AUTOMOBILE SUPPLY CHAIN MANAGEMENT SYSTEM

1st Devendra Mandava, dmandava
Computer Science and Engineering
University at Buffalo
Buffalo, United States
dmandava@buffalo.edu

2nd Hemanth Kongara, hkongara
Computer Science and Engineering
University at Buffalo
Buffalo, United States
hkongara@buffalo.edu

3d Sethu Thakkilapati, sethutha
Computer Science and Engineering
University at Buffalo
Buffalo, United States
sethutha@buffalo.edu

***Abstract*—The Car Supply Chain Management System (CSCMS) serves as a specialized database solution designed to overcome the limitations of Excel in managing the complex automotive supply chain. This database system enhances operational efficiency by providing scalable data management, real-time updates, and collaborative features for multiple users. With relationships mapped between suppliers, products, and customers, the CSCMS ensures seamless transactional flow and data integrity. The adoption of the CSCMS is expected to streamline processes, reduce costs, and elevate customer satisfaction, thereby reinforcing the industry's competitiveness and adaptability.**

***Index Terms*—Automotive supply chain, database system, operational efficiency, data management, real-time updates, collaborative features, data integrity, transactional flow, process streamlining, cost reduction, customer satisfaction, industry competitiveness, adaptability**

I. Introduction

The Car Supply Chain Management System project aims to revolutionize the automotive industry by optimizing the end-to-end processes involved in sourcing, manufacturing, and delivering vehicles to customers. Leveraging a comprehensive database schema, this project meticulously organizes data into distinct tables for Suppliers, Products, Customers, Orders, Order Details, and Payment Information. Through efficient data management and insightful analytics, the project endeavors to enhance supply chain transparency, streamline operations, and ultimately elevate customer satisfaction.

II. Problem Statement

The automotive industry faces formidable challenges in managing its complex supply chain, characterized by multiple suppliers, diverse products, and intricate customer demands. Traditional methods of data management, such as Excel files, are inadequate for handling the voluminous and interconnected nature of supply chain data. This project aims to address these challenges by leveraging a comprehensive database solution to streamline supply chain operations, improve decision-making processes, and enhance overall efficiency.

III. Background

The automotive supply chain is a multifaceted network involving numerous stakeholders, including suppliers, manufacturers, distributors, and customers. Managing this intricate ecosystem poses significant logistical and operational hurdles. Excel files, commonly used for data management in this industry, often lack the scalability, accessibility, and analytical capabilities required to effectively address these challenges. As a result, organizations struggle to gain meaningful insights from their data, leading to inefficiencies, delays, and missed opportunities.

IV. Objective

The primary objective of this project is to develop a robust database infrastructure tailored to the unique needs of the automotive supply chain. By structuring data into distinct entities such as Suppliers, Products, Customers, Orders, and Payment Information, the project aims to provide a centralized repository for all supply chain-related data. This database will enable advanced analytics, predictive modeling, and real-time monitoring, empowering stakeholders to make data-driven decisions with confidence.

V. Contribution to the Problem Domain

This project has the potential to revolutionize automotive supply chain management by offering a comprehensive solution to longstanding challenges. By transitioning from Excel files to a database-driven approach, organizations can unlock new possibilities for efficiency, agility, and innovation. The project's contribution lies in its ability to optimize processes, reduce costs, mitigate risks, and ultimately, enhance customer satisfaction. In an industry where competitiveness hinges on operational excellence, the impact of this project cannot be overstated.

VI. Target Users

Automotive Manufacturers: Automotive manufacturers will use the database to manage their supply chain operations, track inventory levels, monitor production processes, and analyze customer demand patterns. They will leverage the database to optimize production schedules, ensure timely delivery of vehicles, and maintain quality standards across their product lines.

Suppliers: Suppliers of automotive components, parts, and raw materials will utilize the database to coordinate orders, manage inventory levels, and track shipments. They will benefit from real-time visibility into demand forecasts, production schedules, and delivery timelines, enabling them to align their operations with customer requirements more effectively.

Distributors and Dealers: Distributors and dealerships will access the database to place orders, manage inventory at their respective locations, and track sales performance. They will use the database to forecast demand, optimize stocking levels, and analyze customer preferences to tailor their offerings accordingly.

Logistics and Transportation Providers: Logistics and trans- portation companies will leverage the database to optimize route planning, track shipments in transit, and ensure timely delivery of vehicles and components. They will utilize the database to streamline logistical operations, minimize trans- portation costs, and enhance overall supply chain efficiency.

VII. Database Administration

The database will be administered by a dedicated team of IT professionals with expertise in database management and system administration. In a real-life scenario, this team may consist of database administrators (DBAs), system architects, network engineers, and cybersecurity specialists. Their responsibilities will include:

Database Design and Implementation: Designing the database schema, creating tables, defining relationships, and ensuring data integrity and normalization.

Data Management: Importing and exporting data, performing backups and restores, monitoring database performance, and optimizing query execution.

Security Management: Implementing access controls, encryption mechanisms, and authentication protocols to safeguard sensitive data from unauthorized access and cyber threats.

System Maintenance: Installing software updates, applying patches, tuning database parameters, and resolving technical issues to ensure the smooth operation of the database environment.

User Support and Training: Providing technical support to users, troubleshooting database-related issues, and conducting training sessions to educate users on database functionalities and best practices.

By having a dedicated team of database administrators, the organization can ensure the reliability, security, and scalability of the database infrastructure, thereby maximizing its utility and value to the various stakeholders involved in the automotive supply chain.

Added new tables, updated ER diagram and database

VIII. Entity-Relationship (E/R) Diagram

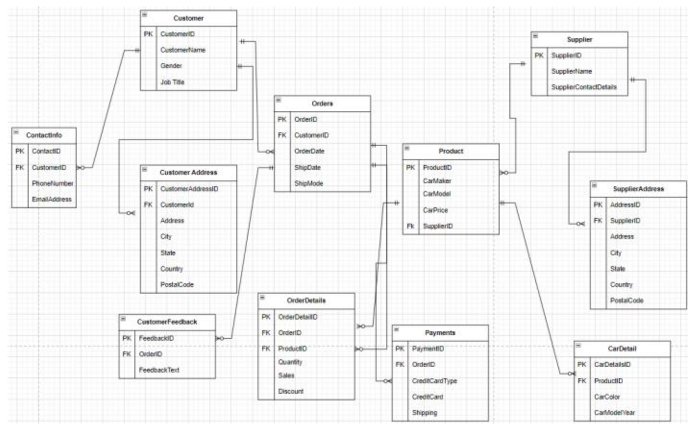


Fig. 1. ER Diagram

IX. Data Description

Supplier's Data: Stores unique identifiers, names, and contact details for each supplier.

