# BUAN 6320.0W2 - Group Project Technical Report
## Group No.3

**Members:** Roopali Reddy Kallem (RXK180054)
Nguyen Thanh Nguyen (NXN210036)
Nihar Vinnakota (NXV230017)
Haritha Jampani (HXJ220031)

## Introduction

This report provides a detailed explanation of database management, emphasizing the hands-on use of SQL commands in a PostgreSQL setting. The database, designed with entities like Customers, Orders, Products, Suppliers, and Payments, mirrors the interconnectedness and dynamism of business processes. Various SQL operations are explored, ranging from table creation and modification to data insertion, with a strong focus on maintaining data integrity and relationships. Advanced queries, utilizing subqueries and aggregate functions, uncover valuable insights and showcase the database's flexibility. Additionally, common challenges, such as managing foreign key constraints during deletion operations, are addressed, demonstrating the commitment to preserving database integrity while accommodating complex data manipulation needs. The overarching goal is to present a comprehensive understanding of the database's capabilities, ensuring a reliable and efficient system that supports real-world business applications.

## Overview

The database at the core of our discussion is structured around six key entities: Customers, Orders, Products, Suppliers, Payments, and Orders_Products, each represented by meticulously designed tables. These entities are intricately linked to mirror real-world business relationships and transactions.

Customers: The foundational entity of our database, capturing essential customer information, forms the basis of our transactional data, storing details like names, contacts, and addresses.
Orders: This table tracks the lifecycle of customer orders, encompassing various facets such as quantities, amounts, statuses, and delivery specifics, thereby connecting customers with their purchases.
Products: Here, we manage our product inventory, detailing items with descriptions, pricing, and stock status, which is pivotal for inventory control and sales analysis.
Suppliers: The Supplier table is crucial for supply chain management, recording supplier details, product associations, and supply frequencies, ensuring seamless inventory replenishment.
Payments: This entity captures the financial transactions associated with customer orders, detailing payment methods, amounts, and statuses, and is vital for financial tracking and analysis.
Orders_Products: Serving as an associative table, it links Orders and Products, providing insights into the composition of each order and the popularity of products.

Our approach also involves the use of ERDs (Entity-Relationship Diagrams) to visually represent the database schema, elucidating the relationships and dependencies between different data elements. This visual aid is instrumental in both understanding and communicating the database structure effectively. Overall, the database is not just a tool for data storage but a comprehensive system for insightful analysis and strategic business support.

## Assumptions and Special Considerations

- If a payment fails initially and the customer retries, we are considering the latest transaction as a new entry in the database.
- As mentioned in the business rules, we are assuming that each product will only be supplied by one supplier.
- Assuming that all entities are unique and can be identified by their primary keys.
- Constraint: ORDER_PRODUCT is an associative entity (i.e., join tables) and as such do not meet the five-attribute minimum, and the 2 foreign keys (Order_ID_FK and Product_ID_FK) combine to form composite PK.

## Requirements Definition Document

**Business Rules**

1. One CUSTOMER can order zero or many ORDERS.
2. One ORDER must be placed by on CUSTOMER
3. One ORDER can contain one or many PRODUCT.
4. Each PRODUCT can be associated with one or more ORDERS.
5. One ORDER can contain one or many ORDER_PRODUCT.
6. Each ORDER_PRODUCT must be associated with one ORDER.
7. One PRODUCT may appear in one or many ORDER_PRODUCT.
8. Each ORDER_PRODUCT corresponds to a specific PRODUCT .
9. One SUPPLIER may provide one or many PRODUCTS.
10. Each PRODUCT must be provided by one and only one SUPPLIER.
11. One CUSTOMER may have one or multiple PAYMENTS.
12. Each PAYMENT must be must be associated with one and only one CUSTOMER

**Entity and Attribute Description**:

Entity Name: CUSTOMER
Entity Description: The primary user of the e-commerce website.
Main Attributes of CUSTOMER:

Customer_ID: (Primary Key) A unique Identifier for the customer's account
Cust_Fname: Customer First name
Cust_Lname: Customer Last name

Cust_phone_no: Customer's phone number to contact
Cust_Email: customer's email to verify or notify customer about their order
Cust_Address: Customer address where to deliver the order
Created_by: who created the account (admin)
Date_created: Date the account was created
Modified_by: who lately updated it or modified the account
Date_modified: date account was last modified on

