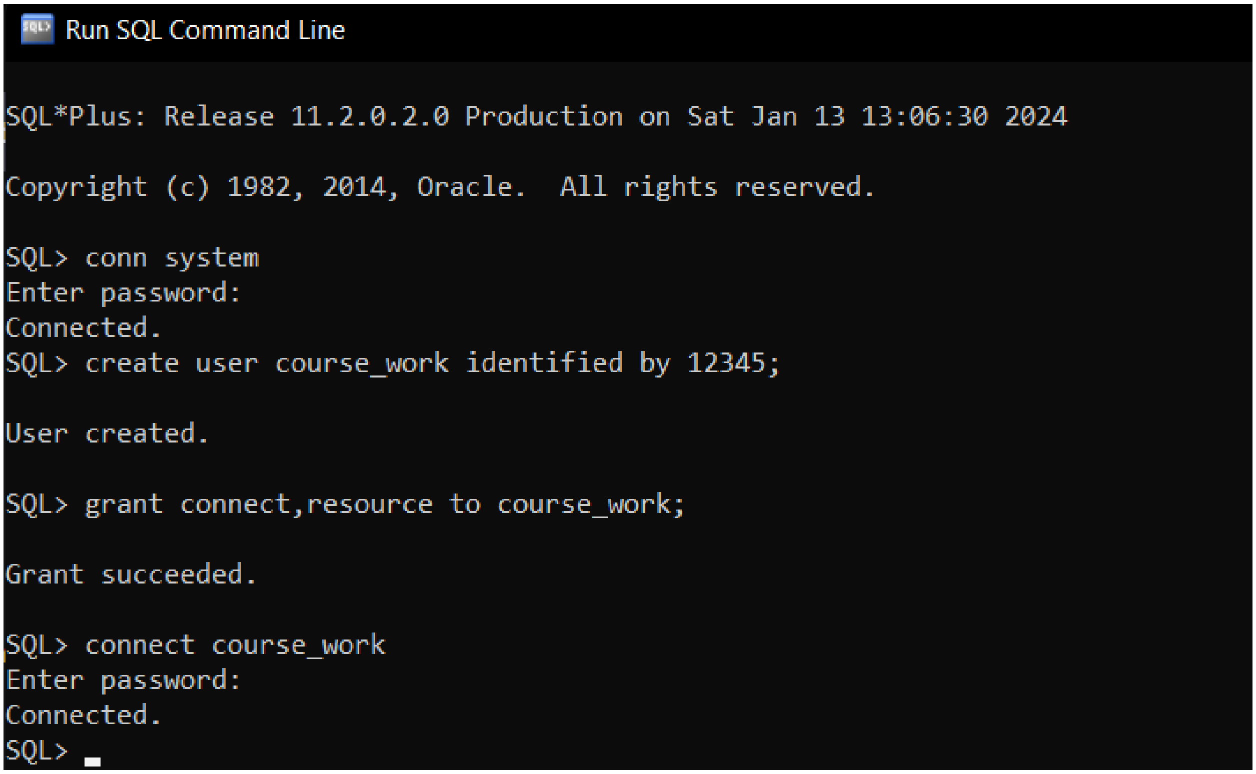
Grant succeeded.

SQL> connect course_work

Enter password:

Connected.

SQL>



Run SQL Command Line

```
SQL*Plus: Release 11.2.0.2.0 Production on Sat Jan 13 13:06:30 2024

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

SQL> conn system
Enter password:
Connected.
SQL> create user course_work identified by 12345;

User created.

SQL> grant connect,resource to course_work;

Grant succeeded.

SQL> connect course_work
Enter password:
Connected.
SQL> _
```

Figure 3: Connecting and creating new user

Creating Customer_Category_Details Table:

Query:

CREATE TABLE Customer_Category_Details (

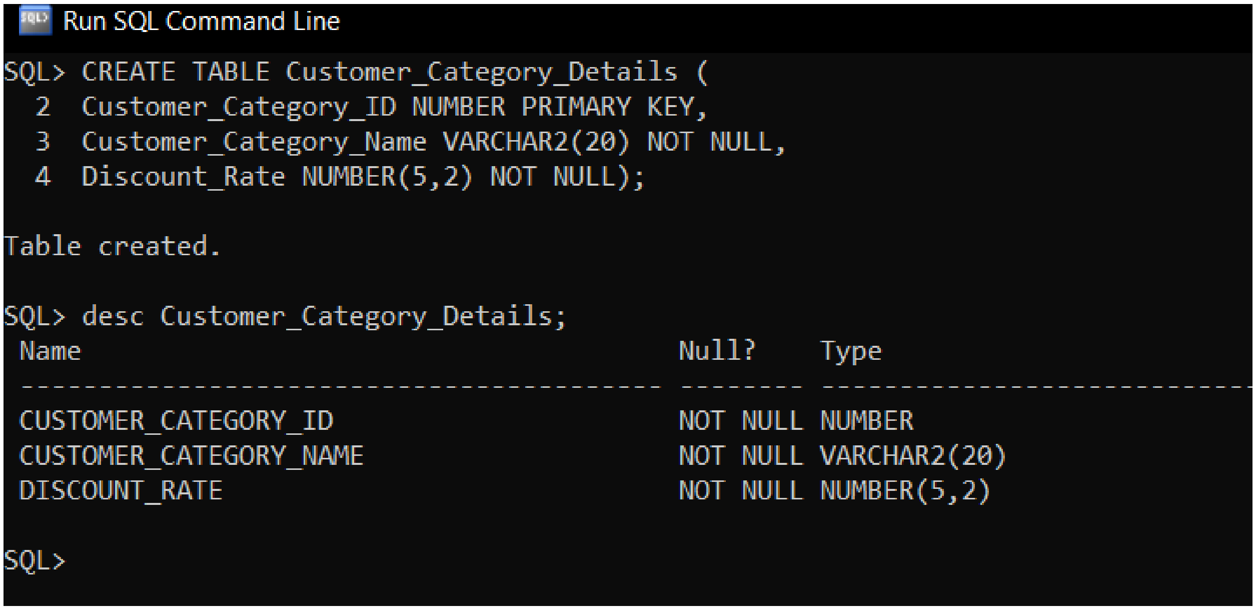
Customer_Category_ID NUMBER PRIMARY KEY,

Customer_Category_Name VARCHAR2(20) NOT NULL,

Discount_Rate NUMBER (5,2) NOT NULL);

Explanation:

Creates a table "Customer_Category_Details" with ID, Name, and Discount Rate columns, where ID is the primary key, and Name and Discount Rate are mandatory, with Discount Rate having a precision of 5 digits, including 2 decimal places.



```
SQL> CREATE TABLE Customer_Category_Details (
  2  Customer_Category_ID NUMBER PRIMARY KEY,
  3  Customer_Category_Name VARCHAR2(20) NOT NULL,
  4  Discount_Rate NUMBER(5,2) NOT NULL);

Table created.

SQL> desc Customer_Category_Details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_CATEGORY_ID                      NOT NULL NUMBER
 CUSTOMER_CATEGORY_NAME                    NOT NULL VARCHAR2(20)
 DISCOUNT_RATE                             NOT NULL NUMBER(5,2)

SQL>
```

Figure 4: Creating and Describing Customer_Category_Details

Creating Address_Details Table:

Query:

CREATE TABLE Address_Details (

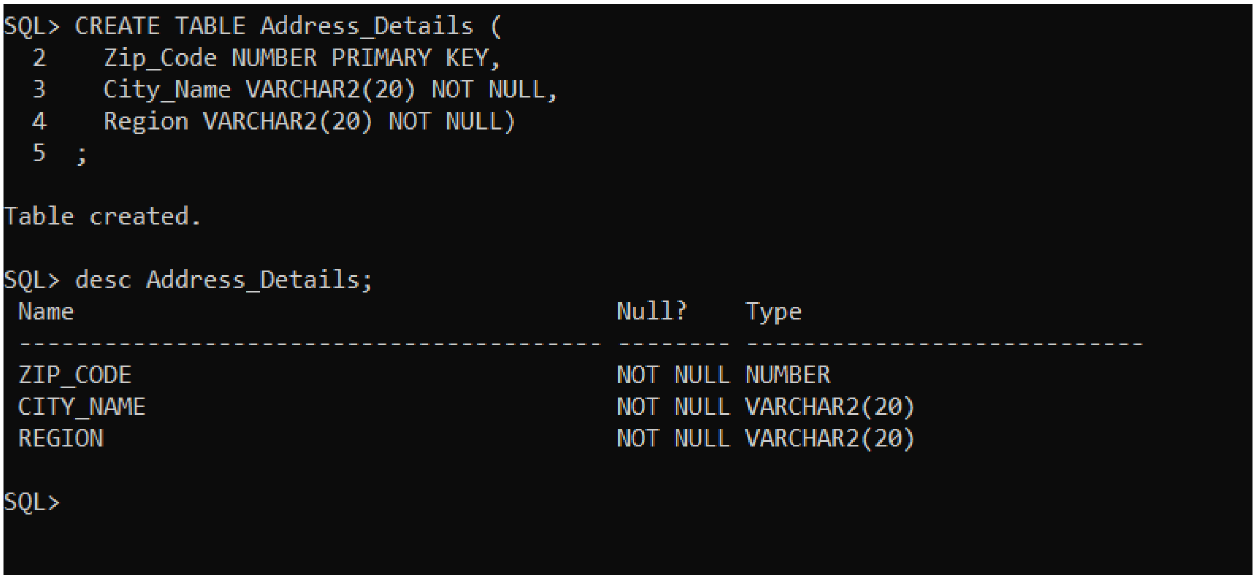
Zip_Code NUMBER PRIMARY KEY,

City_Name VARCHAR2(20) NOT NULL,

Region VARCHAR2(20) NOT NULL);

Explanation:

Creates a table "Address_Details" with Zip Code as the primary key, and non-null columns for City Name and Region.



```
SQL> CREATE TABLE Address_Details (
  2    Zip_Code NUMBER PRIMARY KEY,
  3    City_Name VARCHAR2(20) NOT NULL,
  4    Region VARCHAR2(20) NOT NULL)
  5  ;

Table created.

SQL> desc Address_Details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ZIP_CODE                                  NOT NULL NUMBER
 CITY_NAME                                 NOT NULL VARCHAR2(20)
 REGION                                    NOT NULL VARCHAR2(20)

SQL>
```

Figure 5: Creating and Describing Address_Details.

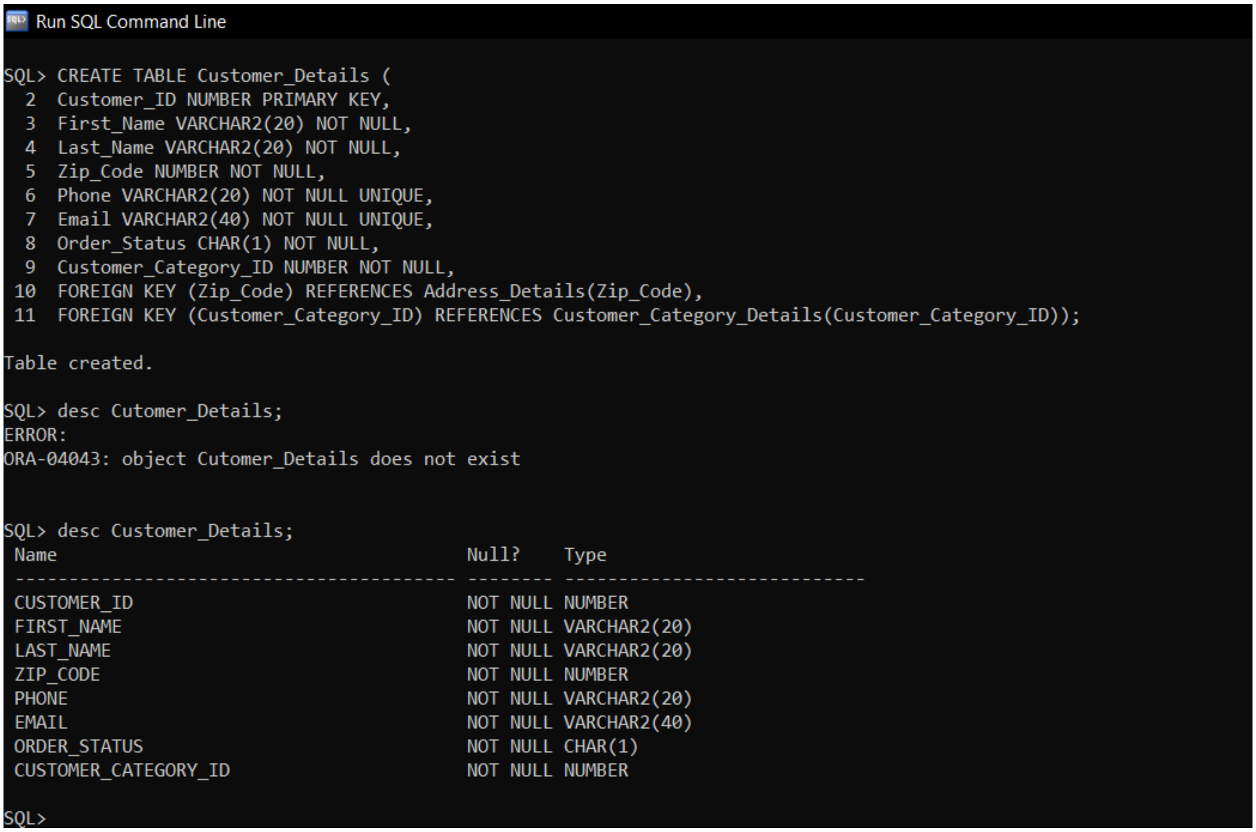
Creating Customer_Details Table:

Query:

```
CREATE TABLE Customer_Details (
Customer_ID NUMBER PRIMARY KEY,
First_Name VARCHAR2(20) NOT NULL,
Last_Name VARCHAR2(20) NOT NULL,
Zip_Code NUMBER NOT NULL,
Phone VARCHAR2(20) NOT NULL UNIQUE,
Email VARCHAR2(40) NOT NULL UNIQUE,
Order_Status CHAR(1) NOT NULL,
Customer_Category_ID NUMBER NOT NULL,
FOREIGN KEY (Zip_Code) REFERENCES Address_Details(Zip_Code),
FOREIGN KEY (Customer_Category_ID) REFERENCES
Customer_Category_Details(Customer_Category_ID));
```

Explanation:

Creates a table "Customer_Details" with Customer ID as the primary key, and non-null columns for First Name, Last Name, Zip Code, Phone (unique), Email (unique), Order Status, and Customer Category ID. Defines foreign key constraints on Zip Code referencing Address_Details and Customer Category ID referencing Customer_Category_Details.



```
SQL> CREATE TABLE Customer_Details (
  2  Customer_ID NUMBER PRIMARY KEY,
  3  First_Name VARCHAR2(20) NOT NULL,
  4  Last_Name VARCHAR2(20) NOT NULL,
  5  Zip_Code NUMBER NOT NULL,
  6  Phone VARCHAR2(20) NOT NULL UNIQUE,
  7  Email VARCHAR2(40) NOT NULL UNIQUE,
  8  Order_Status CHAR(1) NOT NULL,
  9  Customer_Category_ID NUMBER NOT NULL,
 10  FOREIGN KEY (Zip_Code) REFERENCES Address_Details(Zip_Code),
 11  FOREIGN KEY (Customer_Category_ID) REFERENCES Customer_Category_Details(Customer_Category_ID));

Table created.

SQL> desc Cutomer_Details;
ERROR:
ORA-04043: object Cutomer_Details does not exist


SQL> desc Customer_Details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_ID                               NOT NULL NUMBER
 FIRST_NAME                                NOT NULL VARCHAR2(20)
 LAST_NAME                                 NOT NULL VARCHAR2(20)
 ZIP_CODE                                  NOT NULL NUMBER
 PHONE                                     NOT NULL VARCHAR2(20)
 EMAIL                                     NOT NULL VARCHAR2(40)
 ORDER_STATUS                              NOT NULL CHAR(1)
 CUSTOMER_CATEGORY_ID                      NOT NULL NUMBER

SQL>
```

Figure 6: Creating and Describing Customer_Details.

Creating Invoice_Details Table:

Query:

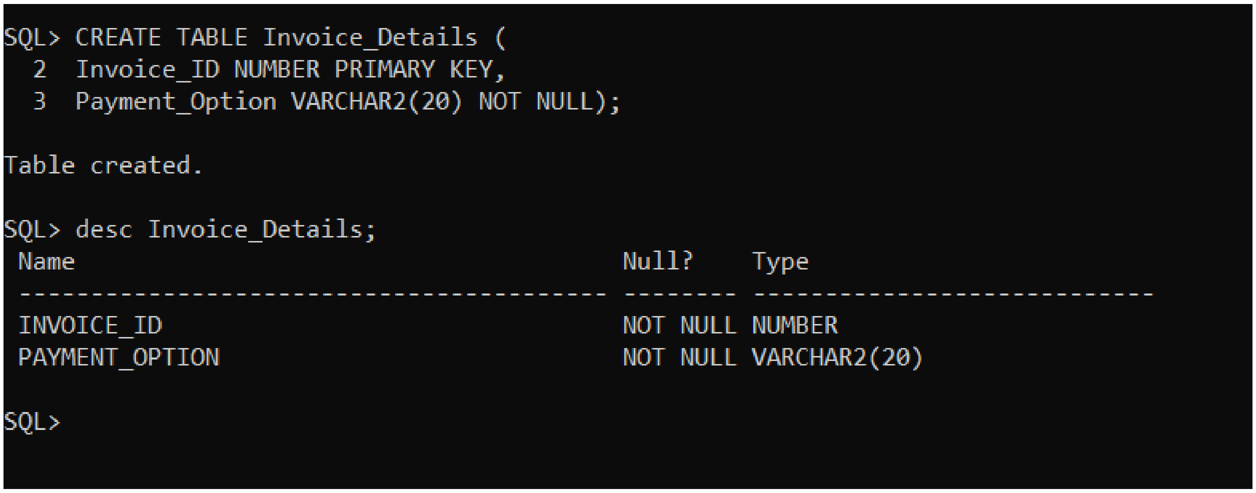
CREATE TABLE Invoice_Details (

Invoice_ID NUMBER PRIMARY KEY,

Payment_Option VARCHAR2(20) NOT NULL);

Explanation:

Creates a table "Invoice_Details" with Invoice ID as the primary key and a non-null Payment Option column.



```
SQL> CREATE TABLE Invoice_Details (
  2  Invoice_ID NUMBER PRIMARY KEY,
  3  Payment_Option VARCHAR2(20) NOT NULL);

Table created.

SQL> desc Invoice_Details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 INVOICE_ID                                NOT NULL NUMBER
 PAYMENT_OPTION                            NOT NULL VARCHAR2(20)

SQL>
```

Figure 7: Creating and Describing Invoice_Details.

