Inventory Management and System

Bhavani Kiran Kukunoor
University at buffalo
50560517
bhavanik@buffalo.edu

Ch.v.m.sai praneeth
University at buffalo
50560459
vchidell@buffalo.edu

Hima venkata sai saketh saketh ram jaladi
University at buffalo
50560439
himavenk@buffalo.edu

I . PROBLEM STATEMENT

A. Description

Businesses in various sectors rely heavily on streamlined inventory management and online order tracking systems to ensure efficient operations and satisfied customers. Without these systems, they face the risk of overselling or stockouts, leading to lost sales, missed opportunities, and disgruntled customers. Similarly, lacking a robust online order tracking mechanism can result in delays in delivery times and customer dissatisfaction, as it becomes challenging to monitor the status of orders. Moreover, incorporating a user review system is imperative to allow customers to provide feedback on their purchases, which is invaluable for understanding preferences and improving product offerings. These systems enable various analyses, including sales forecasting, customer segmentation, and product recommendation systems, which are crucial for strategic planning and targeted marketing efforts.

Manual inventory tracking processes are not only time-consuming but also prone to errors, leading to inaccuracies in data and delays in decision-making. Hence, there arises a need for an integrated inventory management and online order tracking system capable of efficiently managing inventory levels, orders, and shipments, from small-scale to large-scale operations. Such a system should provide real-time monitoring of stock levels, generate alerts for low stock levels, manage purchase orders, and automatically update inventory levels. Additionally, it should track online orders from placement to delivery, offering customers real-time updates on the status of their orders to enhance satisfaction.

Moreover, the system must be user-friendly, efficient, and scalable to accommodate the company's growth. Robust reporting and analytics features are essential for monitoring inventory levels, tracking inventory movements, and identifying trends and patterns to make informed decisions. Furthermore, strong security features are necessary to protect sensitive inventory data from unauthorized access and cyber threats. These systems should also allow customers to receive updates on their orders via email or SMS notifications, improving transparency and enhancing the overall order tracking experience. Ultimately, implementing such systems enables businesses to efficiently manage inventory, reduce the risk of stockouts, improve customer satisfaction, and streamline overall operations, thereby driving growth and success.

B. Database vs Excel Files

Databases outperform Excel files in terms of data organization, consistency, security, scalability, and sharing/collaboration. Unlike Excel, which is limited in its ability to handle complex queries and large datasets, databases excel at managing massive amounts of data efficiently. They provide

greater performance, making it easier to generate reports and visualizations for insightful analysis and informed decision-making.

Furthermore, databases offer a centralized location for data storage, reducing the risk of data loss or corruption that may occur when Excel files are scattered across multiple locations. This centralized approach streamlines data retrieval processes, saving time and effort.

Moreover, databases allow for more advanced data conversions and calculations compared to Excel, offering greater flexibility in data manipulation. This capability is particularly beneficial for businesses requiring advanced analytics or machine learning on their data.

In summary, databases provide a robust and scalable solution for enterprises dealing with large volumes of data that demand secure, consistent, and efficient management. While Excel remains effective for managing small datasets and performing straightforward analytics, databases offer superior capabilities for handling complex data interactions and ensuring data integrity.

II. TARGET USER

Customer service representatives: By utilizing the Database's capabilities to retrieve order histories and specific customer information, and improve customer satisfaction

Sales Representatives: Tasked with managing orders from customers and assuring timely delivery, they manage orders efficiently, keep careful watch of inventories, and provide full analyses that help the Database to continuously observe and enhance sales performance.

Data analysts: They are responsible for analyzing patterns and trends in sales and purchasing habits. They extract complex datasets out of the database and perform in-depth analysis, which provide essential data for strategic decision-making.

III. REAL LIFE SCENARIO

Inventory management systems and order tracking are applied can be observed in a retail company that operates both online and physical stores. A company manages a significant inventory stored across various warehouses. The inventory management system continuously monitors product quantities, locations, and status updates, including shipping and receiving, in real-time.

When a product becomes unavailable, the inventory management system promptly updates its availability on the company's website. Additionally, it sends alerts when inventory levels reach predetermined thresholds, prompting timely action to replenish stock through new orders and warehouse restocking efforts.

By implementing efficient inventory management and order tracking systems, coupled with providing customers with real-time order tracking capabilities, the company can consistently meet customer demand with adequate inventory levels. Furthermore, various real-world scenarios demonstrate the versatility of such datasets:

Supply Chain Optimization: Logistics firms leverage these datasets to evaluate supplier performance and optimize supply chain operations, ensuring timely deliveries and efficient inventory management.

Customer Segmentation: Businesses categorize customers based on demographics and purchasing behavior,

enabling tailored marketing campaigns to specific customer segments, enhancing engagement and loyalty.

Product Recommendation: By analyzing past purchases and browsing behavior, businesses develop personalized product recommendation algorithms, enhancing the overall shopping experience and driving additional sales.

Fraud Detection: Retailers employ these datasets to detect and prevent fraudulent activities, safeguarding against unauthorized transactions and ensuring a secure shopping environment.