Supplier Address's Data: Contains addresses for suppliers, including detailed location information like city and postal code.

Product's Data (Car): Catalogs cars with details such as product ID, maker, model, price, and the associated supplier.

Car Detail's Data: Holds detailed specifications for each car, including color and model year.

Customer's Data: Records customer information including ID, name, gender, and job title.

Contact Info's Data: Captures contact details for each customer, such as phone numbers and email addresses.

Customer Address's Data: Provides address information for customers, including state and country.

Order's Data: Tracks order transactions, including dates, shipping mode, and linkage to customers.

Order Detail's Data: Details items within orders, covering aspects like quantity, sales, and discounts.

Payment's Data: Manages payment information related to orders, including credit card types and numbers.

Customer Feedback's Data: Collects customer feedback on orders, incorporating feedback text and related order ID.

IX. RELATIONSHIP BETWEEN TABLES

1.Supplier and SupplierAddress: One-to-Many**,** Each supplier can have multiple addresses, but each specific address (AddressID) is linked to only one supplier.

2.Supplier and Product (Car): One-to-Many, A supplier can supply multiple products (cars), but each car is provided by only one supplier.

3.Product (Car) and CarDetail: One-to-One, Each product (car) has one set of details specified in CarDetail, uniquely detailing aspects like color and model year for each car.

4.Customer and ContactInfo: One-to-Many, Each customer can have multiple contact information records, but each specific contact record is associated with only one customer.

5.Customer and CustomerAddress: One-to-Many,Customers can have multiple addresses registered, with each address uniquely linked to one customer.

6.Customer and Order: One-to-Many,Each customer can place multiple orders, but each order is associated with one specific customer.

7.Order and OrderDetail: One-to-Many, Each order can contain multiple items (OrderDetail), witheach item's details, such as product ID and quantity, specifically linked to that order.

8.Order and Payment: One-to-One, Each order is associated with a unique payment record, detailing the transaction'spayment method and costs.

9.Order and CustomerFeedback: One-to-One,

Each order can receive one piece of feedback, directly linking customer feedback to the specific order it pertains to.

10.Product (Car) and OrderDetail: One-to-Many, Each car (product) can be part of multiple order details; that is, a car can be ordered multiple times in different orders or even within the same order as different order items.

X. ATTRIBUTES

Supplier's Data

- SupplierID INT (Primary Key): Uniquely identifies each supplier, non-null.
- SupplierName VARCHAR: Represents the name of the supplier, non-null.
- SupplierContactDetails TEXT: Holds contact details such as phone number and email, non-null.

SupplierAddress's Data

- AddressID INT (Primary Key): Uniquely identifies each address, non-null.
- SupplierID INT (Foreign Key to Supplier table): Links to the Supplier ID, non-null.
- Address TEXT: Street address of the supplier, non-null.
- City VARCHAR: City of the supplier's address, non-null.
- State VARCHAR: State of the supplier's address, non-null.
- Country VARCHAR: Country of the supplier's address, non-null.
- PostalCode INT: Postal code of the supplier's address, non-null.

Product's Data (Car)

- ProductID INT (Primary Key): Uniquely defines the product, non-null.
- CarMaker VARCHAR: Represents the maker of the car, non-null.
- CarModel VARCHAR: Represents the model of the car, non-null.
- CarPrice FLOAT: Represents the price of the car, non-null.
- SupplierID INT (Foreign Key to Supplier table): Links to the supplier of the car, non-null.

CarDetail's Data

- CarDetailID INT (Primary Key): Uniquely identifies car detail records, non-null.
- ProductID INT (Foreign Key to Product table): Links to the car product, non-null.
- CarColor VARCHAR: Represents the color of the car, non-null.
- CarModelYear INT: Represents the model year of the car, non-null.

Customer's Data

- CustomerID INT (Primary Key): Uniquely identifies each customer, non-null.
- CustomerName VARCHAR: Represents the full name of the customer, non-null.
- Gender CHAR(1): Represents the gender of the customer, non-null.
- JobTitle VARCHAR: Represents the job title of the customer, non-null.

ContactInfo's Data

- ContactID INT (Primary Key): Uniquely identifies contact information, non-null.
- CustomerID INT (Foreign Key to Customer table): Links to the customer, non-null.
- PhoneNumber VARCHAR: Represents the phone number of the customer, non-null.
- EmailAddress VARCHAR: Represents the email address of the customer, non-null.

CustomerAddress's Data

- CustomerAddressID INT (Primary Key): Uniquely identifies customer addresses, non-null.
- CustomerID INT (Foreign Key to Customer table): Links to the customer, non-null.
- Address TEXT: Street address of the customer, non-null.
- City VARCHAR: City of the customer’s address, non-null.
- State VARCHAR: State of the customer's address, non-null.
- Country VARCHAR: Country of the customer's address, non-null.
- PostalCode INT: Postal code of the customer's address, non-null.

Order's Data

- OrderID INT (Primary Key): Uniquely defines each order, non-null.
- CustomerID INT (Foreign Key to Customer table): Links to the customer who made the order, non-null.
- OrderDate DATE: Represents the date the order was placed, non-null.
- ShipDate DATE: Represents the date the order is shipped,

non-null.

- ShipMode VARCHAR: Represents the mode of shipping used for the order, non-null.

OrderDetail's Data

- OrderDetailID INT (Primary Key): Uniquely defines each order item, non-null.
- OrderID INT (Foreign Key to Order table): Links to the order, non-null.
- ProductID INT (Foreign Key to Product table): Links to the product ordered, non-null.
- Quantity INT: Represents the number of products ordered, non-null.
- Sales FLOAT: Represents the total sales amount, non-null.
- Discount FLOAT: Represents the discount given on the order item, non-null.

Payment's Data

- PaymentID INT (Primary Key): Uniquely defines each payment record, non-null.
- OrderID INT (Foreign Key to Order table): Links to the order, non-null.
- CreditCardType VARCHAR: Represents the type of credit card used, non-null.
- CreditCard VARCHAR: Represents the credit card number, non-null.
- Shipping FLOAT: Represents the shipping cost, non-null.