Creating Order_Details Table:

Query:

CREATE TABLE Order_Details (

Order_ID NUMBER PRIMARY KEY,

Invoice_ID NUMBER NOT NULL,

Order_Date DATE NOT NULL,

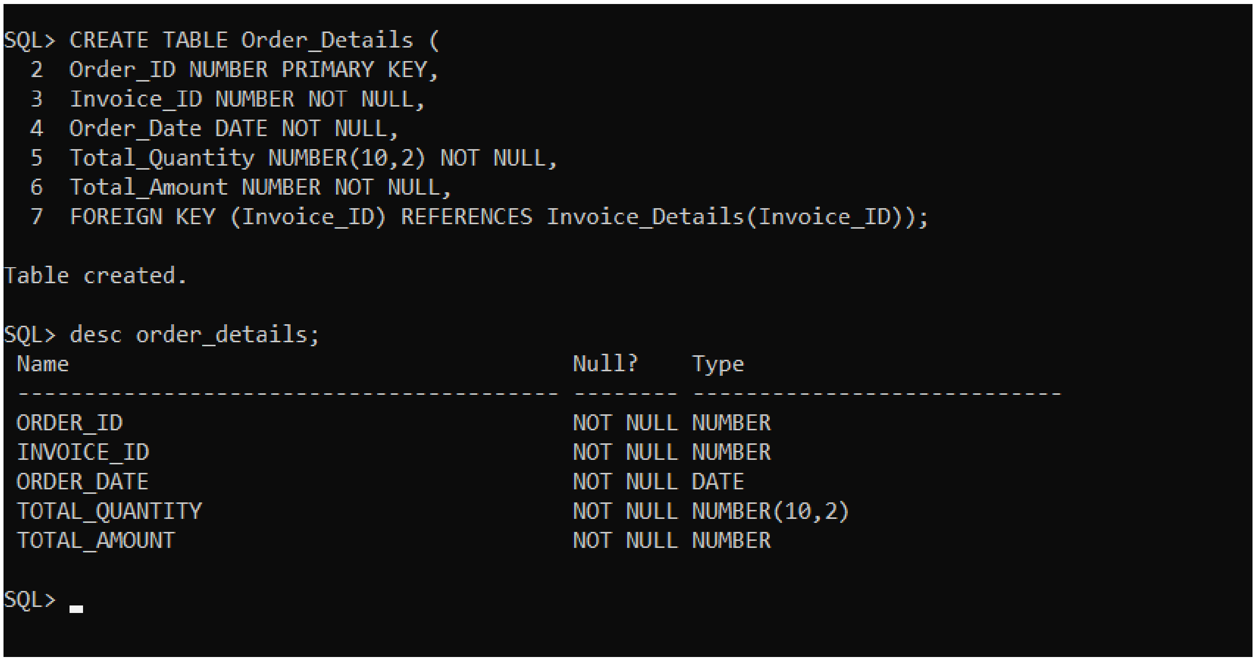
Total_Quantity NUMBER(10,2) NOT NULL,

Total_Amount NUMBER NOT NULL,

FOREIGN KEY (Invoice_ID) REFERENCES Invoice_Details(Invoice_ID));

Explanation:

Creates a table "Order_Details" with Order ID as the primary key, including non-null columns for Invoice ID, Order Date, Total Quantity, and Total Amount. Establishes a foreign key relationship with "Invoice_Details" based on the Invoice ID.



```
SQL> CREATE TABLE Order_Details (
  2  Order_ID NUMBER PRIMARY KEY,
  3  Invoice_ID NUMBER NOT NULL,
  4  Order_Date DATE NOT NULL,
  5  Total_Quantity NUMBER(10,2) NOT NULL,
  6  Total_Amount NUMBER NOT NULL,
  7  FOREIGN KEY (Invoice_ID) REFERENCES Invoice_Details(Invoice_ID));

Table created.

SQL> desc order_details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORDER_ID                                  NOT NULL NUMBER
 INVOICE_ID                                NOT NULL NUMBER
 ORDER_DATE                                NOT NULL DATE
 TOTAL_QUANTITY                            NOT NULL NUMBER(10,2)
 TOTAL_AMOUNT                              NOT NULL NUMBER

SQL> _
```

Figure 8: Creating and Descrbing Order_Details

Creating Customer_Order_Details Table:

Query:

CREATE TABLE Customer_Order_Details (

Customer_ID NUMBER NOT NULL,

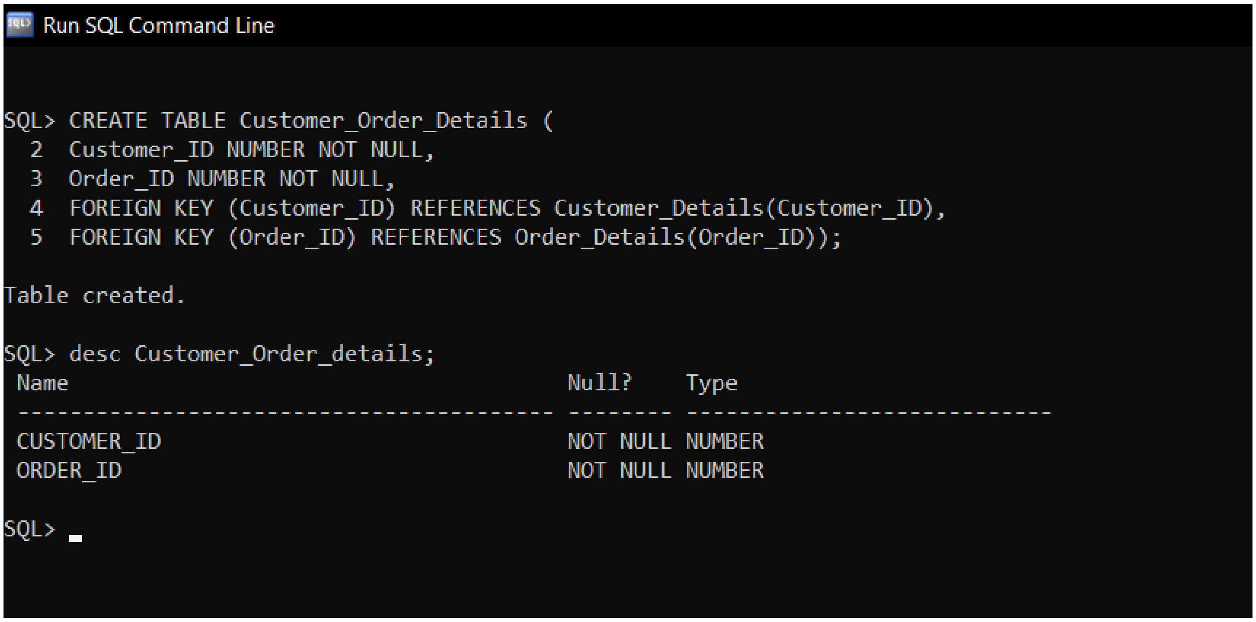
Order_ID NUMBER NOT NULL,

FOREIGN KEY (Customer_ID) REFERENCES Customer_Details (Customer_ID),

FOREIGN KEY (Order_ID) REFERENCES Order_Details (Order_ID));

Explanation:

Creates a table "Customer_Order_Details" with non-null columns for Customer ID and Order ID, establishing foreign key relationships with "Customer_Details" and "Order_Details" tables.



```
SQL> CREATE TABLE Customer_Order_Details (
  2  Customer_ID NUMBER NOT NULL,
  3  Order_ID NUMBER NOT NULL,
  4  FOREIGN KEY (Customer_ID) REFERENCES Customer_Details(Customer_ID),
  5  FOREIGN KEY (Order_ID) REFERENCES Order_Details(Order_ID));

Table created.

SQL> desc Customer_Order_details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_ID                               NOT NULL NUMBER
 ORDER_ID                                  NOT NULL NUMBER

SQL>
```

Figure 9: Creating and Describing Customer_Order_Details.

Creating Inventory_Details Table:

Query:

CREATE TABLE Inventory_Details (

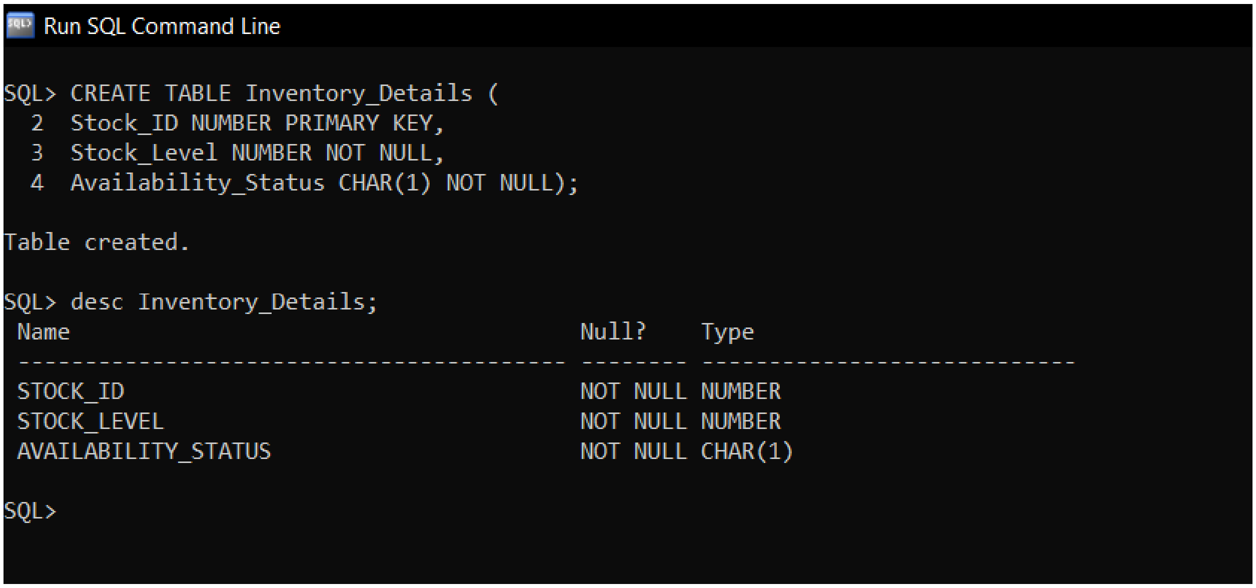
Stock_ID NUMBER PRIMARY KEY,

Stock_Level NUMBER NOT NULL,

Availability_Status CHAR (1) NOT NULL);

Explanation:

Creates a table "Inventory_Details" with Stock ID as the primary key and non-null columns for Stock Level and Availability Status.



```
SQL> CREATE TABLE Inventory_Details (
  2  Stock_ID NUMBER PRIMARY KEY,
  3  Stock_Level NUMBER NOT NULL,
  4  Availability_Status CHAR(1) NOT NULL);

Table created.

SQL> desc Inventory_Details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 STOCK_ID                                  NOT NULL NUMBER
 STOCK_LEVEL                               NOT NULL NUMBER
 AVAILABILITY_STATUS                       NOT NULL CHAR(1)

SQL>
```

Figure 10: Creating and Describing Inventory_Details.

Creating Vendor_Details Table:

Query:

CREATE TABLE Vendor_Details (

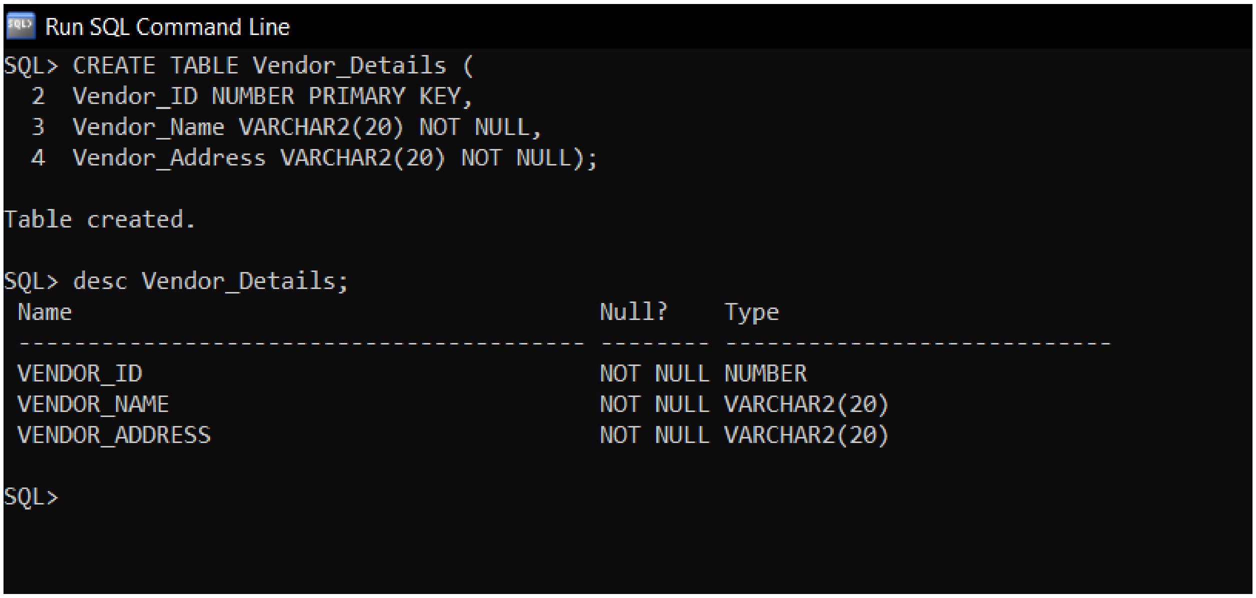
Vendor_ID NUMBER PRIMARY KEY,

Vendor_Name VARCHAR2(20) NOT NULL,

Vendor_Address VARCHAR2(20) NOT NULL);

Explanation:

Creates a table "Vendor_Details" with Vendor ID as the primary key and non-null columns for Vendor Name and Vendor Address.



```
SQL> CREATE TABLE Vendor_Details (
  2  Vendor_ID NUMBER PRIMARY KEY,
  3  Vendor_Name VARCHAR2(20) NOT NULL,
  4  Vendor_Address VARCHAR2(20) NOT NULL);

Table created.

SQL> desc Vendor_Details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 VENDOR_ID                                 NOT NULL NUMBER
 VENDOR_NAME                               NOT NULL VARCHAR2(20)
 VENDOR_ADDRESS                            NOT NULL VARCHAR2(20)

SQL>
```

Figure 11: Creating and Describing Vendor_Details.

Creating Product_Details Table:

Query:

CREATE TABLE Product_Details (

Product_ID NUMBER PRIMARY KEY,

Product_Name VARCHAR2(20) NOT NULL,

Product_Description VARCHAR2(400) NOT NULL,

Product_Category VARCHAR2(20) NOT NULL,

Price NUMBER NOT NULL,

Stock_ID NUMBER NOT NULL,

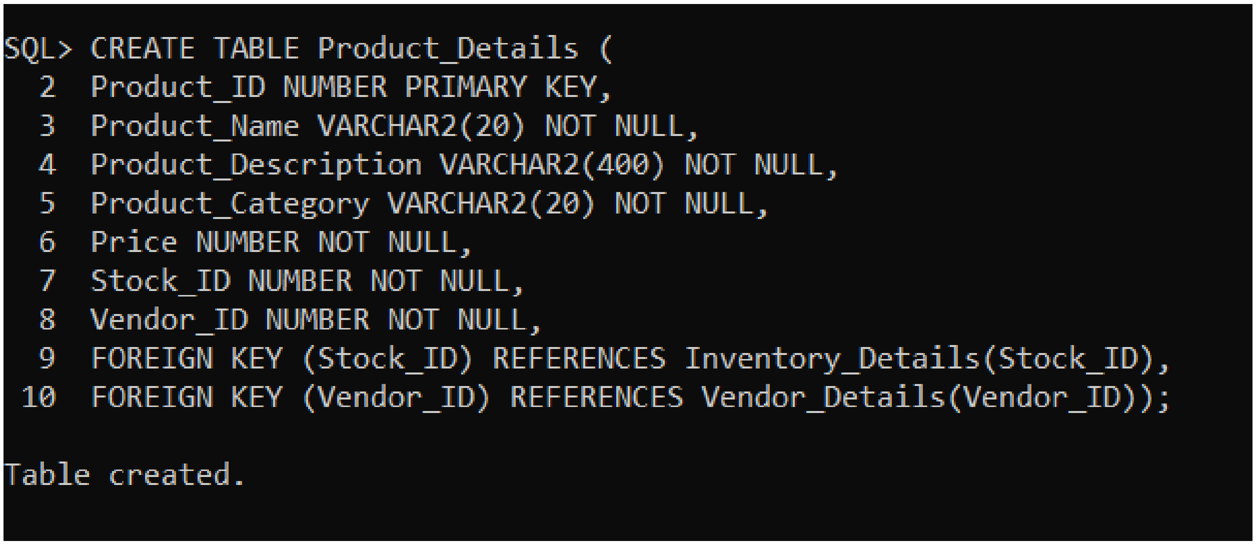
Vendor_ID NUMBER NOT NULL,

FOREIGN KEY (Stock_ID) REFERENCES Inventory_Details(Stock_ID),

FOREIGN KEY (Vendor_ID) REFERENCES Vendor_Details(Vendor_ID));

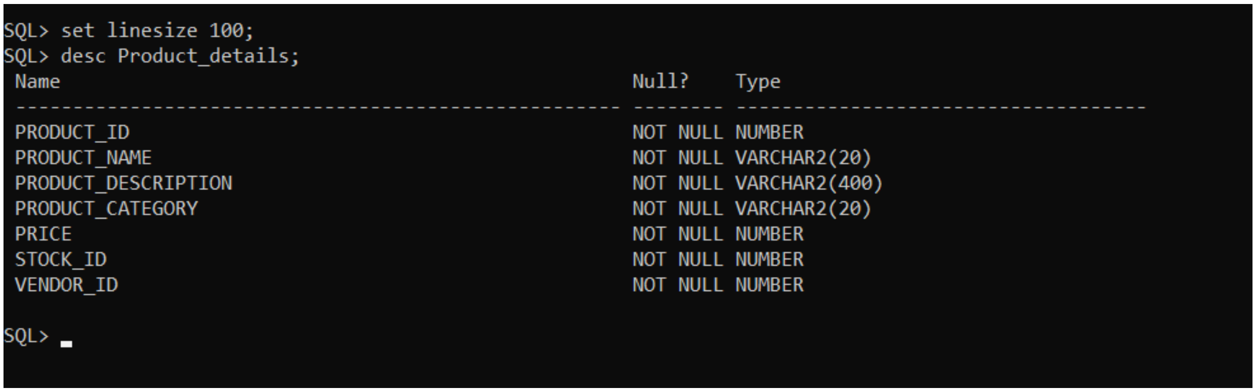
Explanation:

Creates a table "Product_Details" with Product ID as the primary key and non-null columns for Product Name, Product Description, Product Category, Price, Stock ID, and Vendor ID. Establishes foreign key relationships with "Inventory_Details" based on Stock ID and "Vendor_Details" based on Vendor ID.



```
SQL> CREATE TABLE Product_Details (
     Product_ID NUMBER PRIMARY KEY,
     Product_Name VARCHAR2(20) NOT NULL,
     Product_Description VARCHAR2(400) NOT NULL,
     Product_Category VARCHAR2(20) NOT NULL,
     Price NUMBER NOT NULL,
     Stock_ID NUMBER NOT NULL,
     Vendor_ID NUMBER NOT NULL,
     FOREIGN KEY (Stock_ID) REFERENCES Inventory_Details(Stock_ID),
     FOREIGN KEY (Vendor_ID) REFERENCES Vendor_Details(Vendor_ID));

Table created.
```

Figure 12: Creating Product_Details table.



```
SQL> set linesize 100;
SQL> desc Product_details;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ------------------------------------
 PRODUCT_ID                                            NOT NULL NUMBER
 PRODUCT_NAME                                          NOT NULL VARCHAR2(20)
 PRODUCT_DESCRIPTION                                   NOT NULL VARCHAR2(400)
 PRODUCT_CATEGORY                                      NOT NULL VARCHAR2(20)
 PRICE                                                 NOT NULL NUMBER
 STOCK_ID                                              NOT NULL NUMBER
 VENDOR_ID                                             NOT NULL NUMBER

SQL>
```

Figure 13: Describing Product_Details table.

Creating Order_Product_Details Table:

Query:

CREATE TABLE Order_Product_Details (

Product_ID NUMBER NOT NULL,

Order_ID NUMBER NOT NULL,

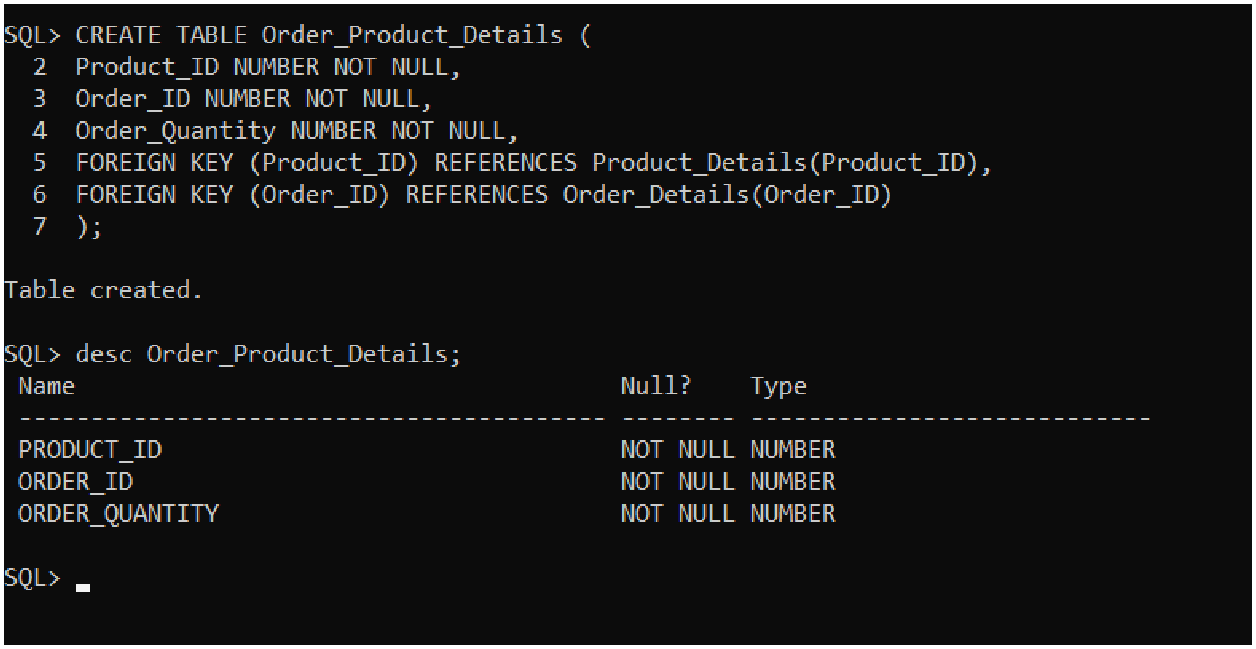
Order_Quantity NUMBER NOT NULL,

FOREIGN KEY (Product_ID) REFERENCES Product_Details (Product_ID),

FOREIGN KEY (Order_ID) REFERENCES Order_Details (Order_ID));

Explanation:

Creates a table "Order_Product_Details" with non-null columns for Product ID, Order ID, and Order Quantity. Establishes foreign key relationships with "Product_Details" based on Product ID and "Order_Details" based on Order ID.



```
SQL> CREATE TABLE Order_Product_Details (
  2  Product_ID NUMBER NOT NULL,
  3  Order_ID NUMBER NOT NULL,
  4  Order_Quantity NUMBER NOT NULL,
  5  FOREIGN KEY (Product_ID) REFERENCES Product_Details(Product_ID),
  6  FOREIGN KEY (Order_ID) REFERENCES Order_Details(Order_ID)
  7  );

Table created.

SQL> desc Order_Product_Details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODUCT_ID                                NOT NULL NUMBER
 ORDER_ID                                  NOT NULL NUMBER
 ORDER_QUANTITY                            NOT NULL NUMBER

SQL>
```

Figure 14:Creating Order_Product_Details table.