Market Analysis: Retailers utilize these datasets to analyze market trends, customer behavior, and seller performance, guiding strategic decisions regarding product offerings and marketing strategies.

```
df = pd.read_csv("ML-Dataset.csv")
```

```
null = df.isnull()
null_counts = null.sum()
print(null_counts)
```

```
RegionName             0
CountryName            0
State                  0
City                   0
PostalCode             0
WarehouseAddress       0
WarehouseName          0
EmployeeName           0
EmployeeEmail          0
EmployeePhone          0
EmployeeHireDate       0
EmployeeJobTitle       0
CategoryName           0
ProductName            0
ProductDescription     0
ProductStandardCost    0
Profit                 0
ProductListPrice       0
CustomerName           0
CustomerAddress        0
CustomerCreditLimit    0
CustomerEmail          0
CustomerPhone          0
Status                 0
OrderDate              0
OrderItemQuantity      0
PerUnitPrice           0
TotalItemQuantity      0
dtype: int64
```

```
Warehouse_names = df['WarehouseName'].unique()
print(Warehouse_names)
```

```
['Southlake Texas' 'San Francisco' 'New Jersy' 'Seattle Washington'
 'Toronto' 'Sydney' 'Mexico City' 'Beijing' 'Bombay']
```

```
Procduct_names = df['ProductName'].unique()
print(Procduct_names)
```

```
['Intel Xeon E5-2699 V3 (OEM/Tray)' 'Intel Xeon E5-2697 V3'
 'Intel Xeon E5-2698 V3 (OEM/Tray)' 'Intel Xeon E5-2697 V4'
 'Intel Xeon E5-2685 V3 (OEM/Tray)' 'Intel Xeon E5-2695 V3 (OEM/Tray)'
 'Intel Xeon E5-2697 V2' 'Intel Xeon E5-2695 V4' 'Intel Xeon E5-2695 V2'
 'Intel Xeon E5-2643 V2 (OEM/Tray)' 'Intel Xeon E5-2690 (OEM/Tray)'
 'Intel Xeon E5-2687W V3' 'Intel Xeon E5-2687W V4'
 'Intel Xeon E5-2667 V3 (OEM/Trav)' 'Intel Xeon E5-2690 V4'
```

```
employee_emails = df['EmployeeEmail'].unique()
print(employee_emails)
```

```
['summer.payne@example.com' 'rose.stephens@example.com'
 'annabelle.dunn@example.com' 'tommy.bailey@example.com'
 'blake.cooper@example.com' 'jude.rivera@example.com'
 'tyler.ramirez@example.com' 'ryan.gray@example.com'
 'elliot.brooks@example.com' 'elliott.james@example.com'
 'albert.watson@example.com' 'mohammad.peterson@example.com'
 'harper.spencer@example.com' 'louie.richardson@example.com'
 'nathan.cox@example.com' 'bobby.torres@example.com'
 'charles.ward@example.com' 'gabriel.howard@example.com'
 'emma.perkins@example.com' 'amelie.hudson@example.com'
 'gracie.gardner@example.com' 'frederick.price@example.com'
```

IV. DATA INSIGHTS

A. Data source

We acquired a dataset from kaggle, then applied pre-processing techniques to obtain clean data.
https://www.kaggle.com/datasets/hetulparmar/inventory-management-dataset/data

B. Data Analysis and Pre-processing

To eliminate any errors to ensure data quality

V. INITIAL DATABASE SCHEMA

The Inventory Management System database contains information on inventory, sales, customer information. It tracks product details, records customer orders, maintains customer information, stores employee details and warehouse information.The dataset consists of multiple relations.
During the initial phase set of attributes present in the database are -

Inventory_Management_System(
RegionName,
CountryName,
State,
City,
PostalCode,
WarehouseAddress,
WarehouseName,
EmployeeName,
EmployeeEmail,
EmployeePhone,
EmployeeHireDate,
EmployeeJobTitle,
CategoryName,
ProductName,
ProductDescription,
ProductStandardCost,
Profit,
ProductListPrice,
CustomerName,
CustomerAddress,
CustomerCreditLimit,
CustomerEmail,
CustomerPhone,
Status,
OrderDate,
OrderItemQuantity,
PerUnitPrice,
TotalItemQuantity
);

Typically, we would divide the relation down into smaller relations that fulfill BCNF in order to achieve BCNF.

VI. BOYCE - CODD NORMAL FORM & FUNCTIONAL DEPENDENCIES

The dataset can be divided into multiple relations - Transaction, Employee, Warehouse, Warehouse address, Customer and Product.
Inorder for the relation to be in Boyce Codd Normal Form, it should satisfy BCNF conditions. whenever X -> Y is a nontrivial FD that holds in *R*, X is a superkey or Candidate key.

1. Transaction

Transaction (
ID INT PRIMARY KEY,
Warehous_name VARCHAR(255),
Employee_name VARCHAR(255),
Product_name VARCHAR(255),
Customer_email VARCHAR(255),
Status VARCHAR(50),
Order_dt DATETIME,
Order_quality INT,
perunit_price DECIMAL(10, 2),
FOREIGN KEY (Warehouse_name)
REFERENCES Warehouse(w_name),
FOREIGN KEY (Employee_name)
REFERENCES Employee(e_email),
FOREIGN KEY (Product_name)
REFERENCES Product(p_name),
FOREIGN KEY (Custom_email)
REFERENCES Customer(c_email)
);

ID is the primary key, so it's a candidate key.
Warehouse_name, Employee_name, Product_name, and Customer_email are foreign keys, each referencing the primary key of their respective tables.
non-key attributes depend on ID which is a primary key.

