INVICTUS WAREHOUSE DIRECT

-Invictus

CONTENTS:

Introduction:

In the dynamic landscape of e-commerce and inventory management, the need for a seamless and comprehensive data application has never been more crucial. Our project, the "Invictus Warehouse Direct," addresses this demand by combining operational and analytical functionalities to streamline customer transactions and empower sales executives with valuable insights. At its core, our system serves as an operational database, facilitating customer interactions with our warehouse through a user-friendly web portal.
In parallel, our application offers a dedicated portal for sales executives, providing them with a comprehensive suite of analytical tools. This dual-purpose approach empowers executives to make data-driven decisions, contributing to effective inventory management and strategic business planning. Our goal is to enhance operational efficiency by identifying underperforming warehouses with excess stock. By pinpointing such warehouses, we aim to strategically close them and seamlessly transfer their stock to more suitable warehouses.

To **initiate the application**, open the Jupyter Notebook and execute the file named 'Invictus warehouse direct app.ipynb'. This main file manages the entire project's execution. Code corresponding to each GUI page is organized within this file, differentiated by appropriate headings. Various classes and functions have been created for different pages, and they are invoked as needed.
Example User Credential for running the app: username – dinesh, password: dinesh777
Employee Credential: username – Diane Murphy, password: DM123

Data Source:

Source link: **https://www.coursera.org/learn/showcase-analyze-data-model-car-database-mysql-workbench/resources/I9gaM**

Utilized MySQL to create existing ***User Credentials*** in login pages.

Data source Description:
The imported dataset is related to a fictional company, a retailer of classic model cars, and other vehicles, which stores the vehicles in four different warehouses.

Objective:

The objective of this project is to utilize MySQL Workbench to examine the data model and identify crucial aspects of the data that can inform strategic decisions on reorganizing or reducing inventory while maintaining timely service to customers. Specific queries will be developed to address key questions, including the relationship between inventory numbers and sales figures, and the identification of slow-moving items that could be potential candidates for removal from the product line. The goal is to provide the company with insightful and data-driven recommendations for inventory reduction, ultimately supporting the decision-making process in closing one of their storage facilities.

Application Design:

Developing tools: Python, MySQL Workbench, PyQt Designer.
It requires additional packages to install:
1. Pandas
2. NumPy
3. Matplotlib

The application consists of two main modules: Operation Module and Analytical Module.

Operation Module:

The operational module GUI is designed to be an efficient and user-friendly platform for order placement and management. Users can log in using their credentials, which enhances security and personalization. Once logged in, they can view which warehouse currently holds the desired product and place orders for products. This allows users to choose a location that is conveniently close to them for order pickup, optimizing the collection process. Additionally, the GUI offers a feature for users to view their previous orders, providing them with a comprehensive history of their transactions and facilitating easy reordering or reference to past purchases. This combination of features makes the operational module both practical and user centric.

Analytical Module:

In this module, analytics are performed on the data gathered from the operational module. Designed three dashboards to provide insights into the sales performance, status of the stocks and performance of the warehouses with a goal of supporting inventory-related business decisions that lead to the closure of a warehouse due to drastic decrease in sales. Real-time operational capabilities of the dashboards provide real-time information to assist in decision-making.

Database Design:

Entity-Relationship Diagram (ERD), Relational Schema, and Star Schema are drawn to represent and organize data structures in different ways. ERD helps in understanding the entities involved and the connections between them. Relational Schema, on the other hand, outlines the structure of a relational database ‘Invictus_db’, detailing tables, attributes, and the relationships between tables. Star Schema designed for ‘Invictus_wh’(warehouse database) consists of central fact tables connected to multiple dimension tables, forming a star-like structure. This schema simplifies complex queries and allows for efficient data retrieval in data warehouse environments, making it suitable for analytical processing.