Entity Name: ORDER
Entity Description: The list of orders of the customers.
Main Attributes of ORDER:
Order_ID: (Primary Key) A unique Identifier for the customer's order
Customer_ID: (Foreign Key) The ID of the customer ordering
Total_quantity: Total quantity of items in current order
Total_Amount: Total amount for the current order
Order_Status: Status of the current order(delivered, received, pending, shipped)
Payment_Status: Status of the payment (paid, unpaid, refunded)
Delivery_Date: Date the order is delivered if delivered
Shipping_Address: Address where the order is to be shipped
Created_by: who created the order
Date_created: Date the order was created
Modified_by: who lately updated it or modified the order
Date_modified: date order was last modified on

Entity Name: PAYMENT
Entity Description: Payment method for the order
Main Attributes of PAYMENT:
Address_on_Card: Address on payment card
Customer_ID: (Foreign Key) The ID of Customer who is making the payment
Payment_Method: The method of payment (MasterCard, VisaCard)
Time_stamp: Date and time of the payment
Payment_Amount: Amount paid
Payment_Status: Status of the payment (paid, unpaid, refunded)
Created_by: who created/saved the Payment
Date_created: Date the Payment method was created/saved
Modified_by: who lately updated it or modified the Payment method
Date_modified: date payment method was last modified on

Entity Name: ORDER_PRODUCT
Entity Description: individual product from each order

Main Attributes of ORDER_PRODUCT:

- Order_ID: (Foreign Key) A unique Identifier for the customer's order
- Product_ID: (Foreign Key) A unique Identifier for the product
- Quantity: Quantity of the current product in current order
- Product_Price: Price of the product being ordered
- Product_Category: Category in which the product belongs
- Created_by: who created the order
- Date_created: Date the order was created
- Modified_by: who lately updated it or modified the order
- Date_modified: date order was last modified on

Entity Name: PRODUCT
Entity Description: List of the Products
Main Attributes of PRODUCT:

- Product_ID: (Primary Key) A unique Identifier for the customer's account
- Inventory_Status: Tells if the product is available or out of stock
- Product_Name: Name of the product
- Description: Description of the product or how to use it
- Estimated_Delivery: Estimated delivery date from the day of order placement
- Product_Price: price of the product
- Created_by: who created the product
- Date_created: Date the product was created
- Modified_by: who lately updated it or modified the product
- Date_modified: date product was last modified on

Entity Name: SUPPLIER
Entity Description:
Main Attributes of SUPPLIER:

- Supplier_ID: (Primary Key) A unique Identifier for the Supplier's account
- Product_ID: (Foreign Key) A unique Identifier for the product
- Supplier_Name: name of the supplier
- Supplier_phone_no: Contact number of the supplier
- Suplly_Frequency: How often does the supplier deliver or stock the supply each month
- Supplying_Since: Date since when the supplier has been associated and supplying the product
- Created_by: who created the supplier details
- Date_created: Date the supplier details was created
- Modified_by: who lately updated it or modified the supplier details
- Date_modified: date supplier details was last modified on

**Relationship and Cardinality Description :**

Relationship: ORDER between CUSTOMER and ORDER
Cardinality: 1:M between Mandatory CUSTOMER and Mandatory ORDER.
Business rule: one CUSTOMER can order zero or many ORDERS, one ORDER must be placed by on CUSTOMER

Relationship: HAS between ORDER and ORDER_PRODUCT
Cardinality: 1:M between Mandatory ORDER and Mandatory ORDER_PRODUCT
Business rule: one ORDER can contain one or many ORDER_PRODUCT, each ORDER_PRODUCT must be associated with one ORDERS