ID ->{ w_name, e_name, p_name, c_email, status, order_dt, order_quality, perunit_price}

Therefore the relation is in BCNF

2. Employee

```
Employee (
   e_email VARCHAR(255) PRIMARY
KEY,
   e_name VARCHAR(255),
   e_phnno VARCHAR(20),
   e_hiredate DATE,
   e_jobtitle VARCHAR(255)
);
```

e_email is a Candidate key, it uniquely identifies other attributes.
e_email ->{ e_name, e_phnno, e_hiredate, e_jobtitle}
Therefore the relation is in BCNF

3. Warehouse

```
Warehouse (
   W_name VARCHAR(255) PRIMARY
KEY,
   W_address VARCHAR(255),
   W_postalcode INT,
   FOREIGN KEY (W_postalcode)
REFERENCES
Warehouse_Address(postal_id)
);
```

Warehouse_name is a candidate key, since it uniquely identifies address, postal code
W_name ->{ W_address, W_postalcode}
Therefore the relation is in BCNF

4. Warehouse Address

```
 Warehouse_Address (
   postal_id INT PRIMARY KEY,
   region_name VARCHAR(255),
   country_name VARCHAR(255),
   state VARCHAR(255),
   city VARCHAR(255)
);
```

Postal_id is a candidate key, since it uniquely identifies location
postal_id ->{ region_name, country_name, state, city}
Therefore the relation is in BCNF

5. Customer

```
 Customer (
   c_email VARCHAR(255) PRIMARY
KEY,
   c_name VARCHAR(255),
   c_phn VARCHAR(20),
   c_address VARCHAR(255),
   c_creditlimit DECIMAL(10, 2)
);
```

c_email is a Candidate key, it uniquely identifies other attributes.
c_email ->{ c_name, c_phnno, c_address, c_creditlimit}
Therefore the relation is in BCNF

6. Product:

```
Product (
   p_name VARCHAR(255) PRIMARY
KEY,
   p_description VARCHAR(255),
   p_standard VARCHAR(255),
   p_profit DECIMAL(10, 2),
   p_list DECIMAL(10, 2)
);
```

p_name is a Candidate key, it uniquely identifies other attributes.
p_name ->{ p_description, p_standard, p_profit, p_list}
Therefore the relation is in BCNF
This database's overall goal is to increase customer happiness, decision-making efficiency, and productivity in the retail or wholesale industry through efficient inventory control and sales tracking.

VII. ER DIAGRAM

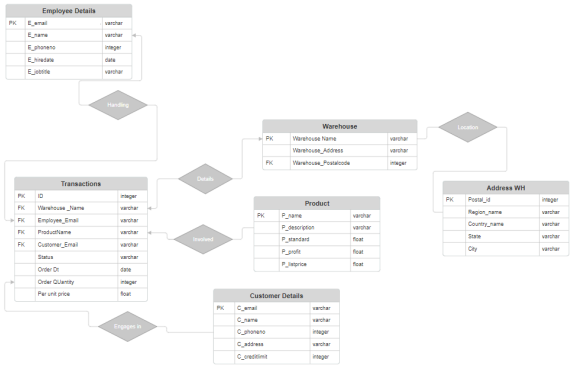


Fig. 1 ER diagram

VIII. DESCRIPTION OF RELATIONS AND IT'S ATTRIBUTES

1. Transaction Relation -
This relation stores information about transactions, including details such as the ID of the transaction, the warehouse where it occurred, the employee who handled it, the product involved, the customer associated with the transaction, transaction status, order date, quantity ordered, and unit price.

- ID: A unique identifier for each transaction.
- Warehouse_name: The name of the warehouse where the transaction took place. (Foreign key referencing Warehouse table)
- Employee_name: The name of the employee who processed the transaction. (Foreign key referencing Employee table)
- Product_name: The name of the product involved in the transaction. (Foreign key referencing Product table)
- Customer_email: The email address of the customer involved in the transaction. (Foreign key referencing Customer table)
- Status: The status of the transaction (e.g., pending, completed, canceled).
- Order_dt: The date and time when the transaction occurred.
- Order_quality: The quantity of the product ordered in the transaction.
- Perunit_price: The price per unit of the product in the transaction.

2. Employee Table:
Information on people who are employed. The email address, name, phone number, hire date, and job title of each employee are listed in this table.

- e_email: The email address of the employee. (Primary key)
- e_name: The name of the employee.
- e_phnno: The phone number of the employee.
- e_hiredate: The date when the employee was hired.
- e_jobtitle: The job title or position of the employee.

3. Warehouse:

Information regarding warehouses, including name, address, and postal code, is stored in this table.

- W_name: The name of the warehouse. (Primary key)
- W_address: The address of the warehouse.
- W_postalcode: The postal code of the warehouse. (Foreign key referencing Warehouse Address table)

4. Warehouse Address:
Warehouse Address table contains detailed address information for warehouses, identified by a unique postal_id.

- postal_id: A unique identifier for each warehouse address. (Primary key)
- region_name: The name of the region where the warehouse is located.
- country_name: The name of the country where the warehouse is located.
- state: The state where the warehouse is located.
- city: The city where the warehouse is located.

5. Customer:
Customers' email address, name, phone number, address, and credit limit are all listed in this table.

- c_email: The email address of the customer. (Primary key)
- c_name: The name of the customer.
- c_phn: The phone number of the customer.
- c_address: The address of the customer.
- c_creditlimit: The credit limit of the customer.

6. Product:
The Product table provides details on the different products that are stocked.

- p_name: The name of the product. (Primary key)
- p_description: Description of the product.
- p_standard: Standard specifications or details of the product.
- p_profit: Profit margin associated with the product.
- p_list: List price of the product.

IX. RELATION BETWEEN TABLES

Customer to Transaction:
Customers can have one-to-many relation with transactions
Each customer can make one or multiple transactions. One customer can have many transactions, but each transaction is associated with only one customer.

Transaction to Product:
Transaction can have many-to-many relation with products
Every transaction can involve one or more items, and every product can have several transactions associated with it. This kind of relationship is usually many-to-many (M:N).

