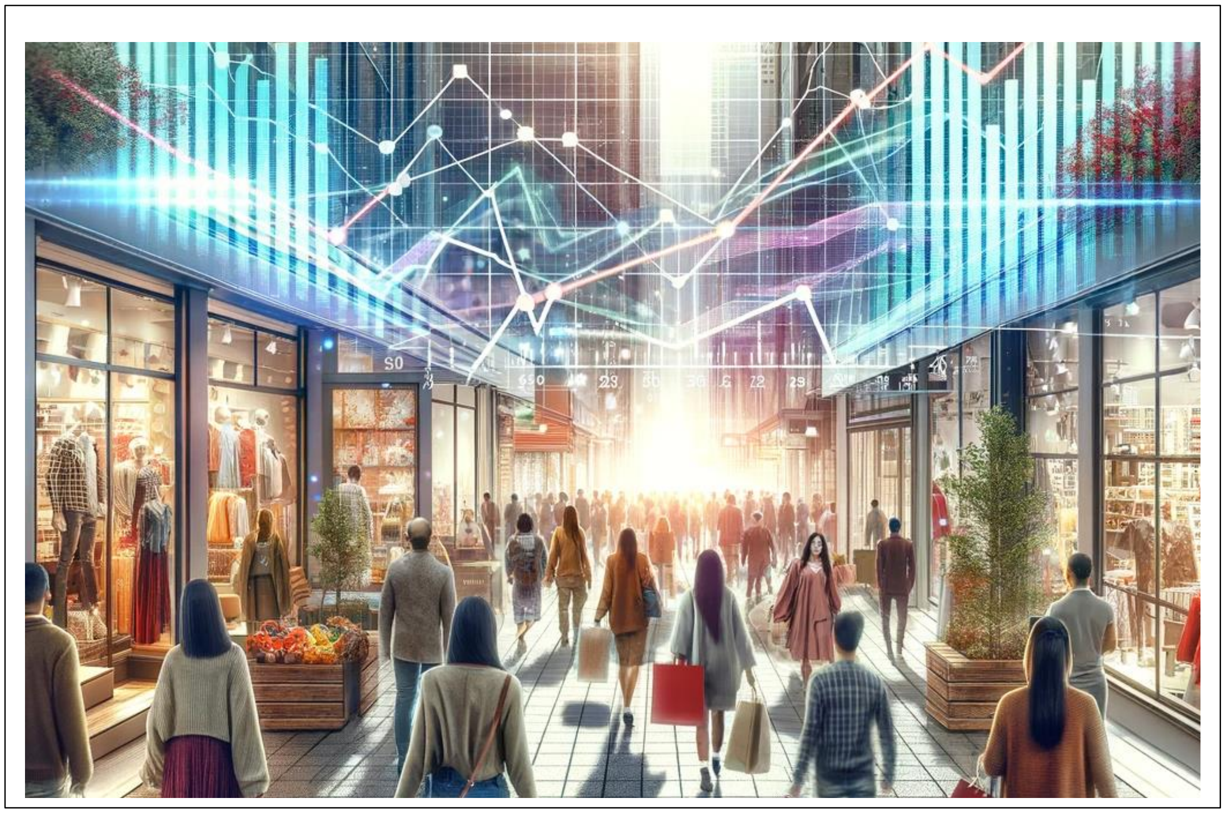
DB SYSTEMS FOR ANALYTICS

CUSTOMER SHOPPING TRENDS



GUIDED BY : PROF. RON MAK

DATA DYNAMOS TEAM:

Shrinivas Bhusannavar – 016990545
Harivardhana Naga Naidu Polireddi – 017437238
Yashasvi Kotra - 017466436
Karthik Nimmagadda – 016996148

INTRODUCTION:

In today’s rapidly changing business environment, the ability to effectively manage and interpret sales data is paramount. The Customer Shopping Analytics Tool is engineered to address this necessity, offering a robust framework for data management and analysis.

The application is designed for businesses, particularly focusing on enhancing their understanding and management of sales and customer data. Its core objective is to provide users with clear, actionable insights into prevailing shopping trends, which in turn facilitates informed decision-making and strategic business planning. It aims to provide clear insights into shopping trends, enabling better decision-making and strategic planning.

The operational aspect of the application is integral to managing daily sales processes, encompassing the collection and storage of sales transaction data to ensure accurate recording and analysis of customer purchases. It includes a billing management system that streamlines the creation and updating of customer bills for efficiency and accuracy, coupled with an inventory tracking system that adjusts stock levels in real-time to avoid overstocking or shortages. Additionally, the application incorporates an employee login management feature to secure access to the system and safeguard data integrity.

The analytical side of the application is all about understanding and using the data it collects to help businesses make better decisions. It looks at shopping trends to see what's happening now and what might happen in the future, which is helpful for businesses when they're figuring out what to stock up on, how to market their products, and how to plan their sales. It also looks closely at what customers like to buy and how they shop, so businesses can make sure they're selling the right things in the right way. Plus, the application can group customers into different categories, like age or how much they spend, which helps businesses know who to target their ads to and how to make shopping feel more personal for their customers.

DATA SOURCE:

LINK: https://www.kaggle.com/datasets/iamsouravbanerjee/customer-shopping-trends-dataset

The dataset provides valuable insights into consumer behaviors and buying patterns. Understanding these preferences is crucial for businesses to customize their products, marketing approaches, and overall customer experience. This dataset encompasses various customer attributes such as age, gender, purchase history, preferred payment methods, shopping frequency, and more. By analyzing this data, one can make informed decisions, optimize their product offerings, and improve customer satisfaction. This dataset serves as a valuable tool for companies seeking to align their strategies with customer needs and preferences. It's worth noting that this dataset is specifically designed for beginners to gain insights into data analysis making it a synthetic resource for learning purposes.

APPLICATION OVERVIEW:

The application, crafted with the Tkinter GUI, is engineered to conduct operational tasks and pinpoint consumer trends over time. At the application's home page, a strong user authentication system is in place to guarantee secure access and govern sensitive features. Login pages have been created for both store employees and administrators, along with a separate screen dedicated to analytics. These pages are pivotal in augmenting the security and user-friendliness of the application.

The application is built using a variety of tools, such as MySQL Workbench for database management, Tkinter Designer for the graphical user interface (GUI), and Python for ETL procedures. It also makes use of a dataset from Kaggle. For data processing, numerical calculations, visualizations, and graphical user interface activities, additional Python packages like Pandas, Numpy, Matplotlib, and Tkinter are used.

OPERATIONAL USE :

Store Employee Page

- Employees are granted access to essential operational functionalities through a dedicated login page.
- Each employee is required to enter a valid username and password to gain access.
- Then we created a Billing System page for employees where they can generate invoices for customer's purchased products.

Store Admin Page

- Store Admins are granted full access to all operational functionalities through a login page.
- Admin is required to enter his/her credentials to access the database.
- Admin has access to Inventory, Employees and Invoices where he/she can have access to critical information.