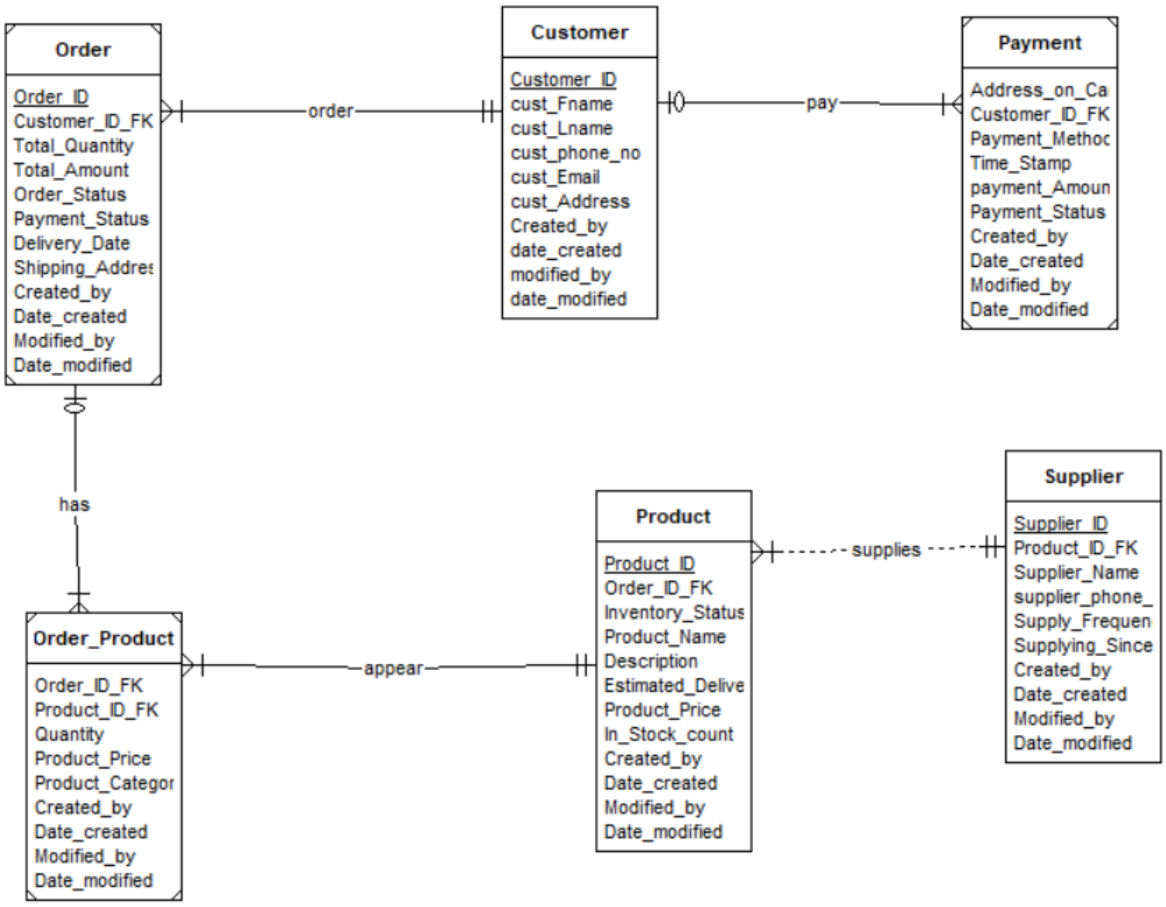
Relationship: APPEAR between ORDER_PRODUCT and PRODUCT
Cardinality: M:1 between Mandatory ORDER_PRODUCT and Mandatory PRODUCT
Business rule: one PRODUCT may appear in one or many ORDER_PRODUCT, each ORDER_PRODUCT corresponds to a specific PRODUCT

Relationship: SUPPLIES between SUPPLIER and PRODUCT
Cardinality: 1:M between Mandatory SUPPLIER and Mandatory PRODUCT
Business rule: one SUPPLIER may provide one or many PRODUCTS, each PRODUCT must be provided by one and only one SUPPLIER

Relationship: PAY between CUSTOMER and PAYMENT
Cardinality: 1:M between Optional CUSTOMER and Mandatory PAYMENT
Business rule: one CUSTOMER may have one or multiple PAYMENTS , each PAYMENT must be must be associated with one and only one CUSTOMER

## Detailed Database Design

**Entity Relationship Diagram**
(Note: Due to limitations in ER-Assistant, the software used to create this entity-relationship diagram, it is not possible to resize the entity boxes to avoid truncating entity and attribute names.)