Employee to Transaction:
Workers may take part in transaction handling. An employee is capable of

managing several transactions, then the relationship is one-to-many.

Warehouse to Transaction:

Transactions involve products that might be stored in a warehouse. This relationship can be represented as one-to-many, indicating that one warehouse can store many products involved in different transactions.

X. TABLES AND KEYS

Tables	Keys
Transaction	Primary key: id Foreign Key: e_email, p_name, c_email Not null: p_name, c_email
Employee	Primary key: e_email Not null: e_email
Warehouse	Primary Key: W_name Foreign Key: postal_id Not null: postal_id
Warehouse Address	Primary Key: postal_id
Customer	Primary key: c_name
Product	Primary key: p_name

We have multiple foreign keys in our transaction table, these foreign keys refer to multiple tables as primary keys. When we delete the primary key in the sub tables, we follow the default value approach to overcome the delete anomaly.
In this approach we set a default to the attribute when the attribute is defined while creating the table

```
-- Create Customer table
CREATE TABLE Customer (
    c_email VARCHAR(255) PRIMARY KEY,
    c_name VARCHAR(255),
    c_phn VARCHAR(20),
    c_address VARCHAR(255),
    c_creditlimit DECIMAL(10, 2)
);
```

Using the customer's email address as the primary key guarantees uniqueness and streamlines searches.
Foreign Keys: None

```
-- Create Employee table
CREATE TABLE Employee (
    e_email VARCHAR(255) PRIMARY KEY,
    e_name VARCHAR(255),
    e_phnno VARCHAR(20),
    e_hiredate DATE,
    e_jobtitle VARCHAR(255)
);
```

Assuring each employee has a unique identity is possible by using the email address as the main key.
Foreign Keys: None

```sql
-- Create Product table
CREATE TABLE Product (
    p_name VARCHAR(255) PRIMARY KEY,
    p_description VARCHAR(255),
    p_standard VARCHAR(255),
    p_profit DECIMAL(10, 2),
    p_list DECIMAL(10, 2)
);
```

Using the product name as the primary key guarantees uniqueness and streamlines queries.
Foreign Keys: None

```sql
-- Create Warehouse table
CREATE TABLE Warehouse (
    W_name VARCHAR(255) PRIMARY KEY,
    W_address VARCHAR(255),
    W_postalcode INT,
    FOREIGN KEY (W_postalcode) REFERENCES Warehouse_Address(postal_id)
);
```

By utilizing the warehouse name as the primary key, you can simplify queries.

postal_id - foreign key preserves referential integrity by connecting each warehouse to its address in the Warehouse Address table. - W_postalcode: References postal_id in the Warehouse Address table.

```sql
-- Create Warehouse Address table
CREATE TABLE Warehouse_Address (
    postal_id INT PRIMARY KEY,
    region_name VARCHAR(255),
    country_name VARCHAR(255),
    state VARCHAR(255),
    city VARCHAR(255)
);
```

Every address record in the table will have a unique identification if postal_id—a unique identifier—is used.
Foreign Keys: None

```sql
CREATE TABLE Transaction (
    ID INT PRIMARY KEY,
    Warehouse_name VARCHAR(255),
    Employee_name VARCHAR(255),
    Product_name VARCHAR(255),
    Customer_email VARCHAR(255),
    Status VARCHAR(50),
    Order_dt DATETIME,
    Order_quality INT,
    perunit_price DECIMAL(10, 2),
    FOREIGN KEY (Warehouse_name) REFERENCES Warehouse(W_name),
    FOREIGN KEY (Employee_name) REFERENCES Employee(e_email),
    FOREIGN KEY (Product_name) REFERENCES Product(p_name),
    FOREIGN KEY (Customer_email) REFERENCES Customer(c_email)
);
```

Every transaction record is uniquely recognized when a transaction ID is used as the primary key.
- Foreign Keys:
- Warehouse_name: In the Warehouse table, it references W_name.
- Justification: By preserving referential integrity, this foreign key connects every transaction to the warehouse in which it took place.
- Employee_name: The Employee table's references e_email
- Justification: Referential integrity is maintained by this foreign key, which connects every transaction to the worker who handled it.
- Product_name: In the Product table, references p_name.
- Justification: Referential integrity is preserved by using this foreign key to connect each transaction to the relevant product.
- Customer_email: In the Customer table, references c_email.
- Justification: Referential integrity is maintained by this foreign key, which connects each transaction to the relevant customer.

XI. DATABASE IMPLEMENTATION

A. Creating Additional Tables

```
CREATE TABLE Order_Details (
    order_id INT PRIMARY KEY,
    customer_email VARCHAR(255),
    order_date DATE,
    total_amount DECIMAL(10,2),
    status VARCHAR(50),
    FOREIGN KEY (customer_email) REFERENCES Customer(c_email)
);
```

Fig 1. Creation of Order_Details Table

```
CREATE TABLE Payment (
    payment_id INT PRIMARY KEY,
    order_id INT,
    payment_date DATE,
    amount_paid DECIMAL(10,2),
    payment_method VARCHAR(100),
    FOREIGN KEY (order_id) REFERENCES Order_Details(order_id)
);
```

Fig 2. Creation of Payment Table

```
CREATE TABLE Shipment (
    shipment_id INT PRIMARY KEY,
    order_id INT,
    shipment_date DATE,
    warehouse_name VARCHAR(255),
    shipping_address VARCHAR(255),
    FOREIGN KEY (order_id) REFERENCES Order_Details(order_id),
    FOREIGN KEY (warehouse_name) REFERENCES Warehouse(w_name)
);
```

Fig 3. Creation of Shipment Table

```
CREATE TABLE Product_Stock (
    product_name VARCHAR(255),
    warehouse_name VARCHAR(255),
    quantity_in_stock INT,
    FOREIGN KEY (product_name) REFERENCES Product(p_name),
    FOREIGN KEY (warehouse_name) REFERENCES Warehouse(w_name),
    PRIMARY KEY (product_name, warehouse_name)
);
```

Fig 4. Creation of Product_Stock Table

B. Loading data to PgAdmin4

Importing a csv file into PostgreSQL database using pgAdmin's Import/Export tool.