Inserting Value on Customer_Category_Details:

Query:

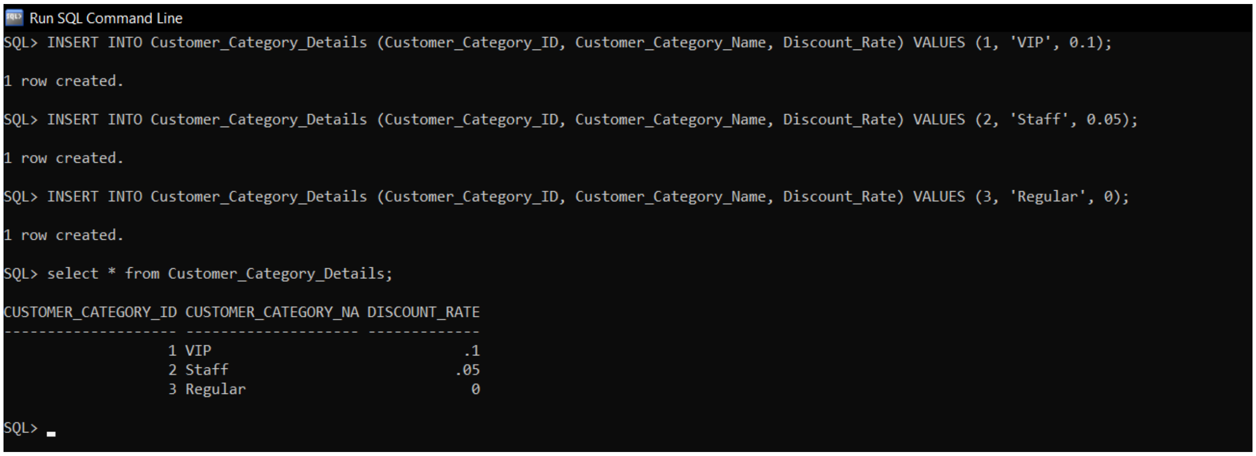
INSERT INTO Customer_Category_Details (Customer_Category_ID, Customer_Category_Name, Discount_Rate) VALUES (1, 'VIP', 0.1);

INSERT INTO Customer_Category_Details (Customer_Category_ID, Customer_Category_Name, Discount_Rate) VALUES (2, 'Staff', 0.05);

INSERT INTO Customer_Category_Details (Customer_Category_ID, Customer_Category_Name, Discount_Rate) VALUES (3, 'Regular', 0);

Explanation:

Inserts three rows into "Customer_Category_Details" table for VIP, Staff, and Regular categories with corresponding discount rates of 0.1, 0.05, and 0, respectively.



```
SQL> INSERT INTO Customer_Category_Details (Customer_Category_ID, Customer_Category_Name, Discount_Rate) VALUES (1, 'VIP', 0.1);

1 row created.

SQL> INSERT INTO Customer_Category_Details (Customer_Category_ID, Customer_Category_Name, Discount_Rate) VALUES (2, 'Staff', 0.05);

1 row created.

SQL> INSERT INTO Customer_Category_Details (Customer_Category_ID, Customer_Category_Name, Discount_Rate) VALUES (3, 'Regular', 0);

1 row created.

SQL> select * from Customer_Category_Details;

CUSTOMER_CATEGORY_ID CUSTOMER_CATEGORY_NA DISCOUNT_RATE
-------------------- -------------------- -------------
                   1 VIP                             .1
                   2 Staff                          .05
                   3 Regular                          0

SQL> _
```

Figure 15: Insertion in Customer_Category_Details

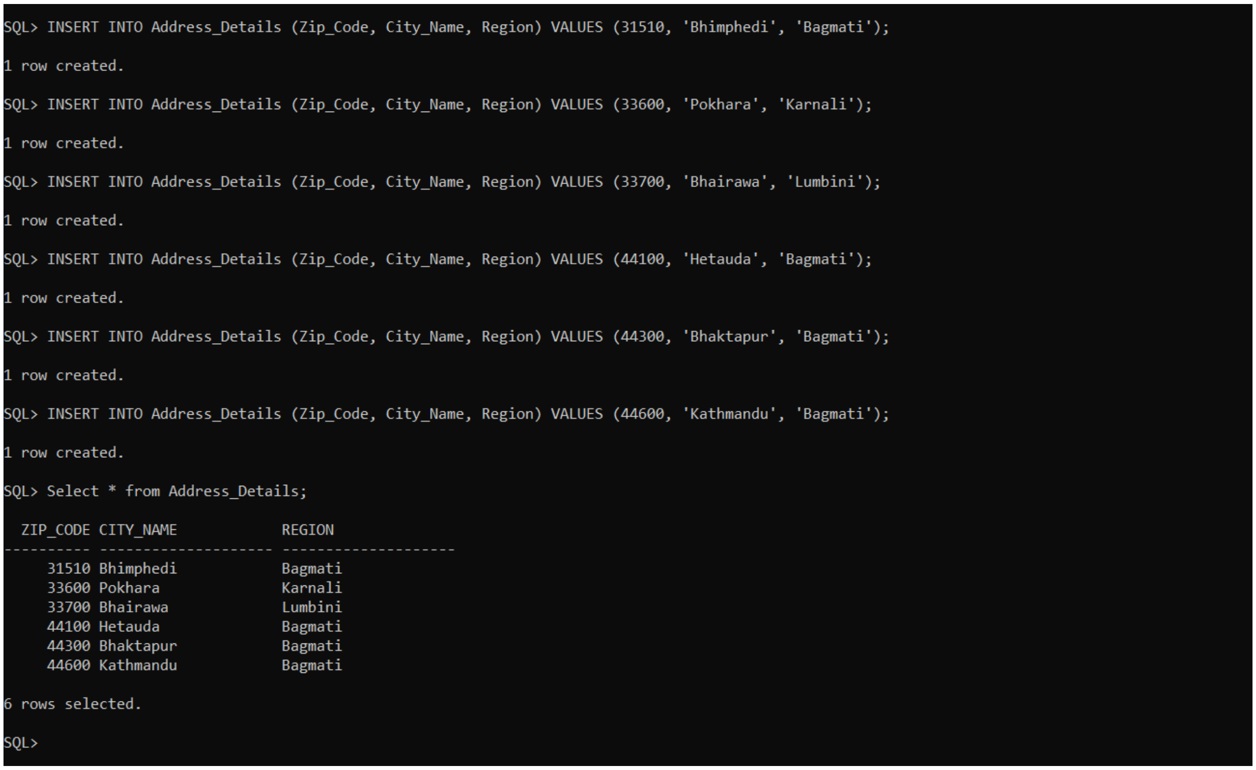
Inserting Value on Address_Details:

Query:

```
INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (31510, 'Bhimphedi', 'Bagmati');
INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (33600, 'Pokhara', 'Karnali');
INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (33700, 'Bhairawa', 'Lumbini');
INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (44100, 'Hetauda', 'Bagmati');
INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (44300, 'Bhaktapur', 'Bagmati');
INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (44600, 'Kathmandu', 'Bagmati');
INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (44700, 'Nuwakot', 'Bagmati');
```

Explanation:

Inserts six rows into the "Address_Details" table, providing values for Zip_Code, City_Name, and Region, representing different locations in Nepal: Bhimphedi, Pokhara, Bhairawa, Hetauda, Bhaktapur, and Kathmandu.



```
SQL> INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (31510, 'Bhimphedi', 'Bagmati');

1 row created.

SQL> INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (33600, 'Pokhara', 'Karnali');

1 row created.

SQL> INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (33700, 'Bhairawa', 'Lumbini');

1 row created.

SQL> INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (44100, 'Hetauda', 'Bagmati');

1 row created.

SQL> INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (44300, 'Bhaktapur', 'Bagmati');

1 row created.

SQL> INSERT INTO Address_Details (Zip_Code, City_Name, Region) VALUES (44600, 'Kathmandu', 'Bagmati');

1 row created.

SQL> Select * from Address_Details;

  ZIP_CODE CITY_NAME            REGION
---------- -------------------- --------------------
     31510 Bhimphedi            Bagmati
     33600 Pokhara              Karnali
     33700 Bhairawa             Lumbini
     44100 Hetauda              Bagmati
     44300 Bhaktapur            Bagmati
     44600 Kathmandu            Bagmati

6 rows selected.

SQL>
```

Figure 16: Insertion and Address_details table

Inserting Value on Customer_Details:

Query:

```sql
INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (1, 'Rajesh', 'Hamal', 33700, '9841909080', 'rajesh@gmail.com', 'N', 3);

INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (2, 'Nikhil', 'Upreti', 44300, '9841076411', 'nikhil@gmail.com', 'N', 3);

INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (3, 'Prabhab', 'Khanal', 44300, '9843347222', 'prabhab@gmail.com', 'Y', 2);

INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (4, 'Alin', 'Basnet', 44600, '9801203873', 'alin@gmail.com', 'Y', 1);

INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (5, 'Renish', 'Khadka', 44100, '9845074066', 'renish@gmail.com', 'Y', 1);

INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (6, 'Sandeep', 'Lamichhane', 31510, '9856923459', 'sandeep@gmail.com', 'Y', 3);

INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (7, 'Paul', 'Shah', 44700, '9876543210', 'paul@gmail.com', 'Y', 3);
```

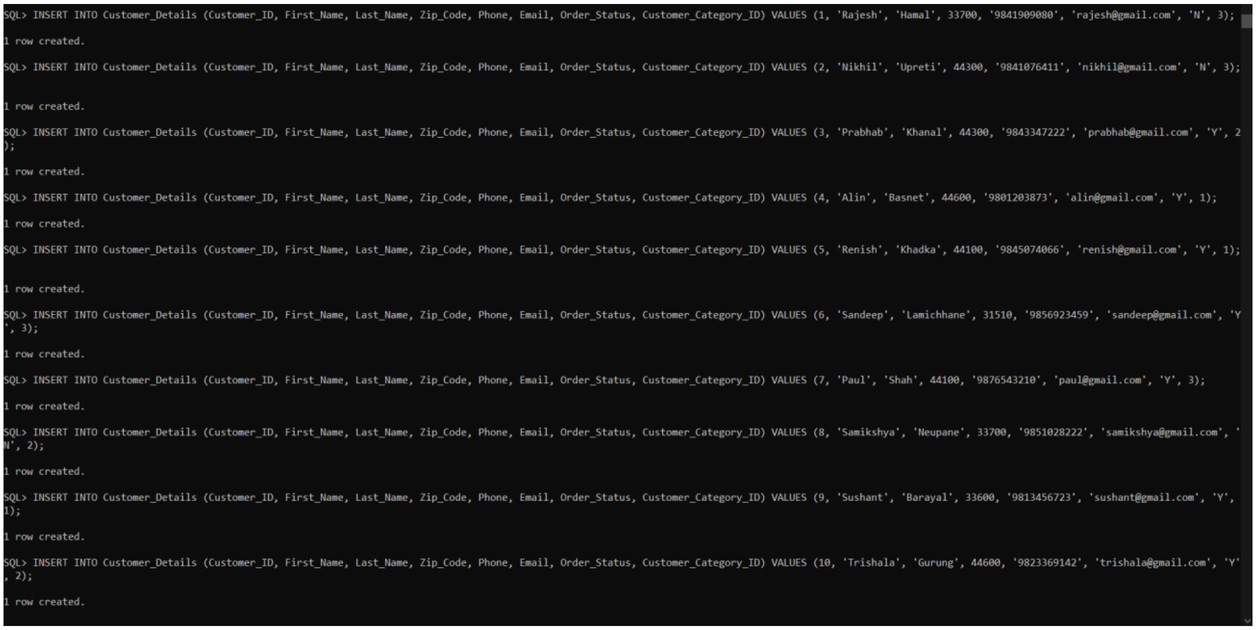
INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (8, 'Samikshya', 'Neupane', 33700, '9851028222', 'samikshya@gmail.com', 'N', 2);

INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (9, 'Sushant', 'Barayal', 33600, '9813456723', 'sushant@gmail.com', 'Y', 1);

INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (10, 'Trishala', 'Gurung', 44600, '9823369142', 'trishala@gmail.com', 'Y', 2);

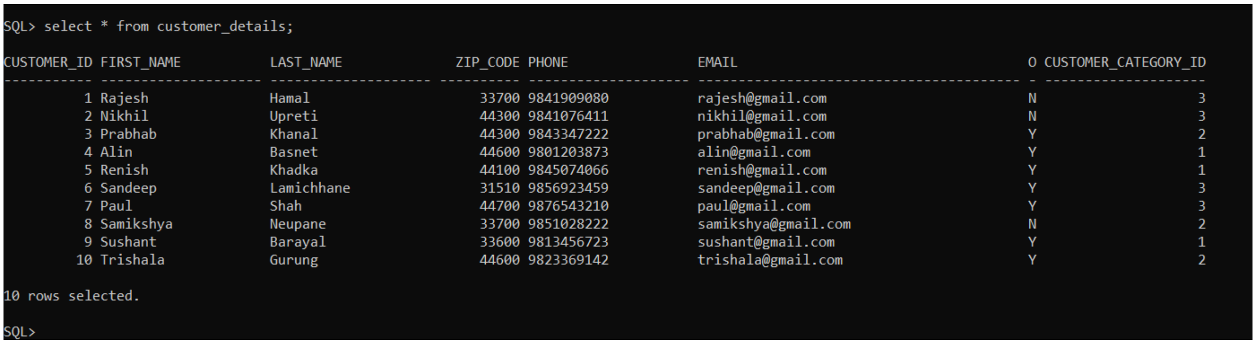
Explanation:

Inserts ten rows into the "Customer_Details" table, providing values for Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, and Customer_Category_ID. Each row represents a customer with unique information, including different Zip Codes, contact details, and customer categories.



```
SQL> INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (1, 'Rajesh', 'Hamal', 33700, '9841909080', 'rajesh@gmail.com', 'N', 3);

1 row created.

SQL> INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (2, 'Nikhil', 'Upreti', 44300, '9841076411', 'nikhil@gmail.com', 'N', 3);


1 row created.

SQL> INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (3, 'Prabhab', 'Khanal', 44300, '9843347222', 'prabhab@gmail.com', 'Y', 2
);

1 row created.

SQL> INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (4, 'Alin', 'Basnet', 44600, '9801203873', 'alin@gmail.com', 'Y', 1);

1 row created.

SQL> INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (5, 'Renish', 'Khadka', 44100, '9845074066', 'renish@gmail.com', 'Y', 1);


1 row created.

SQL> INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (6, 'Sandeep', 'Lamichhane', 31510, '9856923459', 'sandeep@gmail.com', 'Y
', 3);

1 row created.

SQL> INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (7, 'Paul', 'Shah', 44100, '9876543210', 'paul@gmail.com', 'Y', 3);

1 row created.

SQL> INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (8, 'Samikshya', 'Neupane', 33700, '9851028222', 'samikshya@gmail.com', '
N', 2);

1 row created.

SQL> INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (9, 'Sushant', 'Barayal', 33600, '9813456723', 'sushant@gmail.com', 'Y',
1);

1 row created.

SQL> INSERT INTO Customer_Details (Customer_ID, First_Name, Last_Name, Zip_Code, Phone, Email, Order_Status, Customer_Category_ID) VALUES (10, 'Trishala', 'Gurung', 44600, '9823369142', 'trishala@gmail.com', 'Y'
, 2);

1 row created.
```

Figure 17: Insertion in Customer_Details



```
SQL> select * from customer_details;

CUSTOMER_ID FIRST_NAME           LAST_NAME              ZIP_CODE PHONE                EMAIL                                    O CUSTOMER_CATEGORY_ID
----------- -------------------- -------------------- ---------- -------------------- ---------------------------------------- - --------------------
          1 Rajesh               Hamal                     33700 9841909080           rajesh@gmail.com                         N                    3
          2 Nikhil               Upreti                    44300 9841076411           nikhil@gmail.com                         N                    3
          3 Prabhab              Khanal                    44300 9843347222           prabhab@gmail.com                        Y                    2
          4 Alin                 Basnet                    44600 9801203873           alin@gmail.com                           Y                    1
          5 Renish               Khadka                    44100 9845074066           renish@gmail.com                         Y                    1
          6 Sandeep              Lamichhane                31510 9856923459           sandeep@gmail.com                        Y                    3
          7 Paul                 Shah                      44700 9876543210           paul@gmail.com                           Y                    3
          8 Samikshya            Neupane                   33700 9851028222           samikshya@gmail.com                      N                    2
          9 Sushant              Barayal                   33600 9813456723           sushant@gmail.com                        Y                    1
         10 Trishala             Gurung                    44600 9823369142           trishala@gmail.com                       Y                    2

10 rows selected.

SQL>
```

Figure 18: Customer_Details

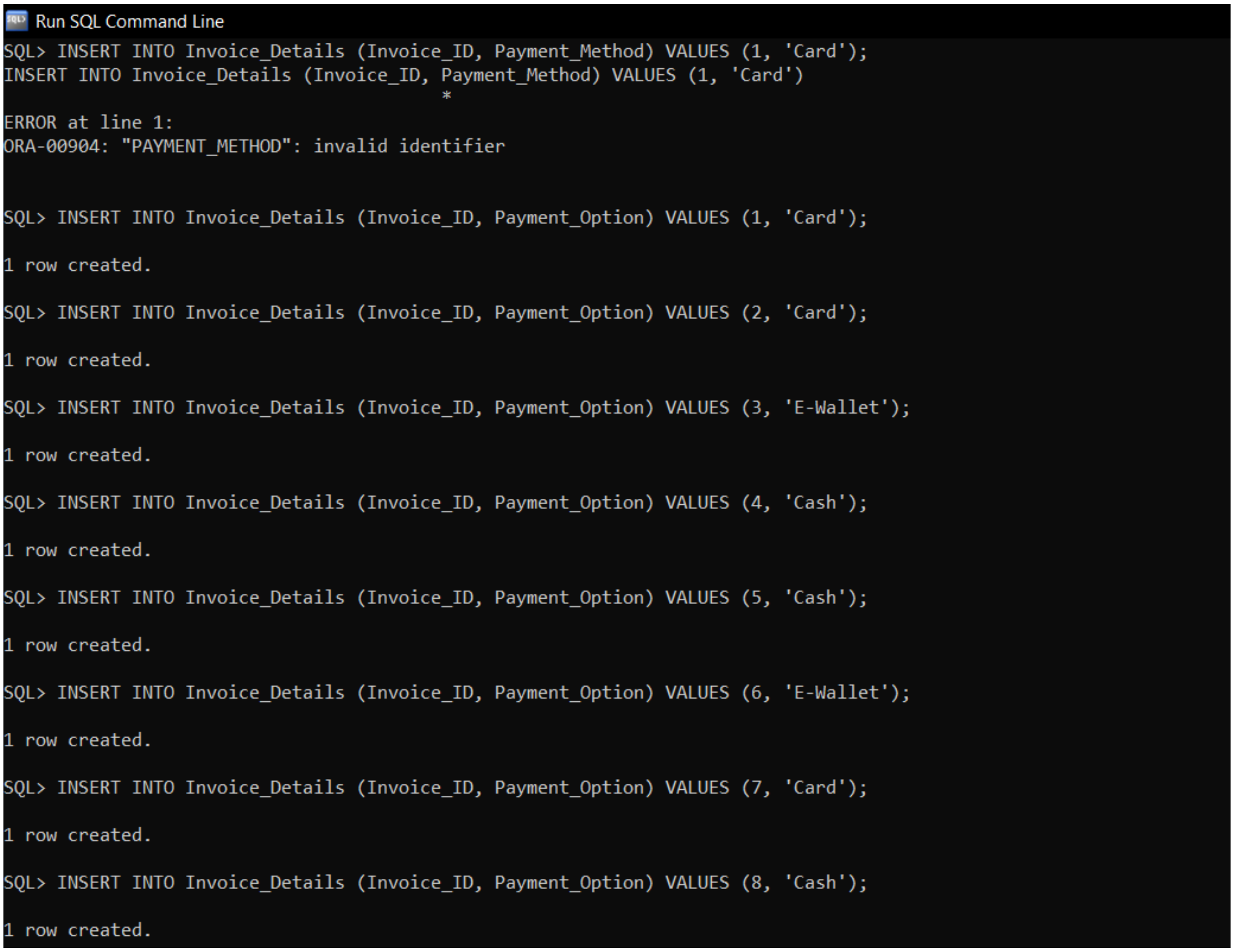
Inserting Value on Invoice _Details:

Query:

```
INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (1, 'Card');
INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (2, 'Card');
INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (3, 'E-Wallet');
INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (4, 'Cash');
INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (5, 'Cash');
INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (6, 'E-Wallet');
INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (7, 'Card');
INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (8, 'Cash');
```