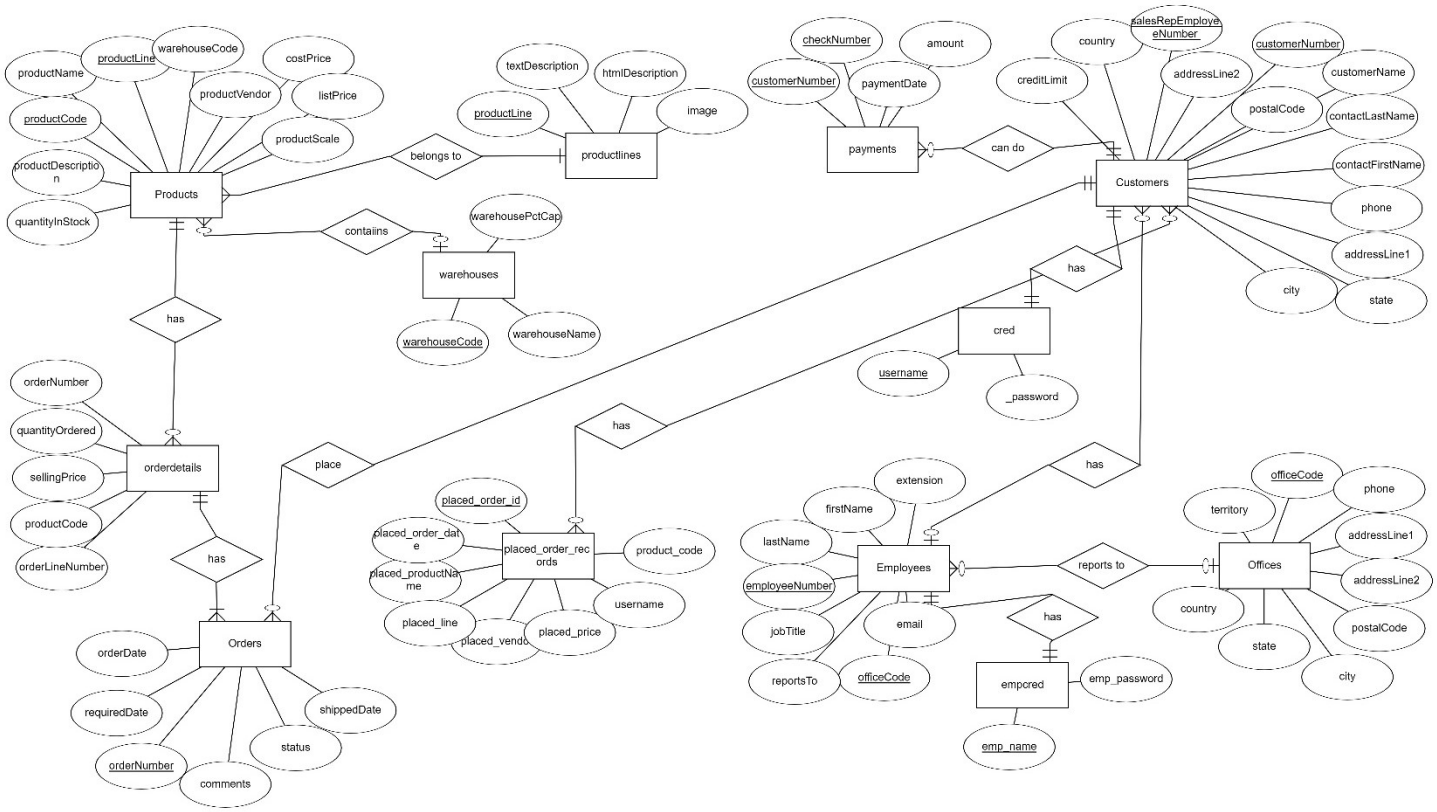

Fig.1. ER Diagram

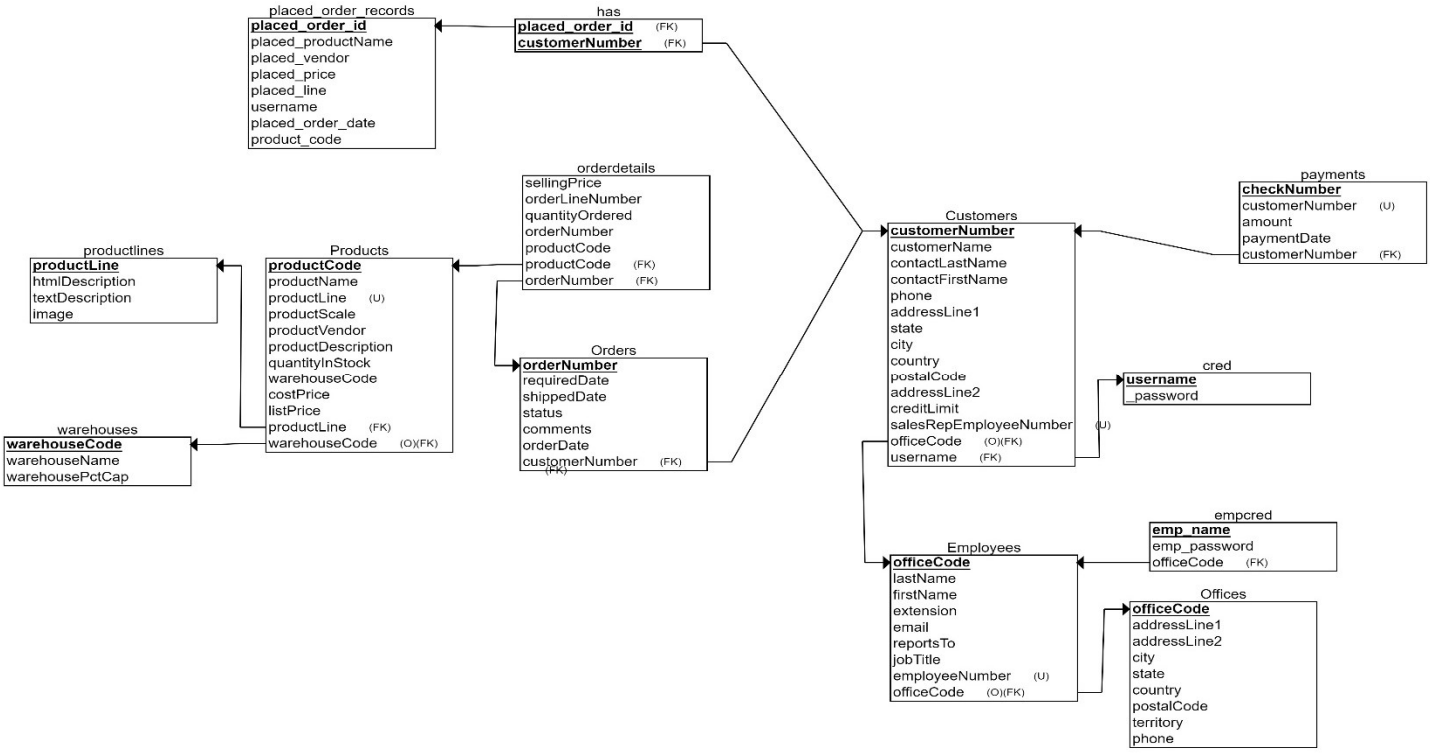

Fig.2.Relational Schema

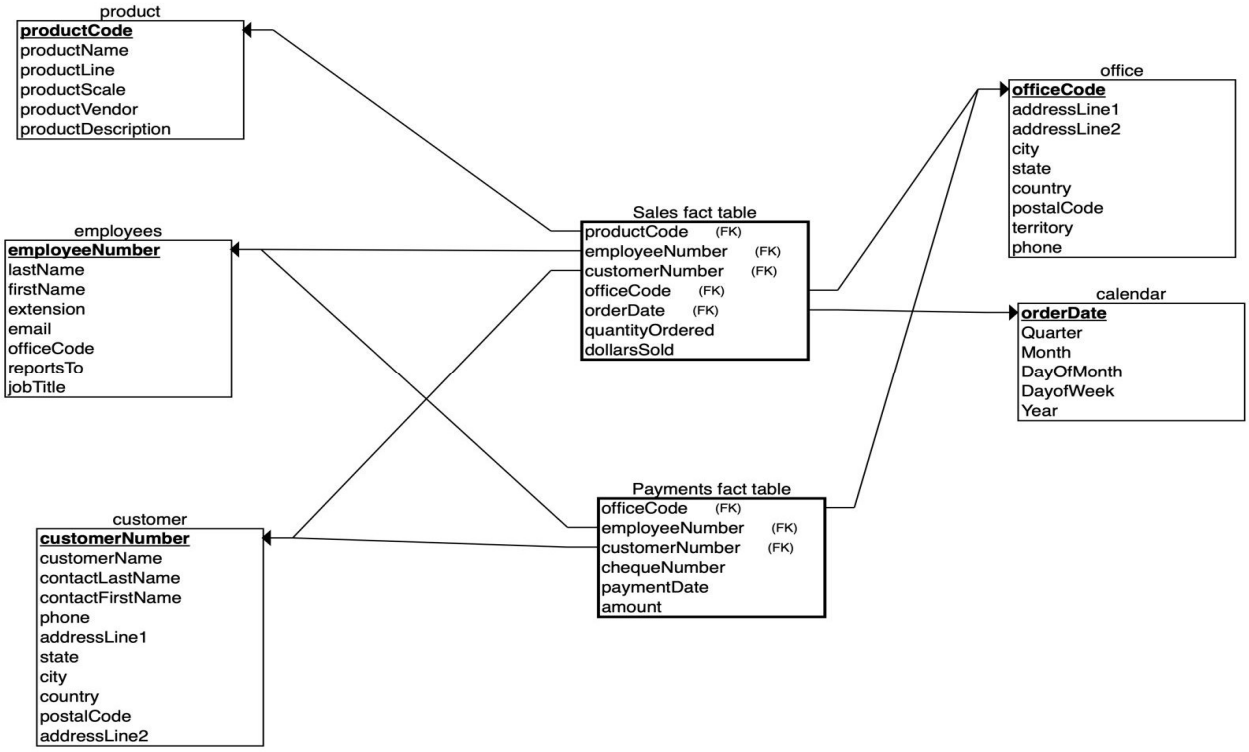

Fig.3.Star Schema

Working of the Operational Module:

The operational module revolves around two primary roles: the user and the warehouse employee, each equipped with distinct functionalities.

For instance, users can log in or create a new account, enabling them to place orders for desired products from available warehouses. Upon placing an order, a unique identifier is generated, and the employee is notified, treating it as a request. Users also have the capability to track their previous orders.

On the other hand, employees can access a comprehensive list of all orders received over time. They can utilize the details of these orders and proceed with processing them. Once an order is processed by an employee, customers can view it in their order history.

This operational module thus streamlines the interaction between users and warehouse employees, providing a seamless process for order placement, verification, and fulfillment.

Graphical User Interface:

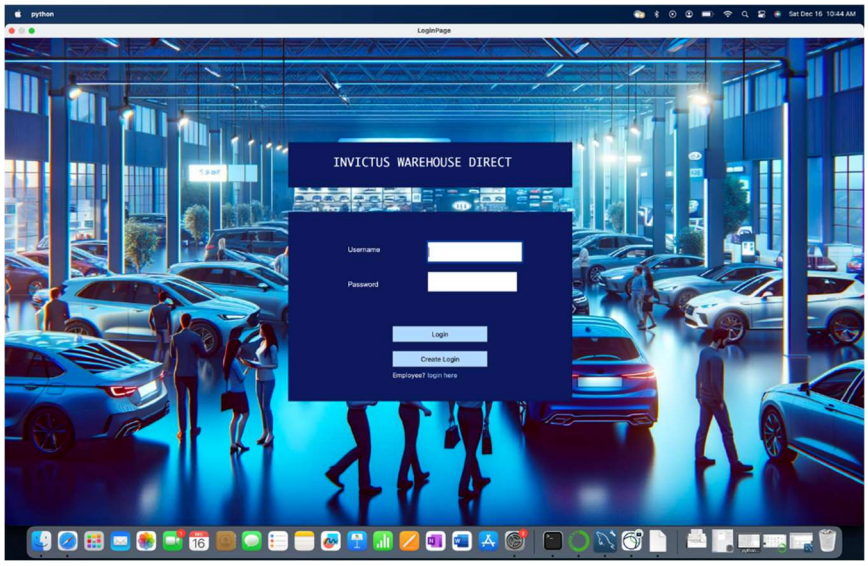