The prepared CSV files are formatted, with columns and rows matching the Tables structure. By using the import tab on the Import/Export dialog, we can browse CSV files. The data then gets successfully processed into the table.

Importing data present in CSV file for a customer table through the Import/Export tool is shown below.

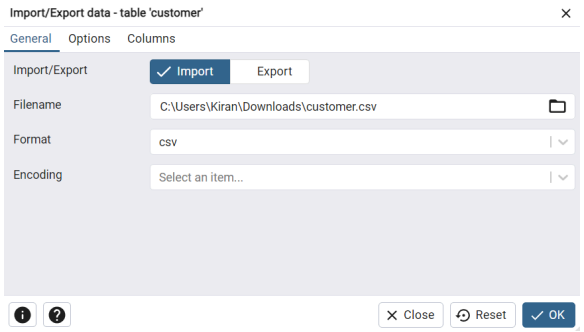


Fig 5. Importing a table's CSV file to postgreSQL

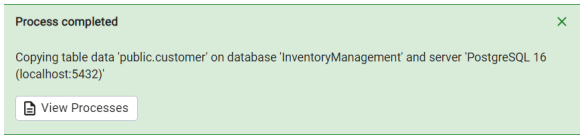


Fig 6. Import process successfully completed

C. Performing Insertion

```
--- Insert data in Customer Table
SELECT * FROM Customer;
INSERT INTO Customer (c_email, c_name, c_phn, c_address, c_creditlimit)
VALUES ('Naruto@buffalo.edu', 'Naruto', '1576567890', '164 Plm St, Buffalo City', 9000.00);
```

400	JohnSnow@gmail.com	John Snow	567897474	11279 Loytan St	2000.00
401	john.lee@buffalo.edu	John lee	1234567890	1234 Elm St, Buffalo City	5000.00
402	Naruto@buffalo.edu	Naruto	1576567890	164 Plm St, Buffalo City	9000.00

Fig 7. Inserting a record into Customer's Table

```
--- Insert data in Product Table
SELECT * FROM Product;
INSERT INTO Product (p_name, p_description, p_standard, p_profit, p_list)
VALUES ('iwatch', 'Apple red  series 4', 8354.11, 554.00, 2432.90);
```

	p_name [PK] character varying (255)	p_description character varying (255)	p_standard character varying (255)	p_profit numeric (10,2)	p_list numeric (10,2)
269	Hynix 6671G DDR2 1GB	Series:Ultrastar 7K3000,Type:7200RPM,Capacity:2TB,Cache:64MB	45.18	14.81	59.99
270	Kingston ValueRAM (KVR1333D3N9/8G) 8 GB DDR3	Series:AV-GP,Type:5400RPM,Capacity:250GB,Cache:8MB	12.63	2.92	15.55
271	Zion ZHY16004096 4 GB Ddr3	Series:FireCuda,Type:Hybrid,Capacity:1TB,Cache:64MB	55.41	12.65	68.06
272	Kingston KVR (KVR16N11/8-SP) 8 GB DDR3	Series:MX300,Type:SSD,Capacity:275GB,Cache:N/A	79.21	18.67	97.88
273	Hynix (HYMD2GB) 2GB DDR2	Series:Barracuda ES,Type:7200RPM,Capacity:1TB,Cache:32MB	34.4	9.59	43.99
274	Silicon Power (SP008GBLFU240B02) 8GB DDR4	Series:Gold,Type:7200RPM,Capacity:10TB,Cache:256MB	313.96	129.68	443.64
275	A-DATA (AX4U300038G16-SRS) 8GB DDR4	Series:MX300,Type:SSD,Capacity:525GB,Cache:N/A	121.92	29.07	150.99
276	Asus SABERTOOTH X99	Series:Caviar Blue,Type:7200RPM,Capacity:250GB,Cache:8MB	15.23	1.76	16.99
277	Widget	Standard red widget	2654.11	456.00	3042.90
278	iwatch	Apple red series 4	8354.11	554.00	2432.90

Fig 8. Inserting a record into Product's Table

D. Performing Deletion

```
-- Delete an employee from Employee Table
SELECT * FROM Employee;
DELETE FROM Employee WHERE e_email = 'rose.stephens@example.co
-- Delete a product from Product Table
SELECT * FROM Product;
DELETE FROM Product WHERE p_name = 'Intel Xeon E5-2699 V3 (OEM
```

Fig 9. Deletion of a record performed on both Employee and Product Table

E. Performing Update

```
--Upadte a Customer creditlimit from Customer Table
SELECT * FROM Customer;
UPDATE Customer SET c_creditlimit = 7500.00 WHERE c_email = 'flor.stone@ray
```

	c_email [PK] character varying (255)	c_name character varying (255)	c_phn character varying (20)	c_address character varying (255)
1	flor.stone@raytheon.com	Flor Stone	13171234104	2904 S Salina St
2	lavera.emerson@plainsallamerican.com	Lavera Emerson	13171234111	5344 Haverford Ave, Philadelphia

