



 slington college

(इस्लिङ्टन कलेज)

CC5051NI Databases

50% Individual Coursework

Autumn 2023

Student Name: Arbit Bhandari

London Met ID: 22068111

Assignment Submission Date: 14 January 2024

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

1	INTRODUCTION.....	1
1.1	Business Rules	3
1.2	Identification of Entities and Attributes	4
1.2.1	Vendor.....	4
1.2.2	Product	5
1.2.3	Order	6
1.2.4	Customer	7
2	INITIAL ERD	8
2.1	Entities and Attributes with identification and representation of the Primary Keys, Foreign Keys.....	8
3	NORMALIZATION.....	10
4	FINAL ERD	17
5	IMPLEMENTATION	18
5.1	CREATE RELATION AND TABLES	18
5.2	INSERTION.....	24
6	Queries	30
6.1	INFORMATION QUERY	30
6.2	TRANSACTION QUERY	33
7	CRITICAL EVALUATION	36
7.1	Critical Evaluation of module.....	36
7.2	Its usage and relation with other subject	36
7.3	Critical Assessment of coursework	37
8	Drop Query and Database Dump file creation	38
9	CONCLUSION	40
10	References	41
11	APPENDIX	42

Table of Figures

Figure 1 Initial ERD	9
Figure 2 Final ERD.....	17
Figure 3 Create Table Vendor.....	18
Figure 4 Describe Vendor	18
Figure 5 Create Product Table	19
Figure 6 Describe Product.....	19
Figure 7 Create Table Customer	20
Figure 8 Describe Customer	20
Figure 9 Create Table Payment	21
Figure 10 Describe Payment.....	21
Figure 11 Create Table Orders	22
Figure 12 Describe Orders	22
Figure 13 Create Table Product_order.....	23
Figure 14 Describe Product_order	23
Figure 15 Inserting into Vendor	24
Figure 16 Selecting From Vendor	24
Figure 17 Inserting into Product	25
Figure 18 Selecting From Product.....	25
Figure 19 Inserting into Customer	26
Figure 20 Selecting From Customer.....	26
Figure 21 Inserting into Payment	27
Figure 22 Selecting From Payment.....	27
Figure 23 Inserting into Orders.....	28
Figure 24 Select from Orders	28
Figure 25 Inserting into Product_order	29
Figure 26 Selecting From Product_order	29
Figure 27 Query 1 of INFORMATION	30
Figure 28 Query 2 of INFORMATION	30
Figure 29 Query 3 of INFORMATION	31
Figure 30 Query 4 of INFORMATION	31
Figure 31 Query 5 of INFORMATION	32
Figure 32 Query 1 of TRANSACTION.....	33
Figure 33 Query 2 of TRANSACTION.....	33
Figure 34 Query 3 of TRANSACTION.....	34
Figure 35 Query 4 of TRANSACTION.....	34
Figure 36 Query 5 of TRANSACTION.....	35
Figure 37 Dropping the table.....	38
Figure 38 Dump File Creation	39
Figure 39 Screenshot of Creating User	42
Figure 40 Screenshot of Commit.....	43
Figure 41 Screenshot of Spool.....	43

Table of Tables

Table 1 Vendor Table..... 4

Table 2 Product table 5

Table 3 Order table 6

Table 4 Customer Table..... 7

1 INTRODUCTION

Gadget Emporium is an e-commerce platform founded by entrepreneur and electronics enthusiast, Mr. John. The platform sells a variety of good electronic devices and accessories. The aim is to serve both individual customers and businesses with high-quality products. The primary goal is to establish an efficient and secure online marketplace that smoothly combines through selection of electronic products. Gadget Emporium offers a thoughtfully selected collection to meet the needs of tech fans and businesses.

The core operations of Gadget Emporium include efficient product management, ensuring organized details of electronic gadgets and accessories. Customers will be categorized as Regular, Staff, or VIP, with corresponding discount rates on purchases. The system will facilitate smooth order processing, recording complete details of products, quantities, and prices. Vendor management will be essential, combining each product with a specific supplier. Real-time tracking of product availability and inventory management will prevent overselling. The platform will integrate with various payment gateways for secure transactions, offering options like cash on delivery, credit/debit cards, or e-wallets. Upon order confirmation, an invoice will be generated, providing detailed information on the order, customer, and payment details, including applicable discounts. In essence, Gadget Emporium aspires to deliver a user-friendly and complete online shopping experience for electronic accessories fans.

In short, Gadget Emporium is a strong foundation for smoothly handling products, connecting with customers, managing orders, and working closely with vendors. The focus is on creating an online marketplace that not only showcases the latest in electronics but also ensures a smooth and satisfying shopping experience for customers.

AIMS

The aim of implementing a database system for "Gadget Emporium" is to make sure the online store runs smoothly. This involves managing products well, organizing customers, handling orders, working with vendors, keeping track of inventory in real-time, ensuring safe payments, and creating invoices easily. The aim is to make everything run efficiently, keep customers happy, and maintain accurate records for the business to grow steadily.

OBJECTIVES

- To Develop a database system to efficiently manage electronic gadgets and accessories
- To keep track of essential product details like names, descriptions, categories, prices, and real-time stock levels
- To Implement an order processing system that records detailed information about each purchase
- To Establish a vendor management system to maintain records of suppliers for electronic gadgets and accessories
- To implement secure payment processing by integrating with various payment gateways

CURRENT BUSINESS ACTIVITIES AND OPERATIONS

"Gadget Emporium operates an online marketplace specializing in electronic devices and accessories. The business manages a diverse product inventory, categorizes customers for personalized discounts, processes orders with multiple items, monitors vendor relationships and inventory levels, and integrates with various payment gateways for smooth transactions. The system generates detailed invoices with customer-specific discounts, ensuring a comprehensive and customer-centric e-commerce experience."

1.1 Business Rules

A business rule is a statement that defines or constrains some aspect of the business. It represents a specific requirement, guideline, or constraint that must be enforced within the database to ensure data accuracy, consistency, and integrity. Business rules are used to govern data-related processes and help maintain the quality of information in a database system (Raipurkar, 2012).

- Each product belongs to only one category.
- Each category can have one or many products.
- Customers can be categorized as Regular (R), Staff (S), and VIP (V).
- Customer address must be stored for delivery purposes.
- An order can have one or more products.
- A product can be included in multiple orders.
- Each product must be associated with a single vendor.
- Each vendor can supply one or more products.
- Each order must be associated with one payment method.

These business rules guide the design and implementation of the database system, promoting consistency, accuracy, and integrity in managing product-category associations, customer categorization, order-product relationships, product-vendor associations, and payment methods for orders.

1.2 Identification of Entities and Attributes

1.2.1 Vendor

Vendor is a business entity or supplier that provides electronic devices and accessories to the store. Each product within the marketplace is linked to a specific vendor, establishing a connection between the products offered and their respective suppliers. The vendor records include details such as contact information, supplier-specific data, and the products supplied by each vendor. This information is essential for efficient inventory management, tracking the sources of products, and maintaining effective relationships with suppliers in the business's supply chain.

Attributes	Data Type	Constraints	Description
Vendor_ID	NUMBER	PRIMARY KEY, UNIQUE	This field records the unique identifier assigned to each vendor in the system.
Vendor_Name	VARCHAR2 (50)	NOT NULL	This field records the name of the vendor who supplies electronic devices and accessories.
Vendor_Email	VARCHAR2 (50)	NOT NULL	This field records the email address of the vendor for communication purposes.
Vendor_Address	VARCHAR2 (50)	NOT NULL	This field records the vendor's address for shipping and delivery.
Vendor_Contact	NUMBER	NOT NULL	This field records the contact number of Vendor.

Table 1 Vendor Table

1.2.2 Product

Product represents individual electronic gadgets and accessories available for sale, associated with unique product IDs. It includes complete details such as product names, descriptions, assigned categories, pricing information, and current stock levels. Each product exclusively belongs to a single category, and categories can include one or multiple products.

Attributes	Data Type	Constraints	Description
Product_ID	NUMBER	PRIMARY KEY, UNIQUE	This field records the unique identifier assigned to each product.
Vendor_ID	NUMBER	FOREIGN KEY, NOT NULL	This field records the unique identifier assigned to each vendor in the system.
Product_Name	VARCHAR2 (50)	NOT NULL	This field records the descriptive name of the electronic gadget or accessory.
Product_Description	VARCHAR2 (50)	NOT NULL	This field records detailed information of the products.
Product_Category	VARCHAR2 (50)	NOT NULL	This field records the categorization of the product into specific groups.
Product_Price	NUMBER	NOT NULL	This field records the price value assigned to the product
Product_Stock Level	NUMBER	NOT NULL	This field records the quantity of the product available in stock.

Table 2 Product table

1.2.3 Order

Order represents a transaction started by a customer for purchasing electronic gadgets or accessories from the "Gadget Emporium". Each order is uniquely identified, capturing details such as the products purchased, quantities, unit prices, and total order amount. Multiple products can be included in a single order, and the same product may appear in multiple orders from various customers. The order records help us keep track of customer transactions and manage our inventory efficiently.

