LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

CC5051NI Databases

50% Individual Coursework

Autumn 2023

Student Name: RENISH SINGH KHADKA

London Met ID: 22068067

Assignment Submission Date: 15 January 2024

Word Count: 9886

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 Table

1. Introduction

In the modern city of Nepal, John, an entrepreneur with a great passion for electronics, embarks on a visionary quest to create a "gadget emporium". Driven by passion and careful planning down to the last detail, John laid the foundation for his online marketplace that appeals to technology enthusiasts and businesses alike. The foundation of this effort was a robust Lyman-style database system designed to seamlessly manage complex customer information, product details, and order processing.

Designed as a haven for cutting-edge technology, his Gadget Emporium offers a wide selection of electronics and accessories. From cutting-edge IoT devices to innovative smart kitchen appliances, the digital shelf was meant to showcase the latest technological wonders. Database systems played an important role, ensuring that not only basic details but also preferences and purchase history were recorded in the customer profile, allowing for a personalized and customized shopping experience for him. The product database is designed to cover a wide range of electronic wonders, making Gadget Emporium a one-stop destination for technology enthusiasts.

As Gadget Emporium takes shape, a seamless ecosystem has emerged that allows technology to be seamlessly integrated into everyday life. An online marketplace showcased his IoT items such as smart home security systems and connected thermostats, while a smart kitchen section showcased devices that can be controlled remotely. A database system coordinated back-end operations, so orders flowed easily from customer to warehouse, ensuring on-time delivery and customer satisfaction. His Gadget Emporium, the centerpiece of Nepal, is a testament to John's vision and commitment, setting a new standard for the harmonious integration of technology into everyday life.

1.1 Aim and Objective

The main purpose of the database system proposed for the online marketplace "Gadget Emporium" is to create a robust and efficient infrastructure to seamlessly manage and organize customer, product, and order information. The other aim of this Gadget Emporium is to: -

- Implement efficient inventory tracking for timely restocking.
- Integrate secure payment gateways for seamless transactions.
- Gain real-time visibility from supplier to customer.
- Foster vibrant online communities through seamless social media integration.

Some objectives for **Gadget Emporium** include: -

- Efficiently track and store all your electronic devices and accessories details such as name, description, category, price, inventory, and suppliers.
- Classify customers (regulars, employees, VIPs) and apply appropriate discount rates. Save customer addresses for accurate shipping.
- Records details for each order, including products purchased, quantity, unit price, total amount, and payment options.
- Track product suppliers and link each product to its corresponding provider.
- Monitor product availability and inventory levels to prevent overselling.
- We offer secure and seamless transaction options such as cash on delivery.
- We provide a detailed invoice for each order at checkout, including product details, customer information, payment method, and applicable discounts.
- Efficiently manage product catalogs, customer relationships, orders, suppliers, and inventory.
- Integrate secure payment gateways for seamless and secure transactions.
- Real-time visibility of your supply chain for efficient product tracking.
- Ensure robust data security measures and compliance to ensure customer trust and information integrity.

1.2 Current Business Activities and Operations

John is driven by his passion for electronics in his entrepreneurial efforts to create an online marketplace, Gadget Emporium, offering a diverse selection of electronic devices and accessories to both individual consumers and corporate organizations. That’s what I'm aiming for. I have been assigned the role of database architect at this company and am working on developing a robust database system to efficiently manage core business processes. These include complex product management, customer segmentation with associated discounts, comprehensive product details capture for seamless order fulfillment, careful supplier management to maintain supplier data, product availability and inventory levels. Includes real-time tracking, and integration with various payment gateways.

Issuance of detailed invoices after securities trading and order confirmation. The goal is to create a streamlined and efficient operational framework for Gadget Emporium, facilitating effective management of products, customers, and orders while ensuring a seamless and satisfying online shopping experience for customers.

1.2.1 Business Rules

Gadget Emporium has certain rules to perform its business rules. They can be listed as:

- Each product belongs to one category, and each category can contain multiple products.
- Assign different discounts depending on customer category (0%, 5%, 10%).
- Allows multiple products per order and products in multiple orders.
- Maintain records of vendors providing electronic equipment.
- By associating each product with one vendor, each vendor can offer multiple products.
- Track product availability in real time and prevent overselling.
- Add inventory details such as inventory quantity and stock status.
- All orders require a payment option.

- Upon order confirmation, we will issue an invoice containing order, customer, and payment details, including any applicable discounts.

1.2.2 Business Assumption

Some business assumption I made while creating database are:

- I assume that customer can order many or zero, but one customer can order multiple products.
- The Customer_ZipCode in the Customer table is assumed to correspond to the zip code in the Customer_Address table.
- Assume that the category table Customer_Category_Type contains many sorts of consumers that may be segmented depending on factors such as loyalty, regularity, and purchase volume.
- I believe that Order_Status is Ture or False that will show the customer who have order the product or not.
- Assume that Discount_Rate in the category table provides the discount rate applied to each defined customer category to allow for individualized pricing or promotional strategies.
- The Order_ID in the Invoice_Details field is intended to link each invoice to a single order. The company prioritizes data consistency and requires all invoices to be connected to valid orders.
- I assume that the total amount shown on the invoice schedule appropriately represents the complete cost of the products or services associated with a certain purchase. This assumption guarantees the financial accuracy of the invoice details.
- I believe that the links established by the Customer_Order_Details table accurately connect customers to their orders and the individual products requested. This assumption guarantees the accuracy of the customer's purchasing history.

2. Entity Relationship Diagram (ERD)

An entity-relationship diagram (ERD), also known as an entity-relationship model, is a graphical representation of relationships between people, objects, places, concepts, or events in an information technology (IT) system. ERD uses data modeling techniques to help define business processes and serves as the basis for relational databases. (Jacqueline Biscobing, 2019)

Importance of ERD:

- ERDs offer a visual framework for initiating database design, providing a structured representation for relational data.
- ERDs assist in identifying information system requirements across an organization, aiding in the planning and design phases.
- Even after the implementation of a relational database, ERDs continue to serve as a reference for debugging and business process re-engineering if needed.
- ERDs are less effective in representing semi-structured or unstructured data due to their inherent focus on relational structures.
- While valuable for organizing relational data, ERDs may not independently facilitate the seamless integration of data into pre-existing information systems.
- ERDs should be complemented with other tools and methodologies when dealing with diverse data types and integrating into complex existing information systems.

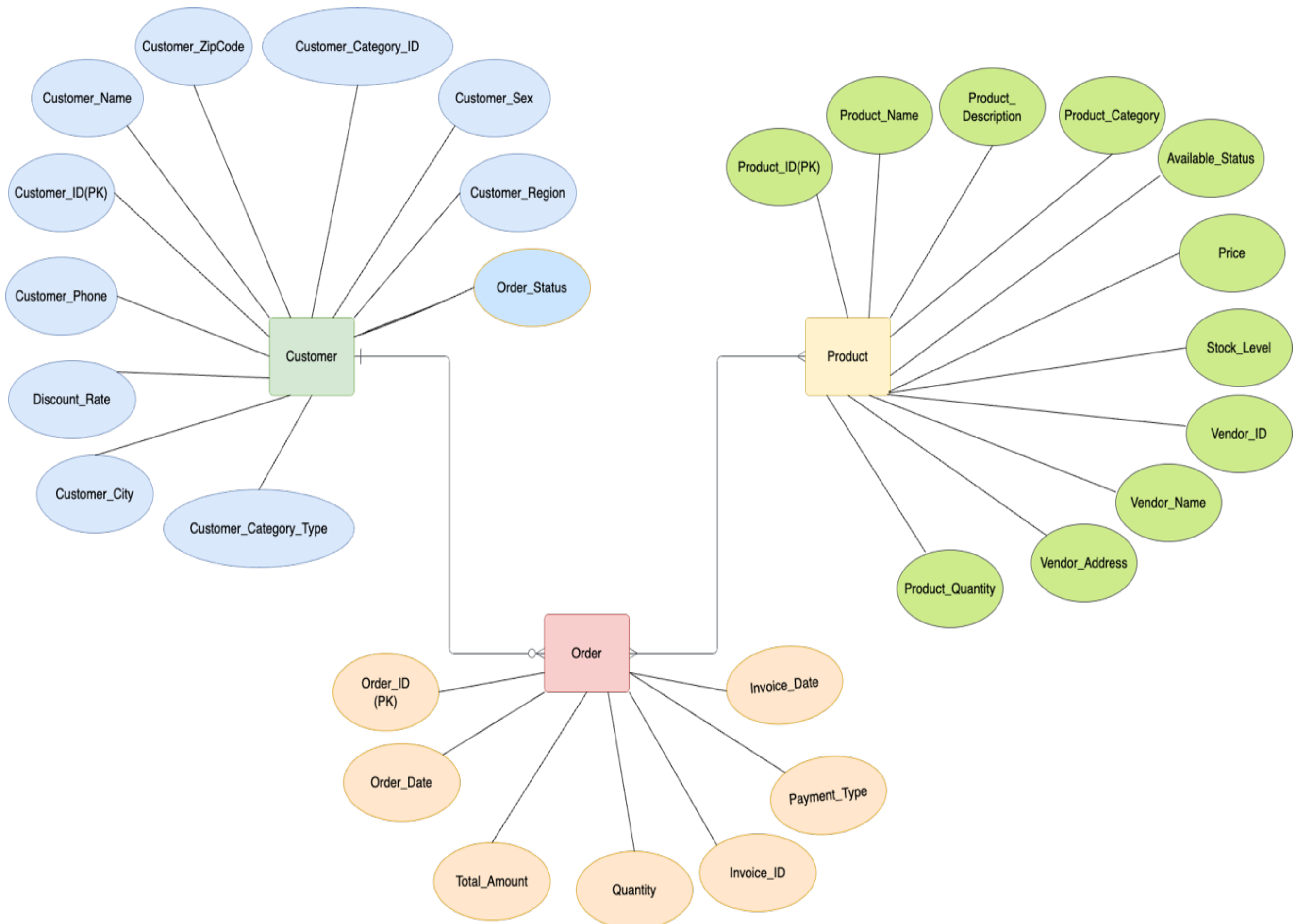


Figure 1: Initial Entity-Relationship Diagram.

The absence of chasm and fan traps in the first Entity-Relationship Diagram (ERD) indicates a well-organized depiction of many-to-one relationships and primary-detail structures. This suggests that the database architecture avoided instances in which too many relationships converge on a single table or follow one another in a way that complicates queries The ERD is expected to specify one-to-many or zero relationships precisely, ensuring that entities on the "one" side are uniquely associated with things on the "many" or "zero" sides. This organization helps to ensure data integrity and gives a straightforward structure for connecting records across multiple entities without creating excessive redundancy.

Normalization approaches are critical for addressing the issues given by partial and transitive dependencies in the ERD. Normalization is the process of structuring the database structure to reduce redundancy and dependence concerns, ensuring that data is logically arranged and easy to maintain. Designers can reduce the likelihood of anomalies and improve overall database efficiency by breaking down huge tables into smaller ones and adhering to specified normalization forms, such as the First Normal Form (1NF) and Third Normal Form (3NF). Normalization provides a more strong and stable foundation for the ERD, supporting data integrity and allowing for seamless data manipulation operations.

3. Identification of Entities and Attributes

3.1 Entities and Attributes

Real things that are easily recognizable and unrecognizable, living or not. Everything within your company needs to be represented in a database. It can be physical or just facts about your company or events happening in the real world. An entity is a place, person, object, event, or concept that stores data in a database. Entity characteristics require attributes and unique keys. Each entity consists of several "attributes" that describe that entity. (Peterson, 2023)

Entity in my Initial Entity Relationship Diagram (ERD): -

- Customer
- Order
- Product

This is a single-valued property of either an entity type or a relationship type. For example, a lecture can have attributes such as time, date, duration, and location. In the example ER diagram, attributes are represented by ellipses. (Peterson, 2023)

Attributes that are in my Initial Entity Relationship Diagram (ERD): -

Attributes of Customer: - *(Customer_ID, Customer_Name, Customer_ZipCode, Customer_Region, Customer_City, Customer_Phone, Customer_Category_ID, Customer_Sex, Discount_Rate, Order_Status)*

Attributes of Order: - *(Order_ID, Order_Date, Total_Amount, Quantity, Invoice_ID, Payment, Invoice_Date)*

Attributes of Product: - *(Product_ID, Product_Name, Product_Description, Available_Status,Price,Stock_Level,Product_category,Product_Quantity,Vendor_ID, Vendor_Name, Vendor_Address)*

3.1.1 Customer

Attributes	Datatype	Constraints	Description
Customer_ID	int	Primary key	A unique identifier assigned to each customer. This field is used as the primary key for individual customer records.
Customer_Name	Varchar2(50)	Not Null	Customer's full name. Usually displayed in first name, last name format.
Customer_ZipCode	int	Not Null	Zip code indicating the customer's geographic location.
Customer_Region	Varchar2(50)	Not Null	Specify the region where customers are located. This could be a broader geographic classification beyond Zip codes.
Customer_City	Varchar2(50)	Not Null	The specific city or town where the customer lives.
Customer_Phone	Varchar2(50)	Not Null	The customer's primary telephone number used for contact purposes.
Customer_Category_ID	Varchar2(50)	PK, FK(Customer)	A numeric code that indicates the customer's category. Often based on purchases.
Customer_Category_Type	Varchar2(50)	Not Null	A descriptive term for the consumer category, such as "Regular," "Staff," or "VIP"
Discount_Rate	Decimal (7,3)	Not Null	Discount percentage available to customers based on category

Customer_Sex	Varchar2(50)	Not Null	Customer gender. It is usually designated by a "Male" for men and an "Female" for women.
Order_Status	Varchar2(5)	Not Null	Indicates the status of the order in "True or False"

Table 1: Customer_Description.

3.1.2 Order

Attributes	Datatype	Constraints	Description
Order_ID	int	Primary key	Unique identifier for each order. It is used as a primary key.
Order_Date	Date	Not Null	The date the order was placed and indicates the start of the transaction.
Total_Amount	Varchar2(50)	Not Null	The order's total monetary value represents the sum of all product prices.
Quantity	int	Not Null	Indicates the number of items purchased as part of an order.
Invoice_ID	int	PK, FK(	A unique identification for the linked invoice that links the order to the corresponding financial record.
Payment_Type	Varchar2(50)	Not Null	The method used for payment (e.g., credit card, cash, online payment).
Invoice_Date	Date	Not Null	Date the invoice was created or processed.

Table 2: Order_Description.

3.1.3 Product

Attributes	Datatype	Constraints	Description
Product_ID	int	Primary Key	Each product is assigned a unique identifier for efficient tracking and management.
Product_Name	Varchar2(50)	Not Null	The complete name of the product, usually used for display and identification.
Product_Description	Varchar2(1000)	Not Null	Detailed description of product features, specifications, and benefits. Often used to educate customers and create attractive product listings.
Available_Status	Varcha2(5)	Not Null	Displays the stock status of the product, whether it is currently in stock or not.
Price	int	Not Null	The price associated with the product.
Stock_Level	int	Not Null	Amount of product currently available in inventory. Used to manage inventory levels and prevent stockouts.
Product_Category	Varchar2(500)	Not Null	
Vendor_ID	int	Primary Key,	A unique identifier assigned to a product provider or supplier.
Vendor_Name	Varchar2(50)	Not Null	The name of the firm or person that supplies the goods.

Vendor_Address	Varchar2(50)	Not Null	The address or location details of the vendor.

Table 3: Product_Description.

4. Normalization

Normalization is a process that eliminates data redundancy and improves the integrity of data in tables. Normalization also helps organize data in your database. This is a multi-step process that converts data into tabular format and removes duplicate data from relational tables. Normalization organizes database columns and tables so that database integrity constraints correctly enforce their dependencies. It is a systematic technique that decomposes tables to eliminate data redundancy (repetition) and unnecessary features such as abnormal inserts, updates, and deletes. (S, 2023)

4.1 Un-Normalized Form (UNF)

Unnormalized Format (UNF), also known as Unnormalized or Non-First Normal Form (NF2) relationships, is a way to represent data in a data model. This is the simplest form to represent a table in a database and does not correspond to any normalized form. Characterized by data redundancy, often containing complex data structures within a single attribute. (Garge, 2022)

In my ERD, a customer can get multiple orders, each of which may contain a variety of products. Since I began with "Customers," "Customers" is the repeated data, "Orders" is the repeating group of customers, and "Products" is another repeating group of orders.

Customer (*Customer_ID, Customer_Name, Customer_ZipCode, Customer_Region, Customer_City,Customer_Phone,Customer_Category_ID,Customer_Category_Type,Customer_Sex,Discount_Rate,Order_Status{*Order*_ID,Order_Date,Total_Amount,Order_Quantity,Invoice_ID,Payment_Type,Invoice_Date{*Product*_ID,Product_Name,Product_Description, Available_Status, Price, Stock_Level, Product_Quantity, Product_Category,Vendor_ID, Vendor_Name, Vendor_Address}})*

4.2 First Normal Form (1NF)

Moving from Unnormalized Format (UNF) to First Normal Format (1NF) requires separating repeating groups and data for a more structured database. This process clearly organizes entities such as customers, orders, and products into separate tables. The "Customer" table is defined with "Customer_ID" as the primary key, the "Order" table is defined with both "Order_ID" and "Customer_ID" as the primary key, and the "Product table" is defined with "Product_ID" as the primary key. Primary key and Customer_ID and Order_ID are defined as foreign keys to establish the relationship.

Rules to achieve First Normalization Form are: -

- Each column in a table must be unique.
- Separate tables must be created for each set of related data.
- Each table must be identified by a unique or concatenated column called a primary key.
- Rows cannot be duplicated.
- Columns cannot be duplicated.
- Row/column intersection does not contain NULL values.
- Row/column intersection does not contain multi-value field.

  (Rouse, 2011)

Customer 1 (*Customer_ID**(PK)**, Customer_Name, Customer_ZipCode, Customer_Region, Customer_City, Customer_Phone, Order_Status, Customer_Category_ID, Customer_Category_Type, Customer_Sex, Discount_Rate)*

Note: *This is a repeating data for repeating group having Customer_ID Primary Key*

Order 1 (*Order_ID**(PK)**, Customer_ID**(FK)** Order_Date, Total_Amount, Quantity, Invoice_ID. Payment_Type, Invoice_Date)*

Note: *This is a repeating group for another repeating group having Primary Key Order_ID and Foreign Key Customer_ID from Cutomer table.*

Product 1 (*Product_ID**(PK)**,* Order_*ID**(FK)**, Customer_ID**(FK)**, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Product_Quantity, Vendor_ID, Vendor_Name, Vendor_Address)*

Note: *This is a repeating group having Primary Key Product_ID and Foreign Key Customer_ID, Order_ID from Order table.*

4.3 Second Normal Form (2NF)

To normalize a 1NF relationship to 2NF, partial dependencies must be removed. Partial dependencies within a DBMS are non-primary attributes, that is, attributes that are not part of a candidate key are not fully functionally dependent on one of the candidate keys.