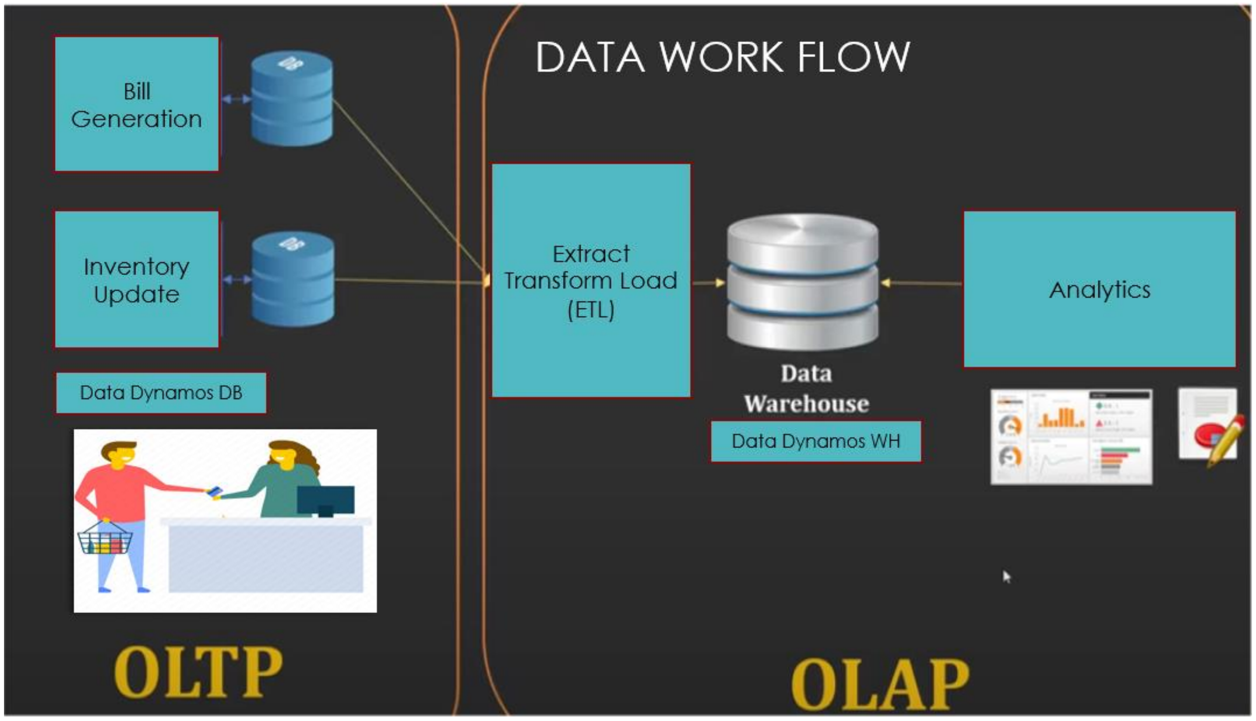
ANALYTICAL USE :

The Analytics page is designed to analyze the customer shopping trends over a time.

- **Revenue Dashboard Page**, analyzed the total revenue by Category, Season and Region.
- **Customer Demographics Page** , analyzed the Customer Demographics by Age, Gender and Category.
- **Roll up Operations Page**, analyzed the Total Sales of the Products by Year, Month and Quarter.

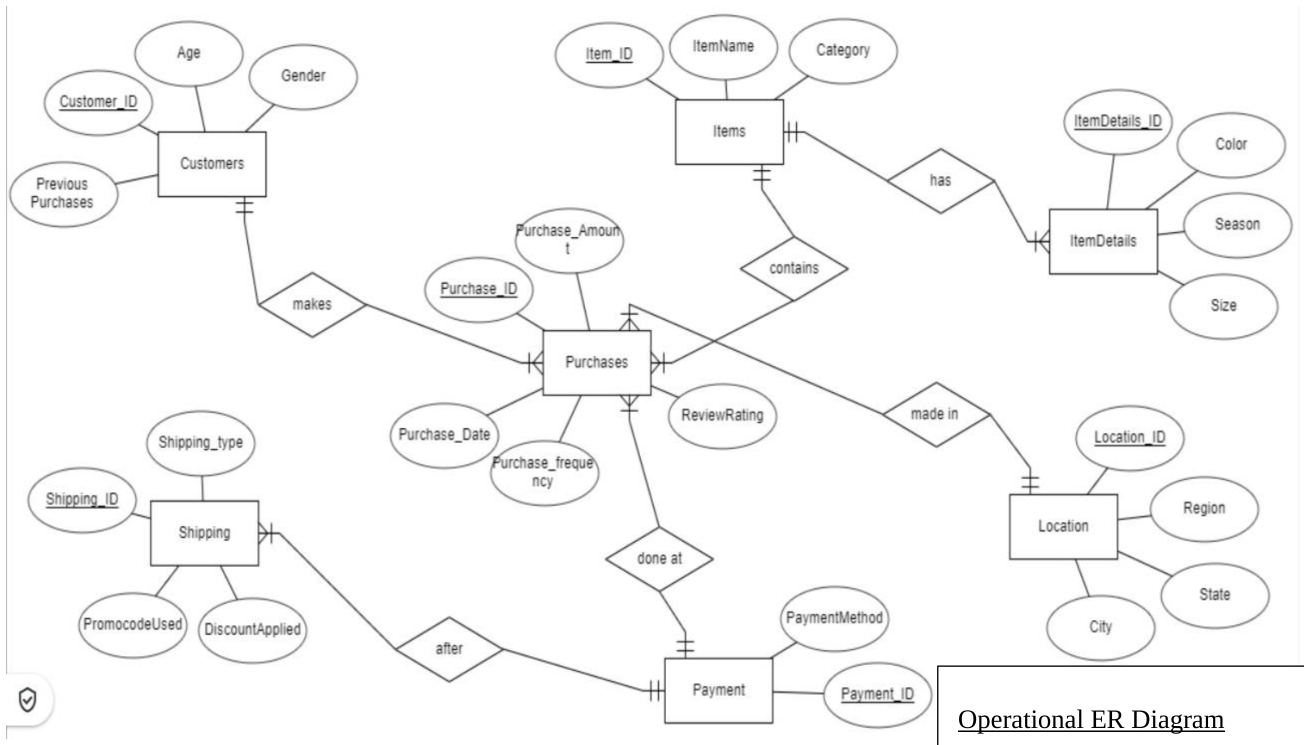
<u>DATA WORKFLOW:</u>

The workflow depicted in the image outlines a systematic approach to data management in a retail context. The process begins with the operational database (OLTP), where transactional data such as sales (bill generation) and product update (inventory update) are recorded. This data is then processed through an ETL (Extract, Transform, Load) procedure, where it is refined and transferred into a data warehouse .The warehoused information becomes the foundation for in-depth analytics, allowing for the extraction of valuable insights into customer behavior, sales trends, and inventory management. This structured data flow is essential for making informed business decisions and strategizing effectively.



DATABASE DESIGN:

Operational Database Design:



Operational ER Diagram

Entities and Attributes:

- **Customer**: This entity stores information about Customers such as their Customer ID, Age, Gender, and Location.
- **Items**: This entity stores information about items such as Item ID, Item Name, Category, Color, Size and Season.
- **Purchase**: This entity stores information about Purchases such as the Purchase ID, Purchase Date, Purchase Amount, and whether a Promo Code was used and Discount was applied.
- **Payment**: This entity stores information about Payments such as the Payment ID, Payment method, and Shipping type.
- **Location**: This entity stores information about Locations such as the Location ID, City, Region, and State.
- **Shipping**: This entity stores information about Shipping such as the shipping ID and shipping details.