Attributes	Data Type	Constraints	Description
Order_ID	NUMBER	PRIMARY KEY, UNIQUE	This field records the unique identifier assigned to each order.
Customer_ID	NUMBER	FOREIGN KEY, NOT NULL	This field records the unique identifier assigned to each customer in the system.
Product_ID	NUMBER	FOREIGN KEY, NOT NULL	This field records the unique identifier assigned to each product.
Order_Date	DATE	NOT NULL	This field records the date when the order was placed.
Order_Quantity	NUMBER	NOT NULL	This field records the total quantity of products in the order.
Order_Total_Amount	NUMBER	NOT NULL	This field records the overall price value of the order.
Payment_ID	NUMBER	NOT NULL	This field records the unique identifier associated with the payment for the order.
Payment_Method	VARCHAR2(50)	NOT NULL	This field records the chosen method of payment for the order

Table 3 Order table

1.2.4 Customer

Customer: Represents individuals engaging with the "Gadget Emporium" online marketplace, categorized as Regular (R), Staff (S), or VIP (V). Each customer record includes personal details, such as name and address, crucial for order delivery. The system assigns distinct discount rates (0%, 5%, and 10%) based on customer categories. This entity enables personalized services and efficient order tracking for a smooth shopping experience.

Attributes	Data Type	Constraints	Description
Customer_ID	NUMBER	PRIMARY KEY, UNIQUE	This field records the unique identifier assigned to each customer in the system.
Customer_Name	VARCHAR2 (50)	NOT NULL	This field records the name of the customer who purchases electronic devices and accessories.
Customer_Address	VARCHAR2 (50)	NOT NULL	This field records the customer's address for shipping and delivery.
Customer_Discount	NUMBER	NOT NULL	This field records any applicable discounts for the customer.
Customer_Category	NUMBER	NOT NULL	This field categorizes the customer based on regular, staff and vip

Table 4 Customer Table

2 INITIAL ERD

An entity relationship diagram (ERD), alternatively referred to as an entity relationship model, is a visual depiction illustrating the connections between individuals, objects, locations, ideas, or occurrences within an information technology (IT) system. Using data modeling methods, an ERD aids in outlining business processes and establishing the groundwork for a relational database (Biscobing, 2023).

2.1 Entities and Attributes with identification and representation of the Primary Keys, Foreign Keys.

ENTITY: **Vendor**

ATTRIBUTES: **Vendor_ID**, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact

- **Primary Key (Vendor_ID)**

ENTITY: **Product**

ATTRIBUTES: **Product_ID**, **Vendor_ID***, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level

- **Primary Key (Product_ID)**
- **Foreign Key (Vendor_ID)**

ENTITY: **Order**

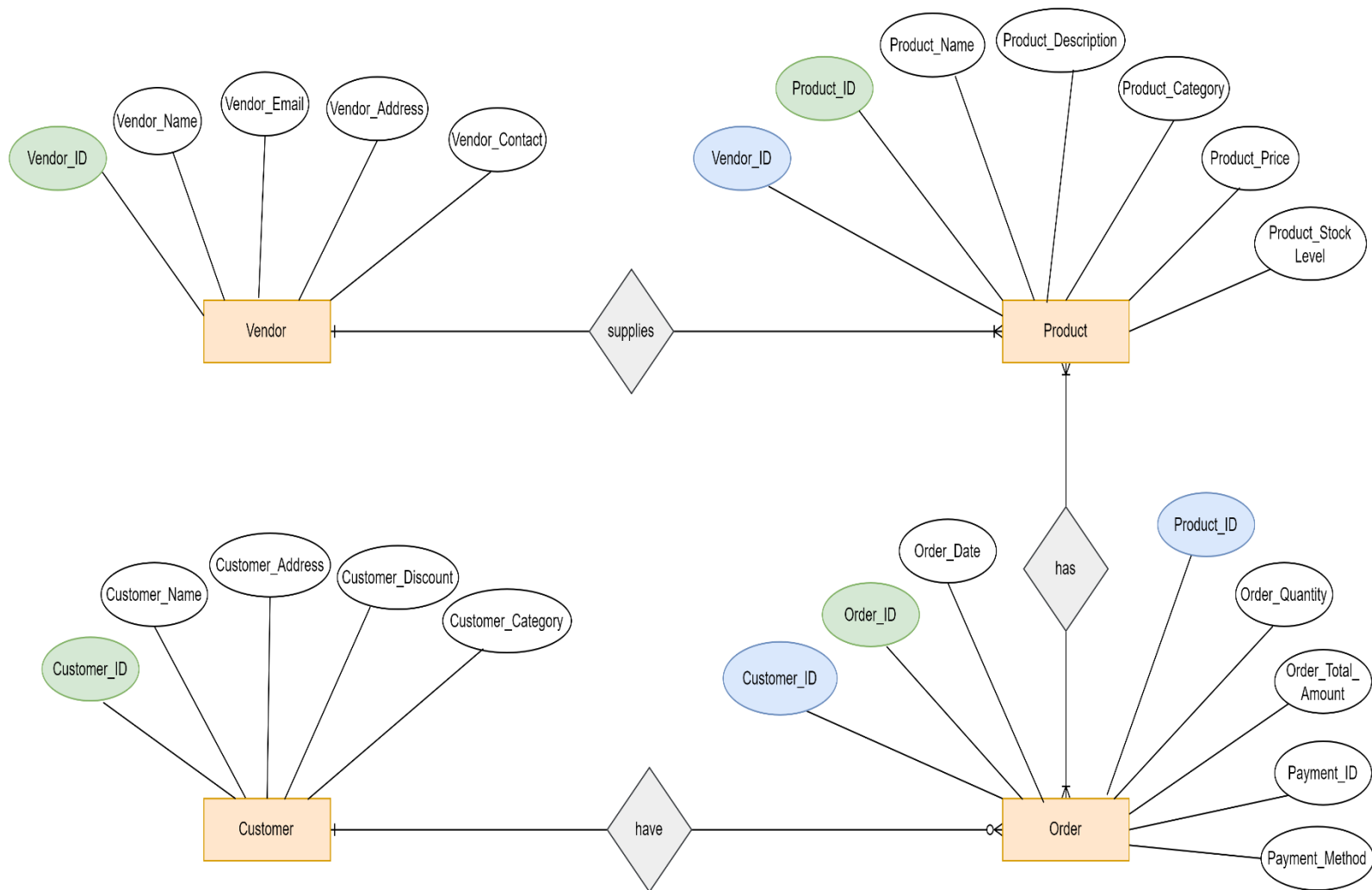
ATTRIBUTES: **Order_ID**, **Customer_ID***, **Product_ID***, Order_Date, Order_Quantity, Order_Total_Amount, Payment_ID, Payment_Method

- **Primary Key (Order_ID)**
- **Foreign Key (Customer_ID, Product_ID)**

ENTITY: **Customer**

ATTRIBUTES: **Customer_ID**, Customer_Name, Customer_Address, Customer_Discount, Customer_Category

- **Primary Key (Customer_ID)**

*Figure 1 Initial ERD*

3 NORMALIZATION

Normalization is a method in database design aimed at minimizing data redundancy and eliminating undesirable issues such as Insertion, Update, and Deletion Anomalies. This technique involves breaking down larger tables into smaller ones and establishing relationships between them. The primary goal of Normalization in SQL is to remove redundant (repetitive) data and ensure the logical storage of data (Peterson, 2023).

Unnormalized Form (UNF)

Unnormalized Form (UNF) is a stage in the database design process where the structure does not comply with standard normalization principles. It often involves data redundancy and inefficient organization. The goal of normalization is to progress from UNF to higher normal forms for a more efficient and well-organized database.

UNF

Product(Product_ID(PK) , Product_Name, Product_Description, Product_Category, Product_Prices, Product_Stock_Level, Vendor_ID, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact {Order_ID, Order_Date, Order_Quantity, Order_Amount, Payment_ID, Payment_Method, Customer_ID, Customer_Name, Customer_Address, Customer_Discount, Customer_Category})

The Unnormalized Form (UNF) have two entities, 'Product' and 'Order,' with associated attributes. To enhance data organization and integrity, we move the UNF to higher normalized forms (1NF, 2NF, 3NF). This involves validating individual values at the basic level, addressing dependencies, and establishing relationships using foreign keys. The final steps include adding constraints, potential further normalization, and optimization for system requirements, resulting in a well-structured relational database design.

1NF

1NF, or First Normal Form, is a fundamental concept in relational database design that helps ensure data integrity by organizing data in a structured and consistent way. The primary goal of 1NF is to eliminate repeating groups of data within a table (Rouse, 2011).

The first normal form states that:

- Remove the repeating groups to a new table and give name to the table.
- Identify the new primary key for the new table (from repeating group).
- Add in a foreign key from UNF table to create a relationship with the new table
- Insert “- 1” while naming the tables so as to identify them as tables from first normal form

FIRST NORMAL FORM

- Here the principle that the candidate primary key must be unique is followed.
- Therefore we need to include an attribute in the new table that links this back to the original table.