CustomerFeedback's Data

- FeedbackID INT (Primary Key): Uniquely identifies each feedback entry, non-null.
- OrderID INT (Foreign Key to Order table): Links to the order, non-null.
- FeedbackText TEXT: Represents the text of the feedback provided by the customer, non-null.

X. ACTIONS ON FOREIGN KEY WHEN PRIMARY KEY DELETED:

No Action (or Restrict): This is the default behavior in many databases. If there are any foreign key dependencies, the system will prevent the deletion of the primary key. This ensures that no orphan records are left in the child tables.

Cascade: When the primary key is deleted, a cascade action will automatically delete all related records in the foreign key table that depend on the primary key. This action ensures that there are no dangling references in the database and helps maintain referential integrity by automatically removing all dependent entities.

Set Null: This action sets the foreign key value in the child table to NULL when the referenced primary key is deleted. It's useful when it's acceptable to retain the dependent records without linking them to the parent record, effectively nullifying the relationship but retaining the child records.

Set Default: Similar to the Set Null action, but instead of setting the foreign key to NULL, it sets it to a default value specified in the table schema. This can be used to reassign dependent records to a default value when the primary entity is removed.

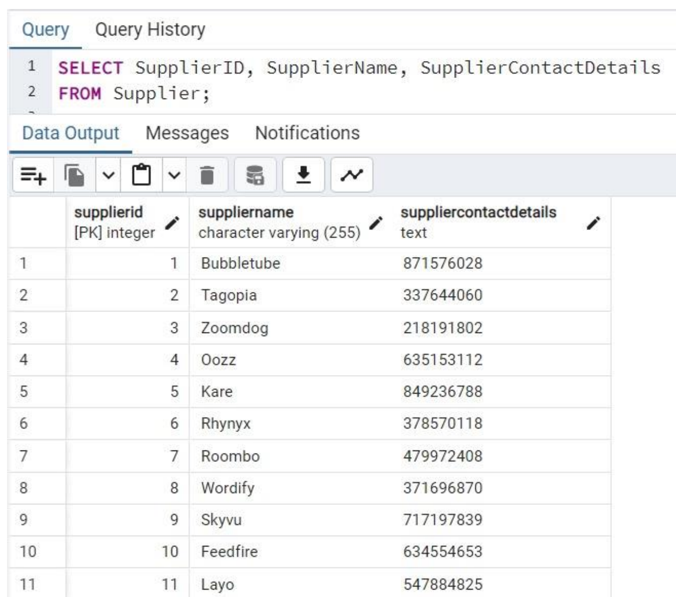
No Action vs. Restrict: While these are often considered the same, depending on the database system, "Restrict" might be an explicitly declared action that prevents the deletion of a primary key if there are any dependencies, whereas "No Action" could potentially defer the integrity check to the end of the transaction.

XI. APPROACH USED TO CREATE TABLES FROM CSV

In our approach to creating database tables from CSV files, we initially preprocessed the data using Python to segment and structure it into distinct tables. This preprocessing involved cleaning, validating, and organizing the data according to the schema requirements of our target database. Following the preprocessing phase, we manually loaded the structured data into PostgreSQL, utilizing its robust SQL capabilities to ensure each dataset was accurately inserted into its respective table. This method allowed for precise control over the data import process, ensuring data integrity and consistency across our database system.

XII. PERFORMED QUERIES:

Query 1 : Retrieve All Suppliers



Query Query History

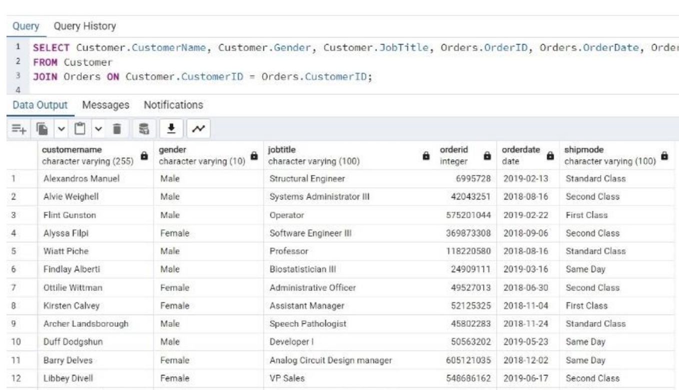
```sql
SELECT SupplierID, SupplierName, SupplierContactDetails
FROM Supplier;
```

Data Output Messages Notifications

	supplierid [PK] integer	suppliername character varying (255)	suppliercontactdetails text
1	1	Bubbletube	871576028
2	2	Tagopia	337644060
3	3	Zoomdog	218191802
4	4	Oozz	635153112
5	5	Kare	849236788
6	6	Rhynyx	378570118
7	7	Roombo	479972408
8	8	Wordify	371696870
9	9	Skyvu	717197839
10	10	Feedfire	634554653
11	11	Layo	547884825

Fig.2.

Query 2 : Customer Information and Their Orders



Query Query History

```sql
SELECT Customer.CustomerName, Customer.Gender, Customer.JobTitle, Orders.OrderID, Orders.OrderDate, Order
FROM Customer
JOIN Orders ON Customer.CustomerID = Orders.CustomerID;
```

Data Output Messages Notifications

	customername character varying (255)	gender character varying (10)	jobtitle character varying (100)	orderid integer	orderdate date	shipmode character varying (100)
1	Alexandros Manuel	Male	Structural Engineer	6995728	2019-02-13	Standard Class
2	Alvie Weighell	Male	Systems Administrator III	42043251	2018-08-16	Second Class
3	Flint Gunston	Male	Operator	575201044	2019-02-22	First Class
4	Alyssa Filpi	Female	Software Engineer III	369873308	2018-09-06	Second Class
5	Wiatt Piche	Male	Professor	118220580	2018-08-16	Standard Class
6	Findlay Alberti	Male	Biostatistician III	24909111	2019-03-16	Same Day
7	Ottilie Wittman	Female	Administrative Officer	49527013	2018-06-30	Second Class
8	Kirsten Calvey	Female	Assistant Manager	52125325	2018-11-04	First Class
9	Archer Landsborough	Male	Speech Pathologist	45802283	2018-11-24	Standard Class
10	Duff Dodgshun	Male	Developer I	50563202	2019-05-23	Same Day
11	Barry Delves	Female	Analog Circuit Design manager	605121035	2018-12-02	Same Day
12	Libbey Divell	Female	VP Sales	548686162	2019-06-17	Second Class
13	Demetria Hazelgrove	Female	Human Resources Assistant IV	579558315	2019-04-25	First Class

Fig.3.