401	Naruto@buffalo.edu	Naruto	1576567890	164 Plm St, Buffalo City
402	flor.stone@raytheon.com	Flor Stone	13171234104	2904 S Salina St

Fig 10. Update command performed on customer's credit limit

```
-- Update P_list of a product from Product Table
SELECT * FROM Product;
UPDATE Product SET p_list = 3005.11 WHERE p_name = 'Intel Xeon E5-2697 V4';
```

	p_name [PK] character varying (255)	p_description character varying (255)	p_standard character varying (255)	p_profit numeric (10,2)	p_list numeric (10,2)
1	Intel Xeon E5-2697 V3	Speed:2.6GHz,Cores:14	2326.27	448.71	2774.98
2	Intel Xeon E5-2698 V3 (OEM/Tray)	Speed:2.3GHz	2035.18	625.54	2660.72
3	Intel Xeon E5-2697 V4	Cores:18,TDP:145W	2144.4	410.59	2554.99
4	Intel Xeon E5-2685 V3 (OEM/Tray)	Speed:2.6GHz,Cores:12	2012.11	489.58	2501.69

	p_name [PK] character varying (255)	p_description character varying (255)	p_standard character varying (255)	p_profit numeric (10,2)	p_list numeric (10,2)
1	Intel Xeon E5-2697 V4	Cores:18,TDP:145W	2144.4	410.59	3005.11

Fig 11. Update command performed on product list

F. Select Queries

1. To fetch customer's details, where the credit limit is higher than 3000.

```
--- SELECT QUERIES
---1. To fetch customer details
SELECT * FROM Customer WHERE c_creditlimit > 3000;
```

	c_email [PK] character varying (255)	c_name character varying (255)	c_phn character varying (20)	c_address character varying (255)	c_creditlimit numeric (10,2)
1	lavera.emerson@plainsallamerican.com	Lavera Emerson	13171234111	5344 Haverford Ave, Philadelphia	5000.00
2	meenakshi.mason@internationalpaper.com	Meenakshi Mason	15652788999	1831 No Wong, Peking	3600.00
3	charlie.sutherland@up.com	Charlie Sutherland	17887873099	Piazza Cacchiatore 23, San Giminiano	3600.00
4	daniel.costner@staples.com	Daniel Costner	18779099109	1597 Legend St, Mysore, Kar	3700.00
5	dianne.derek@dianaher.com	Dianne Derek	18717872090	1606 Sangam Blvd, New Delhi,	5000.00
6	willie.barrera@starbucks.com	Willie Barrera	18990924161	Po Box 2152, Buffalo, NY	5000.00
7	marlene.odom@cummins.com	Marlene Odom	12650900181	1617 Crackers St, Bangalore - India, Kar	5000.00
8	cora.calhoun@kimberly-clark	Cora Calhoun	16782091010	1656 Veterans Rd, Chennai, Tam	3700.00
9	amira.macdonald@fluor.com	Amira Macdonald	17890988761	1668 Chong Tao, Beijing,	5000.00
10	verena.hopper@generalmills.com	Verena Hopper	6710191725	612 Jefferson Ave, Scranton, PA	3600.00

2. Join query between Customer and Order table

```
Select * From Order_details;
SELECT Customer.c_name, Order_details.order_date, Order_details.total_amount
FROM Customer
JOIN Order_details ON Customer.c_email = Order_details.customer_email
WHERE Order_details.status = 'Shipped';
```

	c_name character varying (255)	order_date date	total_amount numeric (10,2)
1	Lavera Emerson	2017-02-20	2774.98
2	Jeni Levy	2017-04-09	2501.69
3	Matthias Hannah	2017-08-15	2431.95
4	Meenakshi Mason	2015-04-09	2269.99
5	Charlie Pacino	2016-12-20	2116.72
6	Daniel Costner	2015-05-02	2042.69
7	Dianne Derek	2016-09-29	2009.46
8	Geraldine Martin	2016-12-02	1908.73
9	Guillaume Edwards	2016-10-27	1904.70
10	Maurice Mahoney	2017-10-27	1899.99

3.Using GROUP BY to find total sales per product

```
---3. Using GROUP BY to find total sales per product:
SELECT Product_name, SUM(perunit_price * Order_quantity) AS Total_Sales
FROM Transaction_Details
GROUP BY Product_name;
```

	product_name character varying (255)	total_sales numeric
1	ASRock C2750D4I	45750.67
2	ASRock EP2C602-4L/D16	319174.94
3	Hynix 1333FSB 4GB DDR3	144477.60
4	HP C2J95AT	39464.35
5	Supermicro X10SAT-O	73089.71
6	PNY VCQM5000-PB	256597.87
7	EVGA Z270 Classified K	277452.14
8	Intel Xeon E5-2698 V3 (OEM/Tray)	73668.08
9	Zotac ZT-70203-10P	43639.70
10	Western Digital WDS256G1X0C	51679.20

4. Using ORDER BY to sort employees by hire date

```
---4. Using ORDER BY to sort employees by hire date:
SELECT e_name, e_hiredate FROM Employee
ORDER BY e_hiredate DESC;
```

	e_name character varying (255)	e_hiredate date
1	Dee Randy	2018-07-09
2	Tannen Biff	2017-04-20
3	Gruber Hans	2017-04-20
4	DeVito Tommy	2017-02-15
5	Navathe Kurt	2017-02-10
6	Roper Katie	2017-01-07
7	Frederick Price	2016-12-24
8	Abigail Palmer	2016-12-19
9	Daisy Ortiz	2016-12-15
10	Felix Bryant	2016-12-12