Fig.4. User login page

The application features a login portal that allows for secure access via a username and password. For newcomers, there is an option to set up a new user account.
The system is designed to accommodate two distinct user roles:

1. ***User***: After logging in, users have the capability to browse and place orders for desired products.
2. ***Employee***: Employees can review customer orders and manage the order fulfillment process.

- Account creation is initiated by selecting the ***'Create Login'*** button.

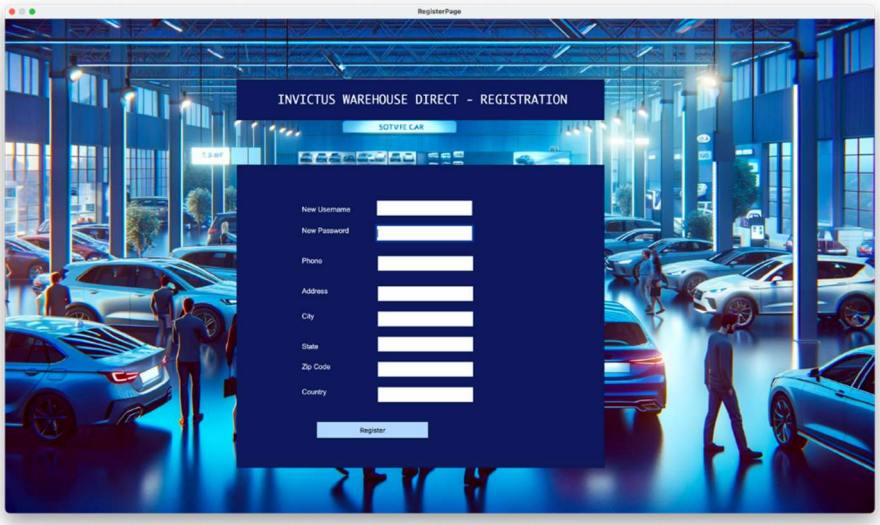

Fig.5. User Registration page

- During this process, individuals are required to input details such as a username, password, phone number, address, city, state, and country.
- The username that is selected must be unique; attempts to register using a username that is already in the system will be unsuccessful.
- Every field in the registration form is required. Users must complete all the details; otherwise, the submission will fail.

- Upon clicking 'Register,' assuming all criteria are met, the account is successfully established, and the new user's information is securely stored in the database.