Rules to achieve Second Normalization Form are: -

- Ensure the table is in the First Normal Form.
- Eliminate dependencies where non-primary attributes are based on only part of the composite primary key.
- Make sure each non-primary attribute is fully dependent on the entire primary key.

(Babbar, 2022)

Checking Partial Dependency for Customer1 Table

The Customer1 table satisfies the Second Normal Form (2NF) criteria since it includes a single candidate key, removing the possibility of partial dependencies. This design option improves data organization while avoiding repetition in the table structure.

Customer 2 (*Customer_ID**(PK)**, Customer_Name, Customer_ZipCode, Customer_Region, Customer_City, Customer_Phone, Customer_Category_ID, Customer_Category_Type, Customer_Sex, Discount_Rate)*

Checking Partial Dependency for Order 1 Table

The Order 1 table, which has both Order_ID and Customer_ID as keys, requires a check for partial dependencies. This evaluation guarantees that all characteristics are completely dependent on the composite key, which is consistent with the Second Normal Form (2NF) criteria for a well-organized database structure.

Order 1 (*Order_ID**(PK)**, Customer_ID**(FK)** Order_Date. Total_Amount, Quantity, Invoice_ID. Payment_Type, Invoice_Date)*

Order_ID determines certain information about the order, such as Order_Date, Quantity. These attributes are functionally dependent in Order_ID.

Order_ID *→ Order_Date, Quantity*

All attributes are specified by Order_ID and Order_ID, Customer_ID, so Customer_ID does not specify any attributes. Hence Customer_ID is foreign key in order table but primary key in customer table, so it doesn't give attributes in order table.

Customer_ID *→ XXX*

Orders are linked to customers using the composite foreign key made up of Order_ID and Customer_ID. This combination makes it easier to get related information such as Invoice_ID, Total_Amount, and Payment, which improves data organization in a relational database structure.

<u>Order_ID</u>, <u>Customer_ID</u> ** → Invoice_ID, Invoice_Date Total_Amount, Payment*

Final 2nf table for order:
Order 2 (*Order_ID****(PK)****, Order_Date, Quantity)*

Order_Details 2 (*Order_ID****(FK)****, Customer_ID****(FK),*** *Invoice_ID, Invoice_Date ,Total_Amount, Payment)*

<u>Checking Partial Dependency for Product 1 Table</u>

With three keys (Product_ID, Order_ID, and Customer_ID) in the Product1 database, it is critical to check for partial dependencies. Product_ID, except for Customer_ID and Order_ID, is the only characteristic that can be uniquely determined for every product. This inspection guarantees that the Second Normal Form (2NF) rules are followed, resulting in a well-organized and normalized database.

Product 1 (*Product_ID****(PK)****,* Order_*ID****(FK)****, Customer_ID****(FK)****, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Product_Quantity, Vendor_ID, Vendor_Name, Vendor_Address)*

By knowing the Product_ID, we can clearly determine all the details about this Product_Name, Description, Availability, Price, Stock, and Supplier Information which are all functionally dependent in Product_ID primary key.

Product_ID *→ (Product_ID**(PK)**, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID, Vendor_Name, Vendor_Address)*

Order_ID is part of the primary key in order table, so it already contributes to the uniqueness of each row. Therefore, it doesn't give attributes in product table because Order_ID is referring foreign key in product table.

Order_ID *→ XXX*

Like Order_ID, Customer_ID is part of the primary key and does not create any partial dependencies.

Customer_ID *→ XXX*

These will be the bridge entity for Order and Product to remove Many to Many relations. The combination of Product_ID and Order_ID already forms a unique identifier for each data record through a composite primary key. There are no partial dependencies here.

<u>Product_ID, Order_ID</u> ** → XXX*

As in the previous case, the combination of Product_ID and Customer_ID acts as a unique identifier for each row in the table without creating partial dependencies.

Product_ID, Customer_ID *→ XXX*

There are no partial dependencies because the (Product_ID, Order_ID, Customer_ID) is the primary key. This combination uniquely identifies the entire line and give the attributes Product_Quantity

<u>Product_ID, Customer_ID, Order_ID</u> ** → Product_Quantity*

Final 2nf table for Product:

Product 2 (*Product_ID**(PK)**, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category,Vendor_ID, Vendor_Name, Vendor_Address)*

Customer_Order_Product_Details 2 (*Product_ID**(FK)**,* Order_*ID**(FK)**, Customer_ID**(FK),** Product_Quantity)*

The Details table brings together information about orders, customers, and products in one place. It simplifies queries that require data from various sources. The Details table collects information about orders, customers, and products in one location. It supports queries that require data from various sources. Detail tables make querying and data retrieval easier, especially in scenarios where you need a combination of order, customer, and product information.

So that's the reason why I am not making these table:

Order_Customer 2(Order_*ID**(FK)**, Customer_ID**(FK))***

Pro_Order_Details 2 (Product_*ID**(FK)**, Order_ID**(FK))***

Pro_Customer_Details 2 (*Product_ID**(FK)**, Customer_ID**(FK))***

Final Table in 2NF

Final table that is formed in Second Normal Form: -

Customer 2 (*Customer_ID**(PK)**, Customer_ZipCode, Customer_Region, Customer_City, Customer_Phone, Order_Status, Customer_Category_ID, Customer_Category_Type, Customer_Sex, Order_status, Discount_Rate)*

Order 2 (*Order_ID**(PK)**, Order_Date, Quantity)*

Order_Details 2 (*Order_ID(FK), Customer_ID(FK), Invoice_ID, Invoice_Date, Total_Amount, Payment_Type)*

Product2 (*Product_ID**(PK)**, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID, Vendor_Name, Vendor_Address)*

Details 2(*Product_ID(FK), Customer_ID(FK), Order_ID(FK), Product_Quantity)*

4.4 Third Normalization Form(3NF)

Normalizing 2NF relations to 3NF entails eliminating transitive dependencies in the database management system. A functional relationship X -> Z is considered transitive if the following three functional dependencies are met:

Rules to achieve Third Normalization Form are: -

- Verify that the table is in second normal form (2NF) by removing partial dependencies.
- Make sure your table does not contain any transitive dependencies. On-prime attributes should not depend on other non-prime attributes.
- Make sure each non-primary attribute depends directly on the primary key and not indirectly on other non-primary attributes.

(Babbar, 2022)

Checking transitive dependency for Customer

Transitive dependencies were discovered when the Customer table was converted from 2NF to 3NF. To address this, direct dependencies were built via Customer_ID, guaranteeing that each attribute receives its value directly from the main key. This is consistent with the Third Normal Form (3NF) principles, which emphasize database structure for increased efficiency and decreased redundancy.

Customer2 (*Customer_ID**(PK)**, Customer_Name, Customer_ZipCode, Customer_Region, Customer_City, Customer_Phone, Customer_Category_ID, Customer_Category_Type, Customer_Sex, Order_Status)*

To solve transitive dependency, ensure that Customer_ID uniquely determines Customer_Category_ID, and that Customer_Category_ID determines both Customer_Category_Type and Discount_Rate. This is consistent with the Third Normal Form (3NF) concepts of improving database structure for increased efficiency and data integrity.

Customer_ID(PK)→**Customer_Category_ID**----->Category_Type, Discount_Rate

To address transitive dependence, show that Customer_ID uniquely determines Customer_ZipCode, which in turn uniquely determines both Customer_Region and Customer_City.

Customer_ID(PK)→ *Customer_*ZipCode -----> *Customer_Region, Customer_City*

After removing transitive dependency there will form of two table which is shown as below:

Category 3 *(Customer_Category_ID**(PK)**,* Customer_Category_Type*, Discount_Rate)*

Customer_Address 3 *(Customer_ZipCode, Customer_Region, Customer_City)*

Final 3nf table for Product:

Customer 3 (*Customer_ID**(PK)**, Customer_ZipCode (FK), Customer_Name, Customer_Phone, Customer_Sex, Order_Status)*

Category 3 *(Customer_Category_ID**(PK)**,* Customer_Category_Type*, Discount_Rate)*

Customer_Address 3 *(Customer_ZipCode, Customer_Region, Customer_City)*

Checking transitive dependency for Order

While transitioning from 2NF to 3NF for the Order database, we developed the Order 2 and Order_Details tables. In Order_Details 2, we discovered a transitive dependence issue with Invoice_ID, which was indirectly related to two foreign keys, resulting in additional attribute values. To address this, we explicitly connected Invoice_ID to the appropriate foreign key. This adheres to 3NF principles, simplifying the database for more efficiency and fewer unwanted data repetition.

Order 2 (*Order_ID**(PK)**, Order_Date, Quantity)*

Order_Details 2 (*Order_ID(FK), Customer_ID(FK), Invoice_ID, Total_Amount, Payment_Type)*

To overcome transitive dependence, show that the combination of Order_ID and Customer_ID uniquely produces Invoice_ID, and that Invoice_ID also determines Total_Amount and Payment.

Order_ID, Customer_ID*→ **Invoice_ID** ----- > *Invoice_Date, Total_Amount, Payment_Type*

Then there will be two different table after removing transitive dependency:

Invoice_Details 3 (Order_ID(FK), Custome_ID(FK), Invoice_ID(FK))

Invoice 3 (*Invoice_ID(PK), Invoice_Date ,Total_Amount, Payment_Type)*

Final 3nf table for Product:

Order 3 (*Order_ID**(PK)**, Order_Date, Quantity)*

Invoice_Details 3 (Order_ID(FK), Custome_ID(FK), Invoice_ID(FK))

Invoice 3 (*Invoice_ID(PK), Invoice_Date ,Total_Amount, Payment_Type)*

Checking transitive dependency for Product

After reaching 2NF with the Product 2 table, we discovered a transitive dependence problem with Vendor_ID, resulting in additional characteristics from the Product table. To address this issue, we explicitly connected Vendor_ID to its associated characteristics. This change adheres to 3NF principles, making the database more efficient and minimizing needless data duplication.

Product 2 (*Product_ID**(PK)**, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category Vendor_ID, Vendor_Name, Vendor_Address)*

Details 2(*Product_ID(FK), Customer_ID(FK), Order_ID(FK))*

To solve transitive dependency, ensure that Product_ID uniquely determines Vendor_ID, and that Vendor_ID uniquely determines the vendor's name, address, and contact information.

Product_ID→ Vendor_ID ----- > Vendor_Name, Address, Contact

Then there will be two table Product and Vendor after removing transitive dependency:

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

Final Table in 3NF for Product

Product 3 (*Product_ID**(PK)**, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category)*

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

Final Table in 3NF

Customer (*Customer_ID**(PK)**, Customer_ZipCode (FK), Customer_Name, ,Customer_Phone, Customer_Sex, Order_Status)*

Category*(Customer_Category_ID**(PK)**,Customer_Category_Type, Discount_Rate)*

Customer_Address *(Customer_ZipCode, Customer_Region, Customer_City)*

Invoice_Details (Order_ID(FK), Custome_ID(FK), Invoice_ID(FK))
Invoice (*Invoice_ID(PK), Ioice_Date Total_Amount, Payment_Type*)
Orders (*Order_ID**(PK)**, Order_Date, Quantity)*
Product (*Product_ID**(PK)**, Vendor_ID**(FK),** Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category)*

Vendor *(Vendor_ID**(PK)**, Vendor_Name, Vendor_Address)*

Customer_Order_Details (*Product_ID(FK), Customer_ID(FK), Order_ID(FK), Product_Quantity)*

5. Final Entity Relationship Diagram

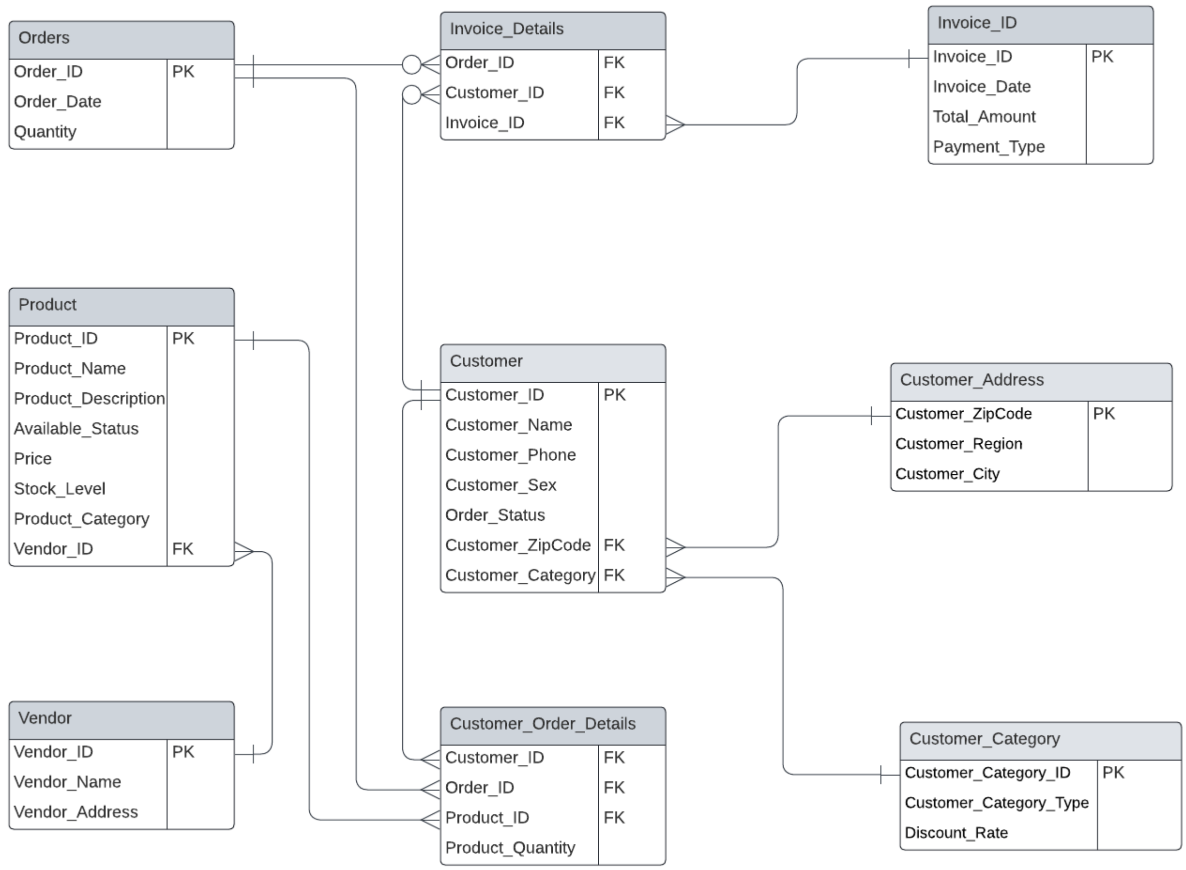


Figure 2: Final Entity-Relation Diagram.

The ERD you see is the outcome of a normalization process, and it has nine tables, each with unique properties from different data types. The diagram also includes two bridge entity, which are purposefully placed to allow connections across tables and ensure meaningful linkages between them.

In the final Entity-Relationship Diagram (ERD), a logical model takes precedence over a physical one, emphasizing the representation of the database's logical structure and the linkages between its constituent entities. This means that the final ERD will focus on displaying the links between things rather than digging into the minutiae of the hardware or software that will be used for database implementation. The fundamental goal is to capture the conceptual and relational features, resulting in a high-level abstraction that transcends the complexities of the eventual technology implementation.

There are many benefits to using logical diagrams when creating the ultimate entity-relationship diagram (ERD). First, logical structures can be seamlessly translated into different physical implementations, increasing adaptability during the design phase. Additionally, it provides a clear and concise visual representation of relationships between entities, making communication and documentation easier. Essentially, logical ERDs streamline the database design process and promote efficiency and effectiveness throughout the process.

Normalization is used to eliminate partial and transitive dependencies in database architecture. This technique restructures the schema by splitting down huge tables into smaller, more logically ordered ones. The database improves data integrity and efficiency by removing redundancy and adhering to normalization forms like the Third Normal Form (3NF). The resulting structure provides accurate representation of entity relationships, reduces anomalies.

Following the normalization process, which comprises converting entities and attributes from Unnormalized Form (UNF) to Third Normal Form (3NF), nine tables are generated. These tables are Customer, Customer_Category, Customer_Address, Order, Invoice_Details, Invoice, Product, Vendor and Customer_Orders _Details. The creation of the normalized entity relationship diagram is the final stage in this process.

6. Implementation

Creating and Granting User:

SQL> connect system

Enter password:

Connected.

SQL> create user coursework_1 identified by renish;

User created.

SQL> grant connect, resource to coursework_1;

Grant succeeded.

SQL> connect coursework_1/renish

Connected.

SQL>

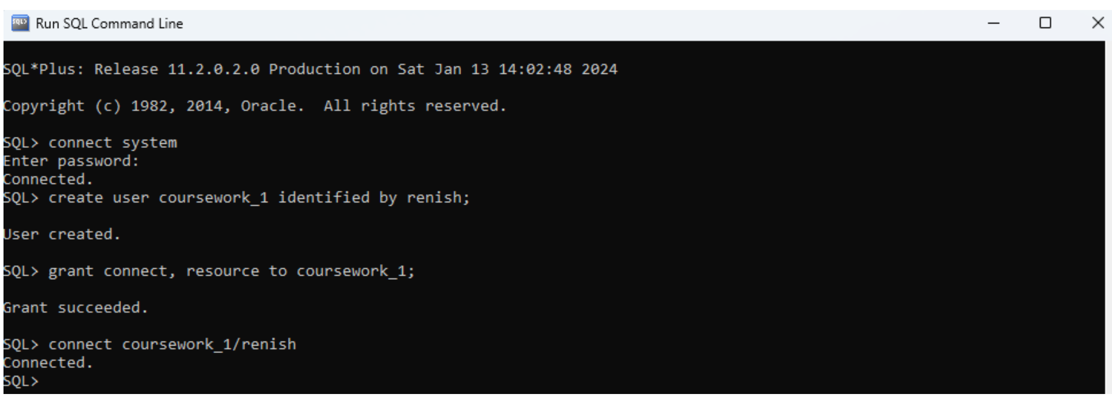


Figure 3:Creating And Granting User.

This Oracle SQL code signs in as the system user, creates a user named "coursework_1" with the password "renish," gives the user the required permissions to login and utilize resources, and then connects to the database using the newly created user's credentials.