Query 3: Detailed Car Information

```sql
SELECT Product.CarMaker, Product.CarModel, Product.CarPrice, CarDetail.CarColor, CarDetail.CarModelYear
FROM Product
JOIN CarDetail ON Product.ProductID = CarDetail.ProductID;
```

	carmaker character varying (100)	carmodel character varying (100)	carprice numeric	carcolor character varying (50)	carmodelyear integer
1	Dodge	Ram 2500	521963.45	Goldenrod	2007
2	Toyota	Tundra	672222.04	Crimson	2010
3	GMC	Savana 1500	504465.72	Crimson	2011
4	Volkswagen	Cabriolet	646077.11	Fuscia	1990
5	Mercury	Mariner	699890.24	Teal	2009
6	Toyota	Land Cruiser	694373.94	Crimson	2005
7	Subaru	Impreza	546977.55	Indigo	1999
8	BMW	X6	528353.8	Purple	2013
9	Mitsubishi	Galant	706964.35	Orange	2011
10	Subaru	Justy	798573.59	Crimson	1988
11	Mercedes-Benz	M-Class	722114.57	Mauv	2002
12	Ford	Taurus X	761182.85	Crimson	2008
13	Mercedes-Benz	CL-Class	568015.89	Red	2011

Fig.4.

Query 4: Orders and Associated Payment Information

```sql
SELECT Orders.OrderID, Orders.OrderDate, Payment.CreditCardType, Payment.CreditCard, Payment.Shipping
FROM Orders
JOIN Payment ON Orders.OrderID = Payment.OrderID;
```

	orderid integer	orderdate date	creditcardtype character varying (50)	creditcard character varying (20)	shipping character varying (20)
1	6995728	2019-02-13	jcb	3549221112237767	Truck
2	42043251	2018-08-16	jcb	3557159608180902	Air
3	575201044	2019-02-22	jcb	3529909223663921	Truck
4	369873308	2018-09-06	china-unionpay	5602235978541517	Air
5	118220580	2018-08-16	laser	6706247841149207	Air
6	24909111	2019-03-16	diners-club-carte-blanche	30528106894581	Truck
7	49527013	2018-06-30	jcb	3589144247894735	Truck
8	52125325	2018-11-04	jcb	3557048508901444	Air
9	45802283	2018-11-24	jcb	3534727482148019	Truck
10	50563202	2019-05-23	diners-club-enroute	201441036983941	Air
11	605121035	2018-12-02	jcb	3558313957113134	Truck
12	548686162	2019-06-17	diners-club-enroute	201524347329142	Truck
13	579558315	2019-04-25	jcb	3542457262271746	Air

Fig.5.

Query 5: Retrieve Customer Feedback with Order Details

```sql
SELECT
    CustomerFeedback.FeedbackID,
    CustomerFeedback.FeedbackText,
    Orders.OrderID,
    Orders.OrderDate,
    Orders.CustomerID
FROM
    CustomerFeedback
JOIN
    Orders ON CustomerFeedback.OrderID = Orders.OrderID;
```

	feedbackid integer	feedbacktext text	orderid integer	orderdate date	customerid integer
1	464898	Good	6995728	2019-02-13	67457594
2	874500	Okay	42043251	2018-08-16	58411135
3	548869	Very Bad	575201044	2019-02-22	5915307
4	581640	Bad	369873308	2018-09-06	51655189
5	745482	Good	118220580	2018-08-16	658110001
6	768011	Very Bad	24909111	2019-03-16	31722328
7	874507	Bad	49527013	2018-06-30	54162018
8	735245	Okay	52125325	2018-11-04	449110060
9	469007	Very Bad	45802283	2018-11-24	548685732
10	182293	Very Bad	50563202	2019-05-23	9551015
11	782360	Okay	605121035	2018-12-02	54838101
12	909336	Good	548686162	2019-06-17	49825119

Fig.6.

XIII. BCNF PROOF:

To ensure that the schema for the Car Supply Chain Management system is in Boyce-Codd Normal Form (BCNF), we must confirm that every non-trivial functional dependency has its left-hand side as a superkey. Here's an analysis of each table based on the functional dependencies.

Supplier

- **Functional Dependency**: SupplierID → SupplierName, SupplierContactDetails

- **Analysis:** SupplierID is the primary key and uniquely determines other attributes in the table, qualifying as a superkey. This table satisfies the BCNF condition.

SupplierAddress

- **Functional Dependency:** AddressID → SupplierID, Address, City, State, Country, PostalCode

- **Analysis:** AddressID is the primary key, making it a superkey that uniquely determines all other attributes in the table. This table is in BCNF.

Product (Car)

- **Functional Dependency:** ProductID → CarMaker, CarModel, CarPrice, SupplierID

- **Analysis:** ProductID, as the primary key, functions as a superkey and uniquely determines the specified attributes, adhering to BCNF requirements.

CarDetail

- **Functional Dependency:** CarDetailID → ProductID, CarColor, CarModelYear

- **Analysis:** CarDetailID is the primary key and a superkey that uniquely determines other related attributes. The table is in BCNF.

Customer

- **Functional Dependency**: CustomerID → CustomerName, Gender, JobTitle

- **Analysis**: CustomerID is the primary key and satisfies the BCNF condition by uniquely determining all other non-key attributes in the table.

ContactInfo

- **Functional Dependency:** ContactID → CustomerID, PhoneNumber, EmailAddress

- **Analysis:** ContactID, as the primary key, qualifies as a superkey, ensuring the table meets BCNF standards.

CustomerAddress

- **Functional Dependency:** CustomerAddressID → CustomerID, Address, City, State, Country, PostalCode

- **Analysis**: CustomerAddressID is the primary key and a superkey, ensuring the table adheres to the BCNF form.

Order

- **Functional Dependency:** OrderID → CustomerID, OrderDate, ShipDate, ShipMode

- **Analysis:** OrderID is the primary key and acts as a superkey, satisfying the BCNF criteria.

OrderDetail

- **Functional Dependency:** OrderDetailID → OrderID, ProductID, Quantity, Sales, Discount

- **Analysis:** OrderDetailID is the primary key and functions as a superkey, making the table BCNF compliant.