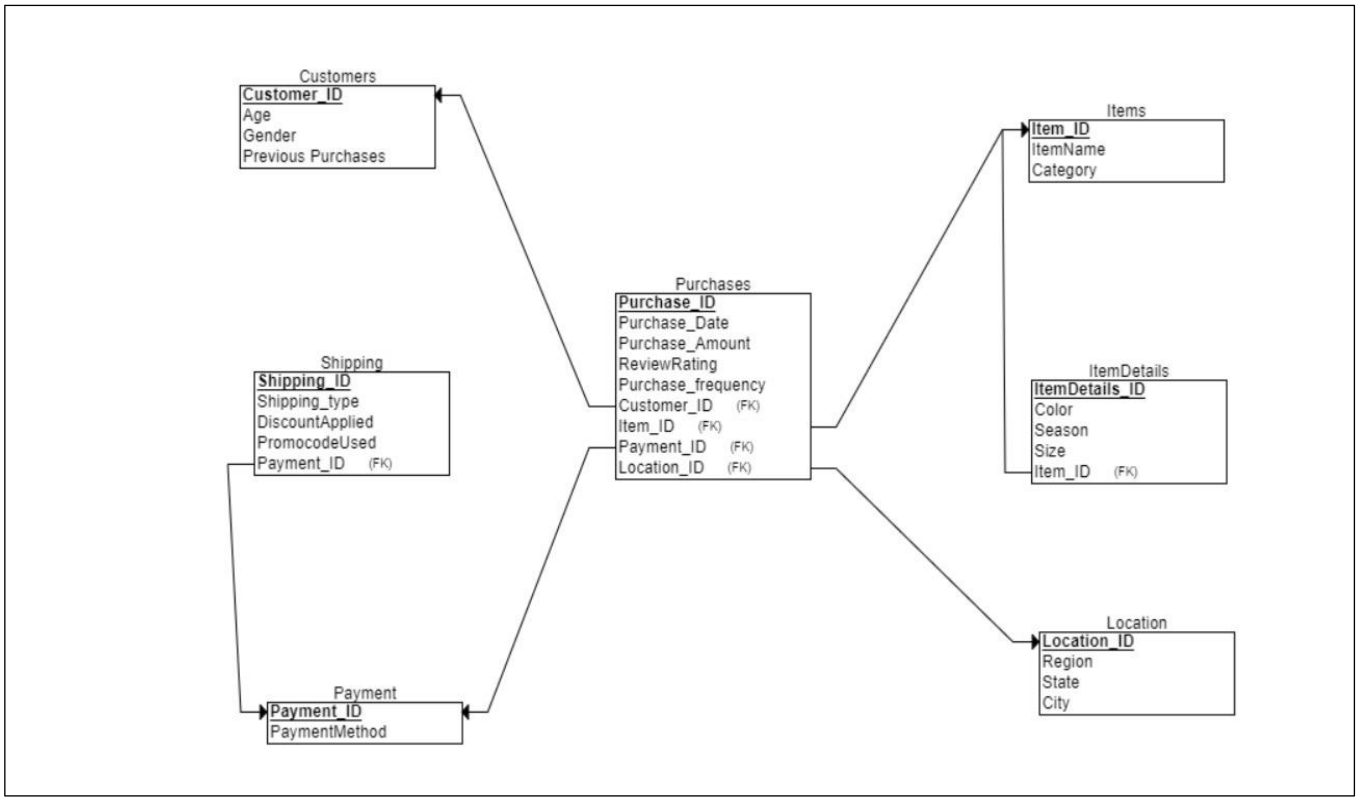
- **ItemDetails**: This entity stores additional details about Items, such as the Item Details ID, Purchase Frequency, and Review Rating.

<u>Relationships</u>:

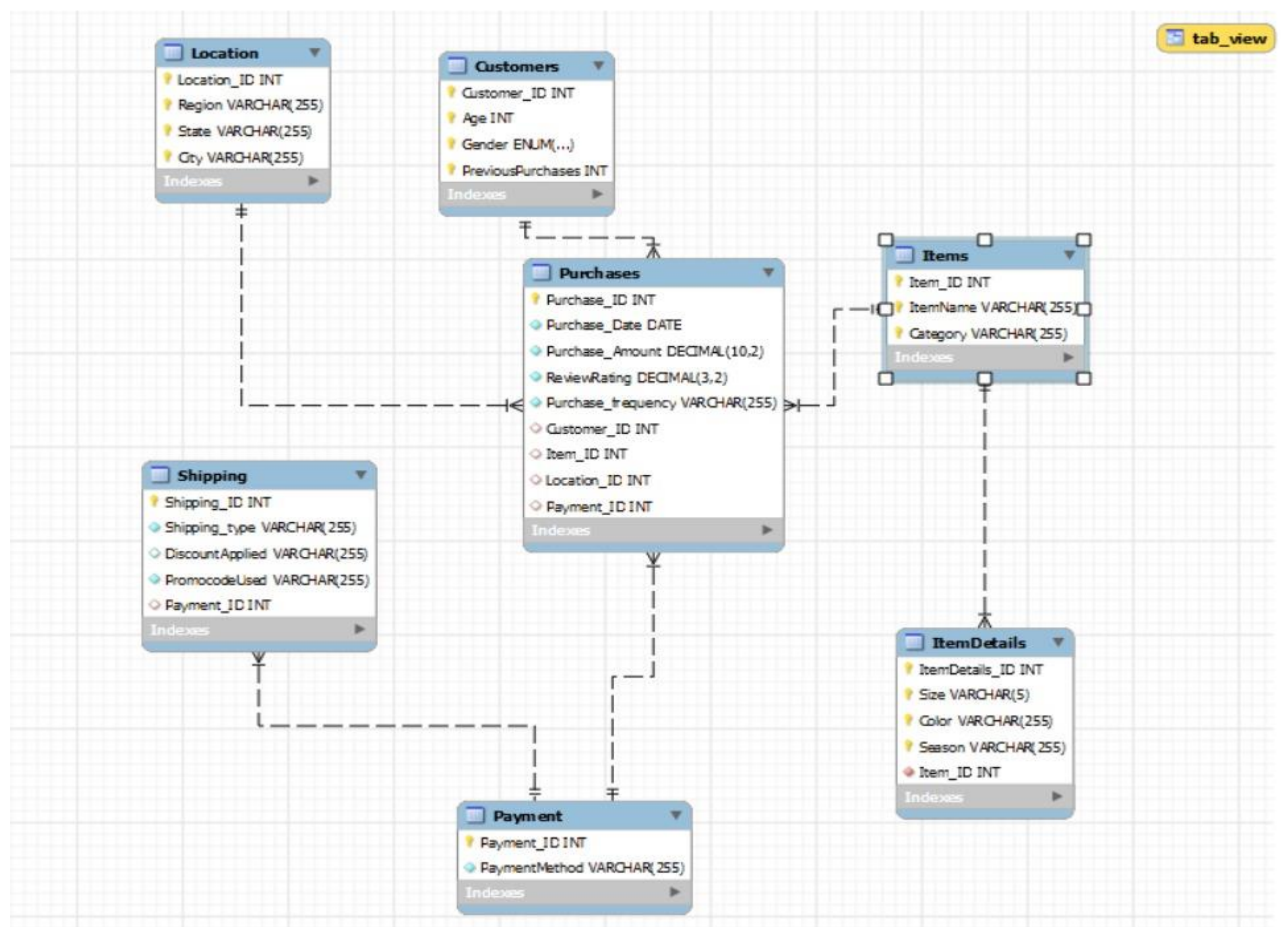
- A customer can make many purchases.
- A purchase contains one or more items.
- An item can be included in many purchases.
- A purchase is made by one customer.
- A purchase is made at one location.
- A purchase is shipped from one location.
- A payment is made for one purchase.
- An item has many item details.