Product-1(Product_ID(PK), Product_Name, Product_Description, Product_Category, Product_Prices, Product_Stock_Level, Vendor_ID, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact)

Order-1(Product_ID(FK)*, Order_ID(PK), Order_Date, Order_Quantity, Order_Amount, Payment_ID, Payment_Method, Customer_ID, Customer_Name, Customer_Address, Customer_Discount, Customer_Category)

In achieving First Normal Form (1NF), the attributes in 'Product-1' and 'Order-1' entities have been made primary and foreign keys has been set. To further improve the design in Second Normal Form (2NF), ensure non-prime attributes are fully dependent on the entire primary key. This step aims to eliminate partial dependencies and enhance data organization.

2NF

Second Normal Form (2NF) ensures that a table in a relational database is free of partial dependencies on its composite primary key. This eliminates data redundancy and enhances data integrity by organizing the database logically (byjus, 2023).

The second normal form states that:

- Look at tables with a composite key.
- Review each non-key attribute to identify whether the attribute is dependent on part of the key or all of the key.
- Remove any partial key and dependants to a new table. Choose a primary key for that table.
- Partial dependencies are relationships where the attribute in focus is not dependent on the entire composite key but only part of the composite key.

Second Normal Form**For Product table**

The "Product" entity doesn't have a composite keys, so there's no chance of partial dependency. Therefore, the "Product" entity is already in the Second Normal Form (2NF).

For Order table

In this table, there is a partial dependency because there are two unique keys.

Product_ID(FK)*, Order_ID(PK)

Product_ID → There is no attribute dependency on the "**Product_ID**"; therefore, no attributes are associated with it.

Order_ID → Order_Date, Order_Amount, Payment_ID, Payment_Method, Customer_ID, Customer_Name, Customer_Address, Customer_Discount, Customer_Category

Product_ID, Order_ID → Order_Quantity

Putting into 2NF, the tables formed are:

Product-2(Product_ID(PK), Product_Name, Product_Description, Product_Category, Product_Prices, Product_Stock_Level, Vendor_ID, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact)

Order-2(Order_ID(PK), Order_Date, Order_Amount, Payment_ID, Payment_Method, Customer_ID, Customer_Name, Customer_Address, Customer_Discount, Customer_Category)

Product_Order-2(Product_ID(PK)(FK), Order_ID(PK)(FK), Order_Quantity)

In Second Normal Form (2NF), we organize our data so that it's efficient and avoids redundancy. If we have a "Product" entity and it doesn't use a combination of keys (composite keys), then it's already good for 2NF because there's no partial dependency.

However, for the "Order" entity, there's a bit of an issue. It has unique keys, like **Product_ID** and **Order_ID**. This can lead to partial dependency, which we want to avoid. To fix this, we create three tables: one for products ("Product-2"), one for orders ("Order-2"), and a linking table ("Product_Order-2") to show the relationship between products and orders. This linking table uses composite primary keys to make sure everything is connected properly.

Now, moving on to Third Normal Form (3NF), we want to make sure there are no transitive dependencies. This means that non-prime attributes (attributes that aren't part of the key) shouldn't depend on other non-prime attributes. By doing this, we make our data even more organized, reduce redundancy, and improve data integrity.

3NF

Third Normal Form (3NF) is a database normalization concept in relational database design. It builds upon the first two normal forms (1NF and 2NF) and aims to eliminate redundancy and improve data integrity in a relational database (geeksforgeeks, 2023).

The third normal form states that:

- Identify any dependencies between non-key attributes within each table in 2NF
- Remove them to a new table
- Decide on a primary key
- This primary key of new table becomes the foreign key in the original table (existing table in 2NF)

Third Normal Form**For Product table**

Product_ID → Product_Name, Product_Description, Product_Category, Product_Prices, Product_Stock_Level

Product_ID → Vendor_ID → Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact

(Transitive Dependency)

❖ Vendor_ID → Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact

For Product table

Order_ID \rightarrow Order_Date, Order_Amount

Order_ID \rightarrow Payment_ID \rightarrow Payment_Method

(Transitive Dependency)

❖ Payment_ID \rightarrow Payment_Method

Order_ID \rightarrow Customer_ID \rightarrow Customer_Name, Customer_Address, Customer_Discount,
Customer_Category

(Transitive Dependency)

❖ Customer_ID \rightarrow Customer_Name Customer_Address, Customer_Discount,
Customer_Category

FOR Product_order

No transitive dependency, Since only one non key attributes

Putting into 3NF, the tables formed are:

Product-3(Product_ID(PK), Product_Name, Product_Description, Product_Category, Product_Prices, Product_Stock_Level, **Vendor_ID(FK)***)

Vendor-3(Vendor_ID(PK), Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact)

Order-3(Order_ID(PK), Order_Date, Order_Amount, **Payment_ID(FK)***, **Customer_ID(FK)***)

Customer-3(Customer_ID(PK), Customer_Name, Customer_Address, Customer_Discount, Customer_Category)

Payment-3(Payment_ID(PK), Payment_Method)

Product-Order-3(Product_ID(PK)(FK)*, Order_ID(PK)(FK)*, Order_Quantity)

We're making our database really organized, and we're now in the Third Normal Form (3NF) stage. In the "Product-3" section, we made sure that all the details about a product only depend on one main thing - its ID. This way, everything is neatly connected without unnecessary complications.

The "Vendor-3" part is also well-organized. Each vendor has its own ID, and the connection with "Product-3" follows the rules of good organization, making things easy to manage.

Now, for the "Order-3" part, we sorted things out nicely. We separated out details about payments into "Payment-3" and details about customers into "Customer-3". Each of them has its own ID. The links between "Order-3" and these new sections are smartly done using these IDs. This makes sure that everything is in order, and we're not repeating information too much.

4 FINAL ERD

A final ERD (Entity-Relationship Diagram) is a visual representation of the structure of a database, depicting entities, attributes, relationships, and constraints. It serves as a blueprint for database development and ensures a clear understanding of data organization (Biscobing, 2019).

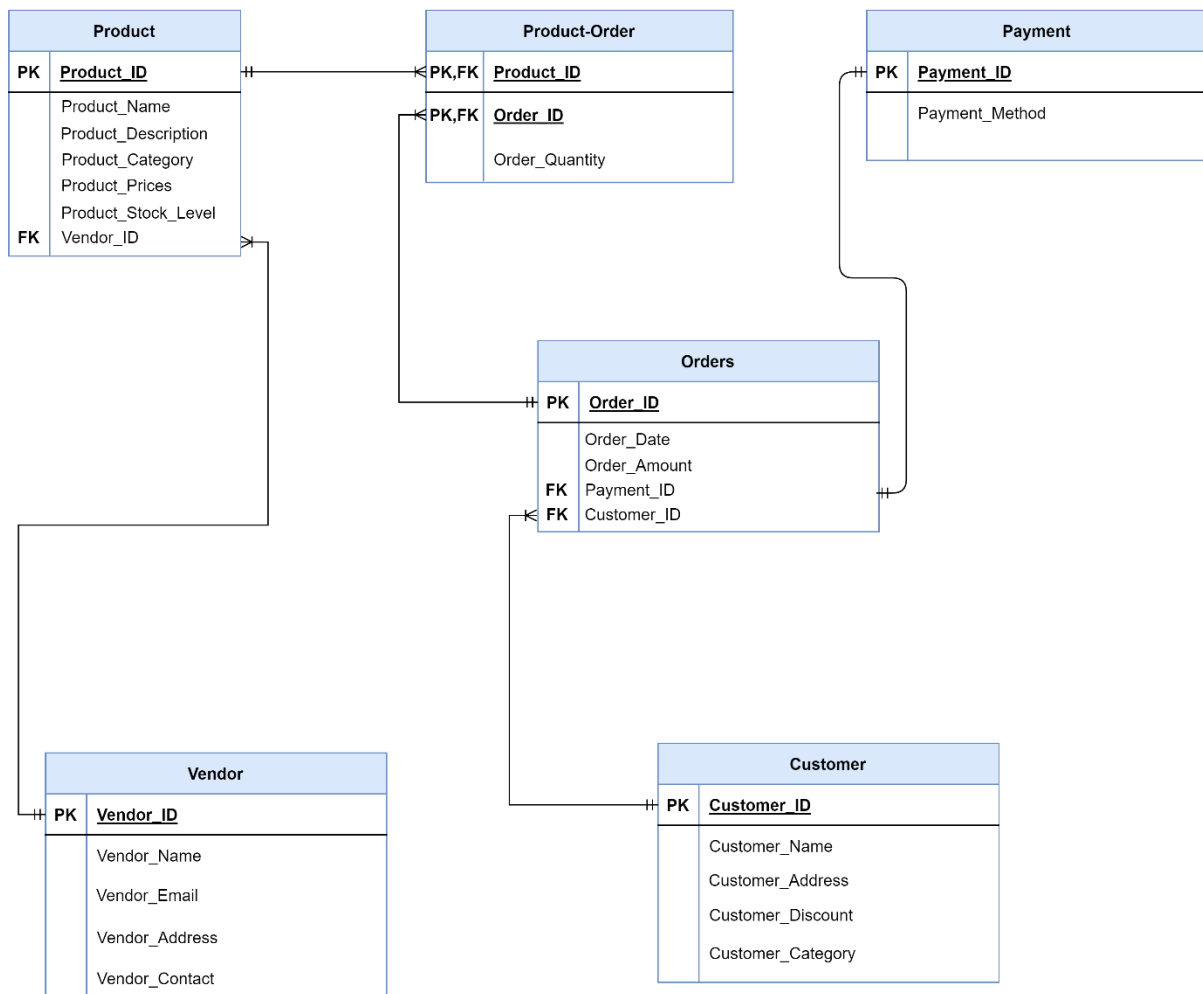


Figure 2 Final ERD

5 IMPLEMENTATION

5.1 CREATE RELATION AND TABLES

Vendor Table

The "Vendor" table consists of a numerical primary key ("Vendor_ID") and four required text fields limited to 50 characters each: "Vendor_Name" for the vendor's name, "Vendor_Email" for email addresses, "Vendor_Address" for physical addresses, and "Vendor_Contact" for contact information. This structure facilitates the storage and recall of essential vendor details in a relational database.

```
SQL> CREATE TABLE Vendor (  
 2     Vendor_ID NUMBER PRIMARY KEY,  
 3     Vendor_Name VARCHAR2(50) NOT NULL,  
 4     Vendor_Email VARCHAR2(50) NOT NULL,  
 5     Vendor_Address VARCHAR2(50) NOT NULL,  
 6     Vendor_Contact VARCHAR2(50) NOT NULL  
 7 );
```

Table created.