Payment

- **Functional Dependency:** PaymentID → OrderID, CreditCardType, CreditCard, Shipping

- **Analysis:** PaymentID is the primary key and superkey, ensuring this table is in BCNF.

CustomerFeedback

- **Functional Dependency:** FeedbackID → OrderID, FeedbackText

- **Analysis:** FeedbackID is the primary key, serving as a superkey. This table meets the BCNF conditions.

Each table's functional dependencies confirm that the primary key functions as a superkey, which functionally determines all other attributes in the table. These conditions satisfy the requirements for BCNF in all tables, eliminating the need for further decomposition to address anomalies or integrity issues. The Entity-Relationship (E/R) diagram thus remains unchanged from its initial state, reflecting a well-structured relational database design that adheres to BCNF criteria.

XIV. USE OF INDEXING:

When dealing with a larger dataset, a major challenge faced was the increased retrieval times for queries, particularly those that required grouping operations. As the size of the dataset grew, the duration required to sift through the data and extract pertinent information also rose markedly. To tackle this problem, indexing techniques were implemented to enhance the efficiency of data retrieval. The time it took to execute a query before the introduction of indexing is illustrated in figure 7

Query:1

Query | Query History

```sql
explain analyze
SELECT p.CarMaker, p.CarModel, SUM(od.Quantity) AS TotalQuantity
FROM OrderDetail od
JOIN Orders o ON od.OrderID = o.OrderID
JOIN Product p ON od.ProductID = p.ProductID
WHERE o.OrderDate >= (
    SELECT DATE_TRUNC('month', MAX(OrderDate))
    FROM Orders
) AND o.OrderDate < (
    SELECT (DATE_TRUNC('month', MAX(OrderDate)) + INTERVAL '1 month')
    FROM Orders
)
GROUP BY p.CarMaker, p.CarModel
ORDER BY TotalQuantity DESC;
```

Data Output | Messages | Explain | Notifications

```
QUERY PLAN
text
            -> Hash  (cost=20.08..20.08 rows=4 width=4) (actual time=0.461..0.462 rows=46 loops=1)
                 Buckets: 1024  Batches: 1  Memory Usage: 10kB
                 -> Seq Scan on orders o  (cost=0.00..20.08 rows=4 width=4) (actual time=0.292..0.438 rows=46 loops=1)
                      Filter: ((orderdate >= $0) AND (orderdate < $1))
                      Rows Removed by Filter: 826
         -> Index Scan using product_pkey on product p  (cost=0.28..0.35 rows=1 width=18) (actual time=0.004..0.004 rows=1 loops...
            Index Cond: (productid = od.productid)
Planning Time: 0.352 ms
Execution Time: 1.065 ms
```

Fig.7.

To speed up query performance on the 'orderdate' column, the command CREATE INDEX idx_orderdate ON orders(orderdate); was used to create an index.

Query | Query History

```sql
explain analyze
SELECT p.CarMaker, p.CarModel, SUM(od.Quantity) AS TotalQuantity
FROM OrderDetail od
JOIN Orders o ON od.OrderID = o.OrderID
JOIN Product p ON od.ProductID = p.ProductID
WHERE o.OrderDate >= (
    SELECT DATE_TRUNC('month', MAX(OrderDate))
    FROM Orders
) AND o.OrderDate < (
    SELECT (DATE_TRUNC('month', MAX(OrderDate)) + INTERVAL '1 month')
    FROM Orders
)
GROUP BY p.CarMaker, p.CarModel
```

Data Output | Messages | Explain | Notifications

```
QUERY PLAN
text
                 Buckets: 1024  Batches: 1  Memory Usage: 10kB
                 -> Bitmap Heap Scan on orders o  (cost=4.32..11.30 rows=4 width=4) (actual time=0.098..0.109 rows=46 loops=1)
                      Recheck Cond: ((orderdate >= $1) AND (orderdate < $3))
                      Heap Blocks: exact=7
                      -> Bitmap Index Scan on idx_orderdate  (cost=0.00..4.32 rows=4 width=0) (actual time=0.092..0.092 rows=46 loops=1)
                           Index Cond: ((orderdate >= $1) AND (orderdate < $3))
         -> Index Scan using product_pkey on product p  (cost=0.28..0.35 rows=1 width=18) (actual time=0.001..0.001 rows=1 loops=46)
              Index Cond: (productid = od.productid)
Planning Time: 2.382 ms
Execution Time: 0.513 ms
```

Fig.8.

Query:2

Query | Query History

```sql
explain analyze
SELECT s.SupplierName, p.CarModel, p.CarPrice
FROM Product p
JOIN Supplier s ON p.SupplierID = s.SupplierID
WHERE p.CarPrice > (SELECT AVG(CarPrice) FROM Product)
ORDER BY p.CarPrice DESC;
```

Data Output | Messages | Explain | Notifications

```
QUERY PLAN
text
Sort  (cost=72.26..72.99 rows=291 width=22) (actual time=1.039..1.063 rows=442 loops=1)
  Sort Key: p.carprice DESC
  Sort Method: quicksort  Memory: 59kB
  InitPlan 1 (returns $0)
    -> Aggregate  (cost=17.90..17.91 rows=1 width=32) (actual time=0.249..0.250 rows=1 loops=1)
         -> Seq Scan on product  (cost=0.00..15.72 rows=872 width=8) (actual time=0.007..0.084 rows=872 loops=1)
  -> Hash Join  (cost=21.54..42.44 rows=291 width=22) (actual time=0.618..0.872 rows=442 loops=1)
       Hash Cond: (s.supplierid = p.supplierid)
       -> Seq Scan on supplier s  (cost=0.00..14.72 rows=872 width=11) (actual time=0.012..0.104 rows=872 loops=1)
       -> Hash  (cost=17.90..17.90 rows=291 width=19) (actual time=0.592..0.592 rows=442 loops=1)
            Buckets: 1024  Batches: 1  Memory Usage: 32kB
            -> Seq Scan on product p  (cost=0.00..17.90 rows=291 width=19) (actual time=0.262..0.500 rows=442 loop...
                 Filter: (carprice > $0)
                 Rows Removed by Filter: 430
Planning Time: 0.244 ms
Execution Time: 1.122 ms
```

Fig.9.

To optimize query performance, three indexes were created: `CREATE INDEX idx_product_carprice ON product(carprice);` to facilitate faster searches based on car prices, `CREATE INDEX idx_supplier_supplierid ON supplier(supplierid);` to enhance lookups by supplier ID, and `CREATE INDEX idx_product_supplierid ON product(supplierid);` to improve access times when querying products by their supplier ID.