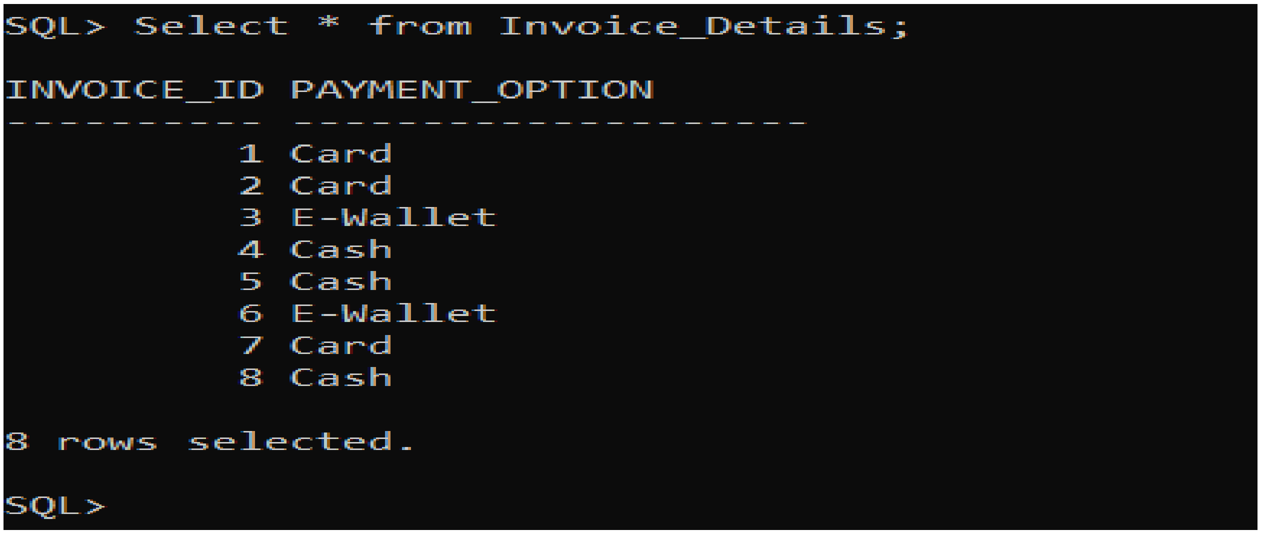
Explanation:

Inserts eight rows into the "Invoice_Details" table, providing values for Invoice_ID and Payment_Option, representing different payment options for invoices, including Card, E-Wallet, and Cash.



```
SQL> INSERT INTO Invoice_Details (Invoice_ID, Payment_Method) VALUES (1, 'Card');
INSERT INTO Invoice_Details (Invoice_ID, Payment_Method) VALUES (1, 'Card')
                                         *

ERROR at line 1:
ORA-00904: "PAYMENT_METHOD": invalid identifier


SQL> INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (1, 'Card');

1 row created.

SQL> INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (2, 'Card');

1 row created.

SQL> INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (3, 'E-Wallet');

1 row created.

SQL> INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (4, 'Cash');

1 row created.

SQL> INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (5, 'Cash');

1 row created.

SQL> INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (6, 'E-Wallet');

1 row created.

SQL> INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (7, 'Card');

1 row created.

SQL> INSERT INTO Invoice_Details (Invoice_ID, Payment_Option) VALUES (8, 'Cash');

1 row created.
```

Figure 19: Insertion in Invoice_Details



```
SQL> Select * from Invoice_Details;

INVOICE_ID PAYMENT_OPTION
---------- --------------------
         1 Card
         2 Card
         3 E-Wallet
         4 Cash
         5 Cash
         6 E-Wallet
         7 Card
         8 Cash

8 rows selected.

SQL>
```

Figure 20: Invoice_details

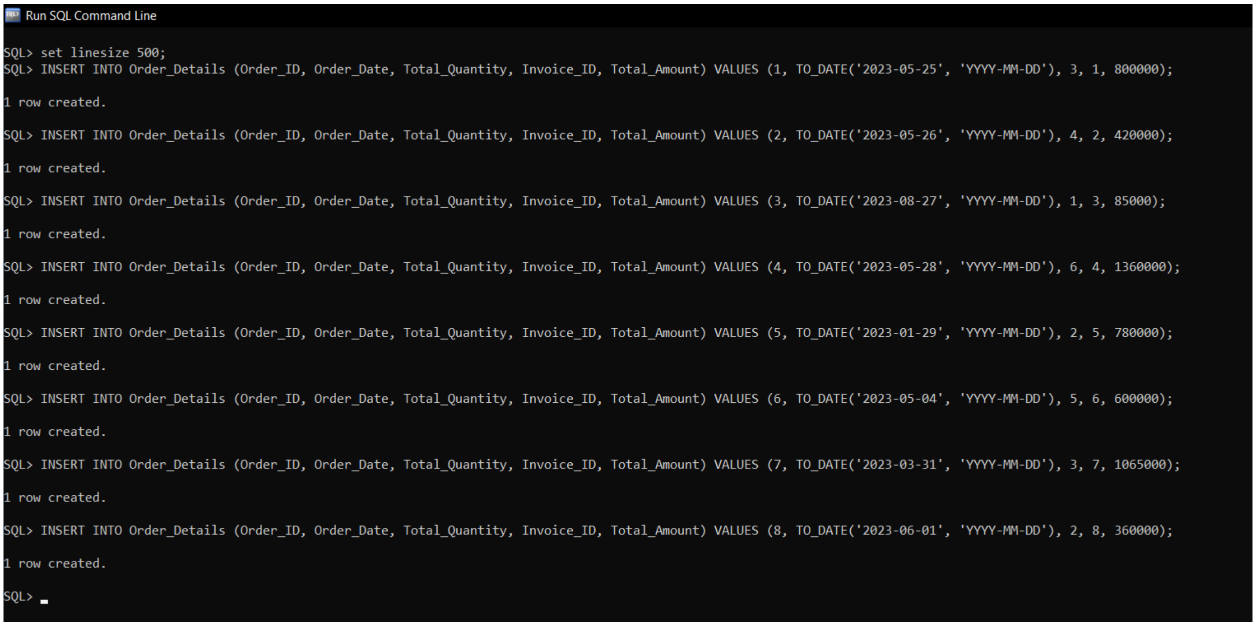
Inserting Value on Order_Details:

Query:

```
INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (1, TO_DATE('2023-05-25', 'YYYY-MM-DD'), 3, 1, 800000);

INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (2, TO_DATE('2023-08-26', 'YYYY-MM-DD'), 4, 2, 420000);

INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (3, TO_DATE('2023-08-27', 'YYYY-MM-DD'), 1, 3, 85000);

INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (4, TO_DATE('2023-05-28', 'YYYY-MM-DD'), 6, 4, 1360000);

INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (5, TO_DATE('2023-01-29', 'YYYY-MM-DD'), 2, 5, 780000);

INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (6, TO_DATE('2023-05-04', 'YYYY-MM-DD'), 5, 6, 600000);

INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (7, TO_DATE('2023-03-31', 'YYYY-MM-DD'), 3, 7, 1065000);

INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (8, TO_DATE('2023-06-01', 'YYYY-MM-DD'), 2, 8, 360000);
```

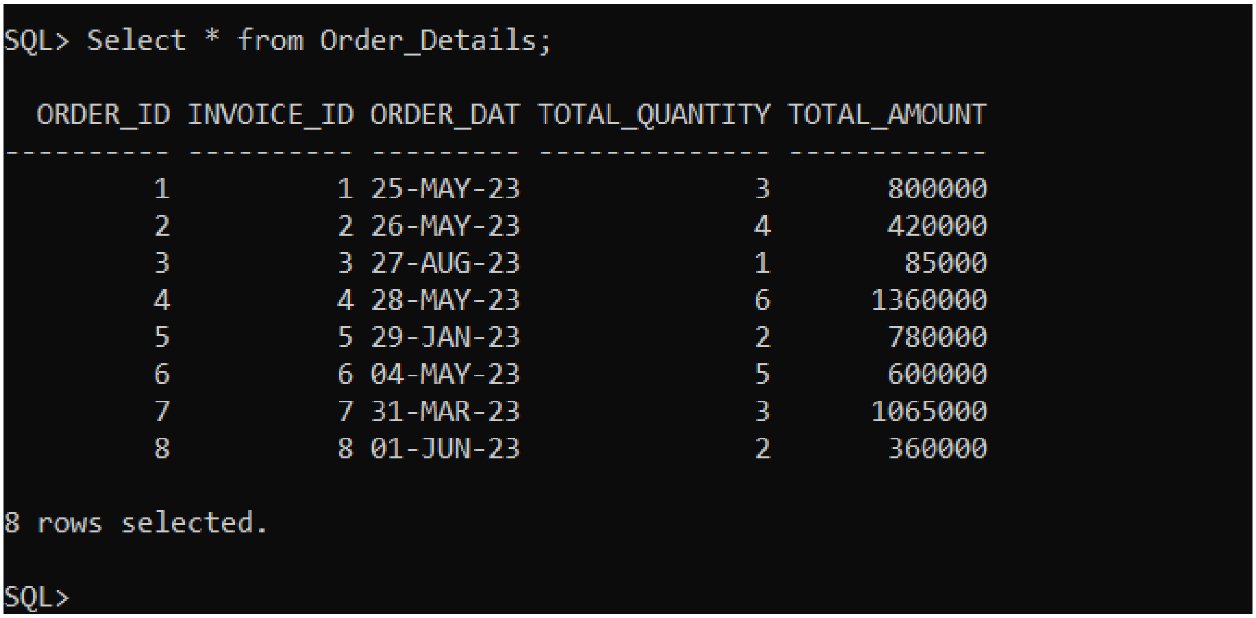
Explanation:

Inserts eight rows into the "Order_Details" table, providing values for Order_ID, Order_Date, Total_Quantity, Invoice_ID, and Total_Amount. These rows represent different orders with associated details such as order date, total quantity, invoice ID, and total amount.



```
SQL> set linesize 500;
SQL> INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (1, TO_DATE('2023-05-25', 'YYYY-MM-DD'), 3, 1, 800000);

1 row created.

SQL> INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (2, TO_DATE('2023-05-26', 'YYYY-MM-DD'), 4, 2, 420000);

1 row created.

SQL> INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (3, TO_DATE('2023-08-27', 'YYYY-MM-DD'), 1, 3, 85000);

1 row created.

SQL> INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (4, TO_DATE('2023-05-28', 'YYYY-MM-DD'), 6, 4, 1360000);

1 row created.

SQL> INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (5, TO_DATE('2023-01-29', 'YYYY-MM-DD'), 2, 5, 780000);

1 row created.

SQL> INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (6, TO_DATE('2023-05-04', 'YYYY-MM-DD'), 5, 6, 600000);

1 row created.

SQL> INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (7, TO_DATE('2023-03-31', 'YYYY-MM-DD'), 3, 7, 1065000);

1 row created.

SQL> INSERT INTO Order_Details (Order_ID, Order_Date, Total_Quantity, Invoice_ID, Total_Amount) VALUES (8, TO_DATE('2023-06-01', 'YYYY-MM-DD'), 2, 8, 360000);

1 row created.

SQL>
```

Figure 21: Insertion in Order_Details



```
SQL> Select * from Order_Details;

  ORDER_ID INVOICE_ID ORDER_DAT TOTAL_QUANTITY TOTAL_AMOUNT
---------- ---------- --------- -------------- ------------
         1          1 25-MAY-23              3       800000
         2          2 26-MAY-23              4       420000
         3          3 27-AUG-23              1        85000
         4          4 28-MAY-23              6      1360000
         5          5 29-JAN-23              2       780000
         6          6 04-MAY-23              5       600000
         7          7 31-MAR-23              3      1065000
         8          8 01-JUN-23              2       360000

8 rows selected.

SQL>
```

Figure 22: Order_Details

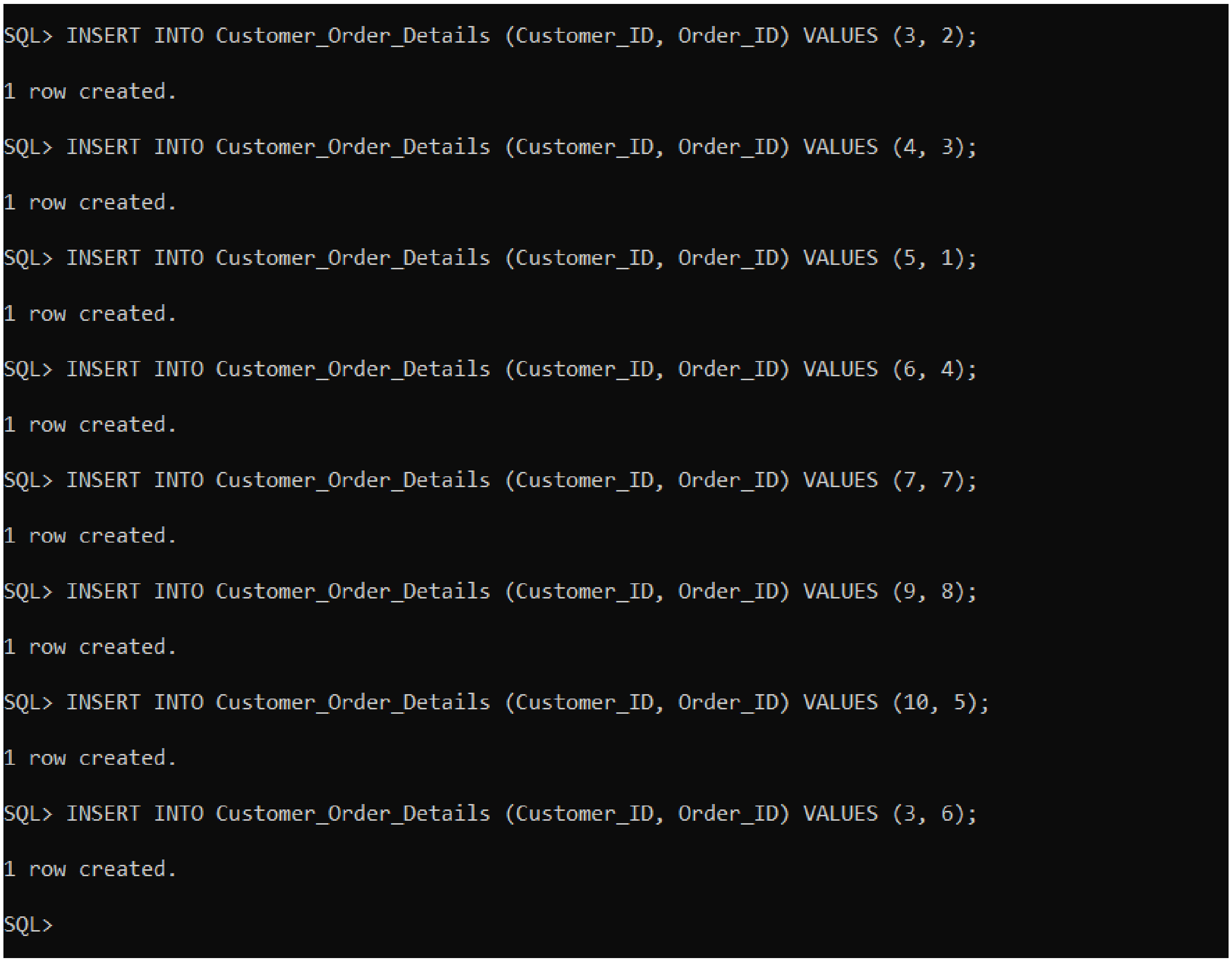
Inserting Value on Customer_Order_Details:

Query:

```
INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (3, 2);
INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (4, 3);
INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (5, 1);
INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (6, 4);
INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (7, 7);
INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (9, 8);
INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (10, 5);
INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (3, 6);
```

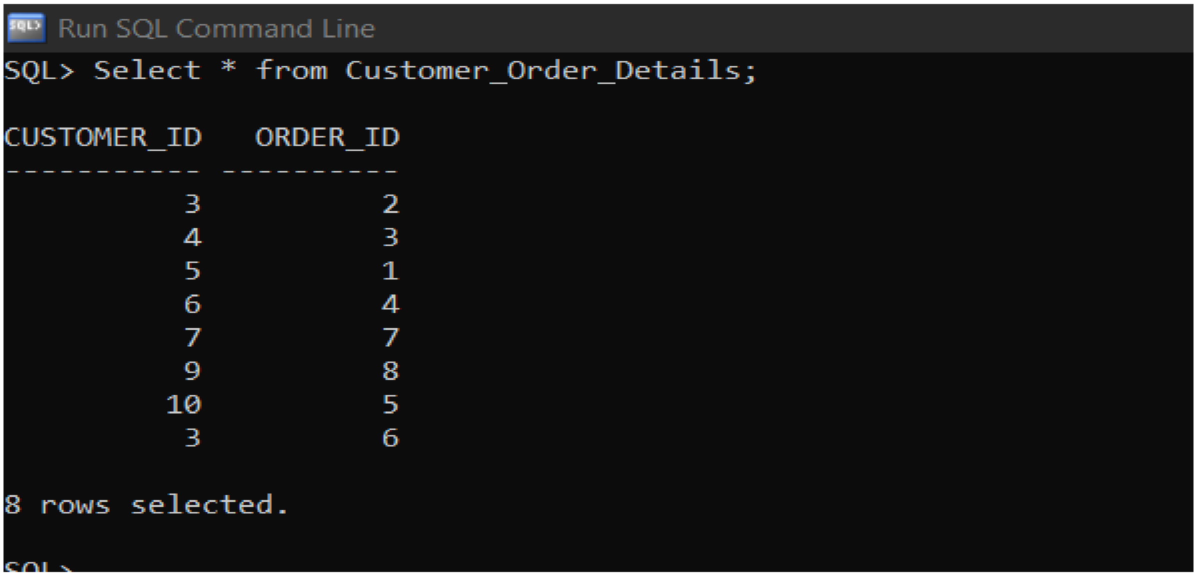
Explanation:

Inserts eight rows into the "Customer_Order_Details" table, establishing relationships between customers and orders by specifying values for Customer_ID and Order_ID.



```
SQL> INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (3, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (4, 3);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (5, 1);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (6, 4);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (7, 7);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (9, 8);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (10, 5);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Customer_ID, Order_ID) VALUES (3, 6);

1 row created.

SQL>
```

Figure 23: Insertion in Customer_Order_Details



```
SQL> Select * from Customer_Order_Details;

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

8 rows selected.

SQL>
```

Figure 24: Customer_Order_Details

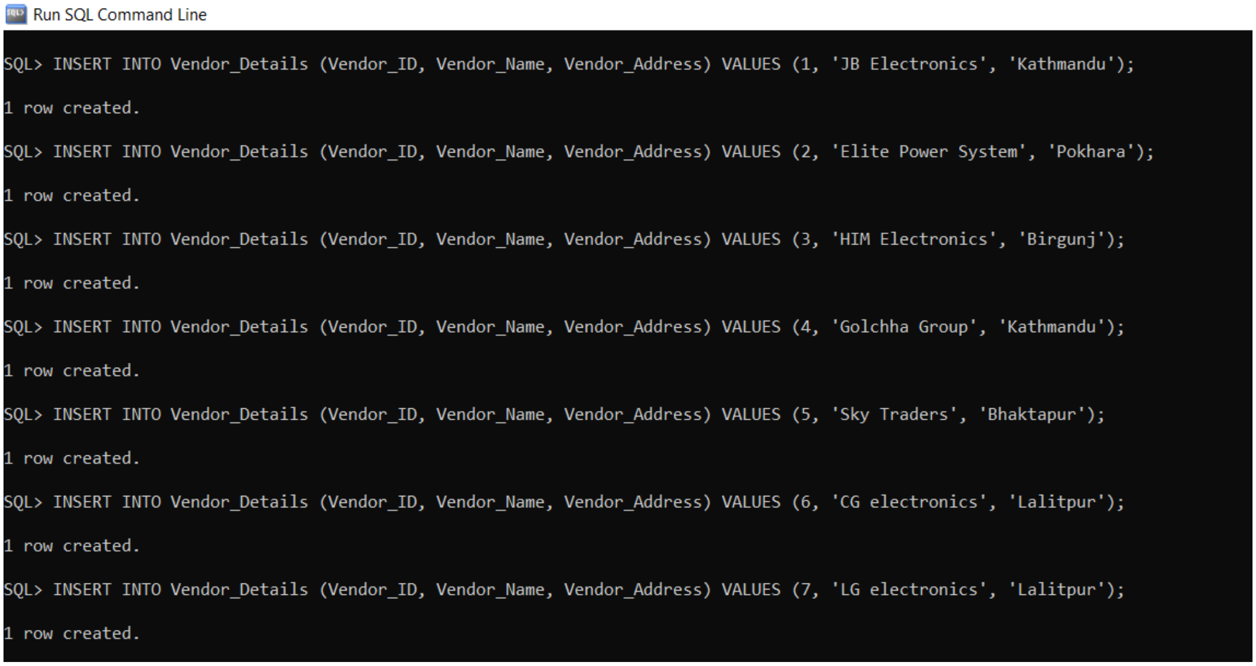
Inserting Value on Vendor_Details:

Query:

```
INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address)
VALUES (1, 'JB Electronics', 'Kathmandu');
INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address)
VALUES (2, 'Elite Power System', 'Pokhara');
INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address)
VALUES (3, 'HIM Electronics', 'Birgunj');
INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address)
VALUES (4, 'Golchha Group', 'Kathmandu');
INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address)
VALUES (5, 'Sky Traders', 'Bhaktapur');
INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address)
VALUES (6, 'CG electronics', 'Lalitpur');
INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address)
VALUES (7, 'LG electronics', 'Lalitpur');
```