Creating Category Table:

SQL> create table Category(

2 Customer_Category_ID int primary key,

3 Customer_Category_Type varchar2(50) not null,

4 Discount_Rate decimal(7,2) not null);

Table created.

SQL> desc Category;

This SQL code generates a table called "Category" with three columns: "Customer_Category_ID" as an integer primary key, "Customer_Category_Type" as a varchar2(50) with no null values, and "Discount_Rate" as a decimal (7,2) with no null values. The table was successfully constructed.

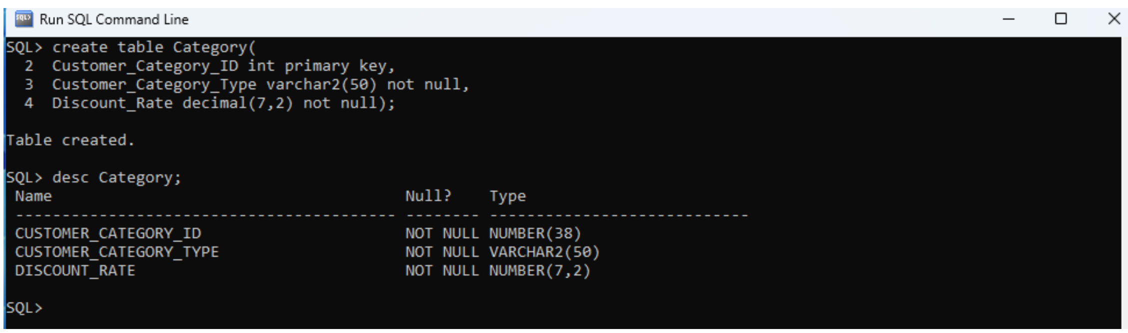

```
Run SQL Command Line
SQL> create table Category(
  2  Customer_Category_ID int primary key,
  3  Customer_Category_Type varchar2(50) not null,
  4  Discount_Rate decimal(7,2) not null);

Table created.

SQL> desc Category;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_CATEGORY_ID                      NOT NULL NUMBER(38)
 CUSTOMER_CATEGORY_TYPE                    NOT NULL VARCHAR2(50)
 DISCOUNT_RATE                             NOT NULL NUMBER(7,2)

SQL>
```

Figure 4: Creating Category Table.

Data implementation of Category Table

SQL> INSERT INTO Category (Customer_Category_ID, Customer_Category_Type, Discount_Rate) VALUES (1, 'Regular', 0);

1 row created.

SQL> INSERT INTO Category (Customer_Category_ID, Customer_Category_Type, Discount_Rate) VALUES (2, 'Staff', 5);

1 row created.

SQL> INSERT INTO Category (Customer_Category_ID Customer_Category_Type, Discount_Rate)VALUES (3, 'VIP', 10);

1 row created.

The provided SQL code inserts data into the Category table. Three rows have been successfully added One for regulars with a discount of 0, and one for staff with a discount of 5 and VIP with a discount of 10.

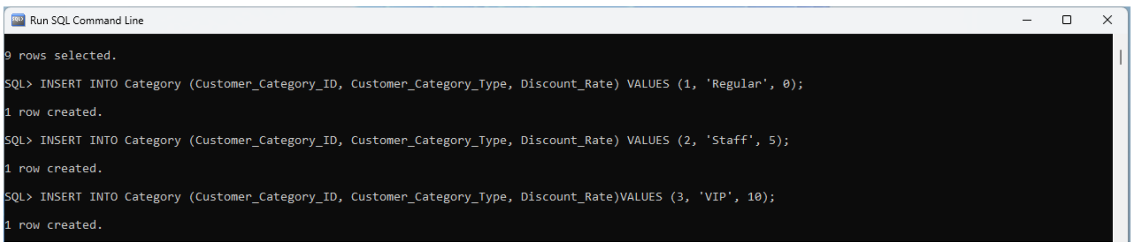


Figure 5: Data Implementation of Category Table.

SQL> set linesize 1000;

SQL> select * from Category;

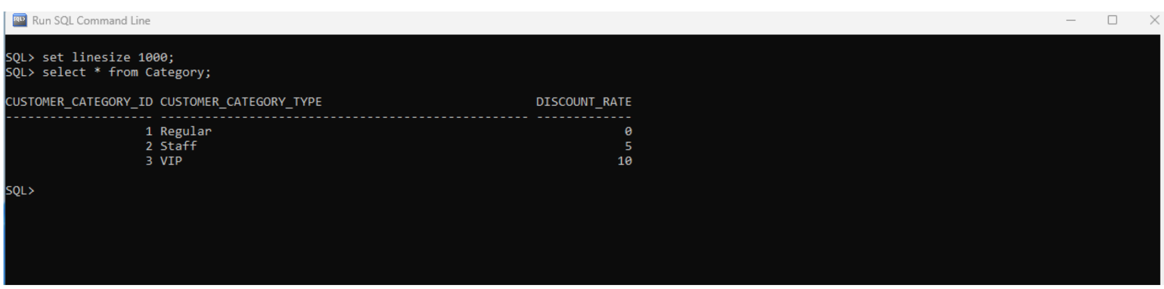


Figure 6: Show Stored Values of Category.

Creating Customer_Address Table:

SQL> create table Customer_Address(

2 Customer_ZipCode int primary key,

3 Customer_Region varchar2(50) not null,

4 Customer_City varchar2(50) not null);

Table created.

SQL> desc Customer_Address;

The provided SQL script creates a table named "Customer_Address" with three columns. 'Customer_ZipCode' is an integer primary key with unique value, 'Customer_Region' is a non-zero varchar2(50), 'Customer_City' is a non-zero varchar2(50) The DESC description command is used to clarify the structural characteristics of the Customer_Address table.

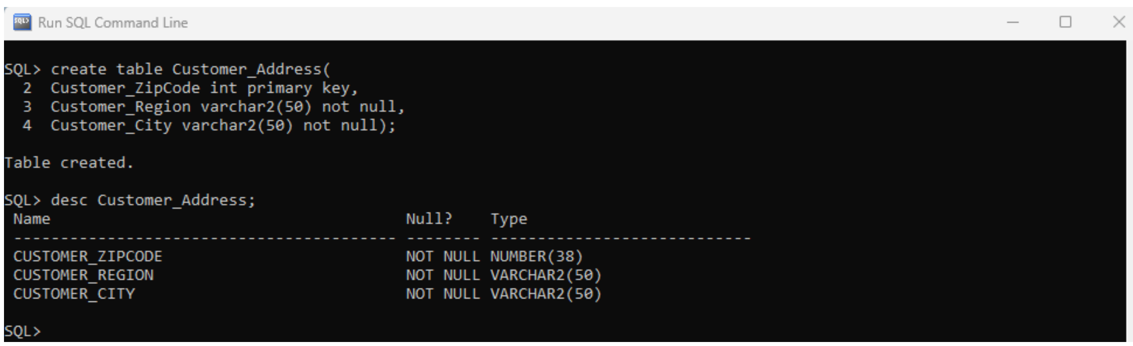


Figure 7:Creating Customer_Address Table.

Data implementation of Cutomer_Address Table

SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (44600, 'Bagmati', 'Kathmandu');

1 row created.

SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (44100, 'Bagmati', 'Hetauda');

1 row created.

SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (44200, 'Bagmati', 'Chitwan');

1 row created.

SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (56100, 'Khoshi', 'Okhaldunga');

1 row created.

SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (45210, 'Bagmati', 'Banepa');

1 row created.

SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (44800, 'Bagmati', 'Bhaktapur');

1 row created.

SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (44700, 'Bagmati', 'Lalitpur');

1 row created.

SQL>

The SQL statement populates the Customer_Address table. Seven records are introduced, each specifying a customer's address and containing Customer_ZipCode, Customer_Region, and Customer_City details. Provided include addresses from various cities such as Kathmandu, Hetauda, Chitwan, Okhaldhunga, Banepa, Bhaktapur, and Lalitpur, with each address falling within the specified region and corresponding postal code.

```
Run SQL Command Line
SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (44600, 'Bagmati', 'Kathmandu');
1 row created.
SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (44100, 'Bagmati', 'Hetauda');
1 row created.
SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (44200, 'Bagmati', 'Chitwan');
1 row created.
SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (56100, 'Khoshi', 'Okhaldunga');
1 row created.
SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (45210, 'Bagmati', 'Banepa');
1 row created.
SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (44800, 'Bagmati', 'Bhaktapur');
1 row created.
SQL> INSERT INTO Customer_Address (Customer_ZipCode, Customer_Region, Customer_City) VALUES (44700, 'Bagmati', 'Lalitpur');
1 row created.
SQL>
```

Figure 8:Data implementation of Cutomer_Address Table.

select * from Customer_Address;

Show the details of Customer_Address which we had insert.

```
Run SQL Command Line
SQL> select * from Customer_Address;

CUSTOMER_ZIPCODE CUSTOMER_REGION                                    CUSTOMER_CITY
---------------- -------------------------------------------------- --------------------------------------------------
           44600 Bagmati                                            Kathmandu
           44100 Bagmati                                            Hetauda
           44200 Bagmati                                            Chitwan
           56100 Khoshi                                             Okhaldunga
           45210 Bagmati                                            Banepa
           44800 Bagmati                                            Bhaktapur
           44700 Bagmati                                            Lalitpur

7 rows selected.

SQL>
```

Figure 9: Show Stored Values of Customer_Address.

Creating Customer Table:

```
SQL> Create table Customer(
  2  Customer_ID int primary key,
  3  Customer_Name varchar2(50) not null,
  4  Customer_phone varchar2(50) not null,
  5  Customer_Sex varchar2(50) not null,
  6  Order_Status varchar2(5) not null,
```

7 Customer_ZipCode int not null,

8 Customer_Category_ID int not null,

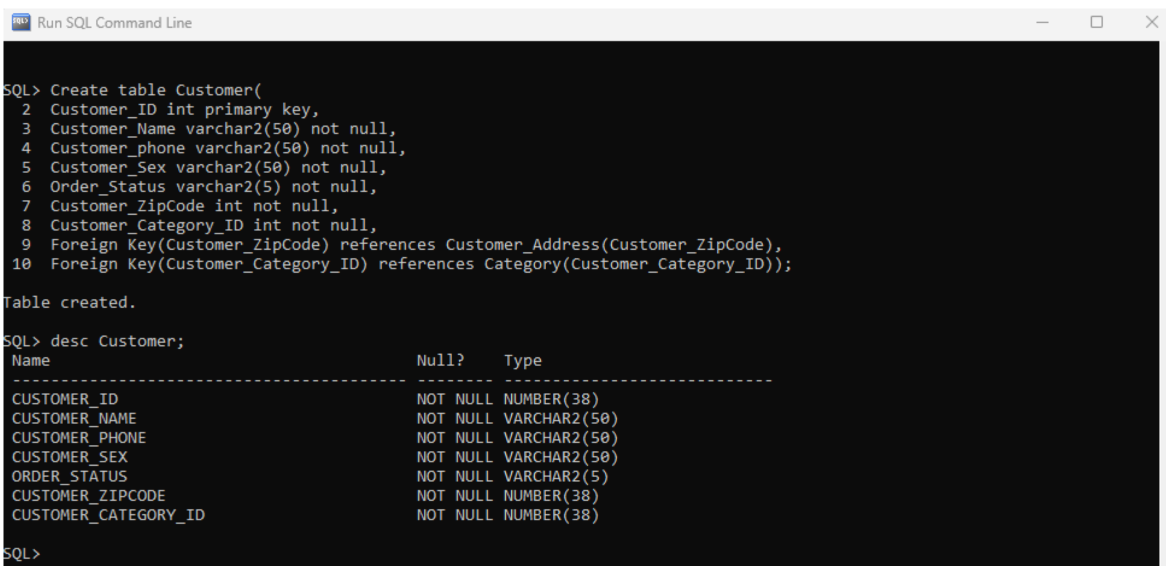
9 Foreign Key(Customer_ZipCode) Customer_Address(Customer_ZipCode),

10 Foreign Key(Customer_Category_ID) Category(Customer_Category_ID));

Table created.

SQL> desc Customer;

The SQL code creates a table named Customer with columns such as customer ID, name, phone number, gender, order status, zip code, and category ID. Two foreign keys link the Customer_Address and Category tables. Table created successfully. Use "DESC Customer" to describe the structure.



```
SQL> Create table Customer(
  2  Customer_ID int primary key,
  3  Customer_Name varchar2(50) not null,
  4  Customer_phone varchar2(50) not null,
  5  Customer_Sex varchar2(50) not null,
  6  Order_Status varchar2(5) not null,
  7  Customer_ZipCode int not null,
  8  Customer_Category_ID int not null,
  9  Foreign Key(Customer_ZipCode) references Customer_Address(Customer_ZipCode),
 10  Foreign Key(Customer_Category_ID) references Category(Customer_Category_ID));

Table created.

SQL> desc Customer;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_ID                               NOT NULL NUMBER(38)
 CUSTOMER_NAME                             NOT NULL VARCHAR2(50)
 CUSTOMER_PHONE                            NOT NULL VARCHAR2(50)
 CUSTOMER_SEX                              NOT NULL VARCHAR2(50)
 ORDER_STATUS                              NOT NULL VARCHAR2(5)
 CUSTOMER_ZIPCODE                          NOT NULL NUMBER(38)
 CUSTOMER_CATEGORY_ID                      NOT NULL NUMBER(38)

SQL>
```

Figure 10:Creating Customer Table.

Data implementation of Customer

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID)VALUES (101, 'Renish Khadka', '9845074066', 'male', 'TRUE', 44600, 1);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (102, 'Prajina Bisural', '9865349287', 'female', 'TRUE', 44200, 2);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (103, 'Prajwal Bisural', '986511548', 'male', 'TRUE', 44200, 3);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (104, 'Prabhab Khanal', '9843347222', 'male', 'TRUE', 44600, 2);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (105, 'Alin Basnet', '9801203873', 'male', 'TRUE', 56100, 3);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (106, 'Indu Khadka', '9845074066', 'female', 'TRUE', 44100, 1);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (107, 'Rajendra Khadka', '9869274740', 'male', 'TRUE', 44100, 3);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (108, 'Sushant Barayal', '9861674532', 'male', 'TRUE', 45210, 2);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (109, 'Nirajan Bogati', '9704502233', 'male', 'TRUE', 44800, 1);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (110, 'Aman Khadka', '9845073077', 'male', 'FALSE', 44200, 2);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (111, 'Ayush Moktan', '9849377047', 'male', 'FALSE', 44800, 1);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (112, 'Pranish Maharjan', '9761622122', 'male', 'FALSE', 44600, 2);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (113, 'Aita Lama', '9744248221', 'female', 'FALSE', 44100, 3);

1 row created.

SQL> INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Phone, Customer_Sex, Order_Status, Customer_ZipCode, Customer_Category_ID) VALUES (114, 'Anjan Roka', '9848666488', 'male', 'FALSE', 44100, 1);

1 row created.

SQL>

The provided SQL command inserts data into the Customer table.14 records are added, each representing a customer with details such as her ID, name, phone number, gender, order status, postal code, category ID, etc. Examples include customers and related information such as "Renish Khadka", "Prajina Bisural", "Aman Khadka", etc.

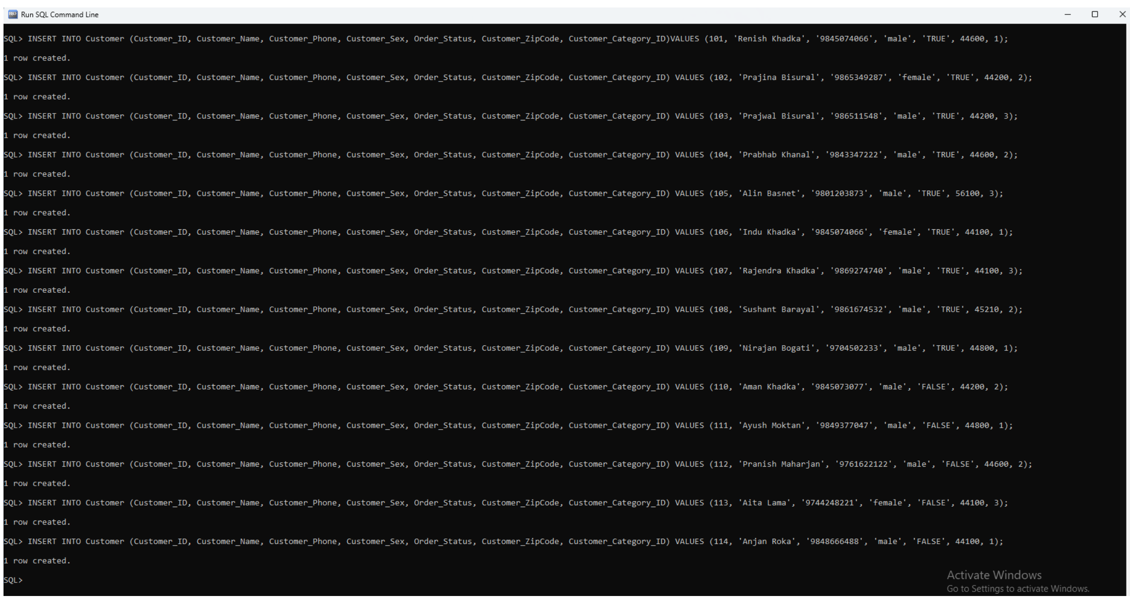


Figure 11:Data implementation of Customer.

SQL> set linesize 4500;

SQL> select * from Customer;

Show the details of all customers which we had insert above.

```
Run SQL Command Line
SQL> set linesize 4500;
SQL> select * from Customer;

CUSTOMER_ID CUSTOMER_NAME        CUSTOMER_PHONE  CUSTOMER_SEX  ORDER CUSTOMER_ZIPCODE CUSTOMER_CATEGORY_ID
----------- -------------------- --------------- ------------- ----- ---------------- --------------------
        101 Renish Khadka        9845074066      male          TRUE             44600                    1
        102 Prajina Bisural      9865349287      female        TRUE             44200                    2
        103 Prajwal Bisural      986511548       male          TRUE             44200                    3
        104 Prabhab Khanal       9843347222      male          TRUE             44600                    2
        105 Alin Basnet          9801203873      male          TRUE             56100                    3
        106 Indu Khadka          9845074066      female        TRUE             44100                    1
        107 Rajendra Khadka      9869274740      male          TRUE             44100                    3
        108 Sushant Barayal      9861674532      male          TRUE             45210                    2
        109 Nirajan Bogati       9704502233      male          TRUE             44800                    1
        110 Aman Khadka          9845073077      male          FALSE            44200                    2
        111 Ayush Moktan         9849377047      male          FALSE            44800                    1

CUSTOMER_ID CUSTOMER_NAME        CUSTOMER_PHONE  CUSTOMER_SEX  ORDER CUSTOMER_ZIPCODE CUSTOMER_CATEGORY_ID
----------- -------------------- --------------- ------------- ----- ---------------- --------------------
        112 Pranish Maharjan     9761622122      male          FALSE            44600                    2
        113 Aita Lama            9744248221      female        FALSE            44100                    3
        114 Anjan Roka           9848666488      male          FALSE            44100                    1

14 rows selected.

SQL>
```

Figure 12:Show stored values of Customer.