5. Subquery to find customers who have placed orders above a certain amount

```
---5. Subquery to find customers who have placed orders above a certain am
SELECT c_name, c_email
FROM Customer
WHERE c_email IN (SELECT c_email FROM Order_details WHERE total_amount > 1
```

	c_name character varying (255)	c_email [PK] character varying (255)
1	Lavera Emerson	lavera.emerson@plainsallamerican.com
2	Fern Head	fern.head@usfoods.com
3	Shyla Ortiz	shyla.ortiz@abbvie.com
4	Jeni Levy	jeni.levy@centene.com
5	Matthias Hannah	matthias.hannah@chs.net
6	Matthias Cruise	matthias.cruise@alcoa.com
7	Meenakshi Mason	meenakshi.mason@internationalpaper.com
8	Christian Cage	christian.cage@emerson.com
9	Charlie Sutherland	charlie.sutherland@up.com
10	Charlie Pacino	charlie.pacino@amgen.com

6. JOIN and GROUP BY to find the number of products sold by each warehouse

```
---6. JOIN and GROUP BY to find the number of products sold by each warehouse
SELECT Warehouse.w_name, COUNT(Transaction_Details.ID) AS Total_Products_Sold
FROM Transaction_Details
JOIN Warehouse ON Transaction_Details.Warehouse_name = Warehouse.w_name
GROUP BY Warehouse.w_name;
```

	w_name [PK] character varying (255)	total_products_sold bigint
1	Mexico City	45
2	Sydney	45
3	Beijing	45
4	Seattle Washington	45
5	Bombay	43
6	Toronto	44

XII. QUERY ANALYSIS EXECUTION

1. Trigger: UpdateStockAfterTransaction

```
--- Query Analysis Execution
CREATE OR REPLACE FUNCTION update_stock()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Product_Stock
    SET quantity_in_stock = quantity_in_stock - NEW.Order_quantity
    WHERE product_name = NEW.Product_name AND warehouse_name = NEW.Warehouse_name;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER UpdateStockAfterTransaction
AFTER INSERT ON Transaction_Details
FOR EACH ROW
EXECUTE FUNCTION update_stock();
```

Potential Issues:

High contention or locking issues may occur if multiple transactions attempt to update the stock for popular products at the same time.

Improvement Plan:

Add Indexes: To speed up lookups, we can ensure that the Product_Stock table contains an index on (product_name, warehouse_name).
And can alter the transaction isolation level to reduce locking and increase concurrency.

2. Trigger: CheckCreditBeforeOrder

```
CREATE OR REPLACE FUNCTION check_credit()
RETURNS TRIGGER AS $$
DECLARE
    credit_limit DECIMAL(10,2);
BEGIN
    SELECT c_creditlimit INTO credit_limit FROM Customer WHERE c_email = NEW.customer_email;
    IF credit_limit < NEW.total_amount THEN
        RAISE EXCEPTION 'Credit limit exceeded for this order.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Data Output | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 116 msec.
```

Potential Issues:

If c_email is not indexed, this SELECT query may be slow, affecting the speed of both the trigger and the Order_details insert.

Improvement Plan:

To add index to c_email: This lookup will be substantially faster using an index on the Customer table's c_email column.
Review Exception Handling: Ensures that exception handling does not add unnecessary overhead, particularly in a high-throughput setting.

3.Trigger: LogJobTitleChange

```
CREATE TRIGGER CheckCreditBeforeOrder
BEFORE INSERT ON Order_details
FOR EACH ROW
EXECUTE FUNCTION check_credit();

CREATE OR REPLACE FUNCTION log_job_title_change()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.e_jobtitle <> NEW.e_jobtitle THEN
        INSERT INTO EmployeeChanges(employee_email, old_jobtitle, new_jobtitle, change_date)
        VALUES (NEW.e_email, OLD.e_jobtitle, NEW.e_jobtitle, CURRENT_DATE);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Data Output | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 118 msec.
```

Potential Issues:

If the EmployeeChanges table lacks efficient indexes or grows in size, the INSERT operation may be sluggish.
If this table is often written to, it may cause disk I/O troubles.

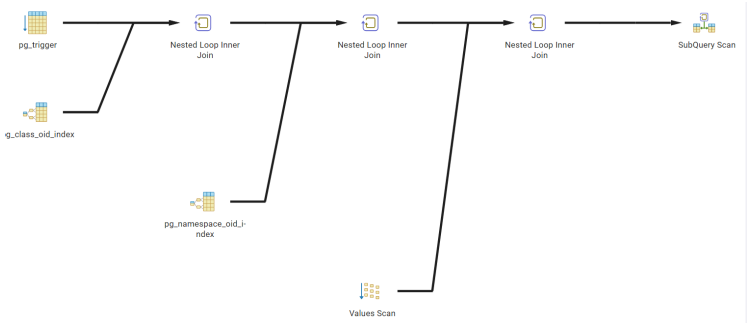
Improvement Plan:

Optimize Indexes: Make sure there are adequate indexes on employee_email and, if applicable, change_date.
Partitioning: If EmployeeChanges becomes huge, try partitioning it by change_date to handle the data volume more efficiently.

Explain Tool:

Using the "Explain" function in pgAdmin, we can learn how PostgreSQL performs Trigger queries and optimizes them for better speed.

```
SELECT trigger_name, event_object_table, event_manipulation
FROM information_schema.triggers
WHERE trigger_schema NOT IN ('pg_catalog', 'information_schema');
```



XIII. INDEXING

A. Challenges

1. Performance degradation: As the dataset size increases, queries are taking longer to complete because they process bigger volumes of data. This can cause users to receive slower responses and have an influence on system performance.
2. Increased memory usage: Storing and processing the data require additional

memory, resulting in greater system requirements and possibly resource limits.