<u>Constraints</u>:

- The Customer ID must be unique for each customer.
- The Item ID must be unique for each item.
- The Purchase ID must be unique for each purchase.
- The Payment ID must be unique for each payment.
- The Location ID must be unique for each location.
- The Shipping ID must be unique for each shipping.
- The Item Details ID must be unique for each Item details.



<u>Relational Schema</u>

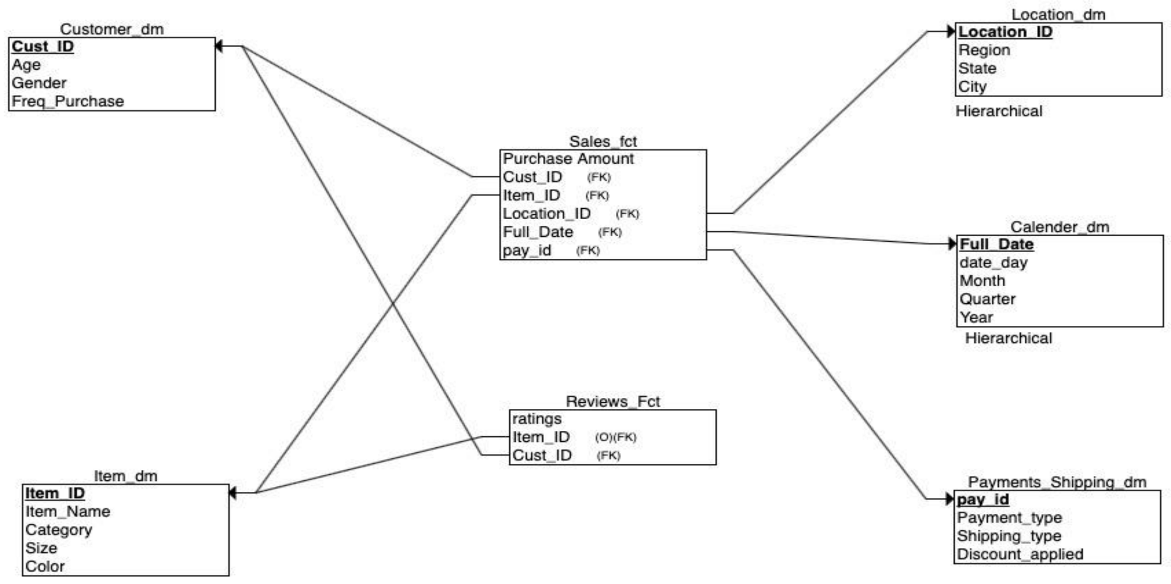


This relational schema can be used to track customer shopping trends by analyzing the data in the database. For example, you could use the data to:

- Identify which items are most popular with certain Customer Demographics.
- Track how Purchase trends change over time.
- Analyze the effectiveness of Promotions and Discounts.

ANALYTICAL DATABASE DESIGN:

OLAP star schema



The OLAP Star Schema depicted represents a multi-dimensional data model tailored for complex queries that are typical in business intelligence applications. At the core of this schema is the **Sales_fact** table, which holds transactional data such as Purchase Amount and links to various dimensions that provide context to these transactions.

The **Customer_dim** dimension table includes customer-specific attributes like Customer ID, Age, Gender, and Freq Purchase (presumably frequency of purchases), allowing businesses to segment their customers and analyze purchasing patterns across different demographic groups.

The **Item_dim** table provides details about the items sold, such as Item ID, Item Name, Size, Category, and Color, enabling an analysis of product popularity and stock management.

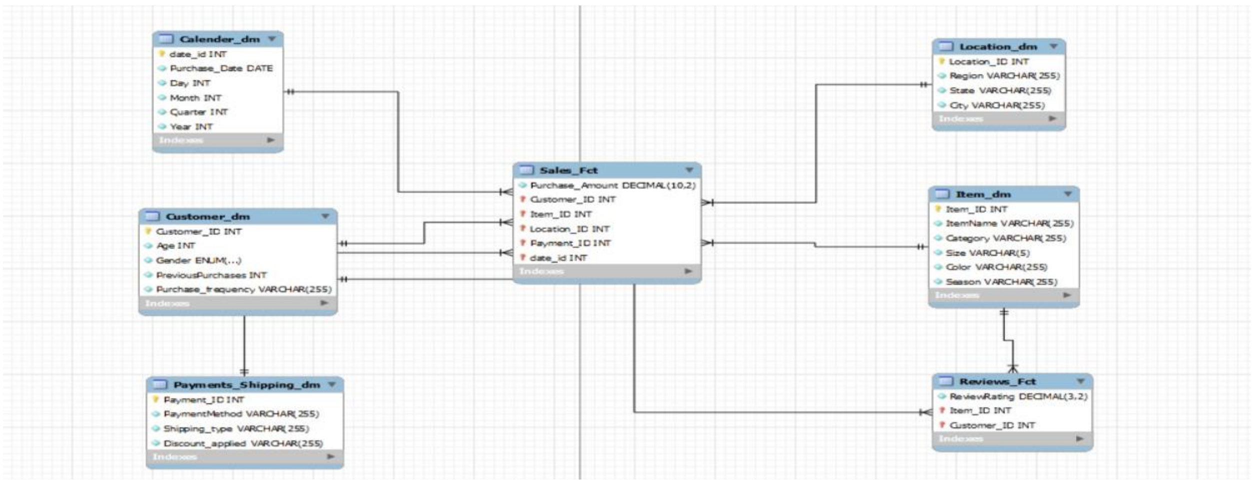
The **Location_dim** table contains location-related data (Location ID, Region, City, State), which can be used to determine regional sales performance and to tailor marketing campaigns geographically.

The **Calendar_dim** table is essential for time-series analysis, with attributes like Full Date, Day, Month, Quarter, and Year. This enables the tracking of sales over time, recognizing trends, and making seasonal adjustments.

Lastly, the **Payments_Shipping_dim** table includes payment and shipping details (Payment Type, Shipping_type, Discount_applied), which can help analyze the effectiveness of promotions and the efficiency of various shipping methods.

Together, these interconnected tables enable a comprehensive analysis of sales data, providing actionable insights that can inform business strategy and decision-making. This schema is essential for reporting, forecasting, and data mining in the context of online shopping analysis.

ANALYTICAL DB STRUCTURE (ERD Diagram)



ERD diagram delves into the nuanced relationships between the different aspects of the sales data. The Sales_Fact table, the centerpiece, captures every transaction with a unique granularity, which is critical for detailed sales analysis. The surrounding dimension tables like Customer_dim, Item_dim, and Location_dim contain descriptive attributes that allow for multifaceted examination of sales data.

For example, the Customer_dim table, by including fields for age and gender, facilitates demographic analysis of purchasing patterns. The Item_dim table allows for inventory tracking and sales performance analysis at the item level. The Location_dim table can provide insights into regional market trends.

ETL (Extract, Transform, Load) processes are critical for the functioning of our Customer Shopping Trends application, ensuring data is accurately and efficiently processed for both operational and analytical purposes.