Creating Order Table:

SQL> create table Orders(

2 Order_ID int primary key,

3 Order_Date DATE not null,

4 Quantity int not null);

Table created.

SQL> desc Orders;

The SQL code creates a table called Orders with three columns: Order_ID as an integer primary key, Order_Date as a non-nullable DATE type, and quantity as a non-nullable integer. The "DESC Orders" command describes the structure of the "Orders" table.

```
Run SQL Command Line
SQL> create table Orders(
  2  Order_ID int primary key,
  3  Order_Date DATE not null,
  4  Quantity int not null);

Table created.

SQL> desc Orders;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORDER_ID                                  NOT NULL NUMBER(38)
 ORDER_DATE                                NOT NULL DATE
 QUANTITY                                  NOT NULL NUMBER(38)

SQL>
```

Figure 13:Creating Order Table.

Data implementation of Orders

SQL> INSERT INTO Orders (Order_ID, Order_Date, Quantity) VALUES (201, TO_DATE('10/05/2023', 'DD/MM/YYYY'), 4);

1 row created.

SQL> INSERT INTO Orders (Order_ID, Order_Date, Quantity) VALUES (202, TO_DATE('11/05/2023', 'DD/MM/YYYY'), 3);

1 row created.

SQL> INSERT INTO Orders (Order_ID, Order_Date, Quantity) VALUES (203, TO_DATE('16/05/2023', 'DD/MM/YYYY'), 2);

1 row created.

SQL> INSERT INTO Orders (Order_ID, Order_Date, Quantity) VALUES (204, TO_DATE('23/05/2023', 'DD/MM/YYYY'), 1);

1 row created.

SQL> INSERT INTO Orders (Order_ID, Order_Date, Quantity) VALUES (205, TO_DATE('12/06/2023', 'DD/MM/YYYY'), 4);

1 row created.

SQL> INSERT INTO Orders (Order_ID, Order_Date, Quantity) VALUES (206, TO_DATE('12/07/2023', 'DD/MM/YYYY'), 4);

1 row created.

SQL> INSERT INTO Orders (Order_ID, Order_Date, Quantity) VALUES (207, TO_DATE('07/08/2023', 'DD/MM/YYYY'), 5);

1 row created.

SQL> INSERT INTO Orders (Order_ID, Order_Date, Quantity) VALUES (208, TO_DATE('20/08/2023', 'DD/MM/YYYY'), 6);

1 row created.

SQL> INSERT INTO Orders (Order_ID, Order_Date, Quantity) VALUES (209, TO_DATE('28/08/2023', 'DD/MM/YYYY'), 7);

1 row created.

SQL>

The SQL command inserts data into the Orders table and adds nine records with details such as Order_ID, Order_Date, and Quantity. Dates are formatted using the TO_DATE function in dd/mm/yyyy format. An example of this is orders for quantities corresponding to different dates.

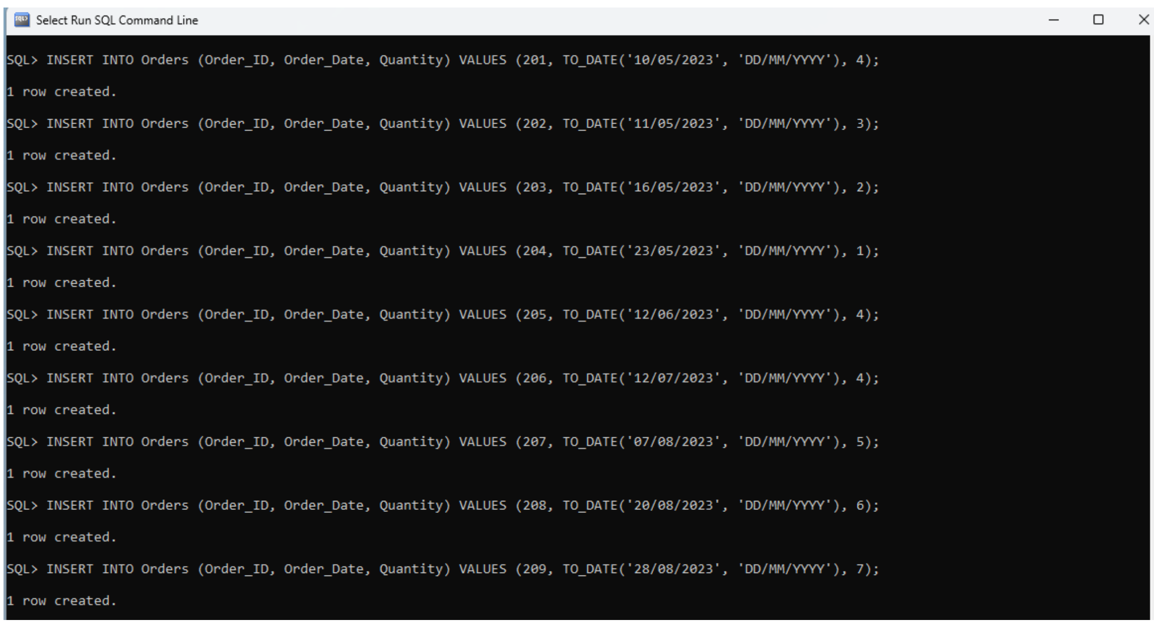


Figure 14:Data implementation of Orders.

SQL> select * from Orders;

To show the all details of Orders which we had insert above.

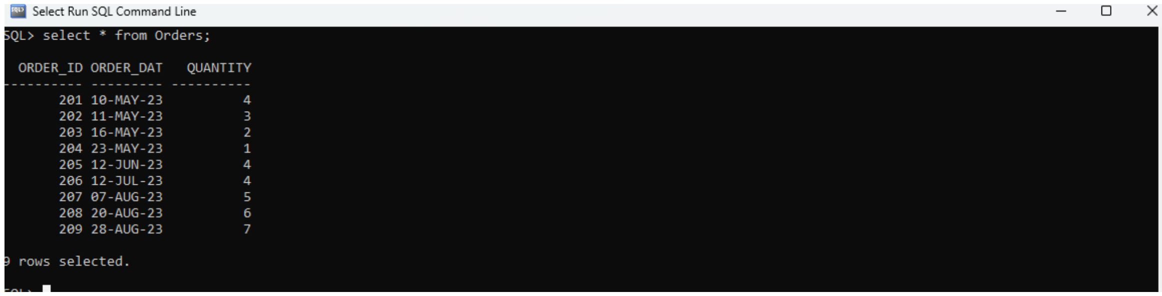


Figure 15: Shows store values of Orders.

Creating Invoice Table:

SQL> create table Invoice(

2 Invoice_ID int primary key,

3 Invoice_Date DATE not null,

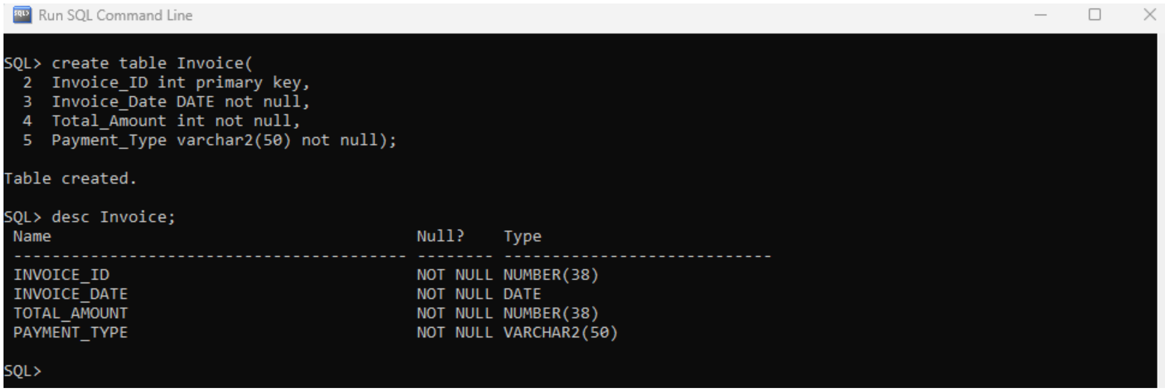
4 Total_Amount int not null,

5 Payment_Type varchar2(50) not null);

Table created.

SQL> desc Invoice;

The SQL code is called "Invoice" with four columns: "Invoice_ID" as an integer primary key, "Invoice_Date" as a non-nullable DATE type, "Total_Amount" as a non-nullable integer, and "Payment_Type". Create a table of names.Used as varchar2(50) and does not allow null values.



```
SQL> create table Invoice(
  2  Invoice_ID int primary key,
  3  Invoice_Date DATE not null,
  4  Total_Amount int not null,
  5  Payment_Type varchar2(50) not null);

Table created.

SQL> desc Invoice;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 INVOICE_ID                                NOT NULL NUMBER(38)
 INVOICE_DATE                              NOT NULL DATE
 TOTAL_AMOUNT                              NOT NULL NUMBER(38)
 PAYMENT_TYPE                              NOT NULL VARCHAR2(50)

SQL>
```

Figure 16:Creating Invoice Table.

Data implementation of Invoice

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (301, 464000, 'cash', TO_DATE('10/05/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (302, 330000, 'card', TO_DATE('11/05/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (303, 64000, 'e-wallet', TO_DATE('16/05/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (304, 70000, 'card', TO_DATE('23/05/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (305, 78000, 'cash', TO_DATE('12/06/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (306, 24000, 'cash', TO_DATE('12/07/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (307, 77500, 'card', TO_DATE('07/08/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (308, 66000, 'e-wallet', TO_DATE('20/08/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (309, 13900, 'card', TO_DATE('28/08/2024', 'DD/MM/YYYY'));

1 row created.

SQL>

The SQL command inserts data into the Invoice table and creates nine records with details such as Invoice_ID, Total_Amount, Payment_Type, and Invoice_Date. Dates are formatted using the TO_DATE function in dd/mm/yyyy format. An example of this is an invoice with different payment methods and corresponding total amounts.

```
Run SQL Command Line
SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (301, 464000, 'cash', TO_DATE('10/05/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (302, 330000, 'card', TO_DATE('11/05/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (303, 64000, 'e-wallet', TO_DATE('16/05/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (304, 70000, 'card', TO_DATE('23/05/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (305, 78000, 'cash', TO_DATE('12/06/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (306, 24000, 'cash', TO_DATE('12/07/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (307, 77500, 'card', TO_DATE('07/08/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (308, 66000, 'e-wallet', TO_DATE('20/08/2024', 'DD/MM/YYYY'));

1 row created.

SQL> INSERT INTO Invoice (Invoice_ID, Total_Amount, Payment_Type, Invoice_Date) VALUES (309, 13900, 'card', TO_DATE('28/08/2024', 'DD/MM/YYYY'));

1 row created.

SQL>
```

Figure 17:Data implementation of Invoice.

SQL> select * from Invoice;

Show all the store data which had insert above.

```
Run SQL Command Line
SQL> select * from Invoice;

INVOICE_ID INVOICE_D TOTAL_AMOUNT PAYMENT_TYPE
---------- --------- ------------ --------------------------------------------------
       301 10-MAY-24       464000 cash
       302 11-MAY-24       330000 card
       303 16-MAY-24        64000 e-wallet
       304 23-MAY-24        70000 card
       305 12-JUN-24        78000 cash
       306 12-JUL-24        24000 cash
       307 07-AUG-24        77500 card
       308 20-AUG-24        66000 e-wallet
       309 28-AUG-24        13900 card

9 rows selected.

SQL>
```

Figure 18:Shows store values of Invoice;

Creating Invoice_Details Table:

SQL> create table Invoice_Details(

2 Order_ID int not null,

3 Customer_ID int not null,

4 Invoice_ID int not null,

5 Foreign Key(Order_ID) references Orders(Order_ID),

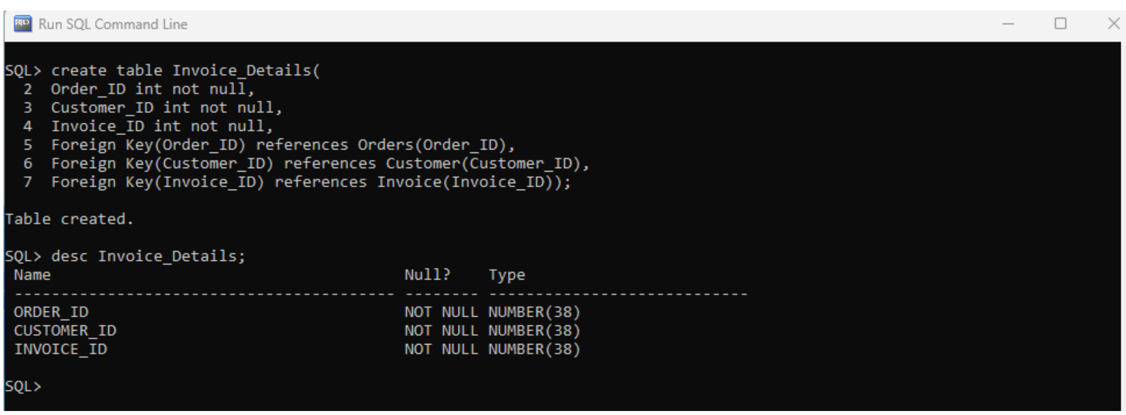
6 Foreign Key(Customer_ID) references Customer(Customer_ID),

7 Foreign Key(Invoice_ID) references Invoice(Invoice_ID));

Table created.

SQL> desc Invoice_Details;

The SQL code creates a table called Invoice_Details with three columns: Order_ID, Customer_ID, and Invoice_ID. Foreign key constraints are added to reference the Orders, Customers, and Invoice tables. The table has been created successfully and its structure can be described in "DESC Invoice_Details".



```
Run SQL Command Line

SQL> create table Invoice_Details(
  2  Order_ID int not null,
  3  Customer_ID int not null,
  4  Invoice_ID int not null,
  5  Foreign Key(Order_ID) references Orders(Order_ID),
  6  Foreign Key(Customer_ID) references Customer(Customer_ID),
  7  Foreign Key(Invoice_ID) references Invoice(Invoice_ID));

Table created.

SQL> desc Invoice_Details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORDER_ID                                  NOT NULL NUMBER(38)
 CUSTOMER_ID                               NOT NULL NUMBER(38)
 INVOICE_ID                                NOT NULL NUMBER(38)

SQL>
```

Figure 19:Creating Invoice_Details Table.

Data implementation of Invoice_Details Table

SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (201, 101, 301);

1 row created.

SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (202, 102, 302);

1 row created.

SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (203, 103, 303);

1 row created.

SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (204, 104, 304);

1 row created.

SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (205, 105, 305);

1 row created.

SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (206, 106, 306);

1 row created.

SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (207, 107, 307);

1 row created.

SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (208, 108, 308);

1 row created.

SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (209, 109, 309);

1 row created.

SQL>

The SQL command inserts data into the Invoice_Details table and adds nine records that establish relationships between orders, customers, and invoices using their respective IDs.

```
Run SQL Command Line
SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (201, 101, 301);
1 row created.
SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (202, 102, 302);
1 row created.
SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (203, 103, 303);
1 row created.
SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (204, 104, 304);
1 row created.
SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (205, 105, 305);
1 row created.
SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (206, 106, 306);
1 row created.
SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (207, 107, 307);
1 row created.
SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (208, 108, 308);
1 row created.
SQL> INSERT INTO Invoice_Details (Order_ID, Customer_ID, Invoice_ID) VALUES (209, 109, 309);
1 row created.
SQL>
```

Figure 20:Data implementation of Invoice_Details Table.

SQL> select * from Invoice_Details

Show all the values which had been insert from above.

```
Run SQL Command Line
SQL> select * from Invoice_Details;

  ORDER_ID CUSTOMER_ID INVOICE_ID
---------- ----------- ----------
       201         101        301
       202         102        302
       203         103        303
       204         104        304
       205         105        305
       206         106        306
       207         107        307
       208         108        308
       209         109        309

9 rows selected.

SQL>
```

Figure 21: Shows store values of Invoice_Details.

Creating Vendor Table:

SQL> create table Vendor(

2 Vendor_ID int primary key,

3 Vendor_Name varchar2(50) not null,

4 Vendor_Address varchar2(50) not null);

Table created.

SQL> desc Vendor;

The SQL code generates a table called "Vendor" with three columns: "Vendor_ID" as an integer primary key, "Vendor_Name" as a varchar2(50) that does not accept null values, and "Vendor_Address" as a varchar2(50) that does not accept null values. You can explain its structure with "DESC Vendor."

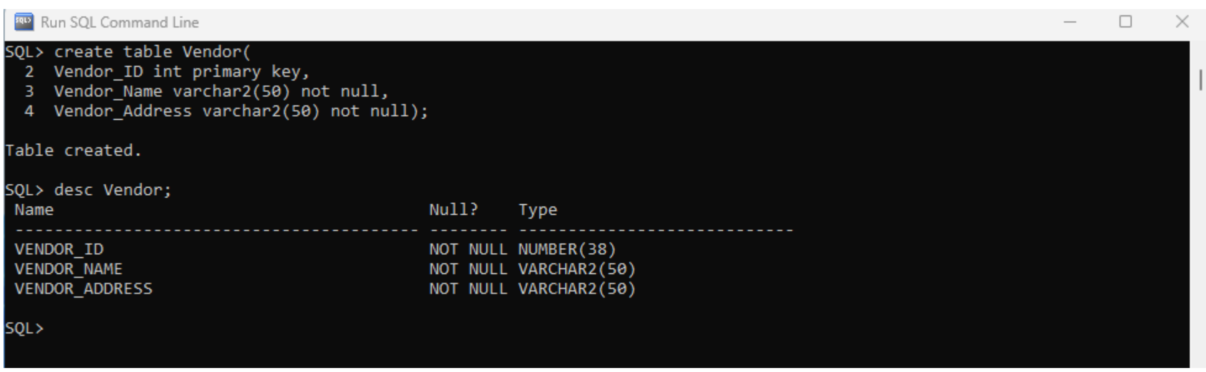


Figure 22:Creating Vendor Table.

Data implementation of Vendor Table

SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (401, 'EvoStore', 'Kathmandu');

1 row created.

SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (402, 'Oliz', 'Hetauda');

1 row created.

SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (403, 'Hukut', 'Chitwan');

1 row created.

SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (404, 'ITTI', 'Okhaldunga');

1 row created.

SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (405, 'Sony', 'Banepa');

1 row created.

SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (406, 'CG', 'Bhaktapur');

1 row created.

SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (407, 'BajajElectronics', 'Lalitpur');

1 row created.

SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (408, 'GigaNepal', 'Kathmandu');

1 row created.

SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (409, 'HimalayanSolution', 'Hetauda');

1 row created.

SQL>

The SQL command inserts data into the Vendor table and creates nine records with details such as Vendor_ID, Vendor_Name, and Vendor_Address. Examples of this are providers such as ``EvoStore,'' ``Oliz,'' and ``Hukut,'' which have addresses associated with different locations.

```
Run SQL Command Line
SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (401, 'EvoStore', 'Kathmandu');
1 row created.
SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (402, 'Oliz', 'Hetauda');
1 row created.
SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (403, 'Hukut', 'Chitwan');
1 row created.
SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (404, 'ITTI', 'Okhaldunga');
1 row created.
SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (405, 'Sony', 'Banepa');
1 row created.
SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (406, 'CG', 'Bhaktapur');
1 row created.
SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (407, 'BajajElectronics', 'Lalitpur');
1 row created.
SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (408, 'GigaNepal', 'Kathmandu');
1 row created.
SQL> INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address) VALUES (409, 'HimalayanSolution', 'Hetauda');
1 row created.
SQL>
```

Figure 23:Data implementation of Vendor Table.

SQL> select * from Vendor;

Show all the details of vendor which insert from above.

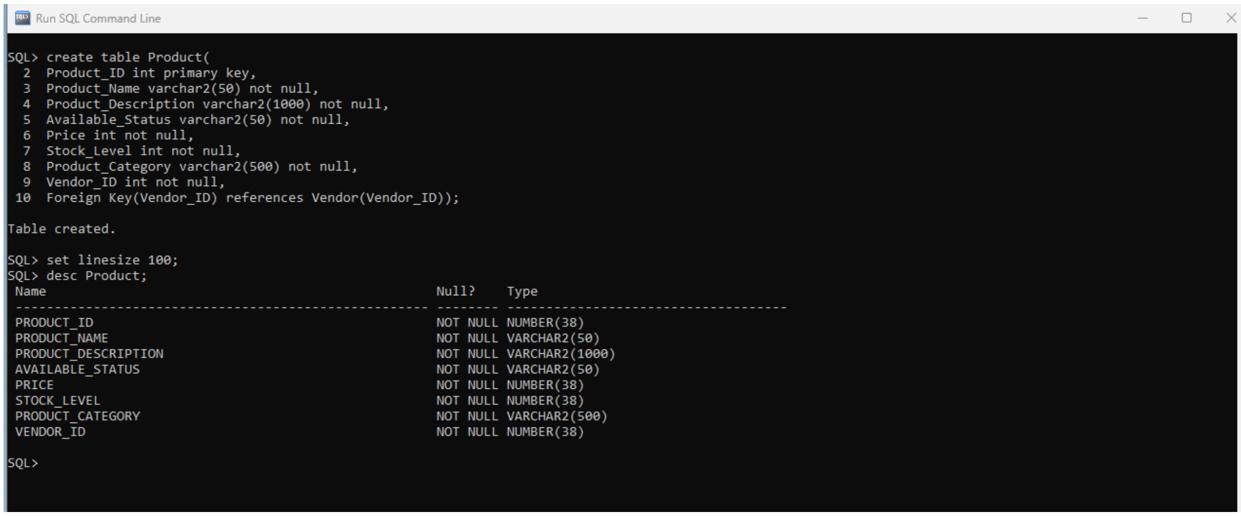
```
Run SQL Command Line
SQL> select * from Vendor;

 VENDOR_ID VENDOR_NAME                                        VENDOR_ADDRESS
---------- -------------------------------------------------- --------------------------------------------------
       401 EvoStore                                           Kathmandu
       402 Oliz                                               Hetauda
       403 Hukut                                              Chitwan
       404 ITTI                                               Okhaldunga
       405 Sony                                               Banepa
       406 CG                                                 Bhaktapur
       407 BajajElectronics                                   Lalitpur
       408 GigaNepal                                          Kathmandu
       409 HimalayanSolution                                  Hetauda

9 rows selected.

SQL>
```

Figure 24: Shows store values of Vendor.

Creating Product Table:

```
SQL> create table Product(
  2  Product_ID int primary key,
  3  Product_Name varchar2(50) not null,
  4  Product_Description varchar2(1000) not null,
  5  Available_Status varchar2(50) not null,
  6  Price int not null,
  7  Stock_Level int not null,
  8  Product_Category varchar2(500) not null,
  9  Vendor_ID int not null,
 10  Foreign Key(Vendor_ID) references Vendor(Vendor_ID));

Table created.

SQL> desc Product;
```

The SQL code creates a "Product" table with columns for product details such as ID, name, description, availability, price, inventory, category, and supplier ID. Each product is assigned to a provider via a foreign key reference. The structure can be described by "DESC Product".



```
SQL> create table Product(
  2  Product_ID int primary key,
  3  Product_Name varchar2(50) not null,
  4  Product_Description varchar2(1000) not null,
  5  Available_Status varchar2(50) not null,
  6  Price int not null,
  7  Stock_Level int not null,
  8  Product_Category varchar2(500) not null,
  9  Vendor_ID int not null,
 10  Foreign Key(Vendor_ID) references Vendor(Vendor_ID));

Table created.

SQL> set linesize 100;
SQL> desc Product;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ----------------------------------
 PRODUCT_ID                                            NOT NULL NUMBER(38)
 PRODUCT_NAME                                          NOT NULL VARCHAR2(50)
 PRODUCT_DESCRIPTION                                   NOT NULL VARCHAR2(1000)
 AVAILABLE_STATUS                                      NOT NULL VARCHAR2(50)
 PRICE                                                 NOT NULL NUMBER(38)
 STOCK_LEVEL                                           NOT NULL NUMBER(38)
 PRODUCT_CATEGORY                                      NOT NULL VARCHAR2(500)
 VENDOR_ID                                             NOT NULL NUMBER(38)

SQL>
```

Figure 25:Creating Product Table.

Data implementation of Product Table

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (501, 'MacbookPro', 'Powerful laptop with high-performance features for professional use.', 'True', 200000, 79, 'Laptop', 401);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (502, 'Airpod', 'Wireless earbuds with excellent sound quality and long battery life.', 'True', 32000, 90, 'Earphone', 401);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (503, 'Iphone15', 'Latest iPhone with advanced camera features and powerful performance.', 'True', 150000, 70, 'Phone', 401);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (504, 'Samsung A80', 'Samsung smartphone with a large display and impressive camera capabilities.', 'True', 70000, 80, 'Phone', 402);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (505, 'Oneplus nord', 'Oneplus smartphone known for its sleek design and high-speed performance.', 'True', 70000, 50, 'Phone', 402);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (506, 'Anker Charger', 'Fast-charging power bank with multiple USB ports for on-the-go charging.', 'True', 5000, 80, 'PowerBank', 408);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (507, 'Logictech MX Keyboard', 'Premium mechanical keyboard with customizable RGB lighting.', 'True', 4500, 70, 'Keyboard', 404);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (508, 'Razer Mouse', 'High-performance gaming mouse with customizable buttons and RGB lighting.', 'True', 7000, 60, 'Mouse', 404);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (509, 'LED screen Intec', 'High-resolution LED monitor for immersive viewing experience.', 'True', 22000, 70, 'Monitor', 407);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (510, 'Graphics Card', 'Powerful graphics card for gaming and graphic-intensive tasks.', 'True', 80000, 80, 'ComputerComponents', 404);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (511, 'RAM', 'High-speed RAM module for improved system performance.', 'True', 15000, 50, 'ComputerComponents', 404);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (512, 'Logictech MSI', 'Precision gaming mouse with customizable DPI settings.', 'True', 10000, 50, 'Mouse', 404);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (513, 'PS5', 'Latest gaming console with cutting-edge graphics and gameplay features.', 'True', 90000, 50, 'Console', 408);

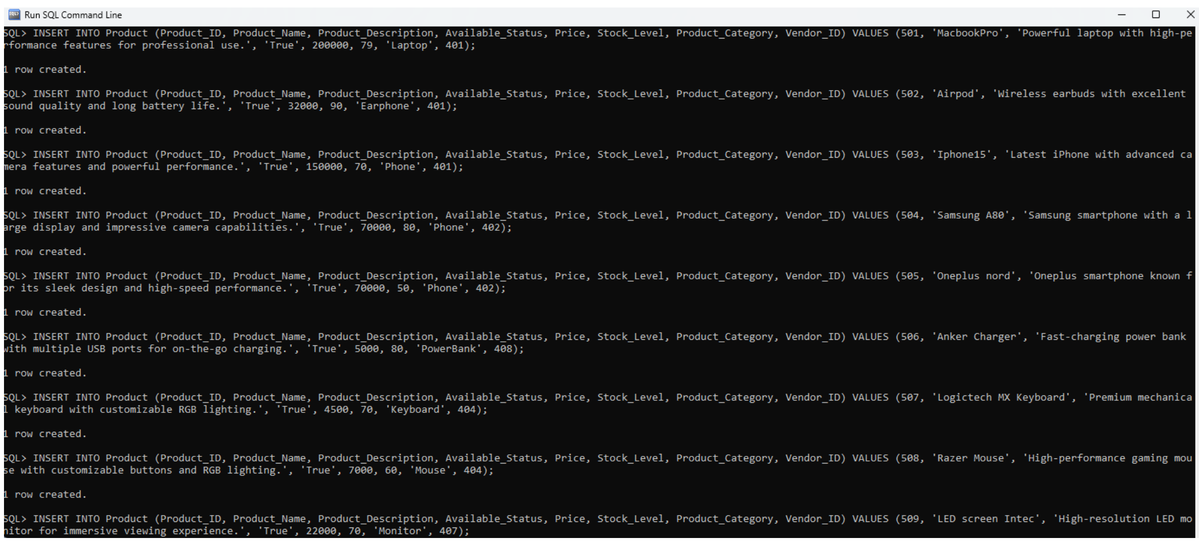
1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (514, 'Router', 'High-performance router for fast and reliable internet connectivity.', 'True', 5000, 70, 'ComputerComponents', 409);

1 row created.

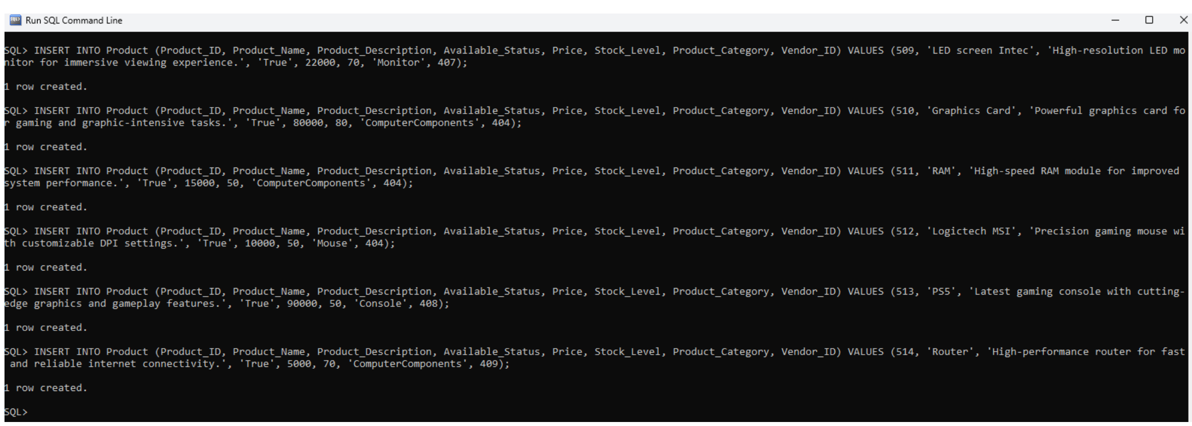
SQL>

The SQL commands insert data into the "Product" table, creating fourteen records with details on various products, including their names, descriptions, availability status, prices, stock levels, categories, and associated vendor IDs.



```
SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (501, 'MacbookPro', 'Powerful laptop with high-performance features for professional use.', 'True', 200000, 79, 'Laptop', 401);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (502, 'Airpod', 'Wireless earbuds with excellent sound quality and long battery life.', 'True', 32000, 90, 'Earphone', 401);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (503, 'Iphone15', 'Latest iPhone with advanced camera features and powerful performance.', 'True', 150000, 70, 'Phone', 401);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (504, 'Samsung A80', 'Samsung smartphone with a large display and impressive camera capabilities.', 'True', 70000, 80, 'Phone', 402);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (505, 'Oneplus nord', 'Oneplus smartphone known for its sleek design and high-speed performance.', 'True', 70000, 50, 'Phone', 402);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (506, 'Anker Charger', 'Fast-charging power bank with multiple USB ports for on-the-go charging.', 'True', 5000, 80, 'PowerBank', 408);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (507, 'Logictech MX Keyboard', 'Premium mechanical keyboard with customizable RGB lighting.', 'True', 4500, 70, 'Keyboard', 404);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (508, 'Razer Mouse', 'High-performance gaming mouse with customizable buttons and RGB lighting.', 'True', 7000, 60, 'Mouse', 404);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (509, 'LED screen Intec', 'High-resolution LED monitor for immersive viewing experience.', 'True', 22000, 70, 'Monitor', 407);
```

Figure 26:Data implementation of Product Table1.

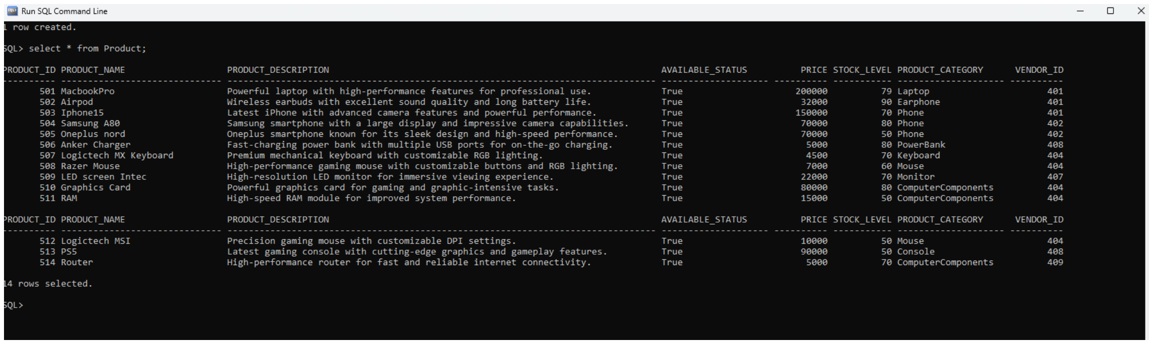


```
SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (509, 'LED screen Intec', 'High-resolution LED monitor for immersive viewing experience.', 'True', 22000, 70, 'Monitor', 407);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (510, 'Graphics Card', 'Powerful graphics card for gaming and graphic-intensive tasks.', 'True', 80000, 80, 'ComputerComponents', 404);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (511, 'RAM', 'High-speed RAM module for improved system performance.', 'True', 15000, 50, 'ComputerComponents', 404);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (512, 'Logictech MSI', 'Precision gaming mouse with customizable DPI settings.', 'True', 10000, 50, 'Mouse', 404);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (513, 'PS5', 'Latest gaming console with cutting-edge graphics and gameplay features.', 'True', 90000, 50, 'Console', 408);

1 row created.

SQL> INSERT INTO Product (Product_ID, Product_Name, Product_Description, Available_Status, Price, Stock_Level, Product_Category, Vendor_ID) VALUES (514, 'Router', 'High-performance router for fast and reliable internet connectivity.', 'True', 5000, 70, 'ComputerComponents', 409);

1 row created.

SQL>
```

Figure 27:Data implementation of Product Table2.

SQL> select * from Product;

Show all the details which had insert in sql.


Run SQL Command Line

1 row created.

SQL> select * from Product;

PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION	AVAILABLE_STATUS	PRICE	STOCK_LEVEL	PRODUCT_CATEGORY	VENDOR_ID
501	MacbookPro	Powerful laptop with high-performance features for professional use.	True	200000	79	Laptop	401
502	Airpod	Wireless earbuds with excellent sound quality and long battery life.	True	32000	90	Earphone	401
503	Iphone15	Latest iPhone with advanced camera features and powerful performance.	True	150000	70	Phone	401
504	Samsung A80	Samsung smartphone with a large display and impressive camera capabilities.	True	70000	80	Phone	402
505	Oneplus nord	Oneplus smartphone known for its sleek design and high-speed performance.	True	70000	50	Phone	402
506	Anker Charger	Fast-charging power bank with multiple USB ports for on-the-go charging.	True	5000	80	PowerBank	408
507	Logictech MX Keyboard	Premium mechanical keyboard with customizable RGB lighting.	True	4500	70	Keyboard	404
508	Razer Mouse	High-performance gaming mouse with customizable buttons and RGB lighting.	True	7000	60	Mouse	404
509	LED screen Intec	High-resolution LED monitor for immersive viewing experience.	True	22000	70	Monitor	407
510	Graphics Card	Powerful graphics card for gaming and graphic-intensive tasks.	True	80000	80	ComputerComponents	404
511	RAM	High-speed RAM module for improved system performance.	True	15000	50	ComputerComponents	404
512	Logictech MSI	Precision gaming mouse with customizable DPI settings.	True	10000	50	Mouse	404
513	PS5	Latest gaming console with cutting-edge graphics and gameplay features.	True	90000	50	Console	408
514	Router	High-performance router for fast and reliable internet connectivity.	True	5000	70	ComputerComponents	409

14 rows selected.

SQL>

Figure 28: Shows store values of Product.