Figure 3 Create Table Vendor

```
SQL> Desc Vendor;
```

Name	Null?	Type
VENDOR_ID	NOT NULL	NUMBER
VENDOR_NAME	NOT NULL	VARCHAR2(50)
VENDOR_EMAIL	NOT NULL	VARCHAR2(50)
VENDOR_ADDRESS	NOT NULL	VARCHAR2(50)
VENDOR_CONTACT	NOT NULL	VARCHAR2(50)

```
SQL>
```

Figure 4 Describe Vendor

Product Table

The "Product" table is designed to manage product information with key attributes such as ID, name, description, category, price, and stock level. The table includes a foreign key, "Vendor_ID," linking to the "Vendor" table. Mandatory constraints ensure data integrity, while optional fields allow for flexibility. This structure aims to streamline product-related data storage and retrieval within the relational database, fostering effective organization and association with vendor details.

```
SQL> CREATE TABLE Product (  
2     Product_ID NUMBER PRIMARY KEY NOT NULL,  
3     Product_Name VARCHAR2(50) NOT NULL,  
4     Product_Description VARCHAR2(50) NOT NULL,  
5     Product_Category VARCHAR2(50) NOT NULL,  
6     Product_Price NUMBER NOT NULL CHECK (Product_Price >= 0),  
7     Product_Stock_Level NUMBER NOT NULL CHECK (Product_Stock_Level >= 0),  
8     Vendor_ID NUMBER NOT NULL,  
9     CONSTRAINT fk_product_vendor FOREIGN KEY (Vendor_ID) REFERENCES Vendor(Vendor_ID)  
10 );
```

Figure 5 Create Product Table

```
SQL> Desc Product;  
Name                               Null?      Type  
-----  
PRODUCT_ID                        NOT NULL   NUMBER  
PRODUCT_NAME                      NOT NULL   VARCHAR2(50)  
PRODUCT_DESCRIPTION               NOT NULL   VARCHAR2(50)  
PRODUCT_CATEGORY                 NOT NULL   VARCHAR2(50)  
PRODUCT_PRICE                    NOT NULL   NUMBER  
PRODUCT_STOCK_LEVEL              NOT NULL   NUMBER  
VENDOR_ID                        NOT NULL   NUMBER
```

Figure 6 Describe Product

Customer Table

The "Customer" table is created to manage customer information with a primary key ("Customer_ID") and essential attributes. These include "Customer_Name" for the customer's name, "Customer_Address" for the address, "Customer_Category" for categorization, and an optional "Customer_Discount" for potential discounts. The structure ensures efficient storage and recall of customer details in a relational database.

```
SQL> CREATE TABLE Customer (  
2     Customer_ID NUMBER PRIMARY KEY NOT NULL,  
3     Customer_Name VARCHAR2(50) NOT NULL,  
4     Customer_Address VARCHAR2(50) NOT NULL,  
5     Customer_Category VARCHAR2(50) NOT NULL,  
6     Customer_Discount NUMBER NOT NULL  
7 );
```

Figure 7 Create Table Customer

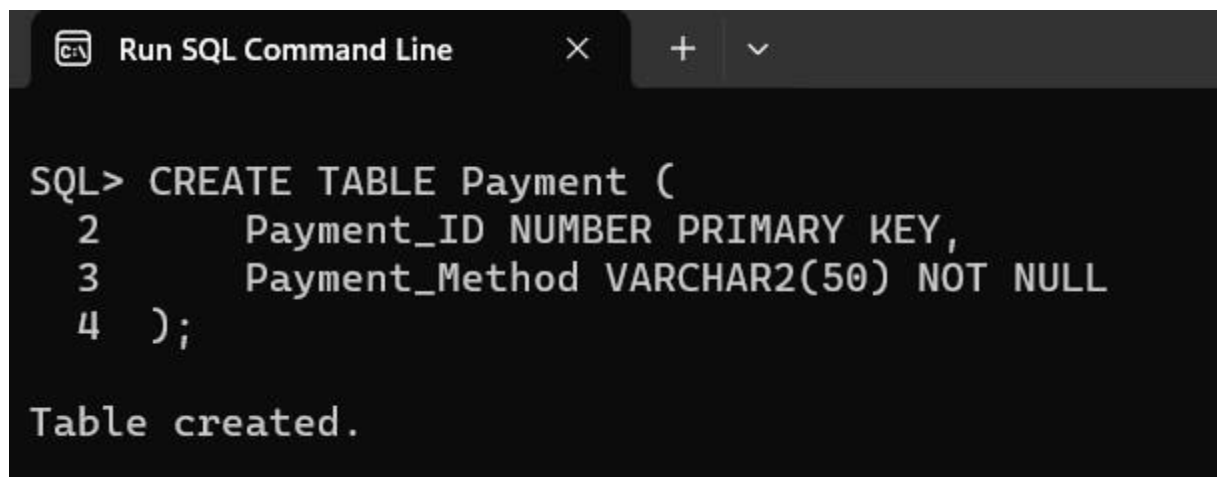
```
SQL> Desc Customer;
```

Name	Null?	Type
CUSTOMER_ID	NOT NULL	NUMBER
CUSTOMER_NAME	NOT NULL	VARCHAR2(50)
CUSTOMER_ADDRESS	NOT NULL	VARCHAR2(50)
CUSTOMER_CATEGORY	NOT NULL	VARCHAR2(50)
CUSTOMER_DISCOUNT	NOT NULL	NUMBER

Figure 8 Describe Customer

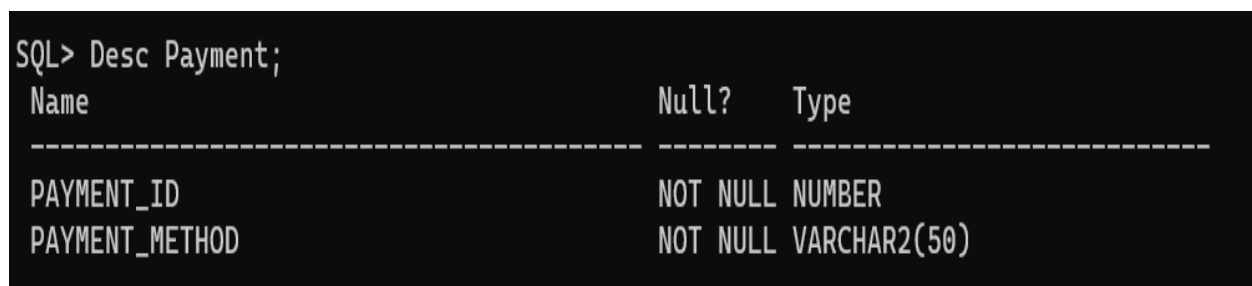
Payment Table

The "Payment" table is established to handle payment information with a primary key ("Payment_ID") and a required attribute, "Payment_Method," specifying the payment method. This structured approach allows for effective storage and recall of payment details in a relational database.



```
SQL> CREATE TABLE Payment (  
2     Payment_ID NUMBER PRIMARY KEY,  
3     Payment_Method VARCHAR2(50) NOT NULL  
4 );  
  
Table created.
```

Figure 9 Create Table Payment



```
SQL> Desc Payment;  
  
Name                               Null?    Type  
-----  
PAYMENT_ID                         NOT NULL NUMBER  
PAYMENT_METHOD                     NOT NULL VARCHAR2(50)
```