After creating a user, the user can login into our application.

a. User Welcome Page:

After a successful login, the user is directed to the welcome page.
The welcome page provides two main options:

- **Place New Order**
- **Previous Orders**
- **Logout**

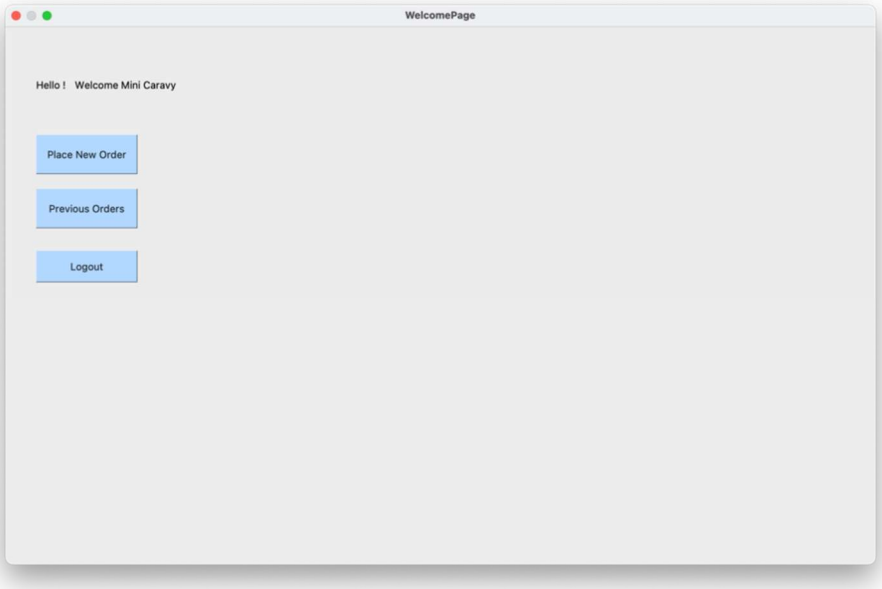

Fig.6. User welcome page

b. Place New Order:

Upon selecting ***"Place New Order,"*** the user is presented with a dropdown menu containing a list of available products.
When a product is selected, the details of the selected product are displayed in a table.
Simultaneously, the availability of the product (warehouse location) is displayed in another table.
After reviewing the product details and availability, the user can press the "***Place Order***" button to initiate the order placement process.

Order Placement Process:

Once the user clicks "**Place Order**," the order is submitted.
The order is then sent to the sales representative for further processing.

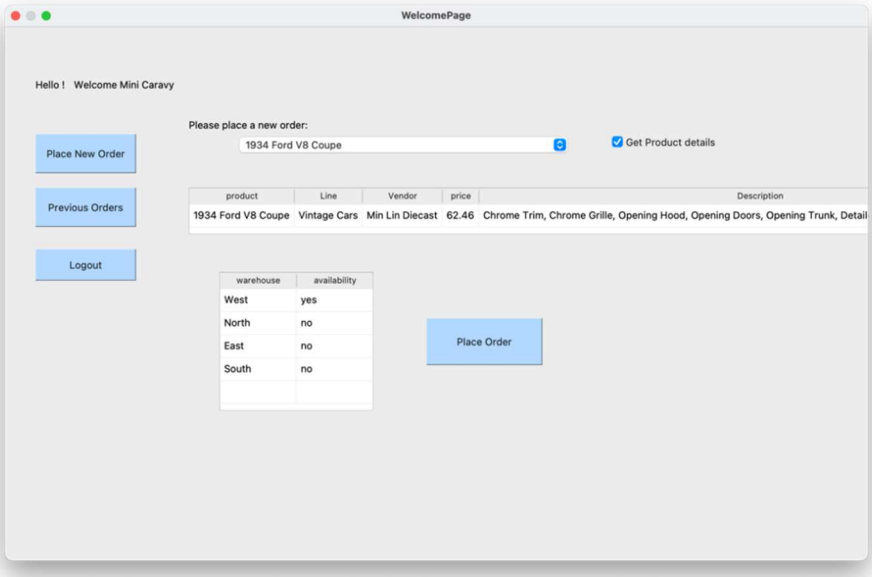

Fig.7.Order placement

c. Previous Orders:

Clicking on "***Previous Orders***" allows users to view a list of their previously placed orders.
Each order entry includes relevant details such as order number, order date, product and Amount.

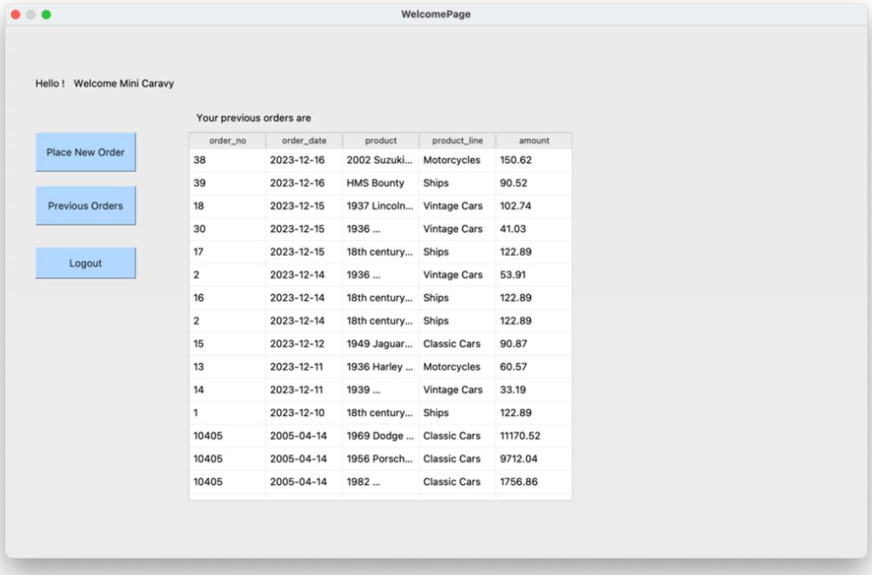

Fig.8. Previous orders page

d. Logout:

Clicking on "Logout" allows the user to securely log out of their account.

e. Employee Login Page:

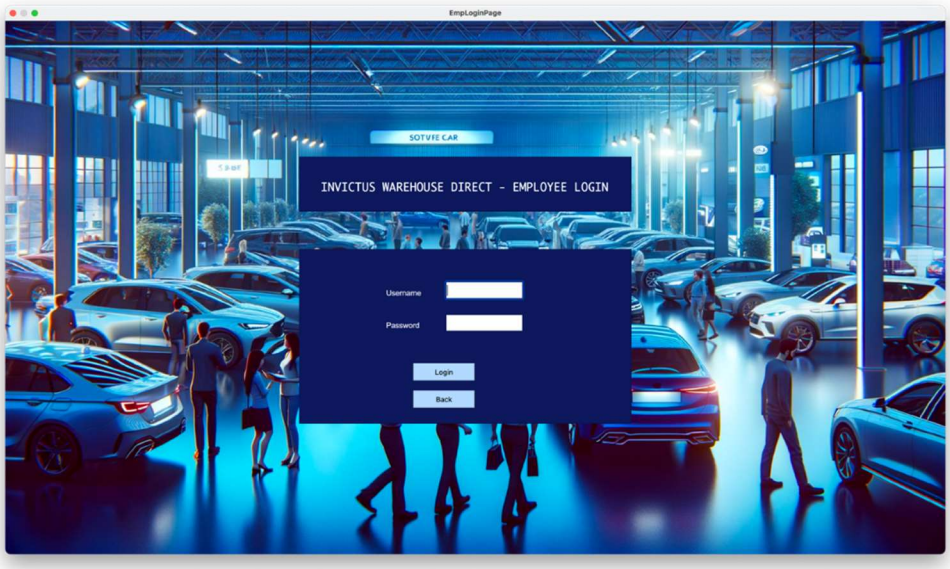

Fig.9.Employee login page

The employee login portal is accessible through the '***Employee'*** label. Employees can enter the portal using their specific credentials.
Upon successful login, employees are presented with two main navigation options:

1. ***Order Requests***: This section displays all orders placed by users, including details such as order ID, order date, line items, vendor details, and product codes.
2. ***Dashboards***: This area provides overview and analytics related to order processing and other relevant operational metrics.

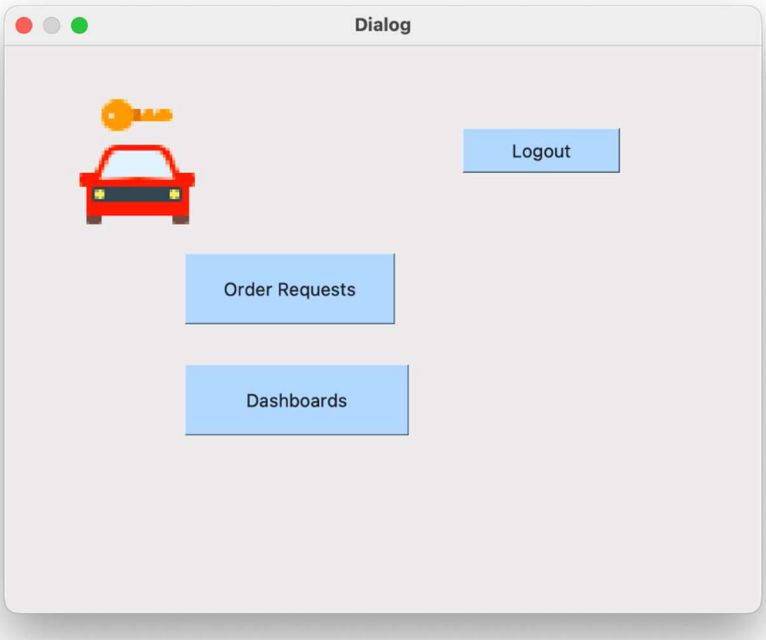

Fig.10.Employee welcome page

Selecting '***Order Requests***' allows employees to review the listed orders. They can then take the necessary steps to process each order accordingly.

After an order has been processed, the corresponding details are made available to the user under the ***'Previous Orders'*** tab, where they can track the status and history of their purchases.

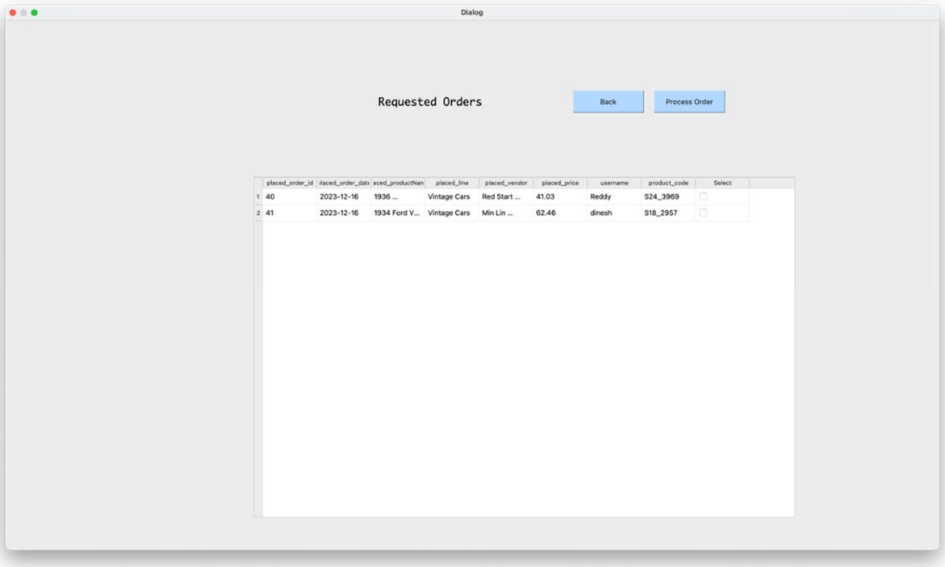

Fig.10. Orders request page

Working of Analytical Module:

The Analytical Module employs data from the warehouse database to create dashboards offering valuable insights into sales, stock levels, and warehouse performances. The sales dashboard offers a detailed view of annual sales performance, highlighting the contributions of both customers and employees. The stocks dashboard analyzes the quantity of overstocked products in each warehouse, while the warehouse dashboard provides information about overall warehouse performance. These analytics will aid us in making decisions regarding which warehouse to consider for closure.