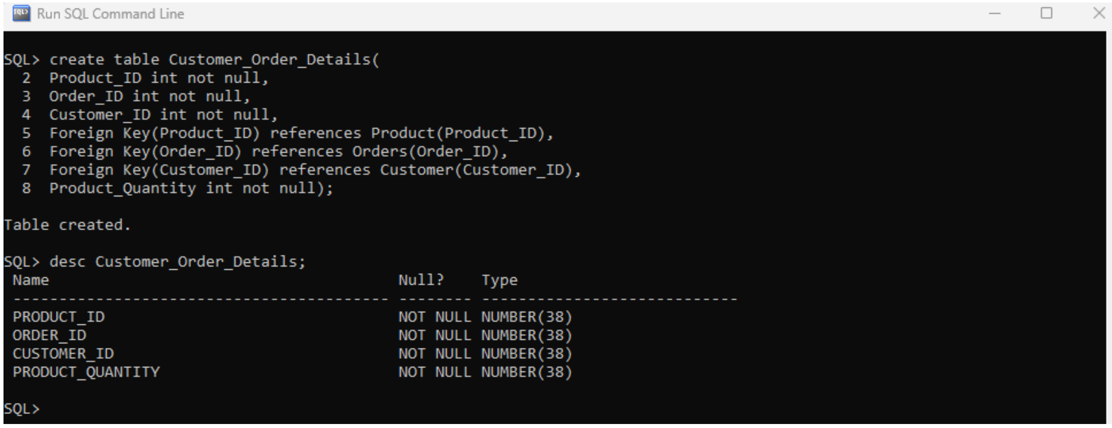
Creating Customer_Order_Details Table:

```
SQL> create table Customer_Order_Details(
  2  Product_ID int not null,
  3  Order_ID int not null,
  4  Customer_ID int not null,
  5  Foreign Key(Product_ID) references Product(Product_ID),
  6  Foreign Key(Order_ID) references Orders(Order_ID),
  7  Foreign Key(Customer_ID) references Customer(Customer_ID),
  8  Product_Quantity int not null);
```

Table created.

SQL> desc Customer_Order_Details;

The SQL code creates a Customer_Order_Details table with columns for product, order, customer ID, and product quantity. Foreign key constraints are associated with the Product, Orders, and Customer tables.



```
Run SQL Command Line

SQL> create table Customer_Order_Details(
  2  Product_ID int not null,
  3  Order_ID int not null,
  4  Customer_ID int not null,
  5  Foreign Key(Product_ID) references Product(Product_ID),
  6  Foreign Key(Order_ID) references Orders(Order_ID),
  7  Foreign Key(Customer_ID) references Customer(Customer_ID),
  8  Product_Quantity int not null);

Table created.

SQL> desc Customer_Order_Details;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODUCT_ID                                NOT NULL NUMBER(38)
 ORDER_ID                                  NOT NULL NUMBER(38)
 CUSTOMER_ID                               NOT NULL NUMBER(38)
 PRODUCT_QUANTITY                          NOT NULL NUMBER(38)

SQL>
```

Figure 29:Customer_Order_Details Table.

Data implementation of Customer_Order_Details Table

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (501, 201, 101, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (502, 201, 101, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (503, 202, 102, 1);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (513, 202, 102, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (502, 203, 103, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (504, 204, 104, 1);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (502, 205, 105, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (508, 205, 105, 2);

1 row created.

SQL> NSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (508, 206, 106, 2);

SP2-0734: unknown command beginning "NSERT INTO..." - rest of line ignored.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (508, 206, 106, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (506, 206, 106, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (507, 207, 107, 3);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (502, 207, 107, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (508, 208, 108, 3);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (511, 208, 108, 3);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (511, 209, 109, 5);

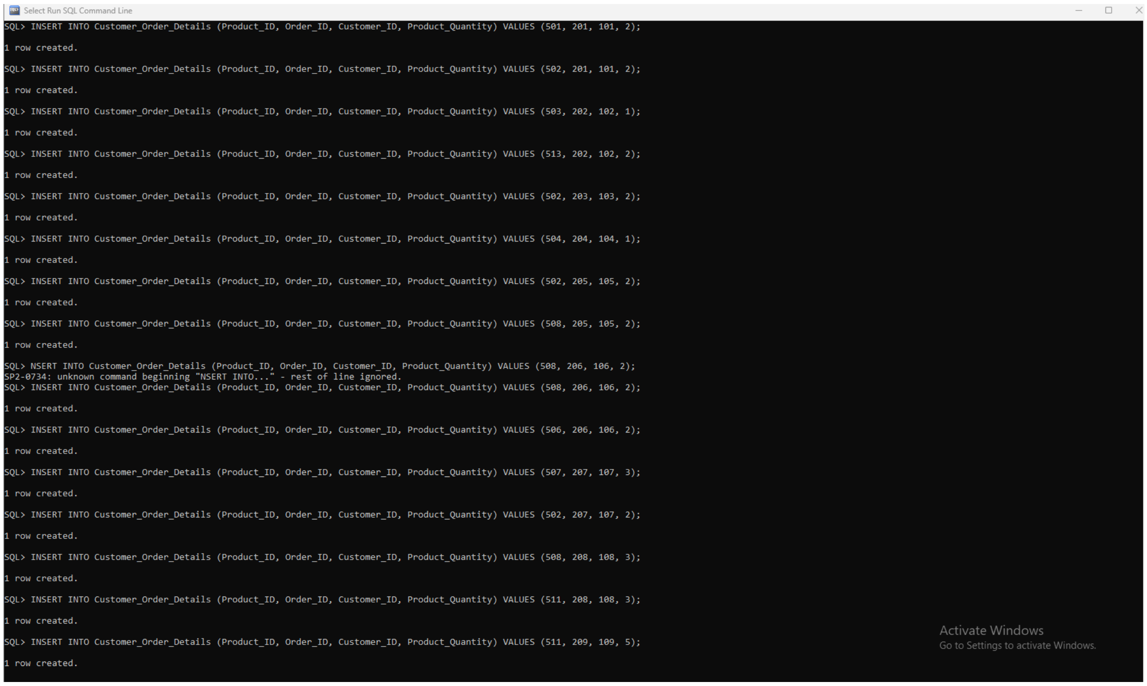
1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (502, 209, 109, 2);

1 row created.

SQL>

The SQL commands insert data into the Customer_Order_Details table and establish relationships between products, orders, and customers. Insert all details in this table.

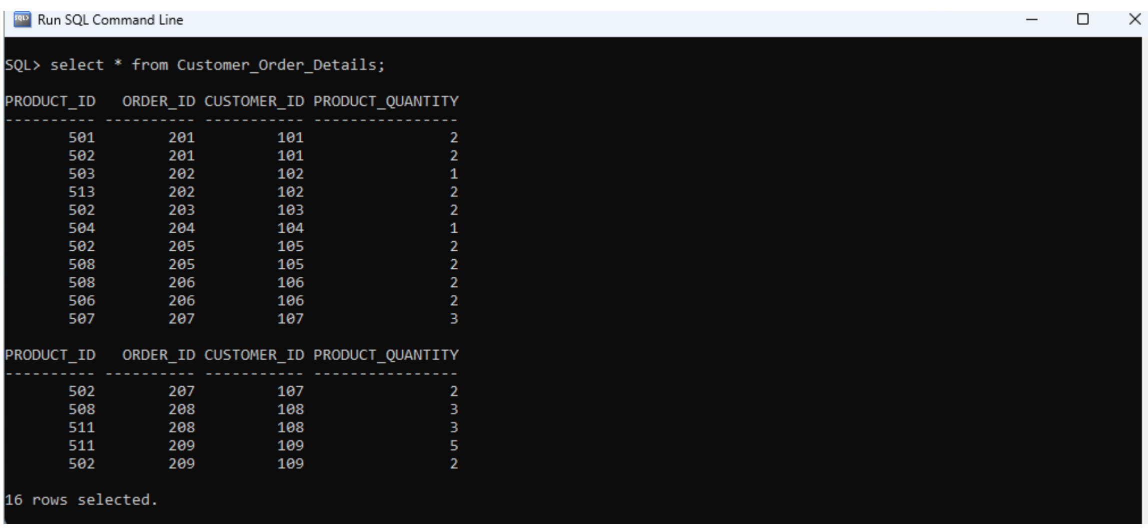


```
Select Run SQL Command Line
SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (501, 201, 101, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (502, 201, 101, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (503, 202, 102, 1);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (513, 202, 102, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (502, 203, 103, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (504, 204, 104, 1);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (502, 205, 105, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (508, 205, 105, 2);

1 row created.

SQL> NSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (508, 206, 106, 2);
SP2-0734: unknown command beginning "NSERT INTO..." - rest of line ignored.
SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (508, 206, 106, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (506, 206, 106, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (507, 207, 107, 3);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (502, 207, 107, 2);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (508, 208, 108, 3);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (511, 208, 108, 3);

1 row created.

SQL> INSERT INTO Customer_Order_Details (Product_ID, Order_ID, Customer_ID, Product_Quantity) VALUES (511, 209, 109, 5);

1 row created.
```


Figure 30:Data implementation of Customer_Order_Details Table.

select * from Customer_Order_Details;

Show all the details of Customer_Order_Details which had been insert from above.



```
Run SQL Command Line
SQL> select * from Customer_Order_Details;

PRODUCT_ID   ORDER_ID CUSTOMER_ID PRODUCT_QUANTITY
---------- ---------- ----------- ----------------
       501        201         101                2
       502        201         101                2
       503        202         102                1
       513        202         102                2
       502        203         103                2
       504        204         104                1
       502        205         105                2
       508        205         105                2
       508        206         106                2
       506        206         106                2
       507        207         107                3

PRODUCT_ID   ORDER_ID CUSTOMER_ID PRODUCT_QUANTITY
---------- ---------- ----------- ----------------
       502        207         107                2
       508        208         108                3
       511        208         108                3
       511        209         109                5
       502        209         109                2

16 rows selected.
```

Figure 31: Shows store values of Customer_Order_Details Table.

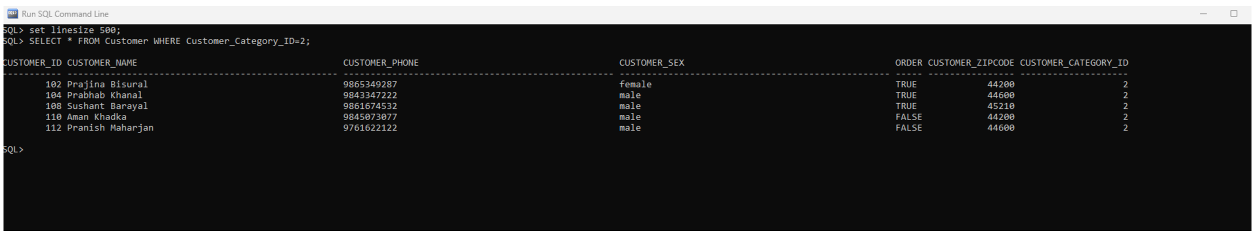
7. Database Querying

7.1 Informational Queries

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

   Query: SELECT * FROM Customer WHERE Customer_Category_ID=2;

   The specified SQL query retrieves all columns from the Customer table where the Customer_Category_ID column has a value of 2.Filter data based on specified criteria.If you want to know how the database executes this query, you can view the execution plan using the EXPLAIN statement.



```
Run SQL Command Line
SQL> set linesize 500;
SQL> SELECT * FROM Customer WHERE Customer_Category_ID=2;

CUSTOMER_ID CUSTOMER_NAME          CUSTOMER_PHONE   CUSTOMER_SEX   ORDER CUSTOMER_ZIPCODE CUSTOMER_CATEGORY_ID
----------- ---------------------- ---------------- -------------- ----- ---------------- --------------------
        102 Prajina Bisural        9865349287       female         TRUE             44200                    2
        104 Prabhab Khanal         9843347222       male           TRUE             44600                    2
        108 Sushant Barayal        9861674532       male           TRUE             45210                    2
        110 Aman Khadka            9845073077       male           FALSE            44200                    2
        112 Pranish Maharjan       9761622122       male           FALSE            44600                    2

SQL>
```

Figure 32:All the customers that are also staff of the company.

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

   Query: SELECT * FROM Orders WHERE Order_Date BETWEEN TO_DATE ('01/05/2023', 'DD/MM/YYYY') AND TO_DATE('28/05/2023', 'DD/MM/YYYY');

   The SQL query retrieves all columns from the Orders table where Order_Date falls within the date range from January 5, 2023 to May 28, 2023. The BETWEEN keyword is used in conjunction with TO_DATE to specify a date range. This query is useful for retrieving orders made during a specific time period.

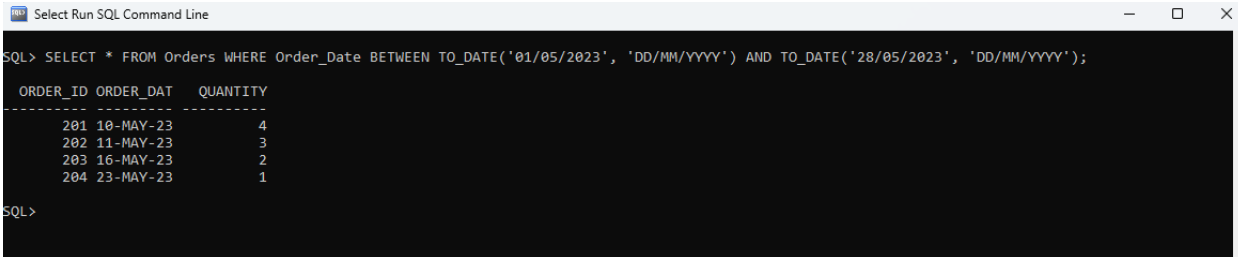
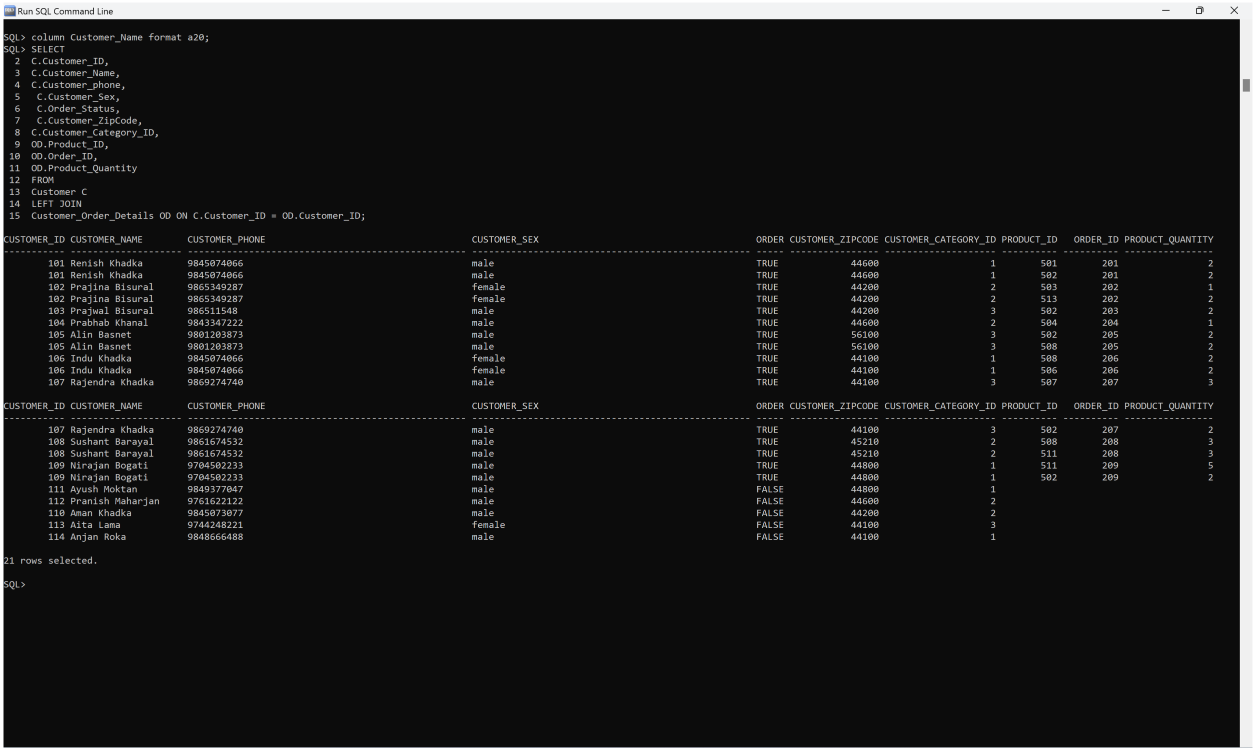


Figure 33:All the orders made for any particular product between the dates 01-05-2023 till 28-05-2023.

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

```
SQL> SELECT
C.Customer_ID,
C.Customer_Name,
C.Customer_phone,
 C.Customer_Sex,
 C.Order_Status,
 C.Customer_ZipCode,
C.Customer_Category_ID,
OD.Product_ID,
OD.Order_ID,
OD.Product_Quantity
FROM
Customer C
LEFT JOIN
Customer_Order_Details OD ON C.Customer_ID = OD.Customer_ID;
```

This SQL query uses a LEFT JOIN between the Customer and Customer_Order_Details tables based on the Customer_ID column to retrieve customer information and related order details.Results include all customers, including those with no orders, and relevant columns such as customer ID, name, phone number, gender, order status, postal code, category ID, product ID, order ID, and product quantity. It shows.



```
Run SQL Command Line

SQL> column Customer_Name format a20;
SQL> SELECT
  2  C.Customer_ID,
  3  C.Customer_Name,
  4  C.Customer_phone,
  5   C.Customer_Sex,
  6   C.Order_Status,
  7   C.Customer_ZipCode,
  8  C.Customer_Category_ID,
  9  OD.Product_ID,
 10  OD.Order_ID,
 11  OD.Product_Quantity
 12  FROM
 13  Customer C
 14  LEFT JOIN
 15  Customer_Order_Details OD ON C.Customer_ID = OD.Customer_ID;
```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_PHONE	CUSTOMER_SEX	ORDER	CUSTOMER_ZIPCODE	CUSTOMER_CATEGORY_ID	PRODUCT_ID	ORDER_ID	PRODUCT_QUANTITY
101	Renish Khadka	9845074066	male	TRUE	44600	1	501	201	2
101	Renish Khadka	9845074066	male	TRUE	44600	1	502	201	2
102	Prajina Bisural	9865349287	female	TRUE	44200	2	503	202	1
102	Prajina Bisural	9865349287	female	TRUE	44200	2	513	202	2
103	Prajwal Bisural	986511548	male	TRUE	44200	3	502	203	2
104	Prabhab Khanal	9843347222	male	TRUE	44600	2	504	204	1
105	Alin Basnet	9801203873	male	TRUE	56100	3	502	205	2
105	Alin Basnet	9801203873	male	TRUE	56100	3	508	205	2
106	Indu Khadka	9845074066	female	TRUE	44100	1	508	206	2
106	Indu Khadka	9845074066	female	TRUE	44100	1	506	206	2
107	Rajendra Khadka	9869274740	male	TRUE	44100	3	507	207	3

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_PHONE	CUSTOMER_SEX	ORDER	CUSTOMER_ZIPCODE	CUSTOMER_CATEGORY_ID	PRODUCT_ID	ORDER_ID	PRODUCT_QUANTITY
107	Rajendra Khadka	9869274740	male	TRUE	44100	3	502	207	2
108	Sushant Barayal	9861674532	male	TRUE	45210	2	508	208	3
108	Sushant Barayal	9861674532	male	TRUE	45210	2	511	208	3
109	Nirajan Bogati	9704502233	male	TRUE	44800	1	511	209	5
109	Nirajan Bogati	9704502233	male	TRUE	44800	1	502	209	2
111	Ayush Moktan	9849377047	male	FALSE	44800	1			
112	Pranish Maharjan	9761622122	male	FALSE	44600	2			
110	Aman Khadka	9845073077	male	FALSE	44200	2			
113	Aita Lama	9744248221	female	FALSE	44100	3			
114	Anjan Roka	9848666488	male	FALSE	44100	1			

```
21 rows selected.

SQL>
```

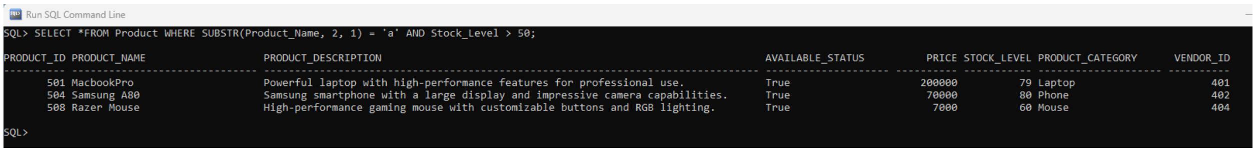
Figure 34:All the customers with their order details and also the customers who have not ordered any products yet.