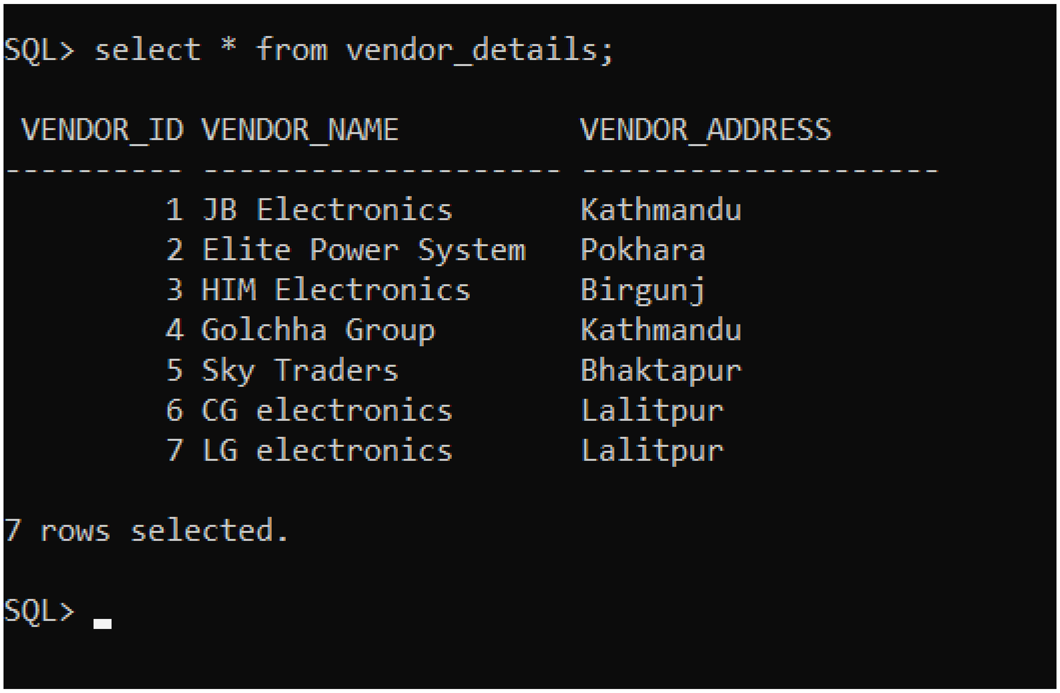
Explanation:

Inserts six rows into the "Vendor_Details" table, providing values for Vendor_ID, Vendor_Name, and Vendor_Address. These rows represent different vendors with unique information.



```
SQL> INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (1, 'JB Electronics', 'Kathmandu');

1 row created.

SQL> INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (2, 'Elite Power System', 'Pokhara');

1 row created.

SQL> INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (3, 'HIM Electronics', 'Birgunj');

1 row created.

SQL> INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (4, 'Golchha Group', 'Kathmandu');

1 row created.

SQL> INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (5, 'Sky Traders', 'Bhaktapur');

1 row created.

SQL> INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (6, 'CG electronics', 'Lalitpur');

1 row created.

SQL> INSERT INTO Vendor_Details (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (7, 'LG electronics', 'Lalitpur');

1 row created.
```

Figure 25: Insertion In Vendor_Details



```
SQL> select * from vendor_details;

 VENDOR_ID VENDOR_NAME          VENDOR_ADDRESS
---------- -------------------- --------------------
         1 JB Electronics       Kathmandu
         2 Elite Power System   Pokhara
         3 HIM Electronics      Birgunj
         4 Golchha Group        Kathmandu
         5 Sky Traders          Bhaktapur
         6 CG electronics       Lalitpur
         7 LG electronics       Lalitpur

7 rows selected.

SQL> _
```

Figure 26: Vendor_Details

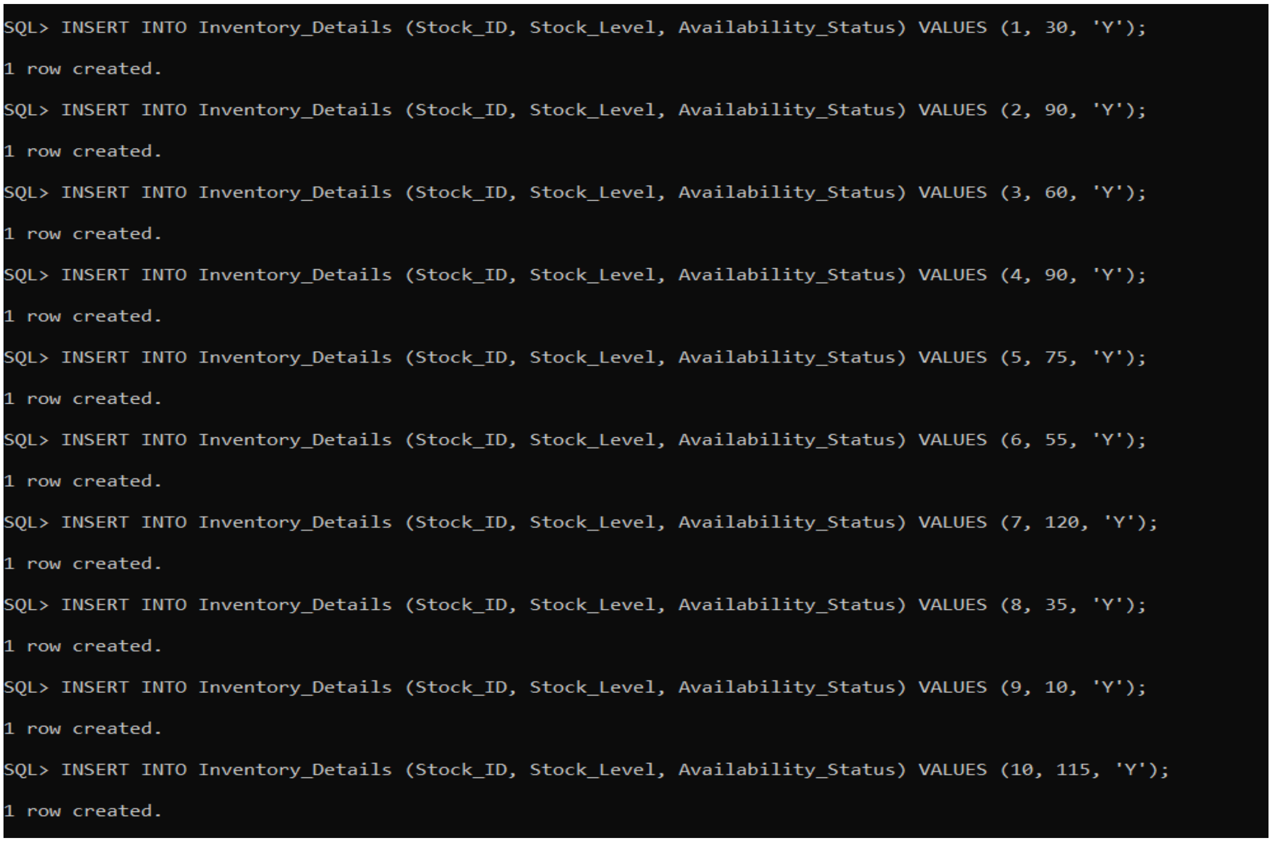
Inserting Value on Inventory_Details:

Query:

```
INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (1, 30, 'Y');
INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (2, 90, 'Y');
INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (3, 60, 'Y');
INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (4, 90, 'Y');
INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (5, 75, 'Y');
INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (6, 55, 'Y');
INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (7, 120, 'Y');
INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (8, 35, 'Y');
INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (9, 10, 'Y');
INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (10, 115, 'Y');
```

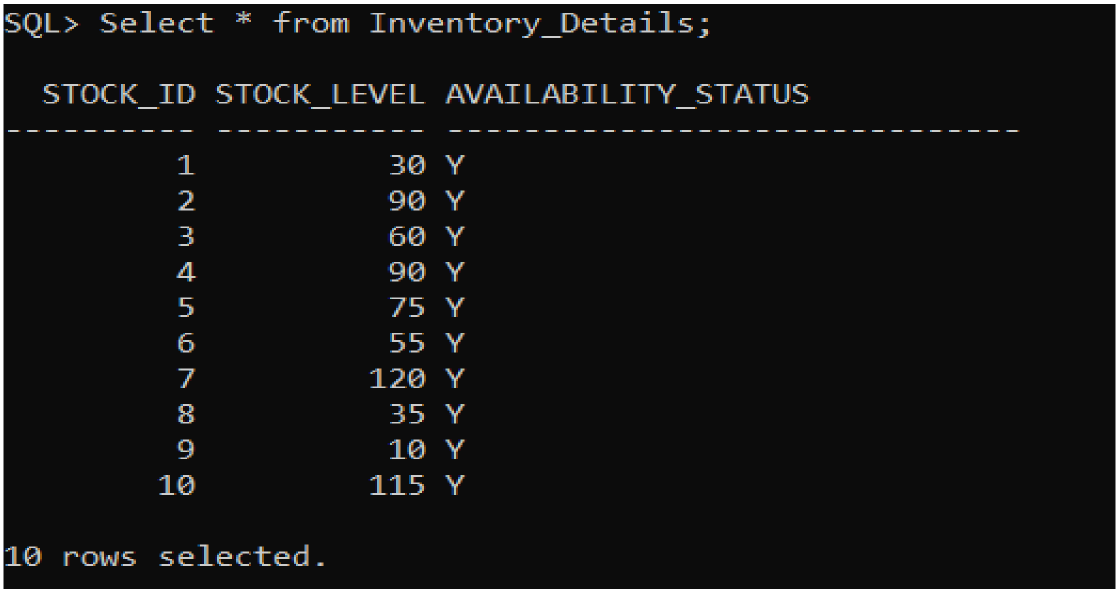
Explanation:

Inserts ten rows into the "Inventory_Details" table, providing values for Stock_ID, Stock_Level, and Availability_Status. Each row represents different inventory items with unique stock levels and availability statuses.



```
SQL> INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (1, 30, 'Y');

1 row created.

SQL> INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (2, 90, 'Y');

1 row created.

SQL> INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (3, 60, 'Y');

1 row created.

SQL> INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (4, 90, 'Y');

1 row created.

SQL> INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (5, 75, 'Y');

1 row created.

SQL> INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (6, 55, 'Y');

1 row created.

SQL> INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (7, 120, 'Y');

1 row created.

SQL> INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (8, 35, 'Y');

1 row created.

SQL> INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (9, 10, 'Y');

1 row created.

SQL> INSERT INTO Inventory_Details (Stock_ID, Stock_Level, Availability_Status) VALUES (10, 115, 'Y');

1 row created.
```

Figure 27: Insertion in Inventory_details



```
SQL> Select * from Inventory_Details;

  STOCK_ID STOCK_LEVEL AVAILABILITY_STATUS
---------- ----------- ------------------------------
         1          30 Y
         2          90 Y
         3          60 Y
         4          90 Y
         5          75 Y
         6          55 Y
         7         120 Y
         8          35 Y
         9          10 Y
        10         115 Y

10 rows selected.
```

Figure 28: Inventory_Details

Inserting Value on Product_Details:

Query:

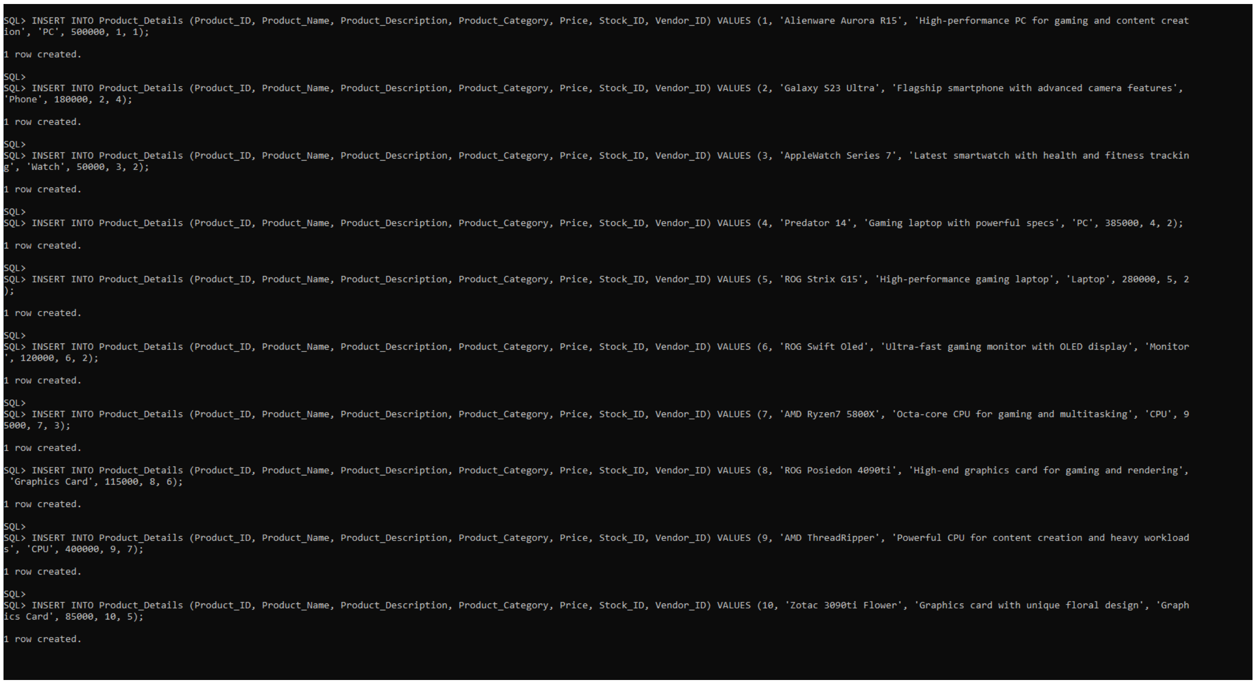
```sql
INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (1, 'Alienware Aurora R15', 'High-performance PC for gaming and content creation', 'PC', 500000, 1, 1);

INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (2, 'Galaxy S23 Ultra', 'Flagship smartphone with advanced camera features', 'Phone', 180000, 2, 4);

INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (3, 'AppleWatch Series 7', 'Latest smartwatch with health and fitness tracking', 'Watch', 50000, 3, 2);

INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (4, 'Predator 14', 'Gaming laptop with powerful specs', 'PC', 385000, 4, 2);

INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (5, 'ROG Strix G15', 'High-performance gaming laptop', 'Laptop', 280000, 5, 2);

INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (6, 'ROG Swift Oled', 'Ultra-fast gaming monitor with OLED display', 'Monitor', 120000, 6, 2);

INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (7, 'AMD Ryzen7 5800X', 'Octa-core CPU for gaming and multitasking', 'CPU', 95000, 7, 3);
INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description,
Product_Category, Price, Stock_ID, Vendor_ID) VALUES (8, 'ROG Posiedon 4090ti',
'High-end graphics card for gaming and rendering', 'Graphics Card', 115000, 8, 6);
```

```
INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description,
Product_Category, Price, Stock_ID, Vendor_ID) VALUES (9, 'AMD ThreadRipper',
'Powerful CPU for content creation and heavy workloads', 'CPU', 400000, 9, 7);
```

```
INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description,
Product_Category, Price, Stock_ID, Vendor_ID) VALUES (10, 'Zotac 3090ti Flower',
'Graphics card with unique floral design', 'Graphics Card', 85000, 10, 5);
```

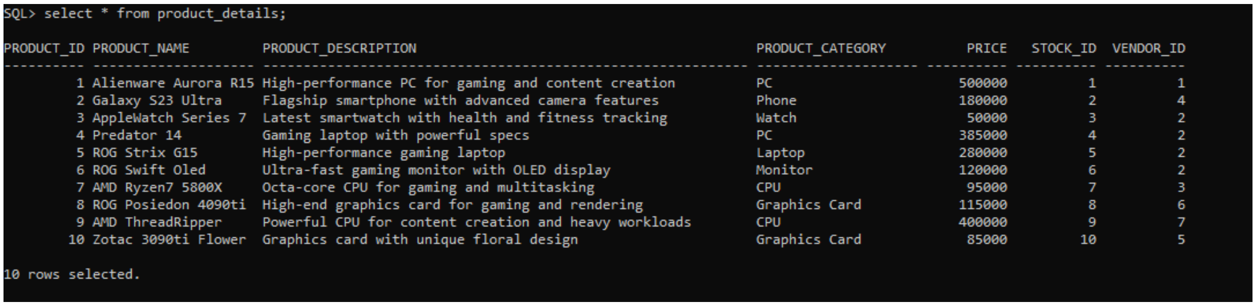
Explanation:

Inserts ten rows into the "Product_Details" table, providing values for Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, and Vendor_ID. Each row represents a different product with unique details, including various categories such as PC, Phone, Watch, Laptop, and components like CPU and Graphics Card.



```
SQL> INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (1, 'Alienware Aurora R15', 'High-performance PC for gaming and content creat
ion', 'PC', 500000, 1, 1);

1 row created.

SQL>
SQL> INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (2, 'Galaxy S23 Ultra', 'Flagship smartphone with advanced camera features',
'Phone', 180000, 2, 4);

1 row created.

SQL>
SQL> INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (3, 'AppleWatch Series 7', 'Latest smartwatch with health and fitness trackin
g', 'Watch', 50000, 3, 2);

1 row created.

SQL>
SQL> INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (4, 'Predator 14', 'Gaming laptop with powerful specs', 'PC', 385000, 4, 2);

1 row created.

SQL>
SQL> INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (5, 'ROG Strix G15', 'High-performance gaming laptop', 'Laptop', 280000, 5, 2
);

1 row created.

SQL>
SQL> INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (6, 'ROG Swift Oled', 'Ultra-fast gaming monitor with OLED display', 'Monitor
', 120000, 6, 2);

1 row created.

SQL>
SQL> INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (7, 'AMD Ryzen7 5800X', 'Octa-core CPU for gaming and multitasking', 'CPU', 9
5000, 7, 3);

1 row created.

SQL> INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (8, 'ROG Posiedon 4090ti', 'High-end graphics card for gaming and rendering',
 'Graphics Card', 115000, 8, 6);

1 row created.

SQL>
SQL> INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (9, 'AMD ThreadRipper', 'Powerful CPU for content creation and heavy workload
s', 'CPU', 400000, 9, 7);

1 row created.

SQL>
SQL> INSERT INTO Product_Details (Product_ID, Product_Name, Product_Description, Product_Category, Price, Stock_ID, Vendor_ID) VALUES (10, 'Zotac 3090ti Flower', 'Graphics card with unique floral design', 'Graph
ics Card', 85000, 10, 5);

1 row created.
```

Figure 29: Insertion in Product_Details(i)



```
SQL> select * from product_details;

PRODUCT_ID PRODUCT_NAME         PRODUCT_DESCRIPTION                                          PRODUCT_CATEGORY          PRICE   STOCK_ID  VENDOR_ID
---------- -------------------- ------------------------------------------------------------ -------------------- ---------- ---------- ----------
         1 Alienware Aurora R15 High-performance PC for gaming and content creation          PC                       500000          1          1
         2 Galaxy S23 Ultra     Flagship smartphone with advanced camera features            Phone                    180000          2          4
         3 AppleWatch Series 7  Latest smartwatch with health and fitness tracking           Watch                     50000          3          2
         4 Predator 14          Gaming laptop with powerful specs                            PC                       385000          4          2
         5 ROG Strix G15        High-performance gaming laptop                               Laptop                   280000          5          2
         6 ROG Swift Oled       Ultra-fast gaming monitor with OLED display                  Monitor                  120000          6          2
         7 AMD Ryzen7 5800X     Octa-core CPU for gaming and multitasking                    CPU                       95000          7          3
         8 ROG Posiedon 4090ti  High-end graphics card for gaming and rendering              Graphics Card            115000          8          6
         9 AMD ThreadRipper     Powerful CPU for content creation and heavy workloads        CPU                      400000          9          7
        10 Zotac 3090ti Flower  Graphics card with unique floral design                      Graphics Card             85000         10          5

10 rows selected.
```

Figure 30: Product_Details

Inserting Value on Order_Product_Details:

Query:

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (1, 1, 1);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (1, 2, 1);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (1, 6, 1);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (2, 7, 2);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (2, 8, 2);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (3, 10, 1);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (4, 3, 2);