Figure 10 Describe Payment

Orders Table

The "Orders" table is designed to track order-related information, featuring a primary key ("Order_ID") along with key attributes: "Order_Date" capturing the order date, "Order_Total_Amount" indicating the total cost (required and non-negative), "Payment_ID" serving as a foreign key referencing the "Payment" table, and "Customer_ID" as a foreign key linking to the "Customer" table. The foreign key constraints ("fk_Payment" and "fk_Customer") enforce referential integrity, ensuring accurate associations with payment and customer details. Additionally, a check constraint ("chk_Order_Total_Amount") guarantees the non-negativity of the order total. This structured design facilitates efficient organization and retrieval of order-related data within the relational database.

```
SQL> CREATE TABLE Orders (
  2     Order_ID NUMBER PRIMARY KEY NOT NULL,
  3     Order_Date DATE NOT NULL,
  4     Order_Total_Amount NUMBER NOT NULL,
  5     Payment_ID NUMBER NOT NULL,
  6     Customer_ID NUMBER NOT NULL,
  7     CONSTRAINT fk_Payment FOREIGN KEY (Payment_ID) REFERENCES Payment(Payment_ID),
  8     CONSTRAINT fk_Customer FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID),
  9     CONSTRAINT chk_Order_Total_Amount CHECK (Order_Total_Amount >= 0)
 10 );
```

Figure 11 Create Table Orders

```
SQL> Desc Orders;
```

Name	Null?	Type
-----	-----	-----
ORDER_ID	NOT NULL	NUMBER
ORDER_DATE	NOT NULL	DATE
ORDER_TOTAL_AMOUNT	NOT NULL	NUMBER
PAYMENT_ID	NOT NULL	NUMBER
CUSTOMER_ID	NOT NULL	NUMBER

Figure 12 Describe Orders

Product_order Table

The "Product_order" table is created to establish a many-to-many relationship between products and orders. It includes three key columns: "Product_ID" and "Order_ID," both mandatory and forming the composite primary key, and "Order_Quantity" specifying the quantity of the product ordered (mandatory with a check for a positive value). Foreign key constraints link "Product_ID" to the "Product" table and "Order_ID" to the "Orders" table, ensuring referential integrity. This design enables the effective representation of product quantities within each order, facilitating organized storage and retrieval of product order-related information in a relational database.

```
SQL> CREATE TABLE Product_Order (  
  2     Product_ID NUMBER NOT NULL,  
  3     Order_ID NUMBER NOT NULL,  
  4     Order_Quantity NUMBER NOT NULL CHECK (Order_Quantity > 0),  
  5     PRIMARY KEY (Product_ID, Order_ID),  
  6     FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID),  
  7     FOREIGN KEY (Order_ID) REFERENCES Orders(Order_ID)  
  8 );
```

Table created.

Figure 13 Create Table Product_order

```
SQL> Desc Product_order;
```

Name	Null?	Type
PRODUCT_ID	NOT NULL	NUMBER
ORDER_ID	NOT NULL	NUMBER
ORDER_QUANTITY	NOT NULL	NUMBER

Figure 14 Describe Product_order

5.2 INSERTION

Inserting data in Vendor

The SQL query adds details for seven vendors to the "Vendor" table. It includes important information such as IDs, names, emails, addresses, and contact numbers. This simple method makes sure vendor data is stored well in the database.

```
SQL> INSERT ALL
2 INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact) VALUES (1, 'Arbit Electronics', 'arbit@gmail.com', 'Kathmandu', '123-456-7890')
3 INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact) VALUES (2, 'Alance Supplies', 'alance@gmail.com', 'Chitwan', '987-654-3210')
4 INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact) VALUES (3, 'Rishav Tech', 'rishav@gmail.com', 'Kathmandu', '555-123-4567')
5 INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact) VALUES (4, 'Riya Devices', 'riya@gmail.com', 'Chitwan', '111-222-3333')
6 INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact) VALUES (5, 'Rajan Technologies', 'rajan@gmail.com', 'Kathmandu', '444-555-6666')
7 INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact) VALUES (6, 'Karan Electronics', 'karan@gmail.com', 'Chitwan', '777-888-9999')
8 INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Email, Vendor_Address, Vendor_Contact) VALUES (7, 'Bishal Supplies', 'bishal@gmail.com', 'Kathmandu', '666-999-8888')
9 SELECT * FROM dual;

7 rows created.
```

Figure 15 Inserting into Vendor

Select from Vendor

```
SQL> SELECT * FROM Vendor;
```

VENDOR_ID	VENDOR_NAME	VENDOR_EMAIL	VENDOR_ADDRESS	VENDOR_CONTACT
1	Arbit Electronics	arbit@gmail.com	Kathmandu	123-456-7890
2	Alance Supplies	alance@gmail.com	Chitwan	987-654-3210
3	Rishav Tech	rishav@gmail.com	Kathmandu	555-123-4567
4	Riya Devices	riya@gmail.com	Chitwan	111-222-3333
5	Rajan Technologies	rajan@gmail.com	Kathmandu	444-555-6666
6	Karan Electronics	karan@gmail.com	Chitwan	777-888-9999
7	Bishal Supplies	bishal@gmail.com	Kathmandu	666-999-8888

```
7 rows selected.
```

Figure 16 Selecting From Vendor

Inserting data in Product

The SQL query adds data for ten different products into the "Product" table. Using the "INSERT ALL" statement with the "VALUES" clause for each product, the query includes details such as product names, descriptions, categories, prices, stock levels, and vendor IDs.

```
SQL> INSERT ALL
2 INTO Product (Product_ID, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level, Vendor_ID) VALUES (1, 'LG Refrigerator', 'Double-door refrigerator', 'Appliances', 1099.99, 40, 1)
3 INTO Product (Product_ID, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level, Vendor_ID) VALUES (2, 'Acer Monitor', 'LCD Monitor', 'Electronics', 249.99, 25, 2)
4 INTO Product (Product_ID, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level, Vendor_ID) VALUES (3, 'HP Scanner', 'Flatbed scanner', 'Electronics', 79.99, 30, 3)
5 INTO Product (Product_ID, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level, Vendor_ID) VALUES (4, 'Canon Printer', 'Color printer', 'Electronics', 149.99, 55, 4)
6 INTO Product (Product_ID, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level, Vendor_ID) VALUES (5, 'D-Link Router', 'Wireless router', 'Networking', 39.99, 50, 5)
7 INTO Product (Product_ID, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level, Vendor_ID) VALUES (6, 'Sony Earphones', 'In-ear headphones', 'Audio', 59.99, 35, 6)
8 INTO Product (Product_ID, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level, Vendor_ID) VALUES (7, 'Logitech Webcam', 'HD webcam', 'Electronics', 89.99, 15, 7)
9 INTO Product (Product_ID, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level, Vendor_ID) VALUES (8, 'Microsoft Tablet', 'Tablet', 'Electronics', 699.99, 10, 2)
10 INTO Product (Product_ID, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level, Vendor_ID) VALUES (9, 'TP-Link Switch', 'Ethernet switch', 'Networking', 29.99, 40, 2)
11 INTO Product (Product_ID, Product_Name, Product_Description, Product_Category, Product_Price, Product_Stock_Level, Vendor_ID) VALUES (10, 'Samsung SSD', 'Solid State Drive', 'Computer Accessories', 129.99, 60, 2)
12 SELECT * FROM dual;

10 rows created.
```

Figure 17 Inserting into Product

Select From Product

```
SQL> Select * From Product;
```

PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION	PRODUCT_CATEGORY	PRODUCT_PRICE	PRODUCT_STOCK_LEVEL	VENDOR_ID
1	LG Refrigerator	Double-door refrigerator	Appliances	1099.99	40	1
2	Acer Monitor	LCD Monitor	Electronics	249.99	25	2
3	HP Scanner	Flatbed scanner	Electronics	79.99	30	3
4	Canon Printer	Color printer	Electronics	149.99	55	4
5	D-Link Router	Wireless router	Networking	39.99	50	5
6	Sony Earphones	In-ear headphones	Audio	59.99	35	6
7	Logitech Webcam	HD webcam	Electronics	89.99	15	7
8	Microsoft Tablet	Tablet	Electronics	699.99	10	2
9	TP-Link Switch	Ethernet switch	Networking	29.99	40	2
10	Samsung SSD	Solid State Drive	Computer Accessories	129.99	60	2

```
10 rows selected.
```

Figure 18 Selecting From Product

Inserting data in Customer

The query inserts data for nine customers into the "Customer" table, covering details such as IDs, names, addresses, categories, and discounts. We've organized customer information in the database for efficient storage. To recall information.

```
SQL> INSERT ALL
2 INTO Customer (Customer_ID, Customer_Name, Customer_Address, Customer_Category, Customer_Discount) VALUES (1, 'Rita Thakur', 'Bhaktapur', 'Regular', 0)
3 INTO Customer (Customer_ID, Customer_Name, Customer_Address, Customer_Category, Customer_Discount) VALUES (2, 'Suman Khatri', 'Lalitpur', 'Staff', 5)
4 INTO Customer (Customer_ID, Customer_Name, Customer_Address, Customer_Category, Customer_Discount) VALUES (3, 'Amit Sharma', 'Kathmandu', 'Regular', 0)
5 INTO Customer (Customer_ID, Customer_Name, Customer_Address, Customer_Category, Customer_Discount) VALUES (4, 'Anju Shakya', 'Butwal', 'Staff', 5)
6 INTO Customer (Customer_ID, Customer_Name, Customer_Address, Customer_Category, Customer_Discount) VALUES (5, 'Rajiv Koirala', 'Pokhara', 'Vip', 10)
7 INTO Customer (Customer_ID, Customer_Name, Customer_Address, Customer_Category, Customer_Discount) VALUES (6, 'Neetu Gupta', 'Kathmandu', 'Regular', 0)
8 INTO Customer (Customer_ID, Customer_Name, Customer_Address, Customer_Category, Customer_Discount) VALUES (7, 'Manish Thapa', 'Biratnagar', 'Staff', 5)
9 INTO Customer (Customer_ID, Customer_Name, Customer_Address, Customer_Category, Customer_Discount) VALUES (8, 'Riya Acharya', 'soltimood', 'Vip', 10)
10 INTO Customer (Customer_ID, Customer_Name, Customer_Address, Customer_Category, Customer_Discount) VALUES (9, 'Sujan Khatri', 'Lalitpur', 'Staff', 5)
11 SELECT * FROM dual;

9 rows created.
```