**DDL Source Code**

```
set search_path to public;

-- Dropping the Tables
DROP TABLE  IF EXISTS Orders_Product;
DROP TABLE  IF EXISTS Customer;
DROP TABLE  IF EXISTS Orders;
DROP TABLE  IF EXISTS Product;
DROP TABLE  IF EXISTS Supplier;
DROP TABLE  IF EXISTS Payment;
```

```
-- Creating Tables

-- Create Customer table
CREATE TABLE Customer (
   Customer_ID VARCHAR(10) PRIMARY KEY,
   cust_Fname VARCHAR(50),
   cust_LName VARCHAR(50),
   Phone_no VARCHAR(11),
   cust_Email VARCHAR(50),
   cust_Address VARCHAR(50)
);

-- Create Orders table
CREATE TABLE Orders (
   Order_ID VARCHAR(10) PRIMARY KEY,
   Customer_ID VARCHAR(10),
   Total_Quantity INTEGER,
   Total_Amount VARCHAR(10),
   Order_Status VARCHAR(20),
   Payment_Status VARCHAR(20),
   Delivery_Date DATE,
   Shipping_Address VARCHAR(50),
         FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
);

-- Create Product table
CREATE TABLE Product (
   Product_ID VARCHAR(10) PRIMARY KEY,
   Inventory_Status VARCHAR(50),
   Product_Name VARCHAR(50),
   Description VARCHAR(50),
   Estimated_Delivery_Date DATE,
   Product_Price VARCHAR(10)
);

-- Create Orders_Product table for the M:N relationship
CREATE TABLE Orders_Product (
   Order_ID VARCHAR(10),
   Product_ID VARCHAR(10),
   Quantity INTEGER,
   Product_Price VARCHAR(10),
   Product_Category VARCHAR(20),
```

```
        FOREIGN KEY (Order_ID) REFERENCES Orders(Order_ID),
        FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID)
);

-- Create Supplier table (Weak Entity)
CREATE TABLE Supplier (
   Supplier_ID VARCHAR(10) PRIMARY KEY,
   Product_ID VARCHAR(10),
   Supplier_Name VARCHAR(50),
   Supplier_Phone_no VARCHAR(11),
   Supply_Frequency VARCHAR(20),
   Supplying_Since DATE,
        FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID)
);

-- Create Payment table (Weak Entity)
CREATE TABLE Payment (
   Payment_ID VARCHAR(10) PRIMARY KEY,
   Customer_ID VARCHAR(10),
   Payment_Method VARCHAR(50),
   Time_Stamp TIMESTAMP,
   Payment_Amount INTEGER,
   Payment_Status VARCHAR(50),
        FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
);




-- Dropping Sequences
DROP SEQUENCE IF EXISTS Customer_id_seq;
DROP SEQUENCE IF EXISTS Order_id_seq;
DROP SEQUENCE IF EXISTS Product_id_seq;
DROP SEQUENCE IF EXISTS Supplier_id_seq;
DROP SEQUENCE IF EXISTS Payment_id_seq;


-- Creating Sequences for Student, Course, Hod and Teacher tables.
CREATE SEQUENCE Customer_id_seq
start with 1000
increment by 1
NO MAXVALUE
```

```
MINVALUE 1000;

CREATE SEQUENCE Order_id_seq
start with 2000
increment by 1
NO MAXVALUE
MINVALUE 2000;

CREATE SEQUENCE Product_id_seq
start with 3000
increment by 1
NO MAXVALUE
MINVALUE 3000;

CREATE SEQUENCE Supplier_id_seq
start with 4000
increment by 1
NO MAXVALUE
MINVALUE 4000;

CREATE SEQUENCE Payment_id_seq
start with 5000
increment by 1
NO MAXVALUE
MINVALUE 5000;



-- Dropping Triggers and Trigger Functions
DROP TRIGGER IF EXISTS Customer_id_auto_increment_trigger on Customer;
DROP FUNCTION IF EXISTS Customer_id_auto_increment;

DROP TRIGGER IF EXISTS Order_id_auto_increment_trigger on Orders;
DROP FUNCTION IF EXISTS Order_id_auto_increment;

DROP TRIGGER IF EXISTS Product_id_auto_increment_trigger on Product;
DROP FUNCTION IF EXISTS Product_id_auto_increment;

DROP TRIGGER IF EXISTS Supplier_id_auto_increment_trigger on Supplier;
DROP FUNCTION IF EXISTS Supplier_id_auto_increment;

DROP TRIGGER IF EXISTS Payment_id_auto_increment_trigger on Payment;
```

```
DROP FUNCTION IF EXISTS Payment_id_auto_increment;



-- Creating Trigger and Trigger Function for Customer Table
CREATE OR REPLACE FUNCTION Customer_id_auto_increment()
RETURNS TRIGGER AS $$
BEGIN
  NEW.Customer_ID = NEXTVAL('Customer_id_seq');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER Customer_id_auto_increment_trigger
BEFORE INSERT ON Customer
FOR EACH ROW
EXECUTE FUNCTION Customer_id_auto_increment();


-- Creating Trigger and Trigger Function for Order Table
CREATE OR REPLACE FUNCTION Order_id_auto_increment()
RETURNS TRIGGER AS $$
BEGIN
  NEW.Order_ID = NEXTVAL('Order_id_seq');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER Order_id_auto_increment_trigger
BEFORE INSERT ON Orders
FOR EACH ROW
EXECUTE FUNCTION Order_id_auto_increment();


-- Creating Trigger and Trigger Function for Product Table
CREATE OR REPLACE FUNCTION Product_id_auto_increment()
RETURNS TRIGGER AS $$
BEGIN
  NEW.Product_ID = NEXTVAL('Product_id_seq');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER Product_id_auto_increment_trigger
BEFORE INSERT ON Product
FOR EACH ROW
EXECUTE FUNCTION Product_id_auto_increment();


-- Creating Trigger and Trigger Function for Supplier Table
CREATE OR REPLACE FUNCTION Supplier_id_auto_increment()
RETURNS TRIGGER AS $$
BEGIN
  NEW.Supplier_ID = NEXTVAL('Supplier_id_seq');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER Supplier_id_auto_increment_trigger
BEFORE INSERT ON Supplier
FOR EACH ROW
EXECUTE FUNCTION Supplier_id_auto_increment();


-- Creating Trigger and Trigger Function for Payment Table
CREATE OR REPLACE FUNCTION Payment_id_auto_increment()
RETURNS TRIGGER AS $$
BEGIN
  NEW.Payment_ID = NEXTVAL('Payment_id_seq');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER Payment_id_auto_increment_trigger
BEFORE INSERT ON Payment
FOR EACH ROW
EXECUTE FUNCTION Payment_id_auto_increment();


-- Dropping View
DROP VIEW IF EXISTS OrderDetails;
```

```
-- Creating view
CREATE OR REPLACE VIEW OrderDetails AS
SELECT
   o.Order_ID,
   c.cust_Fname || ' ' || c.cust_LName AS Customer_Name,
   p.Product_Name,
   op.Quantity,
   op.Product_Price
FROM Orders o
JOIN Customer c ON o.Customer_ID = c.Customer_ID
JOIN Orders_Product op ON o.Order_ID = op.Order_ID
JOIN Product p ON op.Product_ID = p.Product_ID;

---Alter Tables by adding Audit Columns
ALTER TABLE Customer
ADD COLUMN created_by VARCHAR(30),
ADD COLUMN date_created DATE,
ADD COLUMN modified_by VARCHAR(30),
ADD COLUMN date_modified DATE;

ALTER TABLE Orders
ADD COLUMN created_by VARCHAR(30),
ADD COLUMN date_created DATE,
ADD COLUMN modified_by VARCHAR(30),
ADD COLUMN date_modified DATE;

ALTER TABLE Product
ADD COLUMN created_by VARCHAR(30),
ADD COLUMN date_created DATE,
ADD COLUMN modified_by VARCHAR(30),
ADD COLUMN date_modified DATE;


ALTER TABLE Orders_Product
ADD COLUMN created_by VARCHAR(30),
ADD COLUMN date_created DATE,
ADD COLUMN modified_by VARCHAR(30),
ADD COLUMN date_modified DATE;


ALTER TABLE Supplier
ADD COLUMN created_by VARCHAR(30),
```

```
ADD COLUMN date_created DATE,
ADD COLUMN modified_by VARCHAR(30),
ADD COLUMN date_modified DATE;


ALTER TABLE Payment
ADD COLUMN created_by VARCHAR(30),
ADD COLUMN date_created DATE,
ADD COLUMN modified_by VARCHAR(30),
ADD COLUMN date_modified DATE;
```