Operational Database ETL

To populate the operational database, a Python script was developed. This script performs several key functions:

- **Data Extraction**: The script begins by extracting data from excel file, ensuring that all relevant information is gathered for processing.
- **Data Cleaning**: Once the data is extracted, the script performs data cleaning operations. This involves removing any inconsistencies, correcting errors, and ensuring the data is standardized for optimal database storage.
- **Data Loading**: With clean data, the script then uses SQL connectivity to load the data into the operational database. This involves executing SQL commands to insert the data into the appropriate tables and fields within the database.

The operational database serves as the foundation for daily transaction processing and management.

Analytical Database ETL

For analytical purposes, a separate Python script is used:

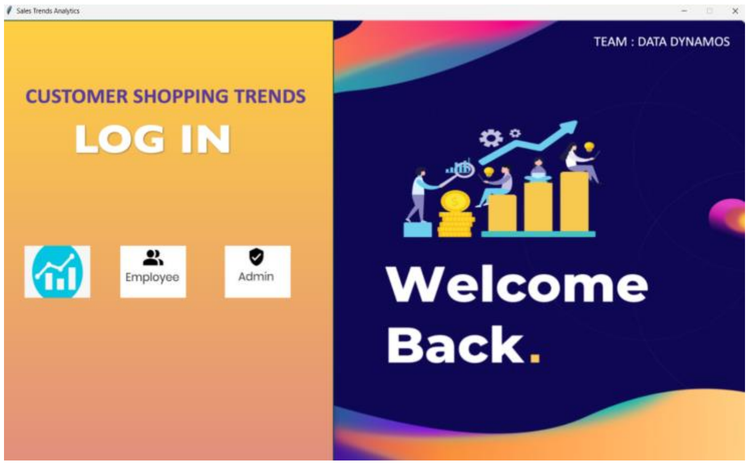
- **Data Reading**: This script starts by reading data from the operational database. It uses SQL 'SELECT' statements to fetch the required data for analysis.
- **Data Transformation**: The retrieved data might undergo some transformation to fit the analytical needs, such as aggregation, summarization, or any other form of manipulation that supports the analysis.
- **Data Loading**: Finally, the script uses SQL 'INSERT' statements to load the transformed data into the analytical database.

The analytical database is tailored to support complex queries and generate insights, which are crucial for understanding customer shopping trends and informing strategic business decisions.

In both ETL processes, Python's robust libraries and SQL's powerful data manipulation capabilities come together to create a streamlined data pipeline that feeds both the day-to-day operational needs and the strategic analytical requirements of the business.

Specifications and Usability of Operational Module

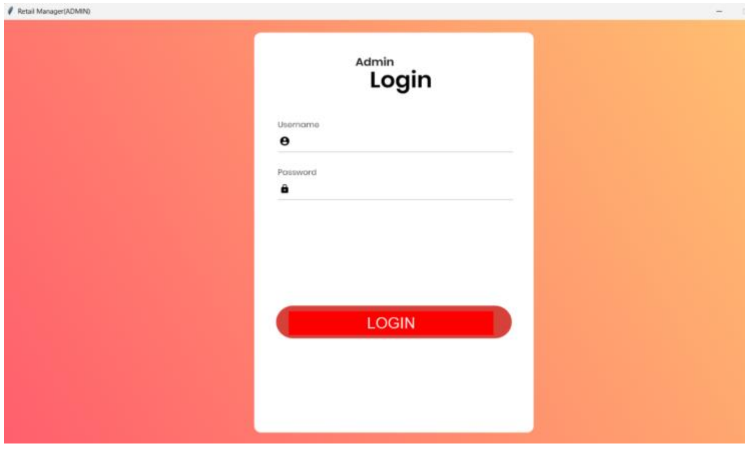
1. Home Login page



The Home page with the login section and the welcome message. The overall design is designed in modern and user-friendly, aimed at professionals dealing with sales and customer trends analysis.

The Login section is designed with three icons which individually take into their respective sections. One for **Admin**, the other for **Employee**, and **Analytical** Dashboard

Store Admin Login Page: on click of admin button in Home Page, the below page opens

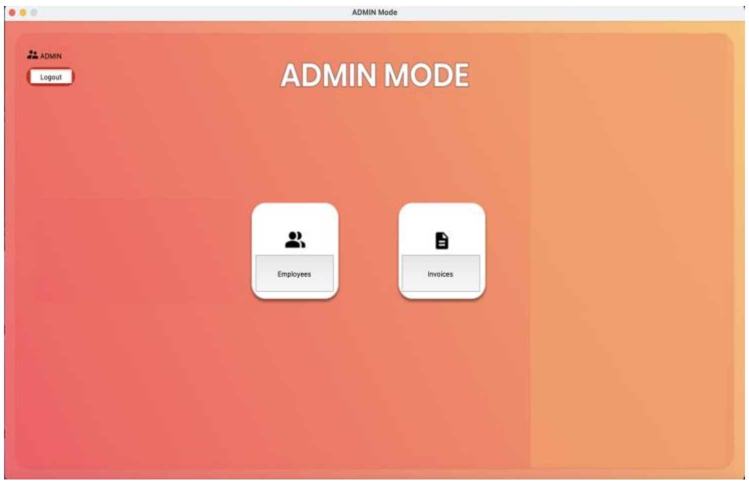


Created a Login Page for Store Admin where they have full access to all operational functionalities.

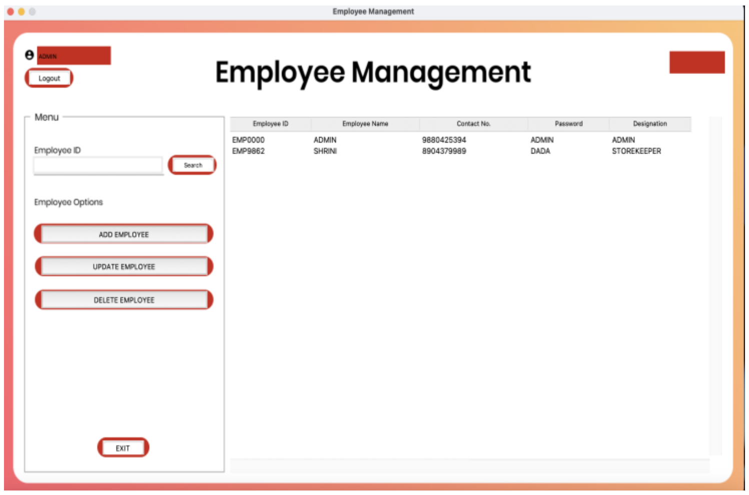
Admin must enter his/her credentials (Username and Password) to enter the admin portal.

Only with **admin** designation can login here

ADMIN PORTAL:



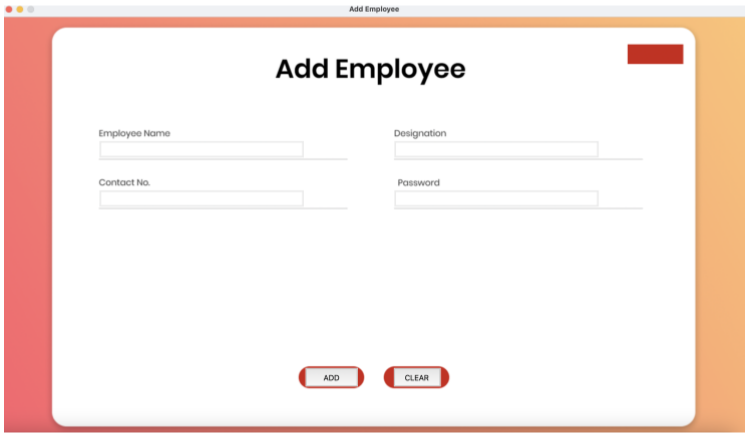
- In Admin Portal there are 2 main tabs which are, Employees, Invoices.
- Admin have the access to all the portals and he can track the status of the invoices generated and employees logins for application.