Figure 19 Inserting into Customer

Select From Customer

```
SQL> Select * From Customer;
```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_ADDRESS	CUSTOMER_CATEGO	CUSTOMER_DISCOUNT
1	Rita Thakur	Bhaktapur	Regular	0
2	Suman Khatri	Lalitpur	Staff	5
3	Amit Sharma	Kathmandu	Regular	0
4	Anju Shakya	Butwal	Staff	5
5	Rajiv Koirala	Pokhara	Vip	10
6	Neetu Gupta	Kathmandu	Regular	0
7	Manish Thapa	Biratnagar	Staff	5
8	Riya Acharya	soltimood	Vip	10
9	Sujan Khatri	Lalitpur	Staff	5

```
9 rows selected.
```

Figure 20 Selecting From Customer

Inserting data in Payment

The query adds Cash, Khalti, Esewa, Debit/Credit Cards, and Cheques as payment methods to the "Payment" table. It ensures organized storage in the database.

```
SQL> INSERT ALL
  2 INTO Payment (Payment_ID, Payment_Method) VALUES (1, 'Cash')
  3 INTO Payment (Payment_ID, Payment_Method) VALUES (2, 'Khalti')
  4 INTO Payment (Payment_ID, Payment_Method) VALUES (3, 'Esewa')
  5 INTO Payment (Payment_ID, Payment_Method) VALUES (4, 'Debit/Credit Cards')
  6 INTO Payment (Payment_ID, Payment_Method) VALUES (5, 'Cheques')
  7 SELECT * FROM dual;

5 rows created.
```

Figure 21 Inserting into Payment

Select From Payment

```
SQL> Select * From Payment;
```

PAYMENT_ID	PAYMENT_METHOD
1	Cash
2	Khalti
3	Esewa
4	Debit/Credit Cards
5	Cheques

Figure 22 Selecting From Payment

Inserting data in Orders

The query populates the "Product_order" table, capturing associations between products and orders. Leveraging the "INSERT ALL" statement with "VALUES" clauses for each product-order pair, the script includes essential details like "Product_ID," "Order_ID," and "Order_Quantity."

```
SQL> INSERT ALL
  2 INTO Orders (Order_ID, Order_Date, Order_Total_Amount, Payment_ID, Customer_ID) VALUES (1, TO_DATE('2023-05-10', 'YYYY-MM-DD'), 30.0, 1, 1)
  3 INTO Orders (Order_ID, Order_Date, Order_Total_Amount, Payment_ID, Customer_ID) VALUES (2, TO_DATE('2023-05-15', 'YYYY-MM-DD'), 40.0, 2, 2)
  4 INTO Orders (Order_ID, Order_Date, Order_Total_Amount, Payment_ID, Customer_ID) VALUES (3, TO_DATE('2023-05-20', 'YYYY-MM-DD'), 50.0, 1, 3)
  5 INTO Orders (Order_ID, Order_Date, Order_Total_Amount, Payment_ID, Customer_ID) VALUES (4, TO_DATE('2023-06-01', 'YYYY-MM-DD'), 60.0, 4, 4)
  6 INTO Orders (Order_ID, Order_Date, Order_Total_Amount, Payment_ID, Customer_ID) VALUES (5, TO_DATE('2023-06-05', 'YYYY-MM-DD'), 70.0, 5, 5)
  7 INTO Orders (Order_ID, Order_Date, Order_Total_Amount, Payment_ID, Customer_ID) VALUES (6, TO_DATE('2023-08-01', 'YYYY-MM-DD'), 80.0, 2, 6)
  8 INTO Orders (Order_ID, Order_Date, Order_Total_Amount, Payment_ID, Customer_ID) VALUES (7, TO_DATE('2023-08-05', 'YYYY-MM-DD'), 90.0, 3, 7)
  9 SELECT * FROM dual;

7 rows created.
```

Figure 23 Inserting into Orders

Select From Orders

```
SQL> Select * from Orders;
```

ORDER_ID	ORDER_DAT	ORDER_TOTAL_AMOUNT	PAYMENT_ID	CUSTOMER_ID
1	10-MAY-23	30	1	1
2	15-MAY-23	40	2	2
3	20-MAY-23	50	1	3
4	01-JUN-23	60	4	4
5	05-JUN-23	70	5	5
6	01-AUG-23	80	2	6
7	05-AUG-23	90	3	7

```
7 rows selected.
```

Figure 24 Select from Orders

Inserting data in Product_order

The query populates the "Product_order" table, capturing associations between products and orders. Leveraging the "INSERT ALL" statement with "VALUES" clauses for each product-order pair, the script includes essential details like "Product_ID," "Order_ID," and "Order_Quantity."

```
SQL> INSERT ALL
 2 INTO Product_order (Product_ID, Order_ID, Order_Quantity) VALUES (1, 1, 5)
 3 INTO Product_order (Product_ID, Order_ID, Order_Quantity) VALUES (2, 1, 7)
 4 INTO Product_order (Product_ID, Order_ID, Order_Quantity) VALUES (3, 3, 3)
 5 INTO Product_order (Product_ID, Order_ID, Order_Quantity) VALUES (4, 1, 2)
 6 INTO Product_order (Product_ID, Order_ID, Order_Quantity) VALUES (5, 2, 1)
 7 INTO Product_order (Product_ID, Order_ID, Order_Quantity) VALUES (6, 3, 2)
 8 INTO Product_order (Product_ID, Order_ID, Order_Quantity) VALUES (7, 4, 1)
 9 INTO Product_order (Product_ID, Order_ID, Order_Quantity) VALUES (8, 5, 2)
10 INTO Product_order (Product_ID, Order_ID, Order_Quantity) VALUES (9, 6, 2)
11 INTO Product_order (Product_ID, Order_ID, Order_Quantity) VALUES (10, 7, 1)
12 SELECT * FROM dual;

10 rows created.
```

Figure 25 Inserting into Product_order

Select From Product_order

```
SQL> Select * From Product_order;
```

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

```
10 rows selected.
```

Figure 26 Selecting From Product_order

6 Queries

6.1 INFORMATION QUERY

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

The query first selects all the customers from the Customer table. Then, it shows the results by only selecting the customers that have a Customer_Category of 'Staff'. The results of the query are then displayed in a table.

```
SQL> SELECT *
2 FROM Customer
3 WHERE Customer_Category = 'Staff';
```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_ADDRES	CUSTOMER_CATEGORY	CUSTOMER_DISCOUNT
2	Suman Khatri	Lalitpur	Staff	5
4	Anju Shakya	Butwal	Staff	5
7	Manish Thapa	Biratnagar	Staff	5

```
SQL>
```

Figure 27 Query 1 of INFORMATION

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

The SQL query retrieves data from the "Orders" table where the "Order_Date" falls within the specified date range.

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

ORDER_ID	ORDER_DAT	ORDER_TOTAL_AMOUNT	PAYMENT_ID	CUSTOMER_ID
1	10-MAY-23	30	1	1
2	15-MAY-23	40	2	2
3	20-MAY-23	50	1	3

```
SQL>
```

Figure 28 Query 2 of INFORMATION

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

The first SQL query retrieves information about customers and their corresponding orders from a database. On the other hand, the second SQL query is designed to fetch information about customers who have not placed any orders.

```
SQL> SELECT c.Customer_ID, c.Customer_Name, c.Customer_Address, c.Customer_Category, c.Customer_Discount, o.Order_ID, o.Order_Date, o.Order_Total_Amount
2 FROM Customer c
3 JOIN Orders o ON c.Customer_ID = o.Customer_ID;
```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_ADDRES	CUSTOMER_CATEGORY	CUSTOMER_DISCOUNT	ORDER_ID	ORDER_DAT	ORDER_TOTAL_AMOUNT
1	Rita Thakur	Bhaktapur	Regular	0	1	10-MAY-23	30
2	Suman Khatri	Lalitpur	Staff	5	2	15-MAY-23	40
3	Amit Sharma	Kathmandu	Regular	0	3	20-MAY-23	50
4	Anju Shakya	Butwal	Staff	5	4	01-JUN-23	60
5	Rajiv Koirala	Pokhara	Vip	10	5	05-JUN-23	70
6	Neetu Gupta	Kathmandu	Regular	0	6	01-AUG-23	80
7	Manish Thapa	Biratnagar	Staff	5	7	05-AUG-23	90