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

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

   This SQL query retrieves all columns from the Product table where the second character of Product_Name is "a" and Stock_Level is greater than 50.The SUBSTR function is used to extract a substring of "Product_Name".

   This condition ensures that only products that meet both criteria are included in the results. This query helps you find products with specific name and inventory characteristics.



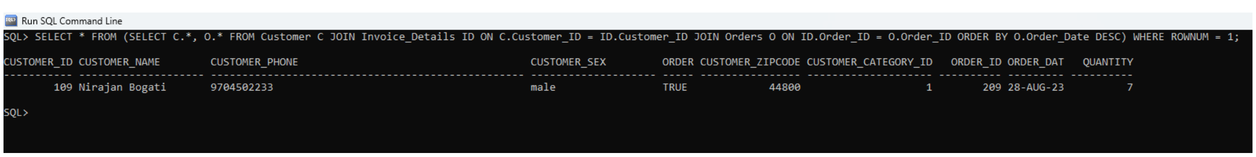
```
Run SQL Command Line
SQL> SELECT *FROM Product WHERE SUBSTR(Product_Name, 2, 1) = 'a' AND Stock_Level > 50;

PRODUCT_ID PRODUCT_NAME                  PRODUCT_DESCRIPTION                                                          AVAILABLE_STATUS          PRICE STOCK_LEVEL PRODUCT_CATEGORY      VENDOR_ID
---------- ----------------------------- ---------------------------------------------------------------------------- -------------------- ---------- ----------- -------------------- ----------
       501 MacbookPro                    Powerful laptop with high-performance features for professional use.         True                     200000          79 Laptop                      401
       504 Samsung A80                   Samsung smartphone with a large display and impressive camera capabilities.  True                      70000          80 Phone                       402
       508 Razer Mouse                   High-performance gaming mouse with customizable buttons and RGB lighting.    True                       7000          60 Mouse                       404

SQL>
```

Figure 35:All product details that have the second letter 'a' in their product name and have a stock quantity more than 50.

5. Find out the customer who has ordered recently.

   Query: SELECT * FROM (SELECT C.*, O.* FROM Customer C JOIN Invoice_Details ID ON C.Customer_ID = ID.Customer_ID JOIN Orders O ON ID.Order_ID = O.Order_ID ORDER BY O.Order_Date DESC) WHERE ROWNUM = 1;

   This SQL query extracts all columns from a derived table in which customer information from the "Customer" table (alias "C") is combined with order details from the "Invoice_Details" and "Orders" tables (aliases "ID" and "O" respectively). The join requirements are based on customer and order IDs, and the results are sorted by order date in descending order. The outer query then chooses only the first row with ROWNUM = 1, resulting in the most recent order information for a client.



```
Run SQL Command Line
SQL> SELECT * FROM (SELECT C.*, O.* FROM Customer C JOIN Invoice_Details ID ON C.Customer_ID = ID.Customer_ID JOIN Orders O ON ID.Order_ID = O.Order_ID ORDER BY O.Order_Date DESC) WHERE ROWNUM = 1;

CUSTOMER_ID CUSTOMER_NAME        CUSTOMER_PHONE                                     CUSTOMER_SEX         ORDER CUSTOMER_ZIPCODE CUSTOMER_CATEGORY_ID   ORDER_ID ORDER_DAT   QUANTITY
----------- -------------------- -------------------------------------------------- -------------------- ----- ---------------- -------------------- ---------- --------- ----------
        109 Nirajan Bogati       9704502233                                         male                 TRUE             44800                    1        209 28-AUG-23          7

SQL>
```

Figure 36:The customer who has ordered recently.

7.2 Transaction query

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

   Query: SELECT EXTRACT(MONTH FROM I.Invoice_Date) AS Month, EXTRACT(YEAR FROM I.Invoice_Date) AS Year, SUM(I.Total_Amount) AS TotalRevenue FROM Invoice I JOIN Invoice_Details ID ON I.Invoice_ID = ID.Invoice_ID JOIN Orders O ON ID.Order_ID = O.Order_ID GROUP BY EXTRACT(MONTH FROM I.Invoice_Date), EXTRACT(YEAR FROM I.Invoice_Date) ORDER BY Year, Month;

   This SQL query calculates the monthly sales total by extracting the month and year from the Invoice_Date in the Invoice table. It works with "Billing Details" and "Orders" to aggregate the total amount and display it by month and year. The editions are arranged in chronological order.

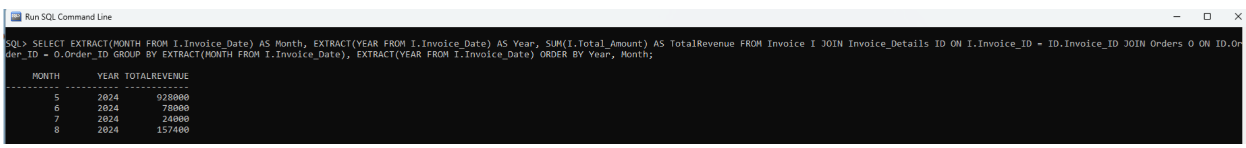
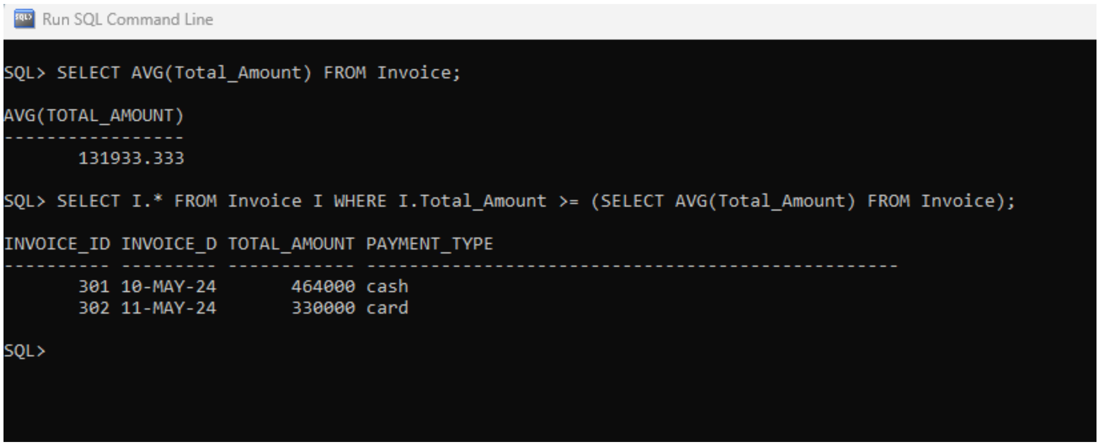


Figure 37:Show the total revenue of the company for each month.

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

   Query: SQL> SELECT AVG(Total_Amount) FROM Invoice;
   SQL> SELECT I.* FROM Invoice I WHERE I.Total_Amount >= (SELECT AVG(Total_Amount) FROM Invoice);

   The first query calculates the average total from the invoice table. The second query selects all columns from the Invoice table whose total amount is greater than or equal to the average calculated by the first query. Basically, you will get the invoices where the total amount is higher than the average value in the invoice table.



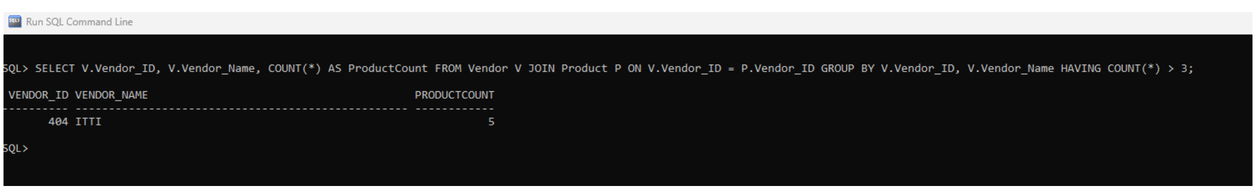
```
Run SQL Command Line

SQL> SELECT AVG(Total_Amount) FROM Invoice;

AVG(TOTAL_AMOUNT)
-----------------
       131933.333

SQL> SELECT I.* FROM Invoice I WHERE I.Total_Amount >= (SELECT AVG(Total_Amount) FROM Invoice);

INVOICE_ID INVOICE_D TOTAL_AMOUNT PAYMENT_TYPE
---------- --------- ------------ ---------------------------------------------
       301 10-MAY-24       464000 cash
       302 11-MAY-24       330000 card

SQL>
```

Figure 38:Those orders that are equal or higher than the average order total value.

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

   Query: SELECT V.Vendor_ID, V.Vendor_Name, COUNT(*) AS ProductCount FROM Vendor V JOIN Product P ON V.Vendor_ID = P.Vendor_ID GROUP BY V.Vendor_ID, V.Vendor_Name HAVING COUNT(*) > 3;

   This SQL query retrieves supplier information and the number of related products from the Vendor and Product tables. Use JOIN on the vendor IDs, group the results by vendor ID and name, and apply the HAVING clause to filter for vendors with three or more related products. Basically, identify vendors that offer three or more products and view their details.



```
Run SQL Command Line

SQL> SELECT V.Vendor_ID, V.Vendor_Name, COUNT(*) AS ProductCount FROM Vendor V JOIN Product P ON V.Vendor_ID = P.Vendor_ID GROUP BY V.Vendor_ID, V.Vendor_Name HAVING COUNT(*) > 3;

 VENDOR_ID VENDOR_NAME                                        PRODUCTCOUNT
---------- -------------------------------------------------- ------------
       404 ITTI                                                          5

SQL>
```

Figure 39:The details of vendors who have supplied more than 3 products to the company.

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

   SQL> column Product_Description format a70;

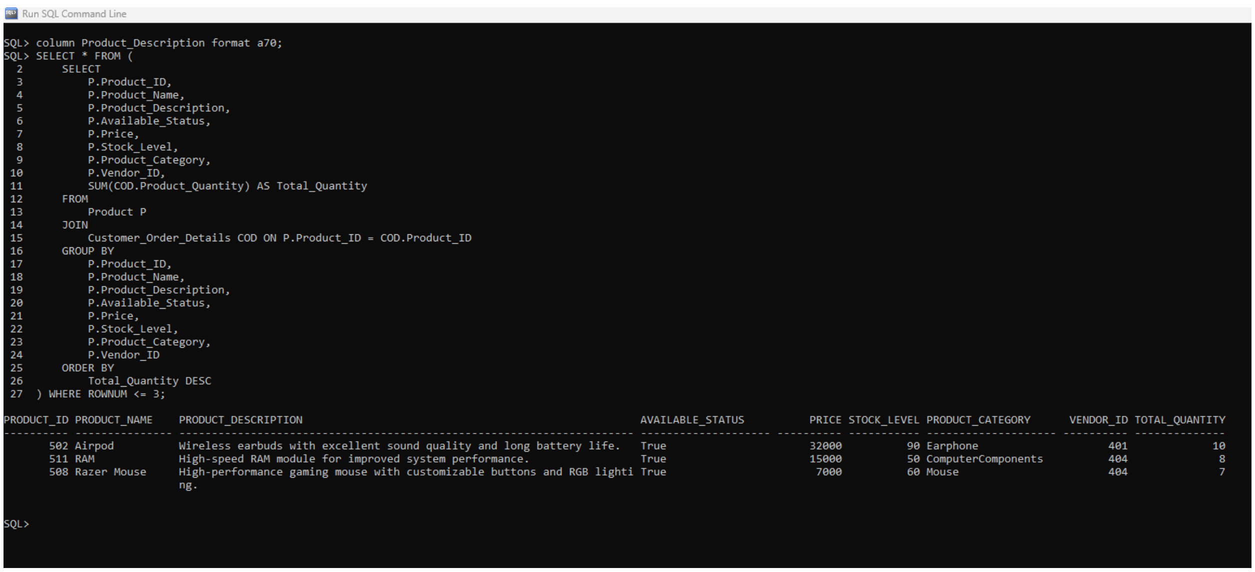
   SQL> SELECT * FROM (

     2   SELECT

```
P.Product_ID,
P.Product_Name,
 P.Product_Description,
P.Available_Status,
P.Price,
P.Stock_Level,
P.Product_Category,
P.Vendor_ID,
SUM(COD.Product_Quantity) AS Total_Quantity
FROM
Product P
JOIN
Customer_Order_Details COD ON P.Product_ID = COD.Product_ID
GROUP BY
 P.Product_ID,
 P.Product_Name,
 P.Product_Description,
 P.Available_Status,
 P.Price,
 P.Stock_Level,
 P.Product_Category,
 P.Vendor_ID
 ORDER BY
```

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

This SQL query combines information from the Product and Customer_Order_Details tables to obtain product details and total sales quantity. Use JOIN on the product IDs, group the results by various product attributes, sum the product quantities, and sort the output in descending order of total quantity. The outer query then selects the top 3 rows based on ROWNUM. Basically, the three best-selling products are identified and displayed.



```
Run SQL Command Line

SQL> column Product_Description format a70;
SQL> SELECT * FROM (
    SELECT
        P.Product_ID,
        P.Product_Name,
        P.Product_Description,
        P.Available_Status,
        P.Price,
        P.Stock_Level,
        P.Product_Category,
        P.Vendor_ID,
        SUM(COD.Product_Quantity) AS Total_Quantity
    FROM
        Product P
    JOIN
        Customer_Order_Details COD ON P.Product_ID = COD.Product_ID
    GROUP BY
        P.Product_ID,
        P.Product_Name,
        P.Product_Description,
        P.Available_Status,
        P.Price,
        P.Stock_Level,
        P.Product_Category,
        P.Vendor_ID
    ORDER BY
        Total_Quantity DESC
  ) WHERE ROWNUM <= 3;
```

PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION	AVAILABLE_STATUS	PRICE	STOCK_LEVEL	PRODUCT_CATEGORY	VENDOR_ID	TOTAL_QUANTITY
502	Airpod	Wireless earbuds with excellent sound quality and long battery life.	True	32000	90	Earphone	401	10
511	RAM	High-speed RAM module for improved system performance.	True	15000	50	ComputerComponents	404	8
508	Razer Mouse	High-performance gaming mouse with customizable buttons and RGB lighti ng.	True	7000	60	Mouse	404	7

```
SQL>
```

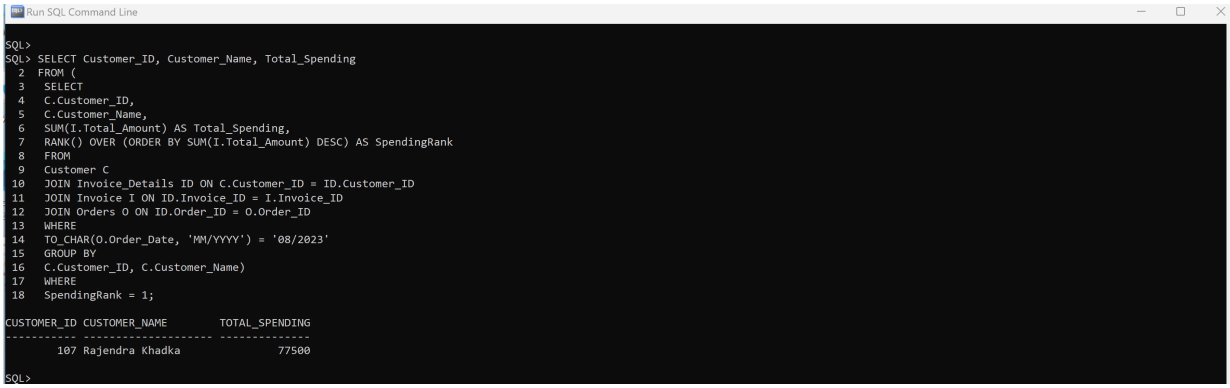
Figure 40:The top 3 product details that have been ordered the most.

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

```
SELECT Customer_ID, Customer_Name, Total_Spending

FROM (

 SELECT

 C.Customer_ID,

 C.Customer_Name,

 SUM(I.Total_Amount) AS Total_Spending,
```

```
RANK() OVER (ORDER BY SUM(I.Total_Amount) DESC) AS SpendingRank
FROM
Customer C
JOIN Invoice_Details ID ON C.Customer_ID = ID.Customer_ID
JOIN Invoice I ON ID.Invoice_ID = I.Invoice_ID
JOIN Orders O ON ID.Order_ID = O.Order_ID
WHERE
TO_CHAR(O.Order_Date, 'MM/YYYY') = '08/2023'
GROUP BY
C.Customer_ID, C.Customer_Name)
WHERE
SpendingRank = 1;
```

This SQL query determines the consumer who spends the most in August 2023. It computes each customer's total spending by combining the "Customer," "Invoice_Details," "Invoice," and "Orders" tables. The RANK() window method is used to sort clients by their total expenditure in descending order. The outer query chooses clients whose spending rank is one, indicating the highest spender for the chosen month and year. The final output contains the customer's ID, name, and total spent.




```
SQL>
SQL> SELECT Customer_ID, Customer_Name, Total_Spending
  2  FROM (
  3   SELECT
  4   C.Customer_ID,
  5   C.Customer_Name,
  6   SUM(I.Total_Amount) AS Total_Spending,
  7   RANK() OVER (ORDER BY SUM(I.Total_Amount) DESC) AS SpendingRank
  8   FROM
  9   Customer C
 10   JOIN Invoice_Details ID ON C.Customer_ID = ID.Customer_ID
 11   JOIN Invoice I ON ID.Invoice_ID = I.Invoice_ID
 12   JOIN Orders O ON ID.Order_ID = O.Order_ID
 13   WHERE
 14   TO_CHAR(O.Order_Date, 'MM/YYYY') = '08/2023'
 15   GROUP BY
 16   C.Customer_ID, C.Customer_Name)
 17   WHERE
 18   SpendingRank = 1;

CUSTOMER_ID CUSTOMER_NAME        TOTAL_SPENDING
----------- -------------------- --------------
        107 Rajendra Khadka               77500

SQL>
```

Figure 41:The customer who has ordered the most in August with his/her total spending on that month.