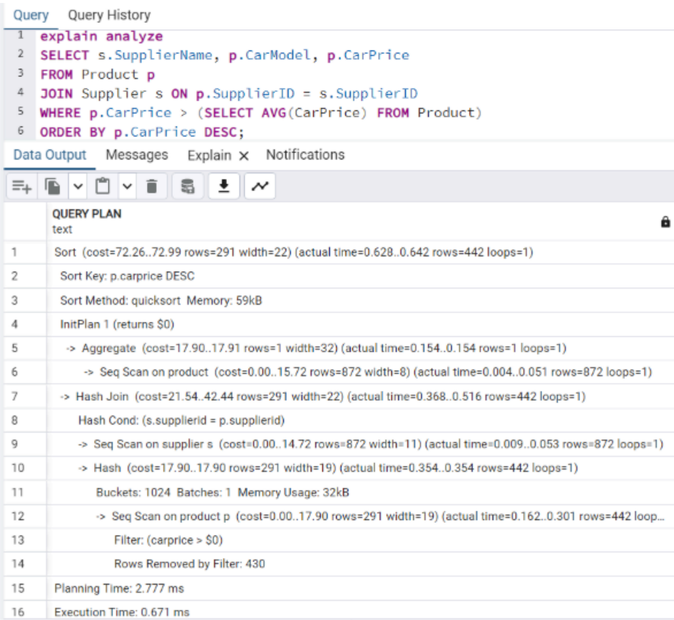


Fig.10.

Query:3

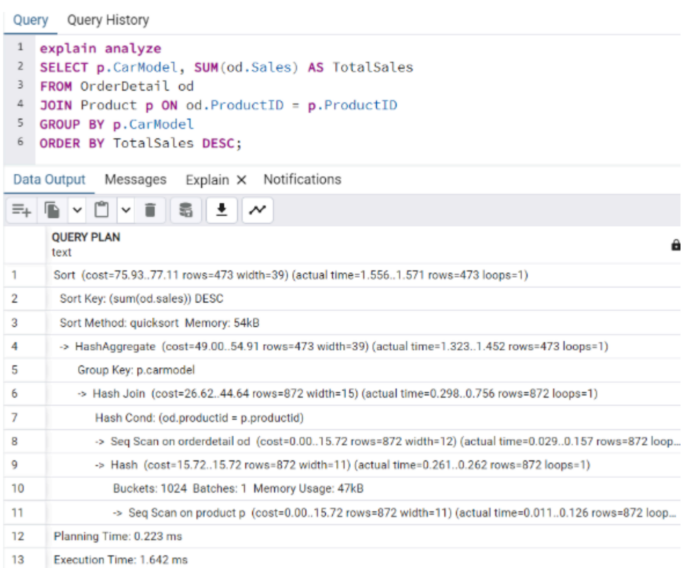


Fig.11.

To improve query efficiency, the commands `CREATE INDEX idx_product_productid ON product(productid);` and `CREATE INDEX idx_orderdetail_productid ON orderdetail(productid);` were used to create indexes, enhancing data retrieval based on product IDs in both the 'product' and 'orderdetail' tables.

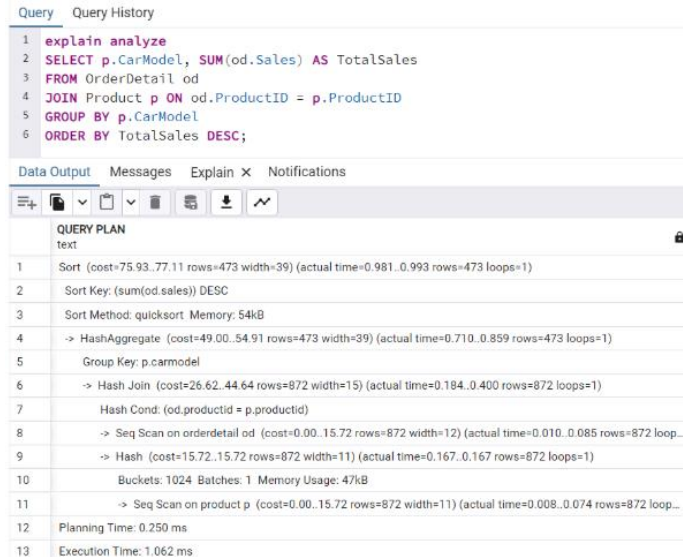


Fig.12.

The execution time taken for a query after use of indexing can be addressed in Fig.8, Fig.10, Fig.12.

Overall, by addressing the retrieval time issue through indexing, able to improve the performance of the database when handling larger datasets, ensuring that queries were executed more efficiently.

XV. QUERYING:

Query1: Insert a new supplier and their address:

Query 2: Insert a new product provided by the last added supplier

Query Query History

```sql
INSERT INTO Product (CarMaker, CarModel, CarPrice, SupplierID)
VALUES ('Toyota', 'Corolla', 20000.00, (SELECT MAX(SupplierID) FROM Supplier));
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 45 msec.
```

Query3: Deleting a Customer and All Related Records

Query Query History

```sql
BEGIN;
DELETE FROM CustomerFeedback
WHERE OrderID IN (
    SELECT OrderID FROM Orders WHERE CustomerID = (SELECT CustomerID FROM Customer WHERE CustomerName = 'Louie Hinsche'));
DELETE FROM Payment
WHERE OrderID IN (
    SELECT OrderID FROM Orders WHERE CustomerID = (SELECT CustomerID FROM Customer WHERE CustomerName = 'Louie Hinsche'));
DELETE FROM OrderDetail
WHERE OrderID IN (
    SELECT OrderID FROM Orders WHERE CustomerID = (SELECT CustomerID FROM Customer WHERE CustomerName = 'Louie Hinsche'));
DELETE FROM Orders
WHERE CustomerID = (SELECT CustomerID FROM Customer WHERE CustomerName = 'Louie Hinsche');
DELETE FROM CustomerAddress
WHERE CustomerID = (SELECT CustomerID FROM Customer WHERE CustomerName = 'Louie Hinsche');
DELETE FROM ContactInfo
WHERE CustomerID = (SELECT CustomerID FROM Customer WHERE CustomerName = 'Louie Hinsche');
DELETE FROM Customer
WHERE CustomerName = 'Louie Hinsche';

COMMIT;
```