**DML Source Code**

```
set search_path to public;

-- Insert a new customer record
INSERT INTO Customer (Customer_ID, cust_Fname, cust_LName, Phone_no, cust_Email, cust_Address,created_by, date_created, modified_by,date_modified)
VALUES
    (NEXTVAL('Customer_id_seq'), 'John', 'Doe', '1234567890', 'john.doe@example.com', '123 Main Street', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Customer_id_seq'), 'Jane', 'Smith', '2345678901', 'jane.smith@example.com', '234 Oak Avenue', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Customer_id_seq'), 'Alice', 'Johnson', '3456789012', 'alice.johnson@example.com', '345 Pine Road', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Customer_id_seq'), 'Bob', 'Brown', '4567890123', 'bob.brown@example.com', '456 Maple Lane', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Customer_id_seq'), 'Charlie', 'Davis', '5678901234', 'charlie.davis@example.com', '567 Cedar Blvd', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Customer_id_seq'), 'Alex', 'Bay', '3456486012', 'alex.bay@example.com', '346 Pine Road', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Customer_id_seq'), 'Cobb', 'Brown', '4567894893', 'cobb.brown@example.com', '456 Mappleson Lane', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Customer_id_seq'), 'Sophie', 'Hightower', '5482101234', 'sophie.hightower@example.com', '567 Cedarpine Blvd', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Customer_id_seq'), 'Cobber', 'Bron', '4567894823', 'cobber.bron@example.com', '456 Map Lane', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Customer_id_seq'), 'Slesha', 'Higher', '9162101234', 'slesha.higher@example.com', '567 Cedarpine Dr', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE);


select * from Customer;

-- Insert a new order record
INSERT INTO Orders (Order_ID, Customer_ID, Total_Quantity, Total_Amount, Order_Status, Payment_Status, Delivery_Date, Shipping_Address,created_by,date_created,modified_by,date_modified)
VALUES
    (NEXTVAL('Order_id_seq'), '1001', 2, 100.00, 'Pending', 'Unpaid', '2023-10-31', '123 Main Street', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Order_id_seq'), '1003', 3, 200.00, 'Delivered', 'Paid', '2023-12-18', '234 Oak Avenue', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
```

```
(NEXTVAL('Order_id_seq'), '1005', 5, 50.00, 'Processing', 'Unpaid', NULL, '345 Pine Road', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Order_id_seq'), '1007', 7, 500.00, 'Delivered', 'Paid', '2023-12-15', '456 Maple Lane', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Order_id_seq'), '1009', 9, 75.00, 'Shipped', 'Paid', '2023-12-22', '567 Cedar Blvd', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Order_id_seq'), '1011', 2, 100.00, 'Pending', 'Unpaid', '2023-10-31', '1253 Main Street', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Order_id_seq'), '1009', 3, 200.00, 'Delivered', 'Unpaid', '2023-12-18', '2314 Oaks Avenue', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Order_id_seq'), '1015', 5, 50.00, 'Processing', 'Unpaid', NULL, '3445 Pine Road', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Order_id_seq'), '1017', 7, 500.00, 'Delivered', 'Paid', '2023-12-15', '4596 Maple Lane', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Order_id_seq'), '1013', 9, 75.00, 'Shipped', 'Paid', '2023-12-22', '5367 Cedarcone Blvd', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE);

select * from Orders;

-- Insert a new product record
INSERT INTO Product (Product_ID, Inventory_Status, Product_Name, Description, Estimated_Delivery_Date, Product_price,created_by,date_created,modified_by,date_modified)
VALUES
(NEXTVAL('Product_id_seq'), 'In Stock', 'Apple iPhone 14 Pro', 'The latest iPhone from Apple', '2023-11-04', 999.00, 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Product_id_seq'), 'In Stock', 'Widget A', 'A useful widget', '2023-12-25', 25.00, 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Product_id_seq'), 'Out of Stock', 'Gadget B', 'An interesting gadget', '2024-01-05', 40.00, 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Product_id_seq'), 'In Stock', 'Tool C', 'A durable tool', '2023-12-30', 15.00, 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Product_id_seq'), 'In Stock', 'Appliance D', 'An essential appliance', '2023-12-20', 100.00, 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Product_id_seq'), 'In Stock', 'Apple iPhone 13 Pro', 'iPhone from Apple', '2023-11-04', 999.00, 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Product_id_seq'), 'In Stock', 'Kids book', 'A writing book for kids', '2023-12-25', 25.00, 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Product_id_seq'), 'Out of Stock', 'Story book', 'An interesting story book', '2024-01-05', 40.00, 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Product_id_seq'), 'In Stock', 'Spanner kit', 'A mechanical tool', '2023-12-30', 15.00, 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
```

```