Graphical Interface – Warehouse:

Analytics welcome page:

Fig.11. Analytics welcome page

- Clicking on the analytics button on the employee welcome page redirects to the 'ANALYSIS OF WAREHOUSES AND PRODUCTS' page.
- This page enables employees to choose the specific dashboard that aligns with their information needs.

a. Sales Dashboard:

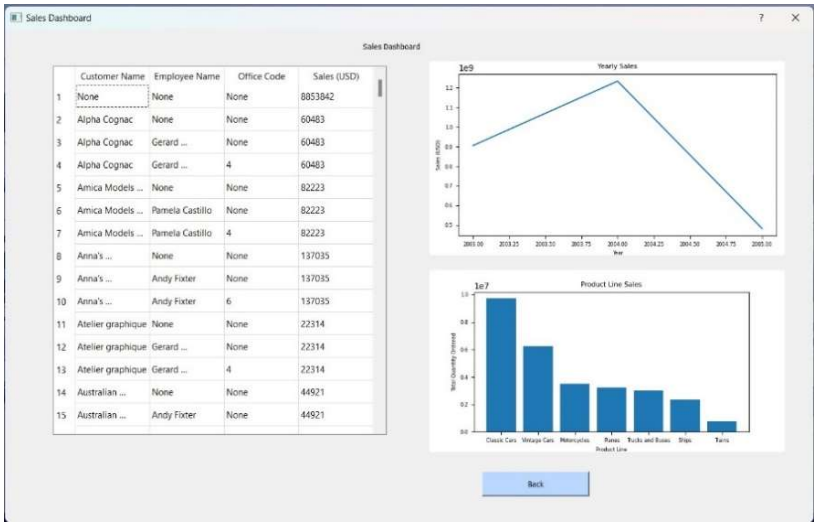

fig.11.Sales Dashboard

- The dashboard provides a high-level view of sales performance over time and by product line, with a detailed data table for granular analysis.
- The data table utilizes the ’ROLL UP’ function of MySQL which helps the employees to monitor the customer by employee performance and total sales contribution of each customer.
- The 'Yearly Sales' graph presents a chronological sequence of sales data. It clearly indicates a substantial decline in sales starting in 2004, which resulted in the strategic decision to shut down one of the warehouses.
- The 'Product Line Sales' chart outlines the distribution of sales among various product lines. This information is crucial in making strategic choices regarding which products should be discontinued in the warehouses, considering both the space they occupy and their sales performance.

b. Stocks Dashboard:

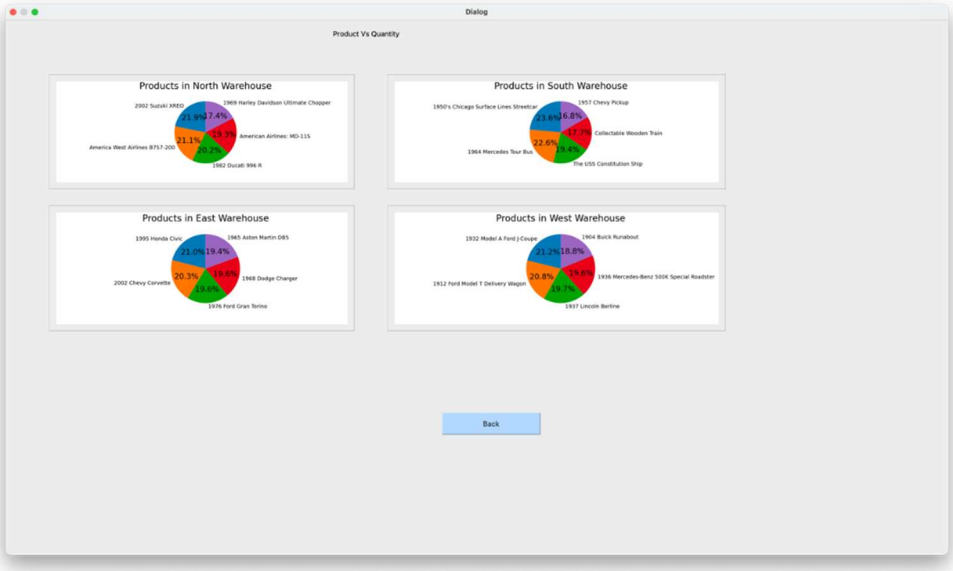

Fig. 12. Stocks Dashboard

- The stock dashboard displays the top 5 products with the highest stock in quantity in each of the four warehouses: East, West, North, and South. This information is visualized through four pie charts. This information can be utilized when deciding which products to eliminate while closing a warehouse and shifting its stocks to another warehouse.

c. Warehouse Dashboard:

Fig.13.Warehouse Dashboard

- The dashboard is designed to monitor and analyze warehouse performance, comparing the number of orders handled by different warehouses.
- The plot depicts that the East warehouse processes the most orders, in contrast to the West warehouse, which processes a relatively smaller number of orders.
- The text box details specific inventory items for the selected warehouse, highlighting products that, despite high stock levels, are yielding the least sales, and are suitable for eliminating.

Technical Aspects:

To initiate the application, open the Jupyter Notebook and execute the file named 'Invictus warehouse direct app.ipynb'. This main file manages the entire project's execution. Code corresponding to each GUI page is organized within this file, differentiated by appropriate headings. Various classes and functions have been created for different pages, and they are invoked as needed.

1. *Common Files:*
   ***Invictus_db.ini* –** When the program requires database access, it reads this INI file to retrieve the database connection details of Invictus_db.
   ***Invicuts_wh.ini* –** Utilizing this .ini file, connection can be established to the 'Invictus_wh' database in the MySQL workbench.
   ***Data225utils.py* –** This is the utility file where the code for connecting python to MySQL and other common code are written and used for the project.