Employee ID	Employee Name	Contact No.	Password	Designation
EMP0000	ADMIN	9880425394	ADMIN	ADMIN
EMP9862	SHRINI	8904379989	DADA	STOREKEEPER

In this Page , store admin can search for all store employees, can add new employees and can delete employees.

And also, admin can update the employee's details so that employee information will be up to date.

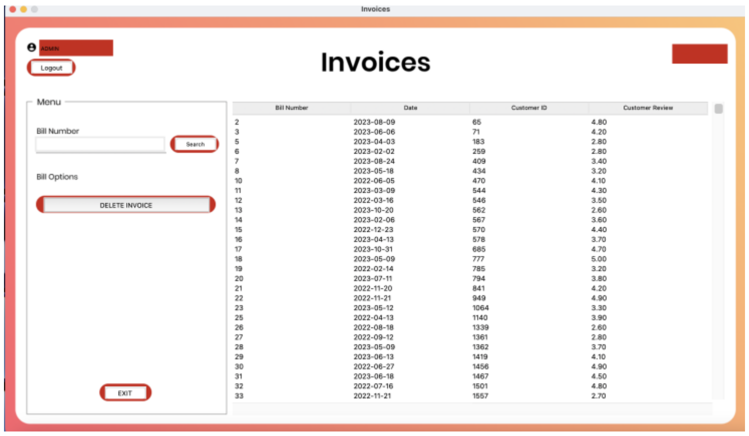


Once , add employee clicked,

This page opens, we can add new employee details.

Employee name, Designation, Contact number , and password.

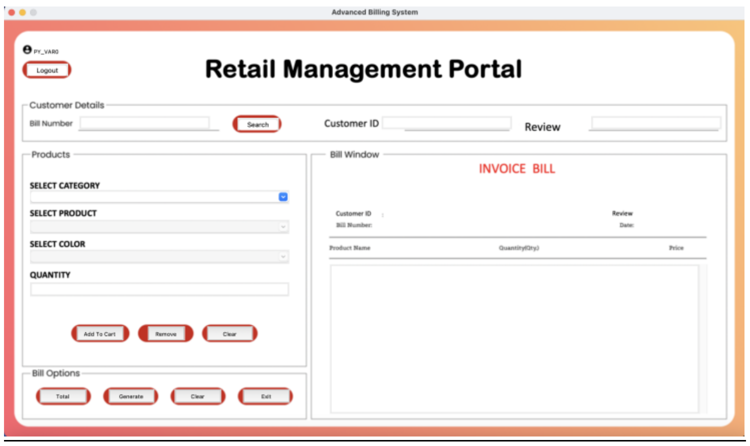
In backend: the insert query is triggred into employee table in database



Bill Number	Date	Customer ID	Customer Review
2	2023-08-09	65	4.80
3	2023-06-06	71	4.20
5	2023-04-03	183	2.80
6	2023-02-02	259	2.80
7	2023-08-24	409	3.40
8	2023-05-18	434	3.20
10	2022-06-05	470	4.10
11	2023-03-09	544	4.30
12	2022-03-16	546	3.50
13	2023-10-20	562	2.60
14	2023-02-06	567	3.60
15	2022-12-23	570	4.40
16	2023-04-13	578	3.70
17	2023-10-31	685	4.70
18	2023-05-09	777	5.00
19	2022-02-14	785	3.20
20	2023-07-11	794	3.80
21	2022-11-20	841	4.20
22	2022-11-21	949	4.90
23	2023-05-12	1064	3.30
25	2022-04-13	1140	3.90
26	2022-08-18	1339	2.60
27	2022-09-12	1361	2.80
28	2023-05-09	1362	3.70
29	2023-06-13	1419	4.10
30	2022-06-27	1456	4.90
31	2023-06-18	1467	4.50
32	2022-07-16	1501	4.80
33	2022-11-21	1557	2.70

Admin can also track the invoices generated daily by the store employees so that there won't be any discrepancies in money transactions and also they can delete the unwanted invoices

Employee Bill portal: on click of employee button in Home Page, the below page opens after login.



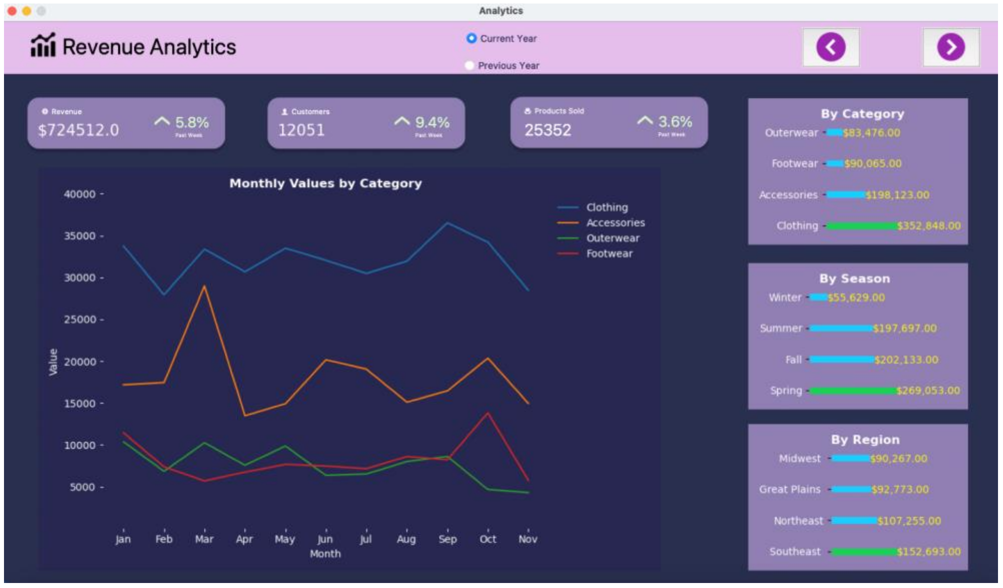
For Store Employees, a Billing System where they can generate bills for the products purchased by customers.

Sore Employees can generate bills by adding Customer ID and Customer Review and by adding Products by Category, Subcategory and Quantity. First, will add to cart and obtain total amount and then will generate the bill for customers.

Then the Bill will be generated to the customer with a unique Bill Number.

In the backend, select with where clause for filters is used and , once bill generated it will be inserted into Purchase table.

Revenue Analytics (OLAP):



Revenue $724512.0 ^ 5.8% Past Week

Customers 12051 ^ 9.4% Past Week

Products Sold 25352 ^ 3.6% Past Week

By Category

Outerwear	$83,476.00
Footwear	$90,065.00
Accessories	$198,123.00
Clothing	$352,848.00

By Season

Winter	$55,629.00
Summer	$197,697.00
Fall	$202,133.00
Spring	$269,053.00

By Region

Midwest	$90,267.00
Great Plains	$92,773.00
Northeast	$107,255.00
Southeast	$152,693.00

Revenue Analytics, designed to provide a comprehensive view of sales data. The dashboard uses advanced data processing techniques involving stored procedures, views, and Python matplotlib scripts to visualize and manipulate data dynamically.