Data Output Messages Notifications

```
COMMIT

Query returned successfully in 43 msec.
```

Query 4 : Deleting a Supplier and All Related Records

Query Query History

```sql
BEGIN;
DELETE FROM CarDetail
WHERE ProductID IN (
    SELECT ProductID FROM Product WHERE SupplierID =
    (SELECT SupplierID FROM Supplier WHERE SupplierName = 'Quality Goods Inc')
);
DELETE FROM Product
WHERE SupplierID = (SELECT SupplierID FROM Supplier WHERE SupplierName = 'Quality Goods Inc');

DELETE FROM SupplierAddress
WHERE SupplierID = (SELECT SupplierID FROM Supplier WHERE SupplierName = 'Quality Goods Inc');

DELETE FROM Supplier
WHERE SupplierName = 'Quality Goods Inc';

COMMIT;
```

Data Output Messages Notifications

```
COMMIT

Query returned successfully in 46 msec.
```

Query 5: Update the price of a specific model from a specific maker

Query Query History

```sql
UPDATE Product
SET CarPrice = CarPrice * 1.1  -- Increase price by 10%
WHERE CarMaker = 'Toyota' AND CarModel = 'Corolla';
```

Data Output Messages Notifications

```
UPDATE 3

Query returned successfully in 49 msec.
```

Query 6: Update contact details for a specific customer

Query Query History

```sql
UPDATE ContactInfo
SET EmailAddress = 'newemail22@domain.com', PhoneNumber = '1234597890'
WHERE CustomerID = (SELECT CustomerID
                    FROM Customer WHERE CustomerName = 'Flint Gunston');
```

Data Output Messages Notifications

```
UPDATE 1

Query returned successfully in 53 msec.
```

Query7: Join multiple tables to get a detailed order list including customer info, order details, and product info

Query Query History

```sql
SELECT c.CustomerName, o.OrderDate, p.CarMaker, p.CarModel, od.Quantity, od.Sales
FROM Orders o
JOIN Customer c ON o.CustomerID = c.CustomerID
JOIN OrderDetail od ON o.OrderID = od.OrderID
JOIN Product p ON od.ProductID = p.ProductID
WHERE o.OrderDate BETWEEN '2019-01-01' AND '2019-5-31'
ORDER BY o.OrderDate DESC;
```

Data Output Messages Notifications

	customername character varying (255)	orderdate date	carmaker character varying (100)	carmodel character varying (100)	quantity integer	sales numeric
1	Gussy Murison	2019-05-31	Pontiac	Grand Prix	2	820595.53
2	Boone Rubie	2019-05-31	Lincoln	LS	2	724227.01
3	Isidora Brugger	2019-05-30	Plymouth	Neon	2	943388.33
4	Sanderson Northbridge	2019-05-30	Audi	S5	1	711993.48
5	Dinny O'Cooney	2019-05-30	Honda	Ridgeline	2	998843.29
6	Tonya Tidball	2019-05-30	Suzuki	Vitara	2	838528.58
7	Andromache Bernardez	2019-05-30	Volkswagen	Cabriolet	1	764462.9
8	Denny Alred	2019-05-30	Volvo	C70	2	956935.45
9	Karim Rosevear	2019-05-29	Pontiac	Trans Sport	1	796593.13
10	Ermina Brambill	2019-05-29	Mitsubishi	GTO	2	743622.93
11	Tristam Le Breton	2019-05-28	Pontiac	Sunbird	2	819189.22
12	Gunar Gobeaux	2019-05-28	Chevrolet	Suburban	2	963512.41
13	Bonita Veracruysse	2019-05-27	Dodge	Ram 2500	1	879833.37

Query8: Aggregate function to get total sales per car model grouped by model

Query Query History

```sql
SELECT p.CarModel, SUM(od.Sales) AS TotalSales
FROM OrderDetail od
JOIN Product p ON od.ProductID = p.ProductID
GROUP BY p.CarModel
ORDER BY TotalSales DESC;
```

Data Output Messages Notifications

	carmodel character varying (100)	totalsales numeric
1	Galant	7040006.14
2	Grand Prix	6793496.12
3	Camaro	6786994.68
4	SL-Class	6226512.28
5	C70	6199948.43
6	Suburban 1500	6086810.14
7	Century	5875109.31
8	M3	5331517.94
9	E-Class	5088724.43
10	Accord	5062198.76
11	CL-Class	4980881.38

Query 9: Subquery to find suppliers who have products costing more than the average price of all products

Query Query History

```sql
SELECT s.SupplierName, p.CarModel, p.CarPrice
FROM Product p
JOIN Supplier s ON p.SupplierID = s.SupplierID
WHERE p.CarPrice > (SELECT AVG(CarPrice) FROM Product)
ORDER BY p.CarPrice DESC;
```

Data Output Messages Notifications

	suppliername character varying (255)	carmodel character varying (100)	carprice numeric
1	Kwideo	Corolla	850035.62985
2	Mycat	Canyon	799454.24
3	Yata	900	798837.47
4	Twitterlist	Mustang	798765.11
5	Feedfire	Justy	798573.59
6	Tazz	5 Series	798206.16
7	Voomm	CL-Class	798192.93
8	Livefish	Ram 2500	798141.78
9	Ainyx	Diamante	797965.14
10	Jamia	Parisienne	797632.01
11	InnoZ	Solara	796767.55

Query 10: Complex query with multiple joins and a subquery to list all products sold in the last month, sorted by quantity sold

Query Query History

```sql
SELECT p.CarMaker, p.CarModel, SUM(od.Quantity) AS TotalQuantity
FROM OrderDetail od
JOIN Orders o ON od.OrderID = o.OrderID
JOIN Product p ON od.ProductID = p.ProductID
WHERE o.OrderDate >= (
    SELECT DATE_TRUNC('month', MAX(OrderDate))
    FROM Orders
) AND o.OrderDate < (
    SELECT (DATE_TRUNC('month', MAX(OrderDate)) + INTERVAL '1 month')
    FROM Orders
)
GROUP BY p.CarMaker, p.CarModel
ORDER BY TotalQuantity DESC;
```

Data Output Messages Explain × Notifications

	carmaker character varying (100)	carmodel character varying (100)	totalquantity bigint
1	Mercedes-Benz	CL-Class	4
2	Pontiac	Bonneville	3
3	Chevrolet	Express 3500	3
4	Cadillac	Seville	2
5	Acura	RL	2
6	Bentley	Azure	2
7	Bentley	Continental GTC	2
8	Infiniti	Q	2
9	Infiniti	QX	2

XVI. QUERY EXECUTION ANALYSIS

Query: The provided SQL query calculates total sales by car model and involves a complex join between the Product and OrderDetail tables, with a subsequent aggregation and sorting operation. The query's execution plan indicates the use of hash joins and aggregates, which are performance-intensive due to full table scans and lack of indexes.