2. *ETL:*
   Created **Dimensional Tables**:

Customer, Office, Employees, Calendar (Hierarchical), products, warehouses, productLines
Created **Fact tables**:
sales_facttable, payments_facttables

Operational Queries:

O.1) Query to display product availability at respective warehouses:(PLACE NEW ORDER)

```
SELECT w.warehouseName,
CASE WHEN p.quantityInStock > 0 AND
p.productName = '{product}' THEN 'yes'
ELSE 'no' END AS stockStatus
FROM invictus_db.warehouses w
LEFT JOIN invictus_db.products p
ON w.warehouseCode = p.warehouseCode
AND p.productName = '{product}'
order by stockStatus desc
```

When the order is placed, and not processed by the employee yet, the respective order details will be stored in **placed_order_records**

O.2) Query to display previous orders of a particular Customer: (PREVIOUS ORDERS)

```
SELECT orderdetails.orderNumber as
order_number,orders.orderDate
 as Date, productName as product, productLine
as Line, quantityOrdered*sellingPrice
as amount FROM products, orderdetails,
orders, customers c WHERE
products.productCode = orderdetails.productCode
and orders.orderNumber
=orderdetails.orderNumber
and c.customerNumber = orders.customerNumber
and c.customerName = '{username}'
order by orders.orderDate DESC
```

When the customer places an order, the details are stored into **'placed_order_records'** table, once the employee processes the order after the item is picked up, the respective details are moved into **'orders'** and **'order_details'** tables.

Analytical Queries:

A.1) Query to display data table in 'SALES DASHBOARD' showing the performance of employee by sales contribution of their customers.

```
SELECT DISTINCT c.customerName,
CONCAT(e.firstName, ' ', e.lastName) AS
employeeFullName, pf.officeCode, SUM(amount) as
salesUSD FROM payments_facttable pf
JOIN employees e ON pf.employeeNumber =
e.employeeNumber JOIN customer c ON
pf.customerNumber = c.customerNumber
JOIN office o ON pf.officeCode = o.officeCode
GROUP BY c.customerName, employeeFullName,
pf.officeCode WITH ROLLUP;
```

Displayed a table in the sales dashboard using **'ROLL UP'** operation of MySQL where the sales performance of both the customers and the employees are shown simultaneously.

A.2) Query to plot the performance of sales over years. (SALES DASHBOARD)

SELECT SUM(dollarsSold), YEAR(orderDate) as Year_ FROM sales_facttable GROUP BY Year_

Utilized this query to draw a timely line chart, to compare the sales performance over previous years.

A.3) Query to draw a bar plot to compare the sales performance by product lines(SALES DASHBOARD)

```
SELECT p.productLine, SUM(s.quantityOrdered) AS
totalQuantityOrdered FROM product p JOIN
sales_facttable s ON p.productCode = s.productCode
GROUP BY p.productLine ORDER BY
totalQuantityOrdered DESC;
```

The above query provides information about which product lines such as classic cars, vintage cars, ships, trains etc., generate more sales.

A.4) Query to display top overstocked products in each of the warehouses (STOCKS DASHBOARD)

SELECT productName, quantityInStock FROM products WHERE warehouseCode='{warehouse_code}' GROUP BY productCode,warehouseCode ORDER BY quantityInStock desc LIMIT 5;

Pie charts generated from the above query can be utilized in analyzing which products are overstocked in each of the warehouses. Further analysis on their respective sales performance will aid us in taking a decision on which product to eliminate.

A.5) Query to display the number of orders received by each warehouse.

SELECT w.warehouseName as Warehouse, COUNT(o.orderNumber) as Orders FROM orderdetails o, warehouses w, products p WHERE w.warehouseCode = p.warehouseCode AND o.productCode = p.productCode GROUP BY w.warehouseCode;

The analytical query A.5 is employed to showcase the comparative performance of individual warehouses by tallying the number of received orders. This information proves beneficial in pinpointing warehouses exhibiting lower performance levels.

A.6) Query to display the information related to selected warehouse.

*CREATE VIEW **RankedProductsView** AS SELECT w.warehouseName,p.productName, p.quantityInStock,ROW_NUMBER() OVER (PARTITION BY w.warehouseName ORDER p.quantityInStock DESC) AS rank_q FROM warehouses w JOIN products p ON w.warehouseCode = p.warehouseCode;*

The above view displays the quantity of stock in each warehouse.

*SELECT CONCAT('In warehouse ', r.warehouseName, ', the following products have generated sales:\n', GROUP_CONCAT(r.productDetails SEPARATOR '\n'), '.') AS warehouseSalesSummary FROM (SELECT r.warehouseName, CONCAT(r.productName, ' with a stock of ', r.quantityInStock,' units, has generated sales worth $',FORMAT(SUM(o.quantityOrdered * o.sellingPrice), 2)) AS productDetails FROM **RankedProductsView** AS r INNER JOIN products AS p ON r.productName = p.productName INNER JOIN orderdetails AS o ON p.productCode = o.productCode WHERE r.rank_q IN (1, 2) GROUP BY r.warehouseName, r.productName, r.quantityInStock) AS r WHERE r.warehouseName = %s GROUP BY r.warehouseName ORDER BY r.warehouseName;*

The above query utilizes MySQL's CONCAT function to populate the warehouse information box with details about the two highest overstocked products that have comparatively lower sales performance. This information proves valuable for determining which products, being both overstocked and contributing minimally to sales, can be considered for removal from the warehouses.

ANALYSIS PERFORMED FOR FINDING SUITABLE WAREHOUSE TO BE CLOSED AND PRODUCTS THAT CAN BE ELIMINATED.

Created a view to display total stock currently present in each of the warehouses and ideal capacity (80%) of each warehouse.

*CREATE VIEW warehouse_view AS SELECT w.warehouseCode, w.warehouseName, w.warehousePctCap, SUM(p.quantityInStock) AS TotalQuantityInStock, CAST((80 * SUM(p.quantityInStock) w.warehousePctCap) AS SIGNED) AS IdealStockCapacity FROM products p*
INNER JOIN warehouses w ON p.warehouseCode =w.warehouseCode GROUP BY w.warehouseCode, w.warehouseName, w.warehousePctCap;

warehouseCode	warehouseName	warehousePctCap	TotalQuantityInStock	IdealStockCapacity
a	North	72	131688	146320
b	East	67	219183	261711
c	West	50	124880	199808
d	South	75	79380	84672

The above query calculates the total quantity of products present at each of the warehouses and the ideal number of products that can fit in each warehouse that is calculated for 80%, which is given by the column IdealStockCapacity.

Analysis:
From the results obtained we can see that West warehouse is currently occupied with 50% of its capacity, making it suitable for combining with another warehouse.