7 rows selected.

```
SQL> select * from customer
2 where customer_id Not in (select c.customer_id from Customer c
3 JOIN Orders o ON c.Customer_ID = o.Customer_ID);
```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_ADDRES	CUSTOMER_CATEGORY	CUSTOMER_DISCOUNT
8	Riya Acharya	soltimood	Vip	10
9	Sujan Khatri	Lalitpur	Staff	5

SQL>

Figure 29 Query 3 of INFORMATION

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

The SQL query retrieves data from the "Product" table, focusing on products with specific criteria. The "SELECT * FROM Product WHERE Product_Name LIKE 'a%' AND Product_Stock_Level > 50;"

```
SQL> SELECT *
2 FROM Product
3 WHERE Product_Name LIKE 'a%'
4 AND Product_Stock_Level > 50;
```

PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION	PRODUCT_CATEGORY	PRODUCT_PRICE	PRODUCT_STOCK_LEVEL	VENDOR_ID
4	Canon Printer	Color printer	Electronics	149.99	55	4
10	Samsung SSD	Solid State Drive	Computer Accessories	129.99	60	2

SQL>

Figure 30 Query 4 of INFORMATION

5. Find out the customer who has ordered recently.

The SQL query aims to identify the customer with the most recent order by extracting customer details and the date of their latest order.

```
SQL> SELECT Customer_ID, Customer_Name, Recent_Order_Date
 2  FROM (
 3      SELECT c.Customer_ID, c.Customer_Name, MAX(o.Order_Date) AS Recent_Order_Date
 4      FROM Customer c
 5      JOIN Orders o ON c.Customer_ID = o.Customer_ID
 6      GROUP BY c.Customer_ID, c.Customer_Name
 7      ORDER BY Recent_Order_Date DESC
 8  )
 9  WHERE ROWNUM = 1;
```

CUSTOMER_ID	CUSTOMER_NAME	RECENT_OR
7	Manish Thapa	05-AUG-23

SQL>

Figure 31 Query 5 of INFORMATION

6.2 TRANSACTION QUERY

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

The SQL query generates a summary of monthly total revenue from the "Orders" table, presenting the results in a structured format.

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

MONTH	TOTALREVENUE
2023-05	120
2023-06	130
2023-08	170

Figure 32 Query 1 of TRANSACTION

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

The SQL query extracts data from the "Orders" table, specifically retrieving records where the total order amount is equal to or exceeds the calculated average order total for all orders.

```
SQL> SELECT *
2 FROM Orders
3 WHERE Order_Total_Amount >= (SELECT AVG(Order_Total_Amount) FROM Orders);
```

ORDER_ID	ORDER_DAT	ORDER_TOTAL_AMOUNT	PAYMENT_ID	CUSTOMER_ID
4	01-JUN-23	60	4	4
5	05-JUN-23	70	5	5
6	01-AUG-23	80	2	6
7	05-AUG-23	90	3	7

```
SQL>
```

Figure 33 Query 2 of TRANSACTION

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

The SQL query retrieves data from the "Vendor" table, focusing on vendors who have supplied more than three products, based on a join with the "Product" table.

```
SQL> SELECT v.Vendor_ID, v.Vendor_Name, v.Vendor_Email, v.Vendor_Address, v.Vendor_Contact, COUNT(p.Product_ID) AS Product_Count
2 FROM Vendor v
3 JOIN Product p ON v.Vendor_ID = p.Vendor_ID
4 GROUP BY v.Vendor_ID, v.Vendor_Name, v.Vendor_Email, v.Vendor_Address, v.Vendor_Contact
5 HAVING COUNT(p.Product_ID) > 3
6 ORDER BY v.Vendor_ID, v.Vendor_Name, v.Vendor_Email, v.Vendor_Address, v.Vendor_Contact, Product_Count DESC;
```

VENDOR_ID	VENDOR_NAME	VENDOR_EMAIL	VENDOR_ADDRESS	VENDOR_CONTACT	PRODUCT_COUNT
2	Alance Supplies	alance@gmail.com	Chitwan	987-654-3210	4

Figure 34 Query 3 of TRANSACTION

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

The SQL query aims to identify the top three products based on their total order quantities.

```
SQL> SELECT p.*, po.Order_Quantity
2 FROM (
3 SELECT po.Product_ID, SUM(po.Order_Quantity) AS Total_Quantity
4 FROM Product_order po
5 GROUP BY po.Product_ID
6 ORDER BY Total_Quantity DESC
7 ) subquery
8 JOIN Product p ON subquery.Product_ID = p.Product_ID
9 JOIN Product_order po ON subquery.Product_ID = po.Product_ID
10 WHERE ROWNUM <= 3;
```

PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION	PRODUCT_CATEGORY	PRODUCT_PRICE	PRODUCT_STOCK_LEVEL	VENDOR_ID	ORDER_QUANTITY
2	Acer Monitor	LCD Monitor	Electronics	249.99	25	2	7
1	LG Refrigerator	Double-door refrigerator	Appliances	1099.99	40	1	5
3	HP Scanner	Flatbed scanner	Electronics	79.99	30	3	3

Figure 35 Query 4 of TRANSACTION

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

The SQL query is designed to identify the top-spending customer for the month of August 2023.

```

SQL> SELECT *
2  FROM (
3      SELECT
4          c.Customer_ID,
5          c.Customer_Name,
6          c.Customer_Address,
7          c.Customer_Category,
8          c.Customer_Discount,
9          TO_CHAR(o.Order_Date, 'YYYY-MM') AS Month,
10         SUM(o.Order_Total_Amount) AS TotalSpending
11     FROM
12         Customer c
13     JOIN
14         Orders o ON c.Customer_ID = o.Customer_ID
15     WHERE
16         TO_CHAR(o.Order_Date, 'YYYY-MM') = '2023-08'
17     GROUP BY
18         c.Customer_ID,
19         c.Customer_Name,
20         c.Customer_Address,
21         c.Customer_Category,
22         c.Customer_Discount,
23         TO_CHAR(o.Order_Date, 'YYYY-MM')
24     ORDER BY
25         TotalSpending DESC
26 )
27 WHERE ROWNUM <= 1;

```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_ADDRES	CUSTOMER_CATEGORY	CUSTOMER_DISCOUNT	MONTH	TOTALSPENDING
7	Manish Thapa	Biratnagar	Staff	5	2023-08	90

SQL>

Figure 36 Query 5 of TRANSACTION

7 CRITICAL EVALUATION

7.1 Critical Evaluation of module

I really enjoyed the Databases module (CC5051NI) in my studies. It was led by **Ujjwal Adhikari** and gave me a solid understanding of designing and using databases. We covered important topics like entity-relationship modeling, normalization, and SQL scripting. The projects were especially helpful, allowing me to apply what I learned to real-life situations and better understand how to manage databases.

One of the best parts was a project where we designed a database for a store called Gadget Emporium. This practical experience was incredibly useful. It not only let me use what I'd learned in a practical way but also improved my problem-solving abilities. The project focused on creating a database for an online store, which is relevant to today's e-commerce and data management trends. Overall, this module provided insights that go beyond just classroom learning.

What I learned here has added to my understanding of software development, showing me how database management connects with other parts of information technology. This well-rounded approach has made a big difference in how I see and learn things in all my classes.

Overall, the Databases module not only equipped me with essential skills but also supported a deeper value for the practical applications of database management in the field of information technology.

7.2 Its usage and relation with other subject

- The coursework is closely tied to e-commerce, supply chain management, and business strategy, showcasing its integrated focus.
- Integration with payment gateways involves considerations in financial technology and cybersecurity.
- The coursework establishes a strong relationship with user experience design by prioritizing features that improve customer satisfaction and trust.

7.3 Critical Assessment of coursework

The way the coursework is set up, covering everything from designing a database to putting it into action, has been well thought out and matches what the module aims to teach. The project about making a database for Gadget Emporium was great because it felt like a real-world situation, making what we learned more useful.

Having both theory and practical parts, like working with SQL, helped me get a complete understand of managing databases. The business rules in the project were helpful, making the database development feel realistic.

One suggestion for improvement could be giving clearer instructions for certain parts of the coursework, like what's expected in the critical evaluation section. This would help students meet the criteria more easily.

In summary, the Databases module and its projects have really helped me understand how databases work, blending what we learn in theory with practical use. The skills I gained aren't just for this module but also apply to the wider world of information technology.