3. Difficulty with data retrieval: Searching, filtering, and sorting through enormous data can become increasingly complex and resource-intensive, making it difficult to acquire the needed information efficiently.

4. Data consistency and integrity: Maintaining data consistency and integrity becomes increasingly important. With larger volumes of data, it might be more difficult to ensure that data is accurate and up to date throughout.

B. Indexing Concepts Adopted

1. The above challenges encountered, using indexing techniques.
As the dataset grows, queries that require filtering or sorting vast amounts of data become slower. Indexes can aid by allowing for more efficient data access.

Indexing the Customer table's c_creditlimit column enables for faster retrieval of records based on credit limit.

```
116 ---Indexing
117
118 CREATE INDEX idx_customer_creditlimit ON Customer(c_creditlimit);
119
```

Data Output | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 137 msec.
```

2. Traditional indexes, such as B-tree indexes, can use a significant amount of memory, particularly for large datasets. To address this, hash indexes were used to speed up lookup operations while reducing memory overhead.

Building a hash index on the Employee table's e_hiredate field can enhance query efficiency when looking for employees by hire date.

```
120 CREATE INDEX idx_employee_hiredate ON Employee USING hash(e_hiredate);
121
```

Data Output | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 142 msec.
```

3. Searching for specific criteria, such as a postal code in the Warehouse table, might be slow without sufficient indexing. Searches for warehouses by postal code can be made more efficient by constructing an index on the w_postalcode field.

```
121 --3.
122 CREATE INDEX idx_warehouse_postalcode ON Warehouse(w_postalcode);
123
```

Data Output | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 75 msec.
```

By employing indexing adapted to the individual requirements of the database, these issues are efficiently resolved, resulting in enhanced overall system performance.

```
SELECT * FROM pg_indexes
WHERE indexname = 'idx_customer_creditlimit' or indexname = 'idx_employee_hiredate'
or indexname = 'idx_warehouse_postalcode' ;
```

	schemaname name	tablename name	indexname name	tablespace name	indexdef text
1	public	customer	idx_customer_creditlimit	[null]	CREATE INDEX idx_customer_creditlimit ON public.customer USING btree (c_creditlimit)
2	public	employee	idx_employee_hiredate	[null]	CREATE INDEX idx_employee_hiredate ON public.employee USING hash (e_hiredate)
3	public	warehouse	idx_warehouse_postalcode	[null]	CREATE INDEX idx_warehouse_postalcode ON public.warehouse USING btree (w_postalco...

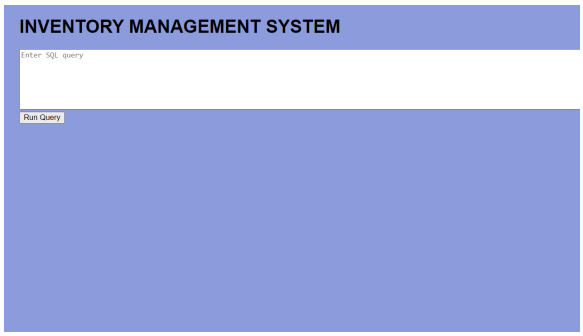
XIV. GUI

The application was created using a combination of technologies:

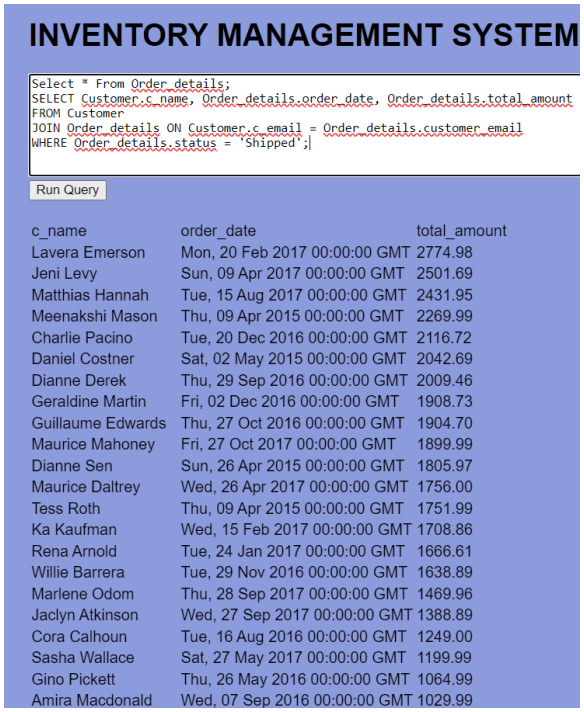
1. Backend: The backend is developed using Python and the Flask framework. Flask is Python's lightweight WSGI web application framework.

2. Frontend: The frontend is developed using HTML, CSS, and JavaScript. HTML defines the structure of the web page, CSS styles the elements, and JavaScript adds interaction to the page.
3. Database: PostgreSQL serves as the database management system, and the psycopg2 Python module is used to connect to and interact with the PostgreSQL database.

To summarize, the program is built with Python on the backend, HTML, CSS, and JavaScript on the frontend, and PostgreSQL for database management.



The fig shows a running website that links with our database to display/visualize query and query results.



XV. REFERENCES

1. https://www.w3schools.com/sql/
2. https://www.postgresql.org/docs/
3.https://www.pgadmin.org/docs/pgadmin4/development/query_tool_toolbar.html
4. https://www.postgresqltutorial.com/