    (NEXTVAL('Product_id_seq'), 'In Stock', 'Drone', 'A remote control gadget', '2023-12-20', 100.00, 'admin', CURRENT_DATE, 'admin', CURRENT_DATE);

select * from Product;

-- Insert a new supplier record
INSERT INTO Supplier (Supplier_ID, Product_ID, Supplier_Name, supplier_phone_no, Supply_Frequency, Supplying_Since, created_by, date_created, modified_by, date_modified)
VALUES
    (NEXTVAL('Supplier_id_seq'), '3001', 'Apple Inc.', '8001234566', 'Monthly', '2010-01-01', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
   (NEXTVAL('Supplier_id_seq'), '3003', 'Acme Corp', '8001234567', 'Monthly', '2018-01-01', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Supplier_id_seq'), '3005', 'Globex Inc', '8002345678', 'Weekly', '2019-05-15', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  (NEXTVAL('Supplier_id_seq'), '3007', 'Soylent Corp', '8003456789', 'Bi-Weekly', '2020-06-20', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
   (NEXTVAL('Supplier_id_seq'), '3009', 'Initech LLC', '8004567890', 'Quarterly', '2018-07-30', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
       (NEXTVAL('Supplier_id_seq'), '3011', 'Apple Inc.', '8001234576', 'Monthly', '2010-01-01', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  (NEXTVAL('Supplier_id_seq'), '3015', 'SR Publication', '8001234777', 'Monthly', '2018-01-01', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  (NEXTVAL('Supplier_id_seq'), '3013', 'Shipping and Co', '8002346378', 'Weekly', '2019-05-15', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
   (NEXTVAL('Supplier_id_seq'), '3017', 'Hard Corp', '8003126789', 'Bi-Weekly', '2020-06-20', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Supplier_id_seq'), '3019', 'Intel LLC', '9104567890', 'Quarterly', '2018-07-30', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE);

select * from Supplier;

-- Insert a new payment record
INSERT INTO Payment (Payment_ID, Customer_ID, Payment_Method, Time_Stamp, payment_Amount, Payment_Status, created_by, date_created, modified_by, date_modified)
VALUES
    (NEXTVAL('Payment_id_seq'), '1001', 'Credit Card', '2023-10-31 17:18:50', 100.00, 'Paid', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
     (NEXTVAL('Payment_id_seq'), '1003', 'Visa', '2023-11-01 10:00:00', 200.00, 'Unpaid', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
    (NEXTVAL('Payment_id_seq'), '1005', 'MasterCard', '2023-11-02 11:00:00', 50.00, 'Paid', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
```

```
(NEXTVAL('Payment_id_seq'), '1007', 'PayPal', '2023-11-03 12:00:00', 500.00, 'Unpaid', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Payment_id_seq'), '1009', 'Visa', '2023-11-04 13:00:00', 275.00, 'Paid', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Payment_id_seq'), '1011', 'Credit Card', '2023-01-31 17:18:50', 100.00, 'Paid', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Payment_id_seq'), '1013', 'Visa', '2023-12-01 10:00:00', 75.00, 'Unpaid', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Payment_id_seq'), '1015', 'MasterCard', '2023-11-02 11:00:00', 50.00, 'Paid', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Payment_id_seq'), '1017', 'PayPal', '2023-11-03 12:00:00', 500.00, 'Unpaid', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
(NEXTVAL('Payment_id_seq'), '1019', 'Visa', '2023-11-04 13:00:00', 0.00, 'Paid', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE);

select * from Payment;

--Insert into order_product table
INSERT INTO Orders_Product (Order_ID, Product_ID, Quantity, Product_Price, Product_Category, created_by, date_created, modified_by, date_modified)
VALUES
  ('2001', '3001', 2, 25.00, 'Electronics', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2003', '3019', 1, 100.00, 'Electronics', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2005', '3013', 2, 25.00, 'HouseHold', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2007', '3019', 2, 100.00, 'Appliances', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2009', '3003', 3, 25.00, 'Technology', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
('2001', '3013', 2, 25.00, 'Electronics', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2003', '3009', 1, 100.00, 'Electronics', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2007', '3009', 1, 100.00, 'Hardware', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2007', '3003', 4, 25.00, 'Appliances', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2007', '3013', 4, 25.00, 'Appliances', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
('2013', '3019', 2, 100.00, 'Technology', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2015', '3013', 2, 25.00, 'Appliances', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
('2017', '3019', 5, 100.00, 'Technology', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2019', '3013', 3, 25.00, 'Appliances', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE),
  ('2011', '3019', 1, 100.00, 'Technology', 'admin', CURRENT_DATE, 'admin', CURRENT_DATE);


select * from Orders_Product;
```