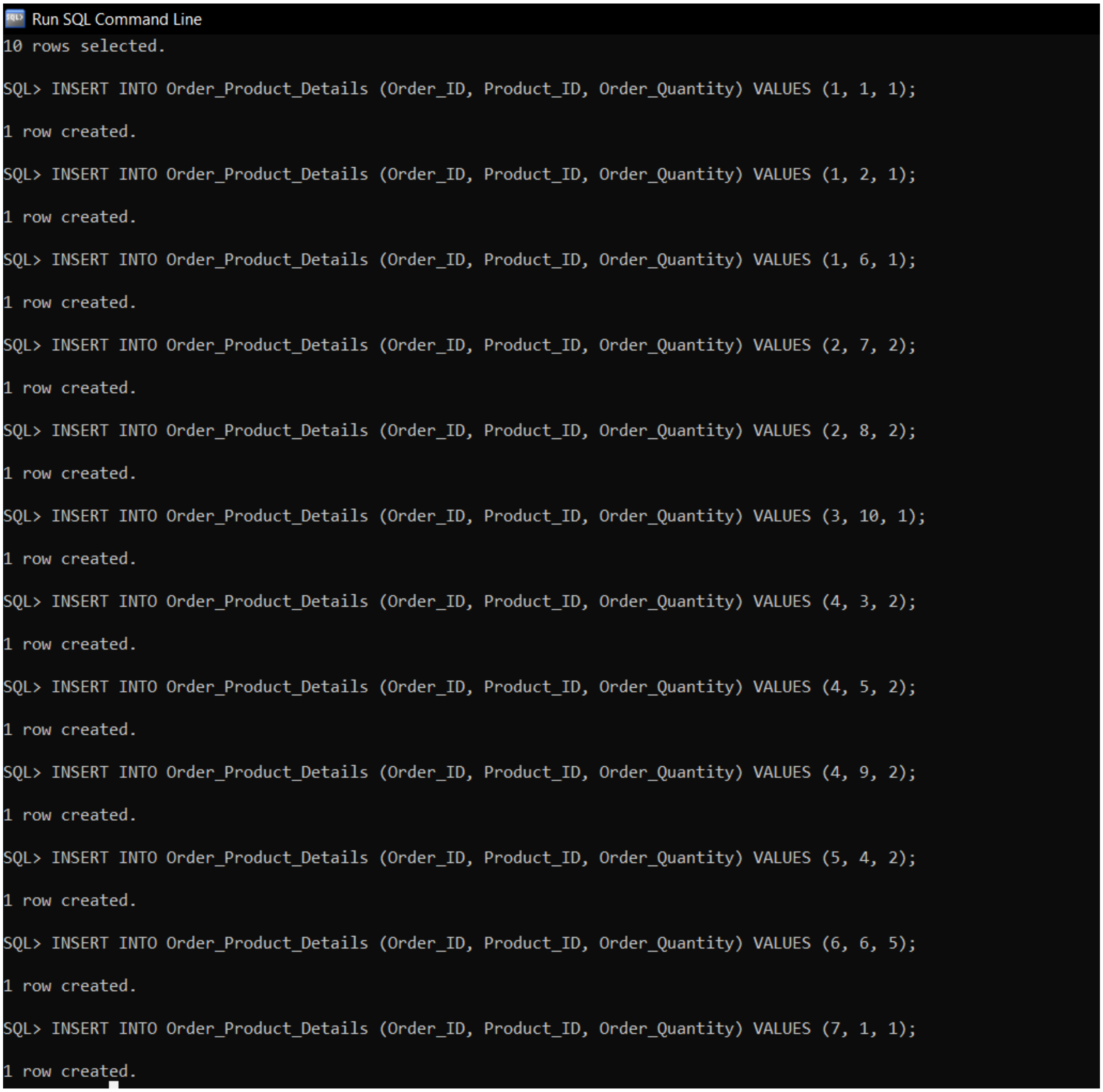
INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (4, 5, 2);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (4, 9, 2);

```
INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity)
VALUES (5, 4, 2);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity)
VALUES (6, 6, 5);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity)
VALUES (7, 1, 1);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity)
VALUES (7, 2, 1);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity)
VALUES (7, 4, 1);

INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity)
VALUES (8, 2, 2);
```

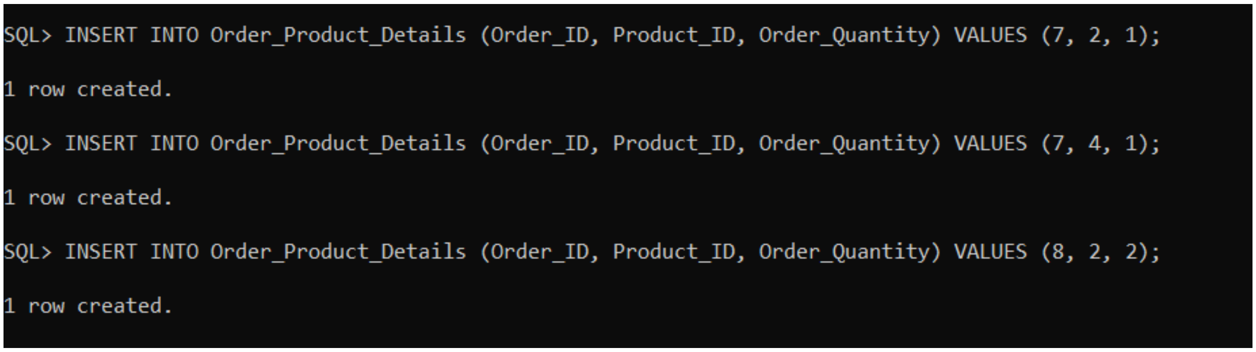
Explanation:

Inserts fifteen rows into the "Order_Product_Details" table, establishing relationships between orders and products by specifying values for Order_ID, Product_ID, and Order_Quantity. Each row represents a product associated with a particular order and quantity.



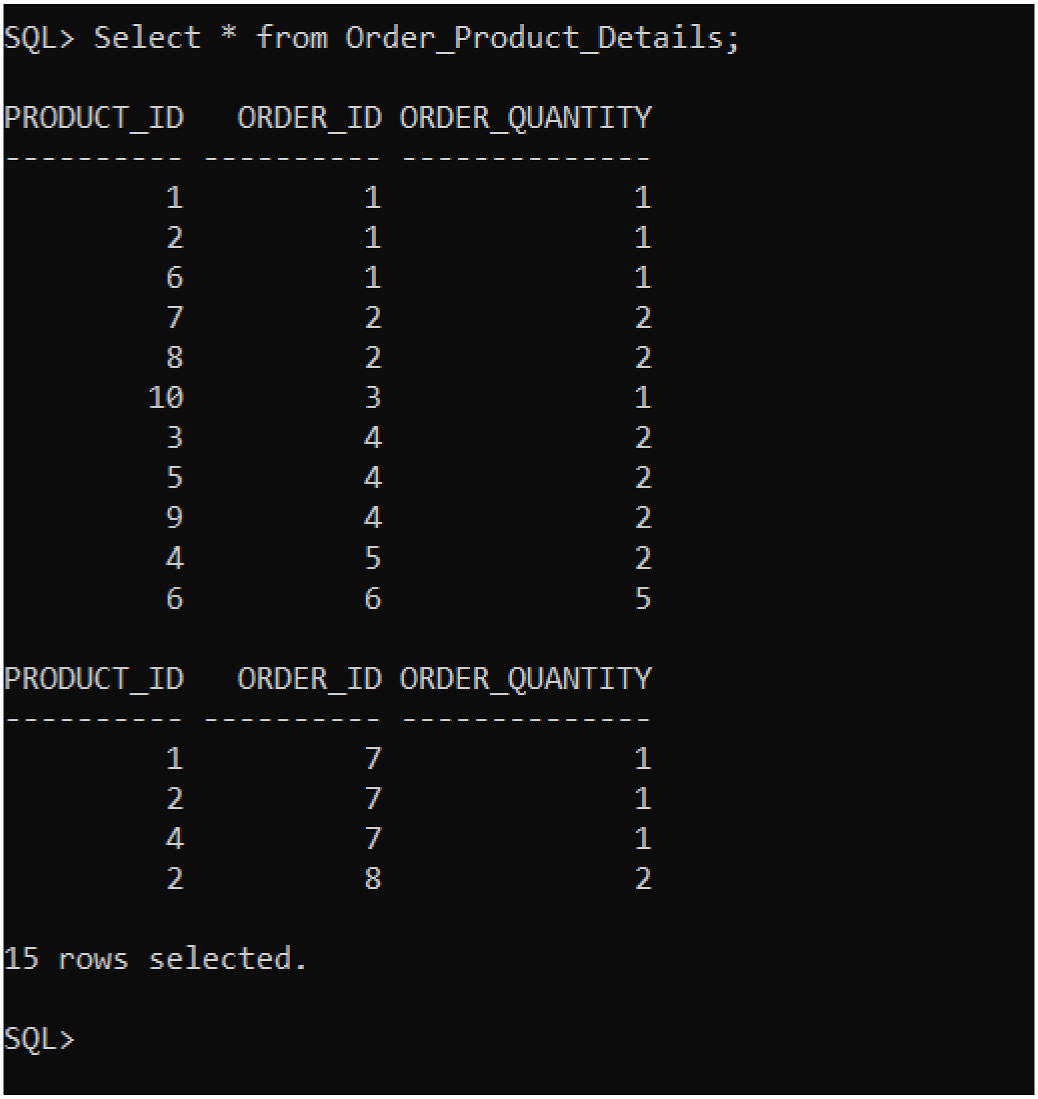
```
10 rows selected.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (1, 1, 1);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (1, 2, 1);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (1, 6, 1);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (2, 7, 2);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (2, 8, 2);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (3, 10, 1);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (4, 3, 2);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (4, 5, 2);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (4, 9, 2);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (5, 4, 2);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (6, 6, 5);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (7, 1, 1);

1 row created.
```

Figure 31: Insertion In Order_Product_Details(i)



```
SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (7, 2, 1);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (7, 4, 1);

1 row created.

SQL> INSERT INTO Order_Product_Details (Order_ID, Product_ID, Order_Quantity) VALUES (8, 2, 2);

1 row created.
```

Figure 32: Insertion in Order_Product_Details(ii)



```
SQL> Select * from Order_Product_Details;

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

PRODUCT_ID   ORDER_ID ORDER_QUANTITY
---------- ---------- --------------
         1          7              1
         2          7              1
         4          7              1
         2          8              2

15 rows selected.

SQL>
```

Figure 33: Order_Product_Details

7. Database Querying

Within the Oracle database, a query, commonly known as SQL or Structured Query Language, functions as a versatile tool for data interaction. The SELECT query retrieves information from tables, offering the flexibility to specify columns, tables, and conditions for data filtering. INSERT allows for the addition of one or multiple records, UPDATE alters existing records, and DELETE eliminates records based on specified criteria. TRUNCATE, categorized as a Data Definition Language (DDL) command, efficiently erases all records from a table; however, distinct from DELETE, it lacks the ability to be rolled back once executed. Collectively, these queries empower users to adeptly manage and manipulate data in the organized landscape of a database. (Pedamkar, 2023)

7.1. Information Query

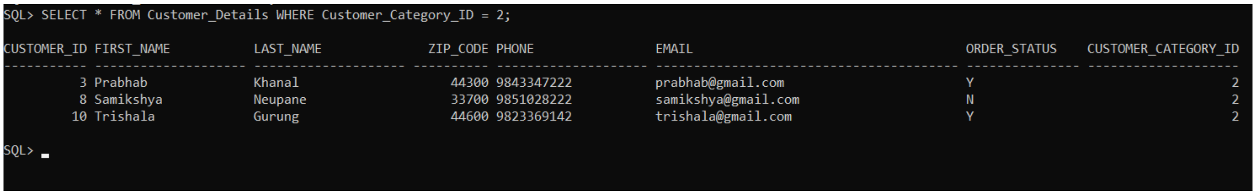
- List all the customers that are also staff of the company.

  Query:

  SELECT * FROM Customer_Details WHERE Customer_Category_ID = 2;

  Explanation:

  This SQL query retrieves all customer details from the `Customer_Details` table where the `Customer_Category_ID` is equal to 2.



```
SQL> SELECT * FROM Customer_Details WHERE Customer_Category_ID = 2;

CUSTOMER_ID FIRST_NAME           LAST_NAME              ZIP_CODE PHONE                EMAIL                                    ORDER_STATUS    CUSTOMER_CATEGORY_ID
----------- -------------------- -------------------- ---------- -------------------- ---------------------------------------- --------------- --------------------
          3 Prabhab              Khanal                    44300 9843347222           prabhab@gmail.com                        Y                                  2
          8 Samikshya            Neupane                   33700 9851028222           samikshya@gmail.com                      N                                  2
         10 Trishala             Gurung                    44600 9823369142           trishala@gmail.com                       Y                                  2

SQL>
```

Figure 34: Customer that are staff

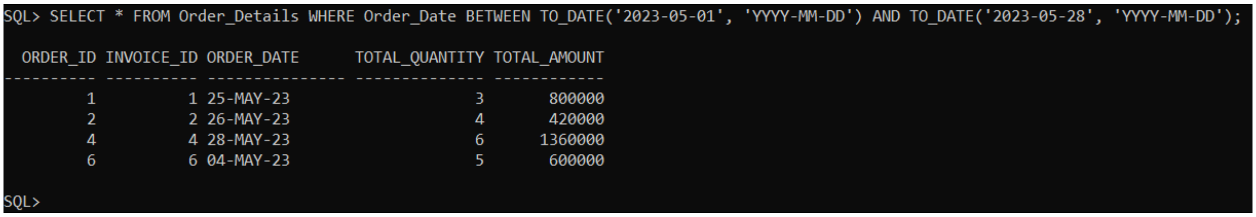
- List all the orders made for any particular product between the dates 01-05-2023 till 28-05-2023.

  Query:

  SELECT * FROM Order_Details WHERE Order_Date BETWEEN TO_DATE('2023-05-01','YYYY-MM-DD') AND TO_DATE('2023-05-28','YYYY-MM-DD');

  Explanation:

  This SQL query retrieves orders made between May 1, 2023, and May 28, 2023, from the `Order_Details` table using the `BETWEEN` clause with specified date ranges.



```
SQL> SELECT * FROM Order_Details WHERE Order_Date BETWEEN TO_DATE('2023-05-01', 'YYYY-MM-DD') AND TO_DATE('2023-05-28', 'YYYY-MM-DD');

  ORDER_ID INVOICE_ID ORDER_DATE      TOTAL_QUANTITY TOTAL_AMOUNT
---------- ---------- --------------- -------------- ------------
         1          1 25-MAY-23                    3       800000
         2          2 26-MAY-23                    4       420000
         4          4 28-MAY-23                    6      1360000
         6          6 04-MAY-23                    5       600000

SQL>
```

Figure 35: Order between the dates 01-05-2023 till 28-05-2023.

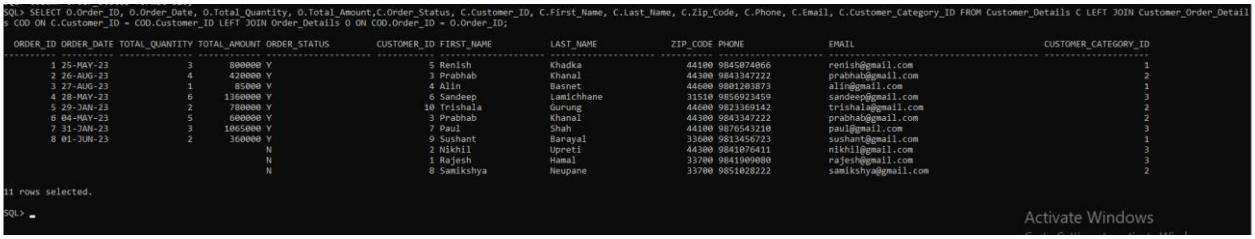
- List all the customers with their order details and also the customers who have not ordered any products yet.

  Query:

  SELECT O.Order_ID, O.Order_Date, O.Total_Quantity, O.Total_Amount, C.Order_Status, C.Customer_ID, C.First_Name, C.Last_Name, C.Zip_Code, C.Phone, C.Email, C.Customer_Category_ID FROM Customer_Details C LEFT JOIN Customer_Order_Details COD ON C.Customer_ID = COD.Customer_ID LEFT JOIN Order_Details O ON COD.Order_ID = O.Order_ID;

  Explanation:

  This SQL query retrieves order details along with customer information, including order status, customer ID, name, contact details, and category. It utilizes a left join between the `Customer_Details`, `Customer_Order_Details`, and `Order_Details` tables to associate customers with their orders, ensuring all customers are included in the result set.



```
SQL> SELECT O.Order_ID, O.Order_Date, O.Total_Quantity, O.Total_Amount,C.Order_Status, C.Customer_ID, C.First_Name, C.Last_Name, C.Zip_Code, C.Phone, C.Email, C.Customer_Category_ID FROM Customer_Details C LEFT JOIN Customer_Order_Detail
s COD ON C.Customer_ID = COD.Customer_ID LEFT JOIN Order_Details O ON COD.Order_ID = O.Order_ID;

  ORDER_ID ORDER_DATE TOTAL_QUANTITY TOTAL_AMOUNT ORDER_STATUS         CUSTOMER_ID FIRST_NAME           LAST_NAME              ZIP_CODE PHONE                EMAIL                                    CUSTOMER_CATEGORY_ID
---------- ---------- -------------- ------------ -------------------- ----------- -------------------- -------------------- ---------- -------------------- ---------------------------------------- --------------------
         1 25-MAY-23               3       800000 Y                              5 Renish               Khadka                    44100 9845074066           renish@gmail.com                                            1
         2 26-AUG-23               4       420000 Y                              3 Prabhab              Khanal                    44300 9843347222           prabhab@gmail.com                                           2
         3 27-AUG-23               1        85000 Y                              4 Alin                 Basnet                    44600 9801203873           alin@gmail.com                                              1
         4 28-MAY-23               6      1360000 Y                              6 Sandeep              Lamichhane                31510 9856923459           sandeep@gmail.com                                           3
         5 29-JAN-23               2       780000 Y                             10 Trishala             Gurung                    44600 9823369142           trishala@gmail.com                                          2
         6 04-MAY-23               5       600000 Y                              3 Prabhab              Khanal                    44300 9843347222           prabhab@gmail.com                                           2
         7 31-JAN-23               3      1065000 Y                              7 Paul                 Shah                      44100 9876543210           paul@gmail.com                                              3
         8 01-JUN-23               2       360000 Y                              9 Sushant              Barayal                   33600 9813456723           sushant@gmail.com                                           1
                                                  N                              2 Nikhil               Upreti                    44300 9841076411           nikhil@gmail.com                                            3
                                                  N                              1 Rajesh               Hamal                     33700 9841909080           rajesh@gmail.com                                            3
                                                  N                              8 Samikshya            Neupane                   33700 9851028222           samikshya@gmail.com                                         2

11 rows selected.

SQL> _
```

Figure 36: Listing all customer with order_details

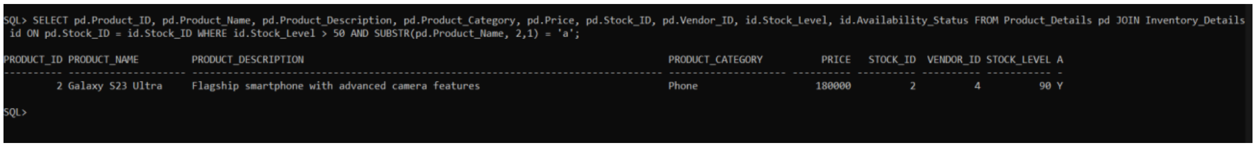
- List all product details that have the second letter ‘a’ in their product name and have a stock quantity more than 50.

  Query:

  SELECT pd.Product_ID, pd.Product_Name, pd.Product_Description, pd.Product_Category, pd.Price, pd.Stock_ID, pd.Vendor_ID, id.Stock_Level, id.Availability_Status FROM Product_Details pd JOIN Inventory_Details id ON pd.Stock_ID = id.Stock_ID WHERE id.Stock_Level > 50 AND SUBSTR(pd.Product_Name,2,1) = 'a';

  Explanation:

  This SQL query retrieves product details and inventory information where the stock level is greater than 50, and the second letter of the product name is 'a'. It involves joining the `Product_Details` and `Inventory_Details` tables and applies conditions for stock level and product name.



```
SQL> SELECT pd.Product_ID, pd.Product_Name, pd.Product_Description, pd.Product_Category, pd.Price, pd.Stock_ID, pd.Vendor_ID, id.Stock_Level, id.Availability_Status FROM Product_Details pd JOIN Inventory_Details
 id ON pd.Stock_ID = id.Stock_ID WHERE id.Stock_Level > 50 AND SUBSTR(pd.Product_Name, 2,1) = 'a';

PRODUCT_ID PRODUCT_NAME         PRODUCT_DESCRIPTION                                                              PRODUCT_CATEGORY          PRICE   STOCK_ID  VENDOR_ID STOCK_LEVEL A
---------- -------------------- -------------------------------------------------------------------------------- -------------------- ---------- ---------- ---------- ----------- -
         2 Galaxy S23 Ultra     Flagship smartphone with advanced camera features                                Phone                    180000          2          4          90 Y

SQL>
```

Figure 37: Second letter ‘a’ in their product name and have a stock > 50

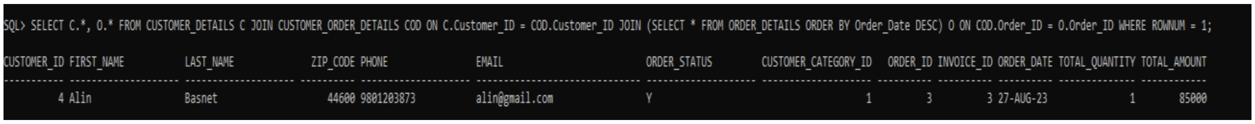
- Find out the customer who has ordered recently.

  Query:

  SELECT C.*, O.* FROM CUSTOMER_DETAILS C JOIN CUSTOMER_ORDER_DETAILS COD ON C.Customer_ID = COD.Customer_ID JOIN (SELECT * FROM ORDER_DETAILS ORDER BY Order_Date DESC) O ON COD.Order_ID = O.Order_ID WHERE ROWNUM = 1;

  Explanation:

  This SQL query retrieves information from the `CUSTOMER_DETAILS`, `CUSTOMER_ORDER_DETAILS`, and `ORDER_DETAILS` tables. It joins these tables based on customer and order relationships, and the subquery selects orders ordered by date in descending order. The main query limits the result to only one row using `ROWNUM = 1`, providing details about the most recent customer order.



```
SQL> SELECT C.*, O.* FROM CUSTOMER_DETAILS C JOIN CUSTOMER_ORDER_DETAILS COD ON C.Customer_ID = COD.Customer_ID JOIN (SELECT * FROM ORDER_DETAILS ORDER BY Order_Date DESC) O ON COD.Order_ID = O.Order_ID WHERE ROWNUM = 1;

CUSTOMER_ID FIRST_NAME          LAST_NAME             ZIP_CODE PHONE                EMAIL                          ORDER_STATUS         CUSTOMER_CATEGORY_ID   ORDER_ID INVOICE_ID ORDER_DATE TOTAL_QUANTITY TOTAL_AMOUNT
----------- ------------------- ------------------- ---------- -------------------- ------------------------------ -------------------- -------------------- ---------- ---------- --------- -------------- ------------
          4 Alin                Basnet                   44600 9801203873           alin@gmail.com                 Y                                       1          3          3 27-AUG-23              1        85000
```

Figure 38: Recent Order

7.2. Transaction Query

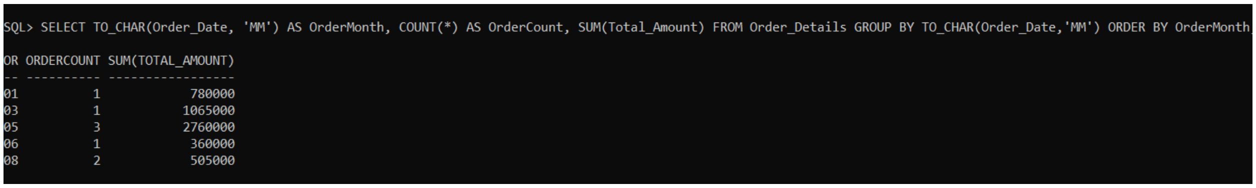
- Show the total revenue of the company for each month.