Query Query History

```sql
SELECT p.CarModel, SUM(od.Sales) AS TotalSales
FROM OrderDetail od
JOIN Product p ON od.ProductID = p.ProductID
GROUP BY p.CarModel
ORDER BY TotalSales DESC;
```

Data Output Messages Explain × Notifications

	carmodel character varying (100)	totalsales numeric
1	Galant	7040006.14
2	Grand Prix	6793496.12
3	Camaro	6786994.68
4	SL-Class	6226512.28
5	C70	6199948.43
6	Suburban 1500	6086810.14
7	Century	5875109.31
8	M3	5331517.94
9	E-Class	5088724.43
10	Accord	5062198.76
11	CL-Class	4980881.38
12	Park Avenue	4529257.99
13	Miata MX-5	4397767.15
14	Ram 2500	4352037.52
15	Sentra	4273939.85

Query Query History

```sql
explain analyze
SELECT p.CarModel, SUM(od.Sales) AS TotalSales
FROM OrderDetail od
JOIN Product p ON od.ProductID = p.ProductID
GROUP BY p.CarModel
ORDER BY TotalSales DESC;
```

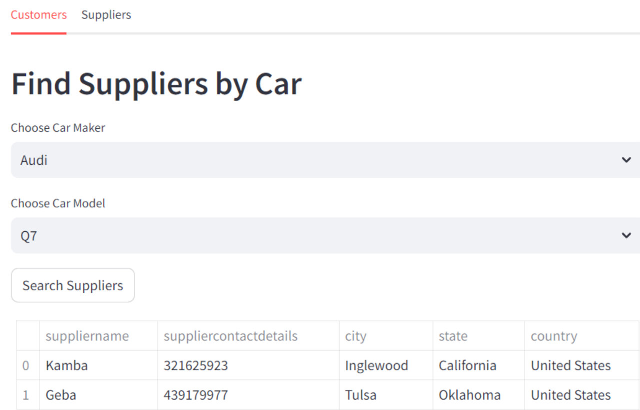
Data Output Messages Explain × Notifications

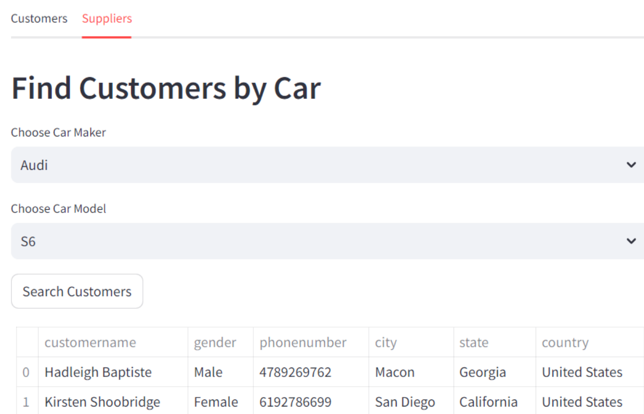
	QUERY PLAN text
1	Sort (cost=75.93..77.11 rows=473 width=39) (actual time=1.556..1.571 rows=473 loops=1)
2	Sort Key: (sum(od.sales)) DESC
3	Sort Method: quicksort Memory: 54kB
4	-> HashAggregate (cost=49.00..54.91 rows=473 width=39) (actual time=1.323..1.452 rows=473 loops=1)
5	Group Key: p.carmodel
6	-> Hash Join (cost=26.62..44.64 rows=872 width=15) (actual time=0.298..0.756 rows=872 loops=1)
7	Hash Cond: (od.productid = p.productid)
8	-> Seq Scan on orderdetail od (cost=0.00..15.72 rows=872 width=12) (actual time=0.029..0.157 rows=872 loop...
9	-> Hash (cost=15.72..15.72 rows=872 width=11) (actual time=0.261..0.262 rows=872 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 47kB
11	-> Seq Scan on product p (cost=0.00..15.72 rows=872 width=11) (actual time=0.011..0.126 rows=872 loop...
12	Planning Time: 0.223 ms
13	Execution Time: 1.642 ms

To optimize this query, indexing is recommended on ProductID in both joined tables for faster joins, CarModel in the Product table to expedite group operations, and Sales in the OrderDetail table to quicken aggregation computations. These indexes are expected to reduce full table scans, enhance data retrieval speed, and improve overall query performance by enabling more efficient join algorithms and faster grouping and sorting operations.

XVII. WEB APPLICATION

In the development of our web application using Streamlit, we aimed to provide a robust tool that enables users to interactively explore customer and supplier data associated with specific car makes and models, drawing information from a PostgreSQL database. This application facilitates the querying of detailed information about suppliers and customers by allowing users to select a car maker and model from dynamically populated dropdown menus. The backend integration is handled through psycopg2, ensuring direct interaction with our database, which is managed securely by fetching credentials from environment variables to protect sensitive information.





We employed Streamlit's caching mechanism to optimize the responsiveness of the application. By caching database connections and data fetching functions, we significantly improved the application's efficiency, reducing the need to establish new database connections or repeat database queries for repeated requests. This not only enhances the user experience by speeding up data retrieval but also reduces the load on the server, leading to a more scalable solution.

Moreover, we focused on user-friendly design elements within the Streamlit framework. The interface is divided into tabs, making it straightforward for users to switch between viewing suppliers and customers. This separation aids in clarity and usability. Each interaction—selecting a car maker and model, followed by the initiation of a search—triggers a backend process that fetches and displays relevant data in a well-organized table format. This setup ensures that users, regardless of their technical background, can easily navigate and utilize the application to gain the insights they need. This approach emphasizes our commitment to creating intuitive, efficient, and secure applications that cater to real-world business needs.

References

Dataset: **[Supply chain management for Car (kaggle.com)]**

Preprocessing Notebook:
https://colab.research.google.com/drive/1Hl2ocgDHfR-BgauiosTObbUIVeFYhX2f?usp=sharing

https://docs.streamlit.io/develop/tutorials/databases/postgresql