**Queries**

----Query 1: Select all columns and all rows from one table
```
SELECT * FROM customer;
```

----Query 2: Select five columns and all rows from one table
```
SELECT order_id, customer_id, total_quantity, total_amount, order_status
FROM orders;
```

----Query 3: Select all columns from all rows from one view
```
SELECT * FROM OrderDetails;
```

----Query 4: Using a join on 2 tables, select all columns and all rows from the tables without the use of a Cartesian product
```
SELECT *
FROM customer c LEFT OUTER JOIN orders o
        ON c.customer_id = o.customer_id ;
```

----Query 5: Select and order data retrieved from one table
```
SELECT * FROM orders
ORDER BY delivery_date ;
```

----Query 6: Using a join on 3 tables, select 5 columns from the 3 tables. Use syntax that would limit the output to 10 rows

```
SELECT c.customer_id, c.cust_lname, o.order_id, o.total_amount, p.payment_status
FROM customer c INNER JOIN orders o ON c.customer_id = o.customer_id
        INNER JOIN payment p ON c.customer_id = p.customer_id
FETCH FIRST 10 ROWS ONLY;
```

----Query 7: Select distinct rows using joins on 3 tables

```
SELECT DISTINCT *
FROM customer c INNER JOIN orders o ON c.customer_id = o.customer_id
        INNER JOIN payment p ON c.customer_id = p.customer_id ;
```