  Query:

  SELECT TO_CHAR(Order_Date, 'MM') AS OrderMonth, COUNT(*) AS OrderCount, SUM(Total_Amount) FROM Order_Details GROUP BY TO_CHAR(Order_Date,'MM') ORDER BY OrderMonth;

  Explanation:

  The SQL query extracts monthly order counts and total revenue from the `Order_Details` table, presenting the results grouped by month and ordered chronologically. It offers a concise summary of the company's monthly performance.



```
SQL> SELECT TO_CHAR(Order_Date, 'MM') AS OrderMonth, COUNT(*) AS OrderCount, SUM(Total_Amount) FROM Order_Details GROUP BY TO_CHAR(Order_Date,'MM') ORDER BY OrderMonth

OR ORDERCOUNT SUM(TOTAL_AMOUNT)
-- ---------- -----------------
01          1            780000
03          1           1065000
05          3           2760000
06          1            360000
08          2            505000
```

Figure 39: Total Amount on Each Month

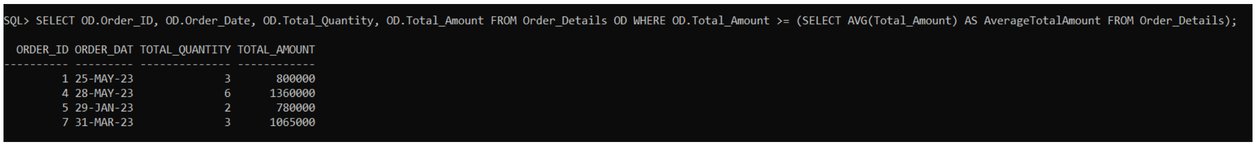
- Find those orders that are equal or higher than the average order total value.

  Query:

  SELECT OD.Order_ID, OD.Order_Date, OD.Total_Quantity, OD.Total_Amount FROM Order_Details OD WHERE OD.Total_Amount >= (SELECT AVG(Total_Amount) AS AverageTotalAmount FROM Order_Details);

  Explanation:

  This query retrieves orders with a total amount greater than or equal to the average total amount across all orders. It accomplishes this by selecting orders from the Order_Details table where the Total_Amount is greater than or equal to the average total amount, calculated using a subquery with the AVG function. The subquery computes the average total amount and is used as a comparison criterion in the main query.



```
SQL> SELECT OD.Order_ID, OD.Order_Date, OD.Total_Quantity, OD.Total_Amount FROM Order_Details OD WHERE OD.Total_Amount >= (SELECT AVG(Total_Amount) AS AverageTotalAmount FROM Order_Details);

  ORDER_ID ORDER_DAT TOTAL_QUANTITY TOTAL_AMOUNT
---------- --------- -------------- ------------
         1 25-MAY-23              3       800000
         4 28-MAY-23              6      1360000
         5 29-JAN-23              2       780000
         7 31-MAR-23              3      1065000
```

Figure 40: Orders that are equal or higher than the average

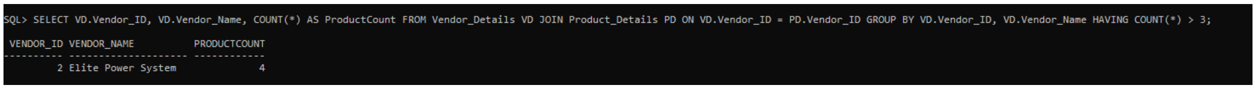
- List the details of vendors who have supplied more than 3 products to the company.

  Query:

  SELECT VD.Vendor_ID, VD.Vendor_Name, COUNT(*) AS ProductCount FROM Vendor_Details VD JOIN Product_Details PD ON VD.Vendor_ID = PD.Vendor_ID GROUP BY VD.Vendor_ID, VD.Vendor_Name HAVING COUNT(*) > 3;

  Explanation:

  This query counts the number of products associated with each vendor and filters for vendors with more than 3 products. It achieves this by joining the Vendor_Details and Product_Details tables on the Vendor_ID column. The results are then grouped by Vendor_ID and Vendor_Name, and the COUNT(*) function is applied to determine the product count for each vendor. The HAVING COUNT(*) > 3 condition is used to filter out vendors with less than 4 products.



```
SQL> SELECT VD.Vendor_ID, VD.Vendor_Name, COUNT(*) AS ProductCount FROM Vendor_Details VD JOIN Product_Details PD ON VD.Vendor_ID = PD.Vendor_ID GROUP BY VD.Vendor_ID, VD.Vendor_Name HAVING COUNT(*) > 3;

 VENDOR_ID VENDOR_NAME          PRODUCTCOUNT
---------- -------------------- ------------
         2 Elite Power System              4
```

Figure 41: Vendors who have supplied more than 3 products

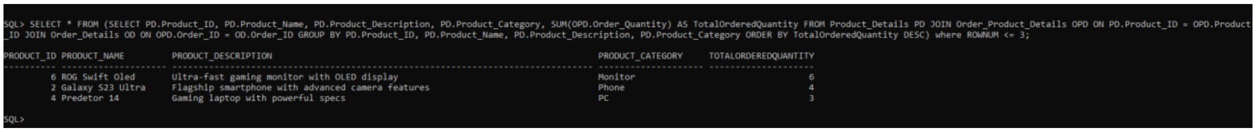
- Show the top 3 product details that have been ordered the most.

  Query:

  SELECT * FROM(SELECT PD.Product_ID, PD.Product_Name, PD.Product_Description, PD.Product_Category, SUM(OPD.Order_Quantity) AS TotalOrderedQuantity FROM Product_Details PD JOIN Order_Product_Details OPD ON PD.Product_ID = OPD.Product_ID JOIN Order_Details OD ON OPD.Order_ID = OD.Order_ID GROUP BY PD.Product_ID, PD.Product_Name, PD.Product_Description, PD.Product_Category ORDER BY TotalOrderedQuantity DESC) where ROWNUM <= 3;

  Explanation:

  This query retrieves the top 3 products based on the highest total ordered quantity. It accomplishes this by joining the `Product_Details`, `Order_Product_Details`, and `Order_Details` tables, linking them through their respective IDs. The `SUM(OPD.Order_Quantity)` function calculates the total ordered quantity for each product. The results are then grouped by product ID, name, description, and category. The final output is sorted in descending order by the total ordered quantity, and the `ROWNUM` restriction is applied to limit the results to the top 3 products.



```
SQL> SELECT * FROM (SELECT PD.Product_ID, PD.Product_Name, PD.Product_Description, PD.Product_Category, SUM(OPD.Order_Quantity) AS TotalOrderedQuantity FROM Product_Details PD JOIN Order_Product_Details OPD ON PD.Product_ID = OPD.Product
_ID JOIN Order_Details OD ON OPD.Order_ID = OD.Order_ID GROUP BY PD.Product_ID, PD.Product_Name, PD.Product_Description, PD.Product_Category ORDER BY TotalOrderedQuantity DESC) where ROWNUM <= 3;

PRODUCT_ID PRODUCT_NAME       PRODUCT_DESCRIPTION                                                              PRODUCT_CATEGORY     TOTALORDEREDQUANTITY
---------- ------------------ -------------------------------------------------------------------------------- -------------------- --------------------
         6 ROG Swift Oled     Ultra-fast gaming monitor with OLED display                                      Monitor                                 6
         2 Galaxy S23 Ultra   Flagship smartphone with advanced camera features                                Phone                                   4
         4 Predetor 14        Gaming laptop with powerful specs                                                PC                                      3

SQL>
```

Figure 42: Top 3 product details that have been ordered the most

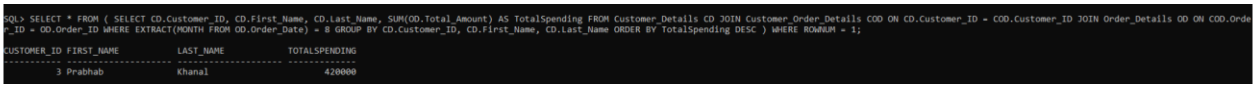
- Find out the customer who has ordered the most in August with his/her total spending on that month.

  Query:

  SELECT * FROM (SELECT CD.Customer_ID, CD.First_Name, CD.Last_Name, SUM(OD.Total_Amount) AS TotalSpending FROM Customer_Details CD JOIN Customer_Order_Details COD ON CD.Customer_ID = COD.Customer_ID JOIN Order_Details OD ON COD.Order_ID = OD.Order_ID WHERE EXTRACT(MONTH FROM OD.Order_Date) = 8 GROUP BY CD.Customer_ID, CD.First_Name, CD.Last_Name ORDER BY TotalSpending DESC) WHERE ROWNUM = 1;

  Explanation:

  This query identifies the customer with the highest spending in August by joining tables and summing total amounts. It uses `ROWNUM = 1` to fetch the top spender and returns their ID, first name, last name, and total spending.



```
SQL> SELECT * FROM ( SELECT CD.Customer_ID, CD.First_Name, CD.Last_Name, SUM(OD.Total_Amount) AS TotalSpending FROM Customer_Details CD JOIN Customer_Order_Details COD ON CD.Customer_ID = COD.Customer_ID JOIN Order_Details OD ON COD.Orde
r_ID = OD.Order_ID WHERE EXTRACT(MONTH FROM OD.Order_Date) = 8 GROUP BY CD.Customer_ID, CD.First_Name, CD.Last_Name ORDER BY TotalSpending DESC ) WHERE ROWNUM = 1;

CUSTOMER_ID FIRST_NAME           LAST_NAME            TOTALSPENDING
----------- -------------------- -------------------- -------------
          3 Prabhab              Khanal                      420000
```

Figure 43: Highest Spender In August

8. Critical Evaluation

The database class (CC5051NI- Databases) is like a super handy tool we learn in Level 5. The teachers are good at making it easy to understand. Instead of just learning facts, they show us how to use databases in the real world—creating, designing, and applying them. It's kind of like being taught how to organize your closet so you can find things easily. The cool part is this knowledge isn't just for the database class; it helps in other subjects too. We figure out how databases work in software engineering and different areas, making it feel like we're not just learning about databases but picking up a skill that works in all sorts of situations. It's like learning the secret language of organization that helps us make sense of stuff in different parts of our studies.

Doing this coursework was like going on a fun rollercoaster ride, full of challenges and discoveries. We were creating a database for Gadget Emporium, and it was like putting together a big puzzle. Figuring out how to organize the data, drawing diagrams, and playing with the information in the database was sometimes tough but surprisingly enjoyable. It was like learning the behind-the-scenes magic of how a computerized database system is made. We got to do hands-on stuff like drawing diagrams, creating tables, and using queries to make the database work. This coursework isn't just a one-time thing; it's like a helpful guidebook that gives you skills and ideas to deal with challenges in the future. It's a cool reference that sticks with you and helps you understand more about how things work.

In essence, this coursework proved instrumental as it served as a pivotal lesson in comprehending the workings of databases and effective management strategies. The acquired knowledge and skills can be likened to specialized tools, empowering individuals to adeptly navigate through any forthcoming challenges or tasks related to databases. Beyond mere academic achievement, the emphasis is on cultivating practical skills that hold considerable significance in real-world scenarios, particularly in the realm of database management.

9. File Creation

9.1. Creating Dump File

- Step 1: Open the file where you need the dump file and type cmd.

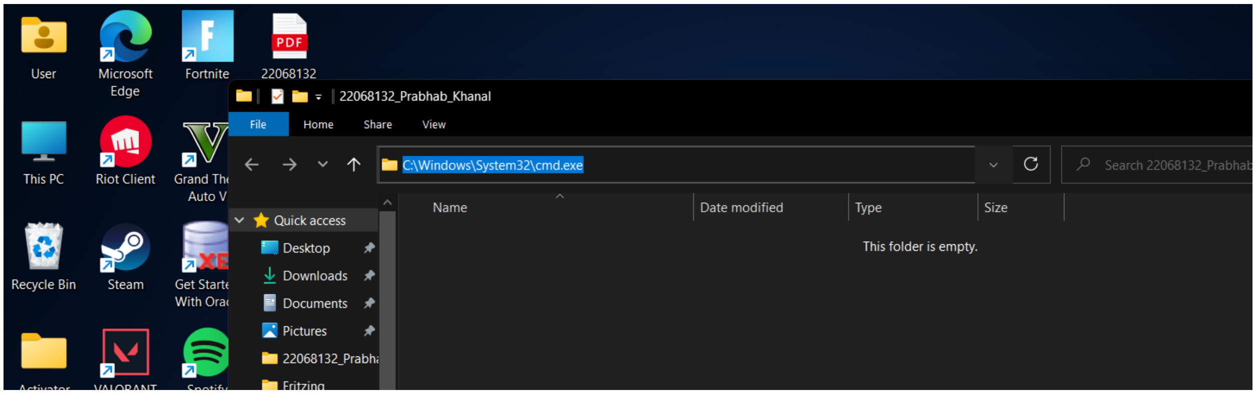
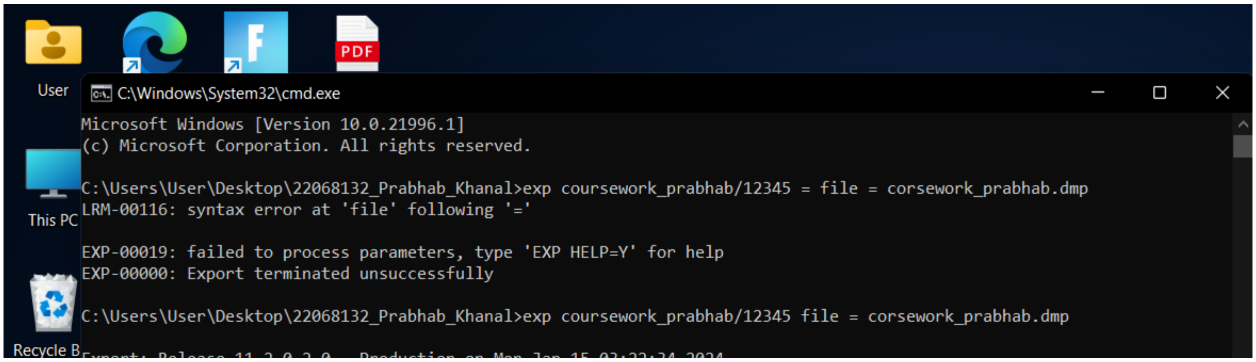


Figure 44:Opening Command prompt from the folder

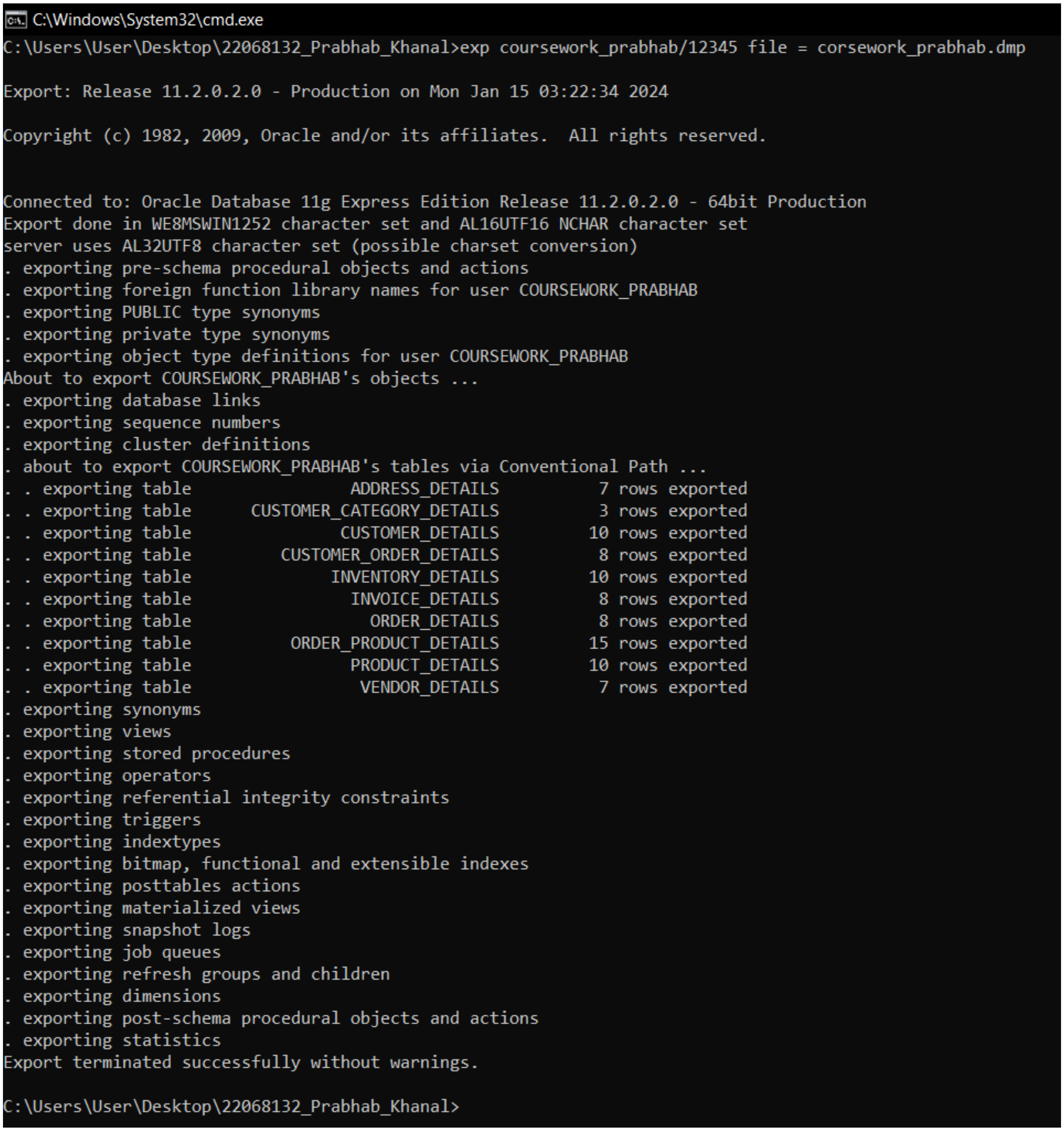
- Step 2: After that type command “exp coursework_prabhab/12345 file = coursework_prabhab.dmp”. After that press enter.



```
Microsoft Windows [Version 10.0.21996.1]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User\Desktop\22068132_Prabhab_Khanal>exp coursework_prabhab/12345 = file = corsework_prabhab.dmp
LRM-00116: syntax error at 'file' following '='

EXP-00019: failed to process parameters, type 'EXP HELP=Y' for help
EXP-00000: Export terminated unsuccessfully

C:\Users\User\Desktop\22068132_Prabhab_Khanal>exp coursework_prabhab/12345 file = corsework_prabhab.dmp
Export: Release 11.2.0.2.0 - Production on Mon Jan 15 03:22:24 2024
```

Figure 45: Code for exporting the dump file.