8. File Creation

8.1 Creating Dump File

Step 1: To create a folder, enter the command terminal by typing cmd in the desired folder.

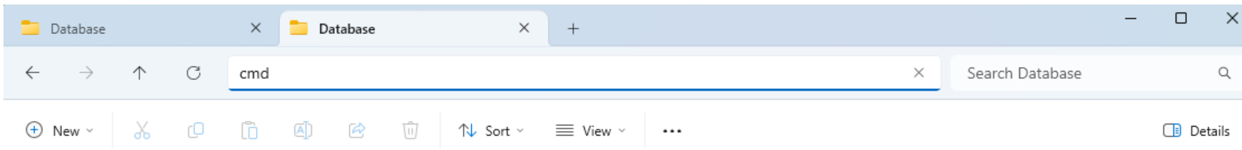


Figure 42: Creating dump file step1.

Step 2: After that type of command “exp coursework_1/renish file="C:\Database\coursework_1.dmp"”. After that press enter.

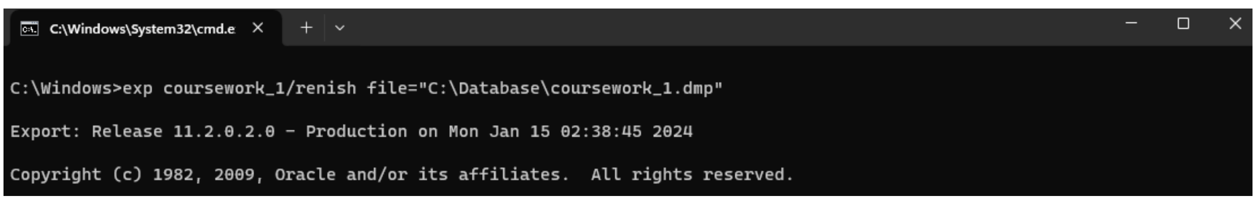


Figure 43:Exporting dump file into coursework_1/renish.

Step 3: After pressing enter the process takes times to create file. After some time "Export terminated successfully with warnings" is seen and dumb file is created**.**


C:\Windows\System32\cmd.e

```
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_1
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions for user COURSEWORK_1
About to export COURSEWORK_1's objects ...
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export COURSEWORK_1's tables via Conventional Path ...
. . exporting table                       CATEGORY          3 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                       CUSTOMER         14 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table               CUSTOMER_ADDRESS          7 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table         CUSTOMER_ORDER_DETAILS         16 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                        INVOICE          9 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                INVOICE_DETAILS          9 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                         ORDERS          9 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                        PRODUCT          0 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                         VENDOR          9 rows exported
EXP-00091: Exporting questionable statistics.
. 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 with warnings.

C:\Windows>
```

Figure 44: Process of creating dump_file.

Step 4: Here the dump file is created successfully.

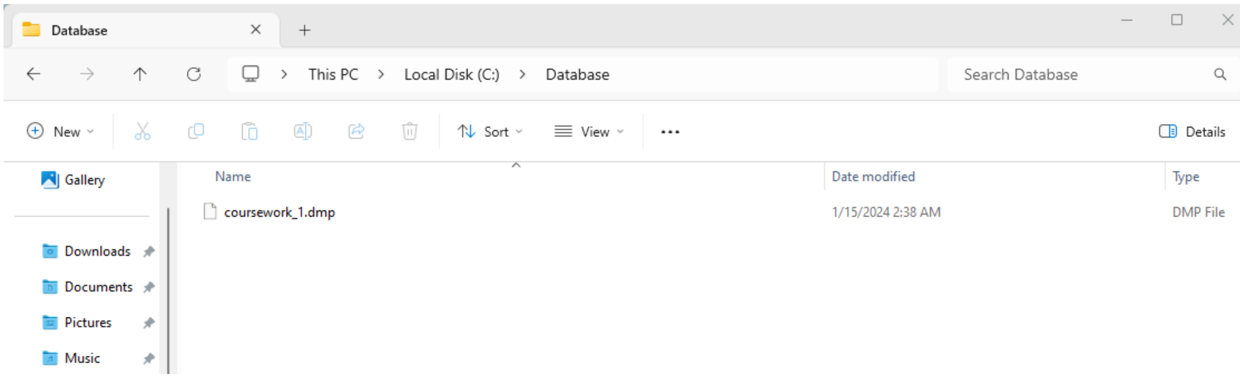


Figure 45: Created dump_file.

8.2 Drop Tables

Query:

SQL> DROP TABLE Category CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Customer_Address CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Customer CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Invoice CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Invoice_Details CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Vendor CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Customer_Order_Details CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Orders CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Product CASCADE CONSTRAINTS;

Table dropped.

This code uses the CASCADE CONSTRAINTS option and associated restrictions to drop multiple tables including Category, Customer_Address, Customer, Invoice, Invoice_Details, Vendor, Customer_Order_Details, Orders, and Product.

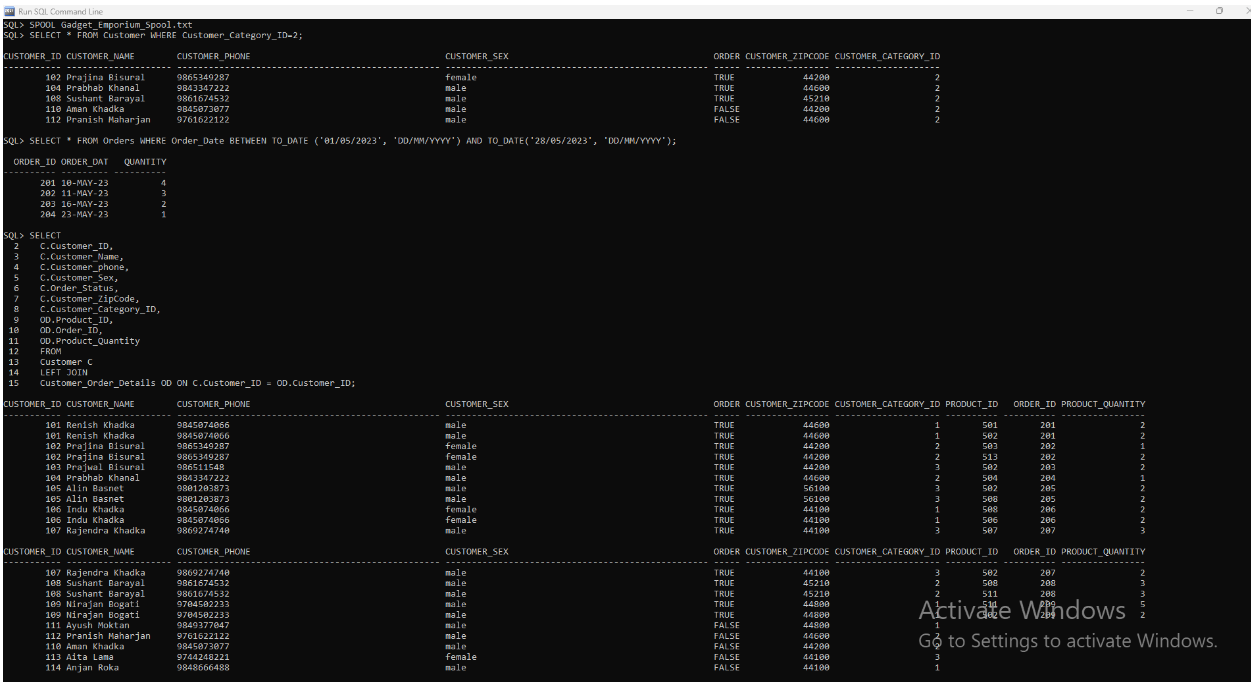
Run SQL Command Line

```
SQL> DROP TABLE Category CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Customer_Address CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Customer CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Invoice CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Invoice_Details CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Vendor CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Customer_Order_Details CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Order CASCADE CONSTRAINTS;
DROP TABLE Order CASCADE CONSTRAINTS
           *
ERROR at line 1:
ORA-00903: invalid table name


SQL> DROP TABLE Orders CASCADE CONSTRAINTS;

Table dropped.

SQL> DROP TABLE Product CASCADE CONSTRAINTS;

Table dropped.

SQL>
```

Figure 46: Drop all the tables.

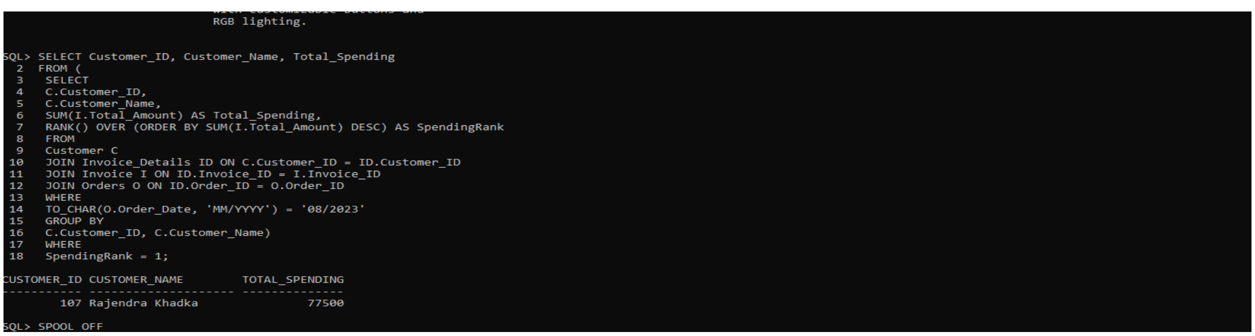
8.3 Spool Creation of Quires.

In Oracle SQL, the term "spool" is often used in the context of SQL, an interactive batch query tool provided by Oracle for interacting with Oracle databases. The SPOOL command in SQLis used to save the output of a SQL query or a series of commands to a file. This file is called a "spool file".



```
Run SQL Command Line
SQL> SPOOL Gadget_Emporium_Spool.txt
SQL> SELECT * FROM Customer WHERE Customer_Category_ID=2;

CUSTOMER_ID CUSTOMER_NAME        CUSTOMER_PHONE                                      CUSTOMER_SEX                                        ORDER CUSTOMER_ZIPCODE CUSTOMER_CATEGORY_ID
----------- -------------------- --------------------------------------------------- --------------------------------------------------- ----- ---------------- --------------------
        102 Prajina Bisural      9865349287                                          female                                              TRUE             44200                    2
        104 Prabhab Khanal       9843347222                                          male                                                TRUE             44600                    2
        108 Sushant Barayal      9861674532                                          male                                                TRUE             45210                    2
        110 Aman Khadka          9845073077                                          male                                                FALSE            44200                    2
        112 Pranish Maharjan     9761622122                                          male                                                FALSE            44600                    2

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

  ORDER_ID ORDER_DAT   QUANTITY
---------- --------- ----------
       201 10-MAY-23          4
       202 11-MAY-23          3
       203 16-MAY-23          2
       204 23-MAY-23          1

SQL> SELECT
  2    C.Customer_ID,
  3    C.Customer_Name,
  4    C.Customer_phone,
  5    C.Customer_Sex,
  6    C.Order_Status,
  7    C.Customer_ZipCode,
  8    C.Customer_Category_ID,
  9    OD.Product_ID,
 10    OD.Order_ID,
 11    OD.Product_Quantity
 12    FROM
 13    Customer C
 14    LEFT JOIN
 15    Customer_Order_Details OD ON C.Customer_ID = OD.Customer_ID;

CUSTOMER_ID CUSTOMER_NAME        CUSTOMER_PHONE            CUSTOMER_SEX              ORDER CUSTOMER_ZIPCODE CUSTOMER_CATEGORY_ID PRODUCT_ID   ORDER_ID PRODUCT_QUANTITY
----------- -------------------- ------------------------- ------------------------- ----- ---------------- -------------------- ---------- ---------- ----------------
        101 Renish Khadka        9845074066                male                      TRUE             44600                    1        501        201                2
        101 Renish Khadka        9845074066                male                      TRUE             44600                    1        502        201                2
        102 Prajina Bisural      9865349287                female                    TRUE             44200                    2        503        202                1
        102 Prajina Bisural      9865349287                female                    TRUE             44200                    2        513        202                2
        103 Prajwal Bisural      986511548                 male                      TRUE             44200                    3        502        203                2
        104 Prabhab Khanal       9843347222                male                      TRUE             44600                    2        504        204                1
        105 Alin Basnet          9801203873                male                      TRUE             56100                    3        502        205                2
        105 Alin Basnet          9801203873                male                      TRUE             56100                    3        508        205                2
        106 Indu Khadka          9845074066                female                    TRUE             44100                    1        508        206                2
        106 Indu Khadka          9845074066                female                    TRUE             44100                    1        506        206                2
        107 Rajendra Khadka      9869274740                male                      TRUE             44100                    3        507        207                3

CUSTOMER_ID CUSTOMER_NAME        CUSTOMER_PHONE            CUSTOMER_SEX              ORDER CUSTOMER_ZIPCODE CUSTOMER_CATEGORY_ID PRODUCT_ID   ORDER_ID PRODUCT_QUANTITY
----------- -------------------- ------------------------- ------------------------- ----- ---------------- -------------------- ---------- ---------- ----------------
        107 Rajendra Khadka      9869274740                male                      TRUE             44100                    3        502        207                2
        108 Sushant Barayal      9861674532                male                      TRUE             45210                    2        508        208                3
        108 Sushant Barayal      9861674532                male                      TRUE             45210                    2        511        208                3
        109 Nirajan Bogati       9704502233                male                      TRUE             44800                    1        511        209                5
        109 Nirajan Bogati       9704502233                male                      TRUE             44800                    1        502        209                2
        111 Ayush Moktan         9849377047                male                      FALSE            44800                    1
        112 Pranish Maharjan     9761622122                male                      FALSE            44600                    2
        110 Aman Khadka          9845073077                male                      FALSE            44200                    2
        113 Aita Lama            9744248221                female                    FALSE            44100                    3
        114 Anjan Roka           9848666488                male                      FALSE            44100                    1
```


Figure 47: Spool start in first command.



```
                          RGB lighting.

SQL> SELECT Customer_ID, Customer_Name, Total_Spending
  2  FROM (
  3   SELECT
  4   C.Customer_ID,
  5   C.Customer_Name,
  6   SUM(I.Total_Amount) AS Total_Spending,
  7   RANK() OVER (ORDER BY SUM(I.Total_Amount) DESC) AS SpendingRank
  8   FROM
  9   Customer C
 10   JOIN Invoice_Details ID ON C.Customer_ID = ID.Customer_ID
 11   JOIN Invoice I ON ID.Invoice_ID = I.Invoice_ID
 12   JOIN Orders O ON ID.Order_ID = O.Order_ID
 13   WHERE
 14   TO_CHAR(O.Order_Date, 'MM/YYYY') = '08/2023'
 15   GROUP BY
 16   C.Customer_ID, C.Customer_Name)
 17   WHERE
 18   SpendingRank = 1;

CUSTOMER_ID CUSTOMER_NAME        TOTAL_SPENDING
----------- -------------------- --------------
        107 Rajendra Khadka               77500

SQL> SPOOL OFF
```

Figure 48: SPOOL OFF command.

9. Critical Evaluation

CC5051NI - Level 5 database engine. We learned Oracle SQL. Oracle SQL is a powerful and widely used relational database management system (RDBMS). Use the SQL language for database queries and operations. As part of our studies, we will learn various aspects of Oracle SQL, including creating and managing database objects, writing SQL queries to retrieve and manipulate data, implementing data constraints, and understanding normalization principles for efficient database design. We think we have covered some aspects. Gaining knowledge of Oracle SQL is valuable in the industry, as Oracle SQL is a widely used database technology for many organizations to manage and derive insights from their data. Emerging Programming Platforms and Technologies, and a variety of other real-world circumstances

Engaging in database coursework has been both enjoyable and demanding, with a combination of excitement and obstacles. The enjoyable side is learning the nuances of Oracle SQL, discovering the power of database design, and seeing personally how data manipulation may lead to significant discoveries. It was all about creating database for GADEGT EMPORIUM However, obstacles have surfaced in dealing with complicated queries, maintaining data standardization for best speed, and overcoming the occasional complexities of Oracle SQL syntax. Faced with these hurdles, the learning curve has been severe but extremely rewarding, with each difficulty presented as an opportunity to improve problem-solving skills and expand understanding. Overall, the coursework provides a dynamic and enriching experience that combines the joy of understanding powerful tools like Oracle SQL with the satisfaction of tackling database management problems.

In summary, the importance of this coursework lies in its role in improving students' understanding of database systems and their management. The acquired knowledge and skills will enable students to skillfully cope with future challenges and tasks related to databases, ensuring a competent and effective approach to these endeavors.

10.Bibliography

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

Peterson, R., 2023. *Entity Relationship (ER) Diagram Model with DBMS Example.* [Online]
Available at: https://www.guru99.com/er-diagram-tutorial-dbms.html
[Accessed 6 January 2024].

Peterson, R., 2023. *guru99.* [Online]
Available at: https://www.guru99.com/er-diagram-tutorial-dbms.html
[Accessed 6 January 2024].

S, R. A., 2023. *What is Normalization in SQL? 1NF, 2NF, 3NF and BCNF in DBMS.* [Online]
Available at: https://www.simplilearn.com/tutorials/sql-tutorial/what-is-normalization-in-sql
[Accessed 6 January 2024].

Garge, S., 2022. *Unnormalized form.* [Online]
Available at: https://alchetron.com/Unnormalized-form
[Accessed 6 January 2023].

Rouse, M., 2011. *What Does First Normal Form Mean?.* [Online]
Available at: https://www.techopedia.com/definition/25955/first-normal-form-1nf
[Accessed 6 January 2024].

Babbar, M., 2022. *Normalization in DBMS.* [Online]
Available at: https://www.scaler.com/topics/dbms/normalization-in-dbms/
[Accessed 11 January 2023].

Babbar, M., 2022. *Normalization in DBMS.* [Online]
Available at: https://www.scaler.com/topics/dbms/normalization-in-dbms/
[Accessed 11 January 2024].