The main feature of the dashboard is the "Monthly Values by Category" line chart, which plots the sales value across different product categories—Clothing, Accessories, Outerwear, and Footwear—over the months from January to November. Each category is represented by a different color line, with the value scale on the vertical axis and time (months) on the horizontal axis.

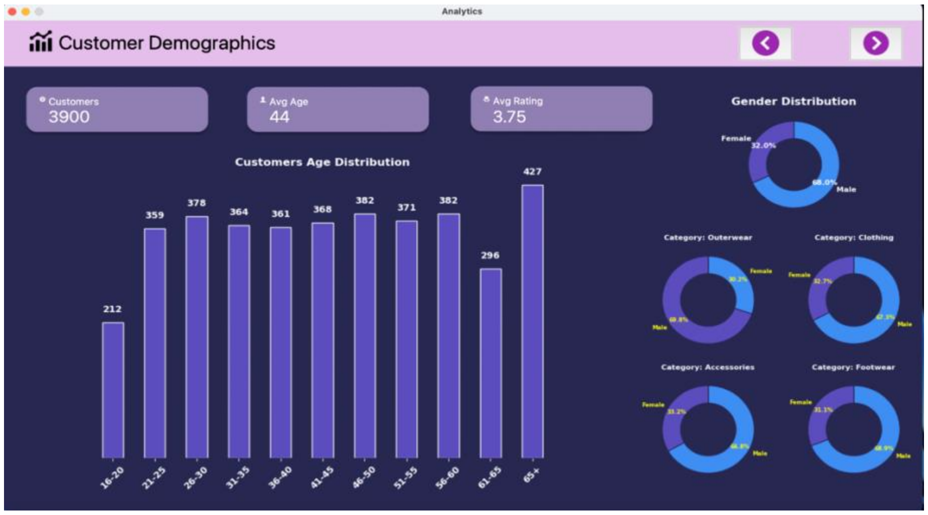
By Category: tab lists sales figures for Outerwear, Footwear, Accessories, and Clothing. Clothing has the highest sales at $352,848.00, indicated by a green bar.

By Season : tab shows sales figures for Winter, Summer, Fall, and Spring, with Spring having the highest sales at $269,053.00, also indicated by a green bar.

By Region : breakdown at the bottom right, detailing sales across different geographical areas—Midwest, Great Plains, Northeast, and Southeast

The use of OLAP techniques in this dashboard enables multidimensional analysis of large volumes of data, facilitating strategic business decisions based on current trends and historical comparisons. The dashboard's layout is crafted to offer clear insights with an emphasis on usability and data-driven decision-making.

Customer Demographics:



Customer Demographics :analytics dashboard that leverages Online Analytical Processing (OLAP) capabilities, crafted through Python's matplotlib library, stored procedures, and views. This dashboard dynamically updates, reflecting changes in customer demographics data when selections are altered.

Customer Overview:
The total customer count is shown as 3,900.
The average age of customers is listed as 44.
The average rating provided by customers is 3.75, on a scale that typically tops out at 5.

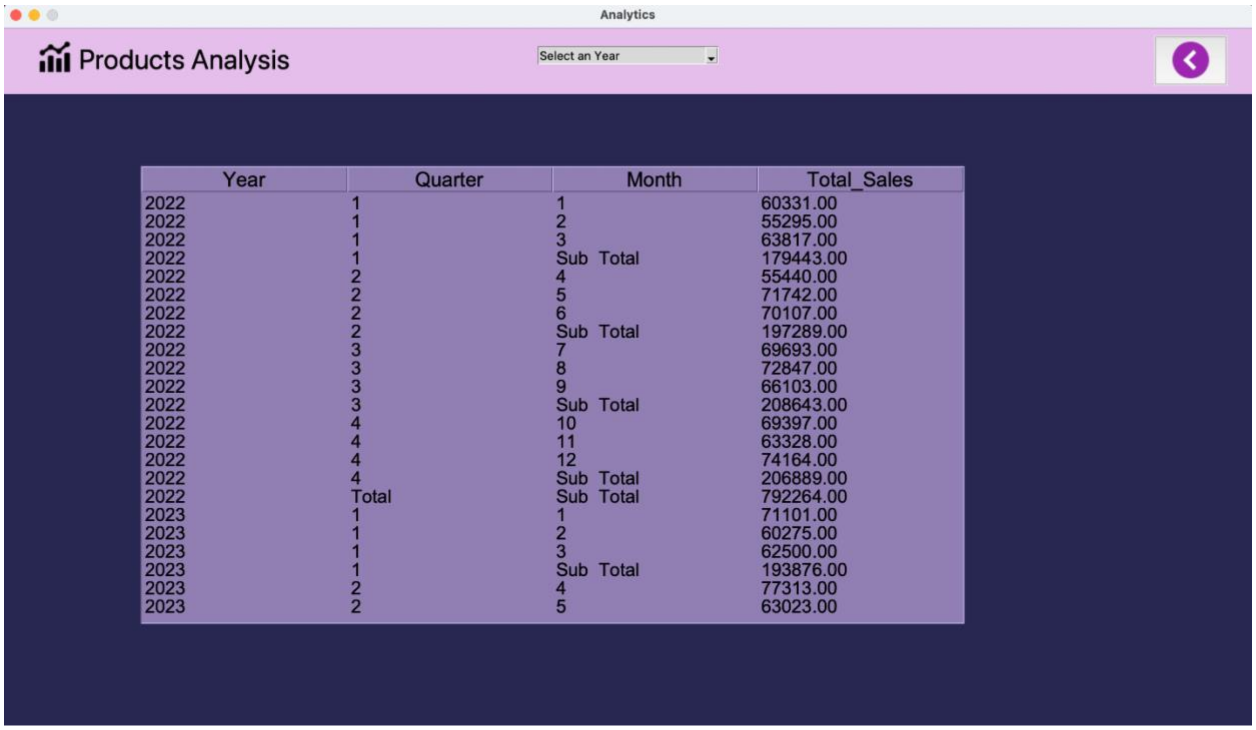
Age Distribution Chart:
A bar chart titled "Customers Age Distribution" shows the spread of customer ages across various age brackets, ranging from "16-20" to "65+".
Each age group is represented by a vertical bar, with its height corresponding to the number of customers in that segment.

Gender Distribution Charts:
A set of donut charts on the right display the "Gender Distribution" across different product categories: Outerwear, Clothing, Accessories, and Footwear.

These gender distribution charts are coded to update dynamically when different categories are selected, providing an interactive experience for the user.

The combination of these visual tools on the dashboard provides a multi-faceted view of the customer base, highlighting key demographic segments and preferences. The insights gained from this data can inform targeted marketing strategies, product development, and sales initiatives.

Product Analysis (Drill Up):



Year	Quarter	Month	Total_Sales
2022	1	1	60331.00
2022	1	2	55295.00
2022	1	3	63817.00
2022	1	Sub Total	179443.00
2022	2	4	55440.00
2022	2	5	71742.00
2022	2	6	70107.00
2022	2	Sub Total	197289.00
2022	3	7	69693.00
2022	3	8	72847.00
2022	3	9	66103.00
2022	3	Sub Total	208643.00
2022	4	10	69397.00
2022	4	11	63328.00
2022	4	12	74164.00
2022	4	Sub Total	206889.00
2022	Total	Sub Total	792264.00
2023	1	1	71101.00
2023	1	2	60275.00
2023	1	3	62500.00
2023	1	Sub Total	193876.00
2023	2	4	77313.00
2023	2	5	63023.00

Product sales analysis report, detailing sales figures over a series of months, quarters, and years.