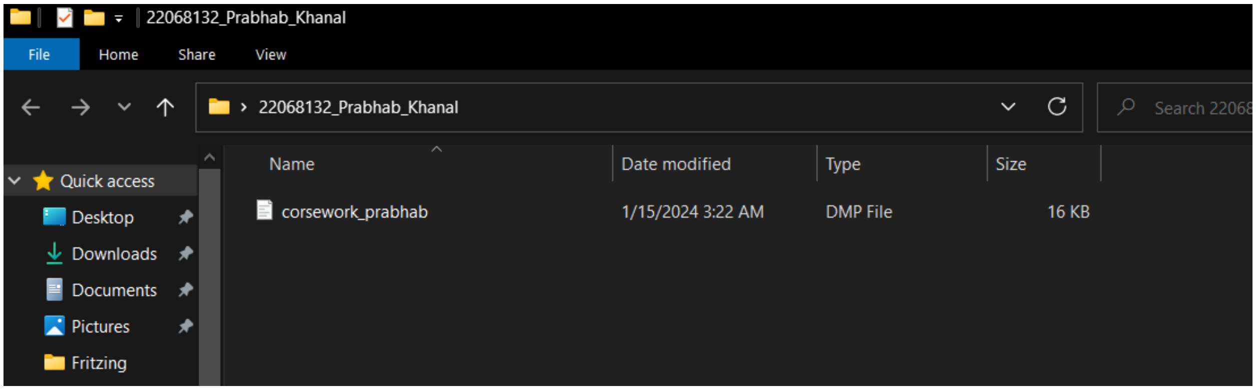
- Step 3: After pressing enter the process takes times for creating dump file. After some time “Export terminated successfully without warning” is seen and the dump file is created.



```
C:\Users\User\Desktop\22068132_Prabhab_Khanal>exp coursework_prabhab/12345 file = corsework_prabhab.dmp

Export: Release 11.2.0.2.0 - Production on Mon Jan 15 03:22:34 2024

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


Connected to: Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
Export done in WE8MSWIN1252 character set and AL16UTF16 NCHAR character set
server uses AL32UTF8 character set (possible charset conversion)
. exporting pre-schema procedural objects and actions
. exporting foreign function library names for user COURSEWORK_PRABHAB
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions for user COURSEWORK_PRABHAB
About to export COURSEWORK_PRABHAB's objects ...
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export COURSEWORK_PRABHAB's tables via Conventional Path ...
. . exporting table                ADDRESS_DETAILS          7 rows exported
. . exporting table      CUSTOMER_CATEGORY_DETAILS          3 rows exported
. . exporting table               CUSTOMER_DETAILS         10 rows exported
. . exporting table         CUSTOMER_ORDER_DETAILS          8 rows exported
. . exporting table              INVENTORY_DETAILS         10 rows exported
. . exporting table                INVOICE_DETAILS          8 rows exported
. . exporting table                  ORDER_DETAILS          8 rows exported
. . exporting table          ORDER_PRODUCT_DETAILS         15 rows exported
. . exporting table                PRODUCT_DETAILS         10 rows exported
. . exporting table                 VENDOR_DETAILS          7 rows exported
. exporting synonyms
. exporting views
. exporting stored procedures
. exporting operators
. exporting referential integrity constraints
. exporting triggers
. exporting indextypes
. exporting bitmap, functional and extensible indexes
. exporting posttables actions
. exporting materialized views
. exporting snapshot logs
. exporting job queues
. exporting refresh groups and children
. exporting dimensions
. exporting post-schema procedural objects and actions
. exporting statistics
Export terminated successfully without warnings.

C:\Users\User\Desktop\22068132_Prabhab_Khanal>
```

Figure 46: Exporting the dump file.

- Step 4: Here the dump file is created successfully.

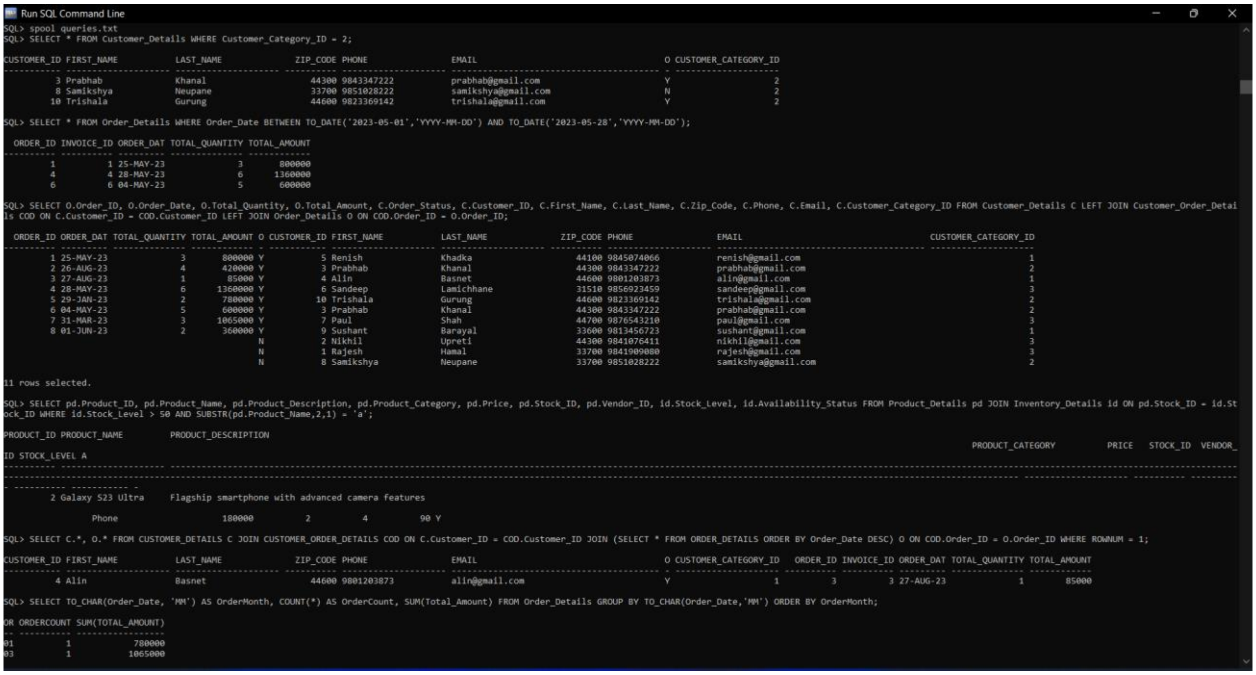


Name	Date modified	Type	Size
corsework_prabhab	1/15/2024 3:22 AM	DMP File	16 KB

Figure 47: Successfully creation of dump file.

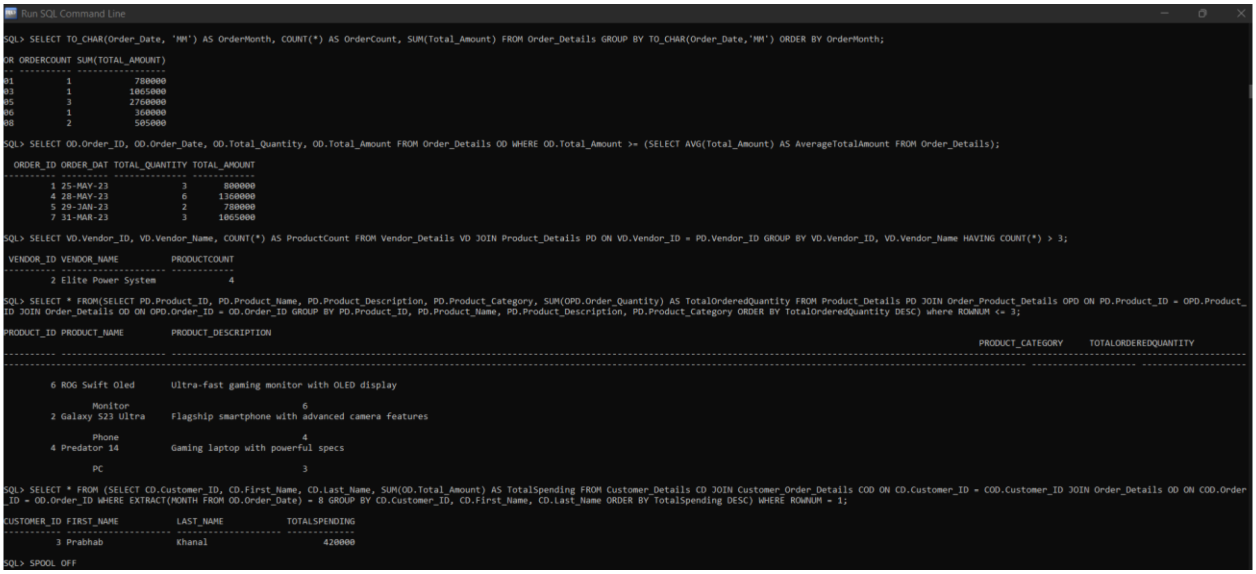
9.2. SPOOLING Query

The SPOOL command in Oracle SQL serves the purpose of storing the output from a SQL Plus session into a designated file. SQL Plus, an Oracle-provided command-line interface and reporting tool, facilitates interactions with Oracle Database. This command proves valuable when there is a need to record and retain the results of SQL statements, query outputs, or any information presented during the SQL*Plus session, allowing users to save it for subsequent reference or analysis. (Deveci, 2021)



```
SQL> spool queries.txt
SQL> SELECT * FROM Customer_Details WHERE Customer_Category_ID = 2;

CUSTOMER_ID FIRST_NAME           LAST_NAME              ZIP_CODE PHONE                EMAIL                                    O CUSTOMER_CATEGORY_ID
----------- -------------------- -------------------- ---------- -------------------- ---------------------------------------- - --------------------
          3 Prabhab              Khanal                    44300 9843347222           prabhab@gmail.com                        Y                    2
          8 Samikshya            Neupane                   33700 9851028222           samikshya@gmail.com                      N                    2
         10 Trishala             Gurung                    44600 9823369142           trishala@gmail.com                       Y                    2

SQL> SELECT * FROM Order_Details WHERE Order_Date BETWEEN TO_DATE('2023-05-01','YYYY-MM-DD') AND TO_DATE('2023-05-28','YYYY-MM-DD');

  ORDER_ID INVOICE_ID ORDER_DAT TOTAL_QUANTITY TOTAL_AMOUNT
---------- ---------- --------- -------------- ------------
         1          1 25-MAY-23              3       800000
         4          4 28-MAY-23              6      1360000
         6          6 04-MAY-23              5       600000

SQL> SELECT O.Order_ID, O.Order_Date, O.Total_Quantity, O.Total_Amount, C.Order_Status, C.Customer_ID, C.First_Name, C.Last_Name, C.Zip_Code, C.Phone, C.Email, C.Customer_Category_ID FROM Customer_Details C LEFT JOIN Customer_Order_Detai
ls COD ON C.Customer_ID = COD.Customer_ID LEFT JOIN Order_Details O ON COD.Order_ID = O.Order_ID;

  ORDER_ID ORDER_DAT TOTAL_QUANTITY TOTAL_AMOUNT O CUSTOMER_ID FIRST_NAME           LAST_NAME              ZIP_CODE PHONE                EMAIL                                    CUSTOMER_CATEGORY_ID
---------- --------- -------------- ------------ - ----------- -------------------- -------------------- ---------- -------------------- ---------------------------------------- --------------------
         1 25-MAY-23              3       800000 Y           5 Renish               Khadka                    44100 9845074066           renish@gmail.com                                            1
         2 26-AUG-23              4       420000 Y           3 Prabhab              Khanal                    44300 9843347222           prabhab@gmail.com                                           2
         3 27-AUG-23              1        85000 Y           4 Alin                 Basnet                    44600 9801203873           alin@gmail.com                                              1
         4 28-MAY-23              6      1360000 Y           6 Sandeep              Lamichhane                31510 9856923459           sandeep@gmail.com                                           3
         5 29-JAN-23              2       780000 Y          10 Trishala             Gurung                    44600 9823369142           trishala@gmail.com                                          2
         6 04-MAY-23              5       600000 Y           3 Prabhab              Khanal                    44300 9843347222           prabhab@gmail.com                                           2
         7 31-MAR-23              3      1065000 Y           7 Paul                 Shah                      44700 9876543210           paul@gmail.com                                              3
         8 01-JUN-23              2       360000 Y           9 Sushant              Barayal                   33600 9813456723           sushant@gmail.com                                           1
                                                 N           2 Nikhil               Upreti                    44300 9841076411           nikhil@gmail.com                                            3
                                                 N           1 Rajesh               Hamal                     33700 9841909080           rajesh@gmail.com                                            3
                                                 N           8 Samikshya            Neupane                   33700 9851028222           samikshya@gmail.com                                         2

11 rows selected.

SQL> SELECT pd.Product_ID, pd.Product_Name, pd.Product_Description, pd.Product_Category, pd.Price, pd.Stock_ID, pd.Vendor_ID, id.Stock_Level, id.Availability_Status FROM Product_Details pd JOIN Inventory_Details id ON pd.Stock_ID = id.St
ock_ID WHERE id.Stock_Level > 50 AND SUBSTR(pd.Product_Name,2,1) = 'a';

PRODUCT_ID PRODUCT_NAME         PRODUCT_DESCRIPTION
                                                                                                                                                                                                       PRODUCT_CATEGORY          PRICE   STOCK_ID  VENDOR_
ID STOCK_LEVEL A
---------- -------------------- ---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------ -------------------- ---------- ---------
- ---------- ----------- -
         2 Galaxy S23 Ultra     Flagship smartphone with advanced camera features

                Phone                     180000          2          4          90 Y

SQL> SELECT C.*, O.* FROM CUSTOMER_DETAILS C JOIN CUSTOMER_ORDER_DETAILS COD ON C.Customer_ID = COD.Customer_ID JOIN (SELECT * FROM ORDER_DETAILS ORDER BY Order_Date DESC) O ON COD.Order_ID = O.Order_ID WHERE ROWNUM = 1;

CUSTOMER_ID FIRST_NAME           LAST_NAME              ZIP_CODE PHONE                EMAIL                                    O CUSTOMER_CATEGORY_ID   ORDER_ID INVOICE_ID ORDER_DAT TOTAL_QUANTITY TOTAL_AMOUNT
----------- -------------------- -------------------- ---------- -------------------- ---------------------------------------- - -------------------- ---------- ---------- --------- -------------- ------------
          4 Alin                 Basnet                    44600 9801203873           alin@gmail.com                           Y                    1          3          3 27-AUG-23              1        85000

SQL> SELECT TO_CHAR(Order_Date, 'MM') AS OrderMonth, COUNT(*) AS OrderCount, SUM(Total_Amount) FROM Order_Details GROUP BY TO_CHAR(Order_Date,'MM') ORDER BY OrderMonth;

OR ORDERCOUNT SUM(TOTAL_AMOUNT)
-- ---------- -----------------
01          1            780000
03          1           1065000
```

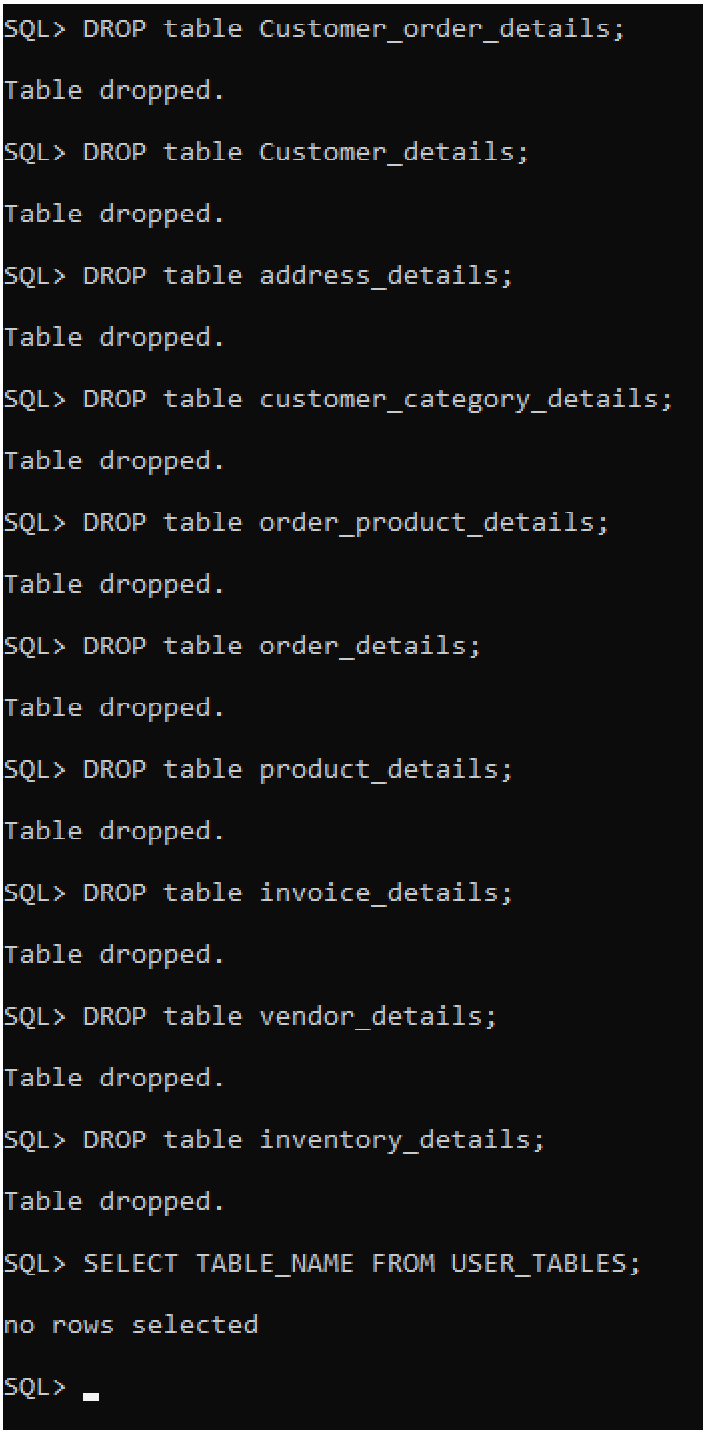
Figure 48: Spooling query(i)



```
SQL> SELECT TO_CHAR(Order_Date, 'MM') AS OrderMonth, COUNT(*) AS OrderCount, SUM(Total_Amount) FROM Order_Details GROUP BY TO_CHAR(Order_Date,'MM') ORDER BY OrderMonth;

OR ORDERCOUNT SUM(TOTAL_AMOUNT)
-- ---------- -----------------
01          1            780000
03          1           1065000
05          3           2760000
06          1            360000
08          2            505000

SQL> SELECT OD.Order_ID, OD.Order_Date, OD.Total_Quantity, OD.Total_Amount FROM Order_Details OD WHERE OD.Total_Amount >= (SELECT AVG(Total_Amount) AS AverageTotalAmount FROM Order_Details);

  ORDER_ID ORDER_DAT TOTAL_QUANTITY TOTAL_AMOUNT
---------- --------- -------------- ------------
         1 25-MAY-23              3       800000
         4 28-MAY-23              6      1360000
         5 29-JAN-23              2       780000
         7 31-MAR-23              3      1065000

SQL> SELECT VD.Vendor_ID, VD.Vendor_Name, COUNT(*) AS ProductCount FROM Vendor_Details VD JOIN Product_Details PD ON VD.Vendor_ID = PD.Vendor_ID GROUP BY VD.Vendor_ID, VD.Vendor_Name HAVING COUNT(*) > 3;

 VENDOR_ID VENDOR_NAME          PRODUCTCOUNT
---------- -------------------- ------------
         2 Elite Power System              4

SQL> SELECT * FROM(SELECT PD.Product_ID, PD.Product_Name, PD.Product_Description, PD.Product_Category, SUM(OPD.Order_Quantity) AS TotalOrderedQuantity FROM Product_Details PD JOIN Order_Product_Details OPD ON PD.Product_ID = OPD.Product_
ID JOIN Order_Details OD ON OPD.Order_ID = OD.Order_ID GROUP BY PD.Product_ID, PD.Product_Name, PD.Product_Description, PD.Product_Category ORDER BY TotalOrderedQuantity DESC) where ROWNUM <= 3;

PRODUCT_ID PRODUCT_NAME         PRODUCT_DESCRIPTION
                                                                                                                                                                            PRODUCT_CATEGORY     TOTALORDEREDQUANTITY
---------- -------------------- ----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------- -------------------- --------------------

         6 ROG Swift Oled       Ultra-fast gaming monitor with OLED display

               Monitor                              6
         2 Galaxy S23 Ultra     Flagship smartphone with advanced camera features

               Phone                                4
         4 Predator 14          Gaming laptop with powerful specs

               PC                                   3

SQL> SELECT * FROM (SELECT CD.Customer_ID, CD.First_Name, CD.Last_Name, SUM(OD.Total_Amount) AS TotalSpending FROM Customer_Details CD JOIN Customer_Order_Details COD ON CD.Customer_ID = COD.Customer_ID JOIN Order_Details OD ON COD.Order
_ID = OD.Order_ID WHERE EXTRACT(MONTH FROM OD.Order_Date) = 8 GROUP BY CD.Customer_ID, CD.First_Name, CD.Last_Name ORDER BY TotalSpending DESC) WHERE ROWNUM = 1;

CUSTOMER_ID FIRST_NAME           LAST_NAME            TOTALSPENDING
----------- -------------------- -------------------- -------------
          3 Prabhab              Khanal                      420000

SQL> SPOOL OFF
```

Figure 49: Spooling query(ii)

9.3. Dropping Tables



```
SQL> DROP table Customer_order_details;

Table dropped.

SQL> DROP table Customer_details;

Table dropped.

SQL> DROP table address_details;

Table dropped.

SQL> DROP table customer_category_details;

Table dropped.

SQL> DROP table order_product_details;

Table dropped.

SQL> DROP table order_details;

Table dropped.

SQL> DROP table product_details;

Table dropped.

SQL> DROP table invoice_details;

Table dropped.

SQL> DROP table vendor_details;

Table dropped.

SQL> DROP table inventory_details;

Table dropped.

SQL> SELECT TABLE_NAME FROM USER_TABLES;

no rows selected

SQL>
```

Figure 50: Dropping all tables.

10. References

Brewer, T., 2024. *Functionly.* [Online]
Available at: https://www.functionly.com/orginometry/org-charts/what-is-an-entity-relationship-diagram-and-how-do-they-work
[Accessed 6 1 2024].

Deveci, M. S., 2021. *ittutorial.* [Online]
Available at: https://ittutorial.org/oracle-spool-to-file-in-sqlplus/
[Accessed 15 1 2024].

Nishadha, 2022. *creately.* [Online]
Available at: https://creately.com/guides/er-diagrams-tutorial/
[Accessed 6 1 2024].

Pedamkar, P., 2023. *EDUCBA.* [Online]
Available at: https://www.educba.com/oracle-queries/
[Accessed 14 1 2024].

Peterson, R., 2023. *Guru99.* [Online]
Available at: https://www.guru99.com/database-normalization.html
[Accessed 6 1 2024].