8 Drop Query and Database Dump file creation

Dropping tables according to order

```
SQL> Drop table Product_order;  
Table dropped.  
  
SQL> Drop Table Orders;  
Table dropped.  
  
SQL> Drop Table Product;  
Table dropped.  
  
SQL> Drop Table Vendor;  
Table dropped.  
  
SQL> Drop Table Payment;  
Table dropped.  
  
SQL> Drop table Customer;  
Table dropped.  
  
SQL>
```

Figure 37 Dropping the table

Dump File Creation

In Oracle databases, a dump file typically refers to a binary file generated by the Data Pump utility (expdp) during the export operation. This file contains a logical backup of database objects such as tables, views, and stored procedures. The dump file can later be used with the Data Pump utility's import operation (impdp) to restore the exported data and schema objects to another Oracle database.

```
Microsoft Windows [Version 10.0.22631.2861]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd C:\Users\arbit\Desktop\dmp

C:\Users\arbit\Desktop\dmp>exp coursework/arbit file= courseworkarbit.dmp

Export: Release 11.2.0.2.0 - Production on Sat Jan 13 16:48:12 2024

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

Connected to: Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
Export done in WE8MSWIN1252 character set and AL16UTF16 NCHAR character set
server uses AL32UTF8 character set (possible charset conversion)

About to export specified users ...
. exporting pre-schema procedural objects and actions
. exporting foreign function library names for user COURSEWORK
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions for user COURSEWORK
About to export COURSEWORK's objects ...
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export COURSEWORK's tables via Conventional Path ...
. . exporting table          CUSTOMER          9 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table          ORDERS             7 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table          PAYMENT            5 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table          PRODUCT           10 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table          PRODUCT_ORDER     10 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table          VENDOR            7 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
```

Figure 38 Dump File Creation

9 CONCLUSION

This assignment is all about making a database for the "Gadget Emporium" online store. We need to create things like categories for products, figure out how they're connected, and organize the data in a specific way. We're using a language called SQL to set up tables, put in data, ask questions, and evaluate how well it all works.

The rules for our database are based on how the store manages products, sorts customers, handles orders, works with vendors, keeps track of inventory, and processes payments. We've written specific questions in SQL to get the information we need. Remember, we have to follow the rules, use a program called Oracle SQL PLUS, and include a special file with all our data.

To do well in this assignment, we need to really understand how databases work, be good at using SQL, and be able to evaluate how effective our setup is.

10 References

Biscobing, J., 2019. *techtarget*. [Online]

Available at: <https://www.techtarget.com/searchdatamanagement/definition/entity-relationship-diagram-ERD>

[Accessed 12 1 2024].

Biscobing, J., 2023. *techtarget*. [Online]

Available at: <https://www.techtarget.com/searchdatamanagement/definition/entity-relationship-diagram-ERD>

[Accessed 3 1 2024].

byjus, 2023. *byjus*. [Online]

Available at: <https://byjus.com/gate/second-normal-form-in-dbms/>

[Accessed 7 1 2024].

geeksforgeeks, 2023. *geeksforgeeks*. [Online]

Available at: <https://www.geeksforgeeks.org/third-normal-form-3nf/>

[Accessed 7 1 2024].

Peterson, R., 2023. *guru99*. [Online]

Available at: https://www.guru99.com/database-normalization.html?gpp&gpp_sid

[Accessed 5 1 2024].

Raipurkar, 2012. *proquest*. [Online]

Available at:

<https://www.proquest.com/docview/1443724655?sourcetype=Scholarly%20Journals>

[Accessed 6 1 2024].

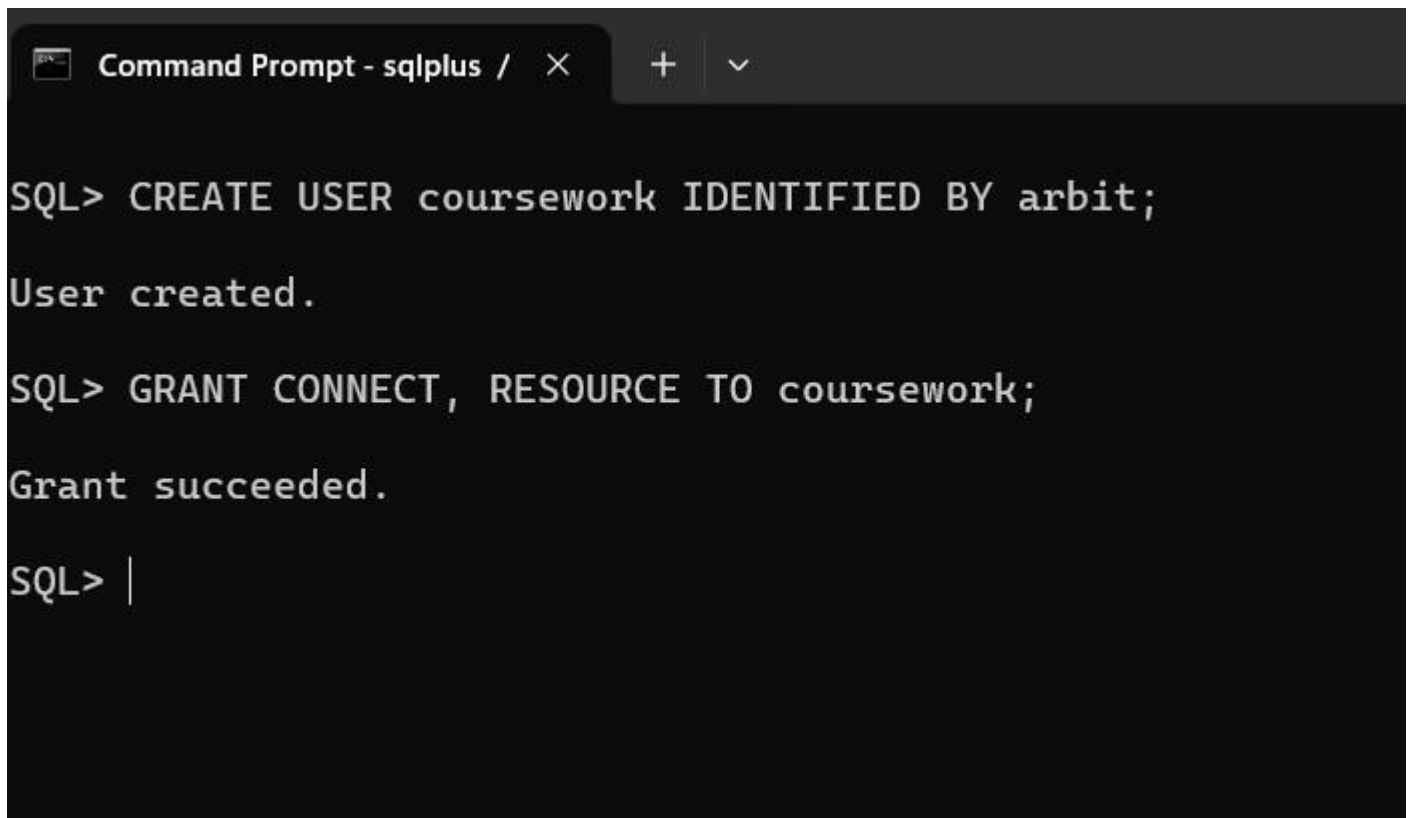
Rouse, M., 2011. *techopedia*. [Online]

Available at: <https://www.techopedia.com/definition/25955/first-normal-form-1nf>

[Accessed 6 1 2023].

11 APPENDIX

I created a user and granted it the required permissions to connect to the database and use necessary resources for my project. This helps in organizing and securing access to the database, ensuring the user has the right permissions to perform the project tasks.

A screenshot of a Windows Command Prompt window titled "Command Prompt - sqlplus /". The window has a dark background with white text. The text shows the execution of two SQL commands. The first command is "SQL> CREATE USER coursework IDENTIFIED BY arbit;" followed by the response "User created.". The second command is "SQL> GRANT CONNECT, RESOURCE TO coursework;" followed by the response "Grant succeeded.". The prompt "SQL> |" is visible at the bottom, indicating the command line is ready for input.

```
Command Prompt - sqlplus /  
  
SQL> CREATE USER coursework IDENTIFIED BY arbit;  
User created.  
  
SQL> GRANT CONNECT, RESOURCE TO coursework;  
Grant succeeded.  
  
SQL> |
```

Figure 39 Screenshot of Creating User

In my project, I've used the COMMIT command in Oracle SQL. This command is crucial because it's like the final stamp that makes all the changes I've made to the database permanent. Once I issue a COMMIT, it locks in the updates, inserts, or deletes I've done during the current transaction. This way, the changes become officially part of the database, and other parts of my project or even other users can now see and work with the updated data.

```
SQL> Commit;

Commit complete.

SQL>
```

Figure 40 Screenshot of Commit

In my project, I've used the SPOOL command a lot to save the results of different queries. I have a total of 10 spool files, but I'll share a snapshot of just one for simplicity. Each spool file has important info, showing the output of various queries I ran. This one screenshot gives you an idea of what's in the multiple spool files, giving a sneak peek into the different analyses and data extractions done throughout the project.

```
SQL> spool C:\Users\arbit\Desktop\spool\Query1.sql
SQL> SELECT *
  2  FROM Customer
  3  WHERE Customer_Category = 'Staff';

CUSTOMER_ID CUSTOMER_NAME      CUSTOMER_ADDRESS      CUSTOMER_CATEGO  CUSTOMER_DISCOUNT
-----
          2 Suman Khatri        Lalitpur              Staff              5
          4 Anju Shakya        Butwal                Staff              5
          7 Manish Thapa        Biratnagar            Staff              5
          9 Sujan Khatri        Lalitpur              Staff              5

SQL> spool off
SQL>
```

Figure 41 Screenshot of Spool