----Query 8: Use GROUP BY and HAVING in a select statement using one or more tables
```
SELECT pr.product_id, pr.product_name, s.supplier_id, s.supplier_name, pr.product_price
FROM product pr INNER JOIN supplier s ON pr.product_id = s.product_id
GROUP BY pr.product_id, pr.product_name, s.supplier_id, s.supplier_name, pr.product_price
```

```
HAVING pr.product_price = '999.00' ;

----Query 9: Use IN clause to select data from one or more tables
SELECT * FROM customer
WHERE customer_id IN ('1001', '1003', '1005');

----Query 10: Select length of one column from one table (use LENGTH function)
SELECT LENGTH(cust_address) FROM customer;


----Query 11: Delete one record from one table. Use select statements to demonstrate the table contents before and after the DELETE statement. Make sure you use ROLLBACK
----afterwards so that the data will not be physically removed
BEGIN;
-- Display data before deletion
SELECT * FROM Product;

-- Temporarily remove reference from Supplier
DELETE FROM Supplier WHERE Product_ID = '3003';

DELETE FROM Orders_Product WHERE Product_ID = '3003';

-- Delete the product
DELETE FROM Product
WHERE Product_Price = '25.00';

-- Display data after deletion
SELECT * FROM Product;

-- Rollback the transaction
ROLLBACK;


----Query 12: Update one record from one table. Use select statements to demonstrate the table contents before and after the UPDATE statement. Make sure you use ROLLBACK
----afterwards so that the data will not be physically removed

BEGIN;

SELECT * FROM supplier;

UPDATE supplier
```

```
SET supply_frequency = 'Mondthly'
WHERE supplier_name = 'Apple Inc.';

SELECT * FROM supplier;
ROLLBACK;

--Advanced Query 1: Supplier Reliability Report
---This query generates a report on supplier reliability by comparing the estimated and actual delivery dates of products. It involves a sub-query to get the earliest order date per product.
SELECT
   s.Supplier_Name,
   p.Product_Name,
   p.Estimated_Delivery_Date,
   MIN(o.Delivery_Date) AS Actual_Delivery_Date,
   CASE
      WHEN MIN(o.Delivery_Date) <= p.Estimated_Delivery_Date THEN 'On Time'
      ELSE 'Delayed'
   END AS Delivery_Status
FROM Supplier s
JOIN Product p ON s.Product_ID = p.Product_ID
JOIN Orders_Product op ON p.Product_ID = op.Product_ID
JOIN Orders o ON op.Order_ID = o.Order_ID
GROUP BY s.Supplier_Name, p.Product_Name, p.Estimated_Delivery_Date
ORDER BY s.Supplier_Name, Delivery_Status;


--Advanced Query 2: Customer Spending Analysis by Payment Method
SELECT
   c.cust_Fname || ' ' || c.cust_LName AS Customer_Name,
   p.Payment_Method,
   COUNT(*) AS Number_of_Orders,
   SUM(p.Payment_Amount) AS Total_Spent,
   CASE
      WHEN SUM(p.Payment_Amount) > 500 THEN 'High Spender'
      WHEN SUM(p.Payment_Amount) BETWEEN 200 AND 500 THEN 'Medium Spender'
      ELSE 'Low Spender'
   END AS Spending_Category
FROM Payment p
JOIN Customer c ON p.Customer_ID = c.Customer_ID
GROUP BY c.cust_Fname, c.cust_LName, p.Payment_Method
ORDER BY Total_Spent DESC;
```

```
--Advanced Query3 with Subquery: Identifying High Spending Customers

SELECT
   c.cust_Fname || ' ' || c.cust_LName AS Customer_Name,
   COUNT(p.Payment_ID) AS Number_of_Payments,
   SUM(p.Payment_Amount::numeric) AS Total_Spending
FROM Customer c
JOIN Payment p ON c.Customer_ID = p.Customer_ID
GROUP BY c.Customer_ID, c.cust_Fname, c.cust_LName
HAVING SUM(p.Payment_Amount::numeric) > (
   SELECT AVG(Total_Amount::numeric)
   FROM (
      SELECT
         Customer_ID,
         SUM(Payment_Amount::numeric) AS Total_Amount
      FROM Payment
      GROUP BY Customer_ID
   ) AS SubQuery
)
ORDER BY Total_Spending DESC;
```

In conclusion, the development and analysis of this relational database system, as detailed in our report, underscore the pivotal role of structured data management in modern business environments. The system, encompassing the entities of Customers, Orders, Products, Suppliers, Payments, and Orders_Products, is a testament to the power of efficient data organization and relational database design. Through the implementation of advanced SQL queries and the strategic handling of data integrity challenges, we have demonstrated the system's capability to not only store and manage data but also to provide valuable insights that are essential for informed business decision-making. The use of Entity-Relationship Diagrams (ERDs) has been instrumental in visualizing and understanding the complex interrelationships between the database entities, further reinforcing the system's integrity and coherence. Overall, this database system stands as a robust, dynamic tool, integral to supporting and driving business processes, and it sets a strong foundation for future enhancements and scalability in the ever-evolving landscape of data-driven business operations.