Query to display the products that are out of the capacity of a warehouse when combined with another one.

SELECT CONCAT (v1.warehouseName, ' + ', v2.warehouseName) AS WarehouseCombination, v1.TotalQuantityInStock + v2.TotalQuantityInStock AS CombinedStockQuantity, CONCAT('IdealStockCapacity of ', v1.warehouseName, ': ', v1.IdealStockCapacity) AS IdealStockCapacityWarehouse1, v1.IdealStockCapacity - (v1.TotalQuantityInStock + v2.TotalQuantityInStock) AS leftoverProduct FROM warehouse_view AS v1
INNER JOIN warehouse_view AS v2 ON v1.warehouseCode < v2.warehouseCode
UNION ALL SELECT CONCAT(v2.warehouseName, ' + ', v1.warehouseName) AS WarehouseCombination, v2.TotalQuantityInStock + v1.TotalQuantityInStock AS CombinedStockQuantity, CONCAT('IdealStockCapacity of ', v2.warehouseName, ': ', v2.IdealStockCapacity) AS IdealStockCapacityWarehouse2, v2.IdealStockCapacity - (v2.TotalQuantityInStock + v1.TotalQuantityInStock)
AS leftoverProductsWarehouse2 FROM warehouse_view AS v1 INNER JOIN warehouse_view AS v2 ON v1.warehouseCode < v2.warehouseCode ORDER BY WarehouseCombination;

	WarehouseCombination	CombinedStockQuantity	IdealStockCapacityWarehouse1	leftoverProducts
▶	East + North	350871	IdealStockCapacity of East: 261711	-89160
	East + South	298563	IdealStockCapacity of East: 261711	-36852
	East + West	344063	IdealStockCapacity of East: 261711	-82352
	North + East	350871	IdealStockCapacity of North: 146320	-204551
	North + South	211068	IdealStockCapacity of North: 146320	-64748
	North + West	256568	IdealStockCapacity of North: 146320	-110248
	South + East	298563	IdealStockCapacity of South: 84672	-213891
	South + North	211068	IdealStockCapacity of South: 84672	-126396
	South + West	204260	IdealStockCapacity of South: 84672	-119588
	West + East	344063	IdealStockCapacity of West: 199808	-144255
	West + North	256568	IdealStockCapacity of West: 199808	-56760
	West + South	204260	IdealStockCapacity of West: 199808	-4452

The above query utilizes UNION ALL function of MySQL to union warehouses table with the same warehouses table, such that we can obtain the total stock in each combination of warehouses when one warehouse is combined with another. Since we already calculated the ideal capacity of each of the warehouses, now we can see whether the combination exceeds the ideal capacity of warehouse after combination. This is displayed in the column 'leftOverProducts'. Finally, we choose the warehouse that is providing less wastage of products after combination of warehouses.

Analysis:
Combining WEST warehouse with SOUTH warehouse gives a total stock quantity of 204,260 products, when this combined stock quantity is subtracted from the Ideal Stock Capacity, it gives the number of products that are exceeding the overall stock capacity of the warehouse. From the results obtained we can see that the least wastage of products, which is (4452) i.e., 4452 products are left out when WEST warehouse is combined with SOUTH warehouse.

Query to display productLines in WEST and SOUTH warehouses:

```
SELECT warehouseCode, productLine
FROM products
WHERE warehouseCode in ('c', 'd')
GROUP BY productLine, warehouseCode
```

	warehouseCode	productLine
▶	d	Trucks and Buses
	c	Vintage Cars
	d	Ships
	d	Trains

Since the WEST and SOUTH warehouses have products from productLines Trucks and Buses, VintageCars, Ships and Trains we try to eliminate least performing products from these productLines.

Query to display least performed products from WEST and SOUTH warehouses:

```
SELECT productCode, productName, quantityInStock,productLine, totalQuantityOrdered, averageQuantityOrdered from
(SELECT prd.productCode,prd.productName,prd.quantityInStock,prd.productLine, sum(ordDet.quantityOrdered) AS
totalQuantityOrdered, AVG(ordDet.quantityOrdered) AS averageQuantityOrdered FROM products AS prd
INNER JOIN orderdetails AS ordDet ON prd.productCode = ordDet.productCode WHERE prd.productLine IN ('Vintage
Cars', 'Ships', 'Trucks and Buses', 'Trains') GROUP BY prd.productCode, prd.productName, prd.quantityInStock ORDER
```

BY totalQuantityOrdered) AS subquery WHERE totalQuantityOrdered < 1000 AND averageQuantityOrdered between 32 and 35 order by averageQuantityOrdered LIMIT 5;

productCode	productName	quantityInStock	productLine	totalQuantityOrdered	averageQuantityOrdered
S18_1367	1936 Mercedes-Benz 500K Special Roadster	8635	Vintage Cars	962	32.0667
S24_1937	1939 Chevrolet Deluxe Coupe	7332	Vintage Cars	938	32.3448
S18_3136	18th Century Vintage Horse Carriage	5992	Vintage Cars	907	32.3929
S32_1268	1980's GM Manhattan Express	5099	Trucks and Buses	911	32.5357
S18_3140	1903 Ford Model A	3913	Vintage Cars	883	32.7037

The above query displays the total number of products orders for each of the productLine, and average of the products ordered for each of the productLine among Trucks and Buses, VintageCars, Ships and Trains, where the totalOrderedQuantity of the products are less than 1000 and average Ordered quantity is between 32 and 35 making them suitable for eliminating.

Analysis:
From the results obtained the products with product codes S18_1367, S18_3136, S18_3140,S24_1937,S32_1268 are suitable for elimination.

ANALYSIS CONCLUSION:

Through analytical evaluation and statistical methods, it has been determined that closing down the South warehouse and combining its operation with the West warehouse is the most efficient strategy. This approach is projected to minimize excess inventory. The analysis also gives a list of products, upon eliminating which the combination of warehouses can be left out with less wastage for the company because the existing inventory levels for these items substantially surpass their market demand.

To address this surplus, offering discounts and promotions on those products is recommended. Running flash-sales or marketing on social media and other strategies can be used by the company to eliminate these products from their warehouses. Therefore, consolidating warehouses emerges as a viable solution to significantly decrease sales compared to the previous year.