The table header reads "Products Analysis" and has a dropdown menu labeled "Select an Year," suggesting that the user can choose different years to view corresponding data. The table is divided into columns labeled "Year," "Quarter," "Month," and "Total Sales."

For the year 2022, the table lists sales data for each month, grouped by quarter. For example, the first quarter (Q1) shows individual monthly sales for January, February, and March, followed by a subtotal for the quarter. This pattern is repeated for the second quarter (Q2), third quarter (Q3), and fourth quarter (Q4) of 2022. The "Sub Total" rows provide the total sales for each quarter, and there is a final row labeled "Total" which aggregates the sales for the entire year of 2022.

The data continues into the year 2023, showing sales figures for the first and second quarters available at the time of the report. Each month's sales are listed, with a subtotal for the first quarter presented.

Instructions to run the application

The launching of the application can be done from the Python terminal with selecting the folder of the project saved and executing the following commands: -

```
python main_final.py
```

The application will run and opens the home page of the customer shopping trends portal.

- From here you can navigate to different portals and pages.
- The credentials for logging into Admin Portal are
  - Username: **EMP0000**
  - Password: **ADMIN**
- The credentials for logging into Employee Portal are
  - Username: **EMP9862**
  - Password: **DADA**

STORED PROCEDURES AND VIEWS QUERIE

GetCustomerCountByYear :

```
DELIMITER $$
CREATE DEFINER=`datadynamos_user`@`%`
PROCEDURE `GetCustomerCountByYear`(IN _year INT)
BEGIN
SELECT
COUNT(*) AS CustomerCount
FROM
Customer_dm c
JOIN Sales_Fct sf ON c.Customer_ID = sf.Customer_ID
JOIN Calender_dm cd ON sf.date_id = cd.date_id
WHERE
cd.Year = _year;
END$$
DELIMITER ;
```

GetMonthlyRevenue

```
DELIMITER $$
CREATE DEFINER=`datadynamos_user`@`%`
 PROCEDURE `GetMonthlyRevenue`(IN _year INT)
BEGIN
SELECT
MONTHNAME(Calender_dm.Purchase_Date) AS Month,
COALESCE(SUM(CASE WHEN Calender_dm.Year = _year
THEN Purchase_Amount ELSE 0 END), 0) AS Revenue
FROM
Sales_Fct
JOIN Item_dm ON Sales_Fct.Item_ID = Item_dm.Item_ID
JOIN Calender_dm ON Sales_Fct.date_id = Calender_dm.date_id
WHERE
Calender_dm.Year = _year
GROUP BY
MONTHNAME(Calender_dm.Purchase_Date);
END$$
DELIMITER ;
```

GetMonthlyRevenueByCategory

GetRevenueByCategory

```
DELIMITER $$
CREATE DEFINER=`datadynamos_user`@`%`
PROCEDURE `GetRevenueByCategory`(IN _year INT)
BEGIN
SELECT
Category,
COALESCE(SUM(CASE WHEN Calender_dm.Year = _year
THEN Purchase_Amount ELSE 0 END), 0) AS Revenue
FROM
Sales_Fct
JOIN Item_dm ON Sales_Fct.Item_ID = Item_dm.Item_ID
JOIN Calender_dm ON Sales_Fct.date_id = Calender_dm.date_id
WHERE
Calender_dm.Year = _year
GROUP BY
Category;
END$$
DELIMITER ;
```

GetRevenueByRegion

```
DELIMITER $$
CREATE DEFINER=`datadynamos_user`@`%`
PROCEDURE `GetRevenueByRegion`(IN _year INT)
BEGIN
SELECT
Region,
COALESCE(SUM(CASE WHEN Calender_dm.Year = _year
THEN Purchase_Amount ELSE 0 END), 0) AS Revenue
FROM
Sales_Fct
JOIN Location_dm ON Sales_Fct.Location_ID = Location_dm.Location_ID
JOIN Calender_dm ON Sales_Fct.date_id = Calender_dm.date_id
WHERE
Calender_dm.Year = _year
GROUP BY
Region
order by COALESCE(SUM(CASE WHEN Calender_dm.Year = _year
THEN Purchase_Amount ELSE 0 END), 0) desc
limit 4;
END$$
DELIMITER ;
```

GetRevenueBySeason

```
DELIMITER $$
CREATE DEFINER=`datadynamos_user`@`%`
PROCEDURE `GetRevenueBySeason`(IN _year INT)
BEGIN
SELECT
Season,
COALESCE(SUM(CASE WHEN Calender_dm.Year = _year
THEN Purchase_Amount ELSE 0 END), 0) AS Revenue
FROM
Sales_Fct
JOIN Item_dm ON Sales_Fct.Item_ID = Item_dm.Item_ID
JOIN Calender_dm ON Sales_Fct.date_id = Calender_dm.date_id
WHERE
Calender_dm.Year = _year
GROUP BY
Season;
END$$
DELIMITER ;
DELIMITER $$
CREATE DEFINER=`datadynamos_user`@`%`
PROCEDURE `GetMonthlyRevenueByCategory`(IN _year INT)
BEGIN
SELECT
Category,
MONTHNAME(Calender_dm.Purchase_Date) AS Month,
COALESCE(SUM(CASE WHEN Calender_dm.Year = _year
THEN Purchase_Amount ELSE 0 END), 0) AS Revenue
FROM
Sales_Fct
JOIN Item_dm ON Sales_Fct.Item_ID = Item_dm.Item_ID
JOIN Calender_dm ON Sales_Fct.date_id = Calender_dm.date
WHERE
Calender_dm.Year = _year
GROUP BY
Category, MONTHNAME(Calender_dm.Purchase_Date);

END$$
DELIMITER ;
```

GetSalesRollup Stored Procedure:

```
DELIMITER $$
CREATE DEFINER=`datadynamos_user`@`%`
PROCEDURE `GetSalesRollup`()
BEGIN
-- Monthly Rollup
SELECT
'Monthly' AS RollupLevel,
i.Category,
i.ItemName,
MONTHNAME(cd.Purchase_Date) AS Period,
SUM(sf.Purchase_Amount) AS Revenue
FROM
Sales_Fct sf
JOIN Item_dm i ON sf.Item_ID = i.Item_ID
JOIN Calender_dm cd ON sf.date_id = cd.date_id
GROUP BY
i.Category, i.ItemName, RollupLevel, Period
WITH ROLLUP;

-- Quarterly Rollup
SELECT
'Quarterly' AS RollupLevel,
i.Category,
i.ItemName,
QUARTER(cd.Purchase_Date) AS Period,
SUM(sf.Purchase_Amount) AS Revenue
FROM
Sales_Fct sf
JOIN Item_dm i ON sf.Item_ID = i.Item_ID
JOIN Calender_dm cd ON sf.date_id = cd.date_id
GROUP BY
i.Category, i.ItemName, RollupLevel, Period
WITH ROLLUP;

-- Yearly Rollup
SELECT
'Yearly' AS RollupLevel,
i.Category,
i.ItemName,
```

VIEWS

TopSellingItemsView

```
CREATE ALGORITHM=UNDEFINED DEFINER=`datadynamos_user`@`%` SQL SECURITY
DEFINER VIEW `datadynamos_wh`.`TopSellingItemsView` AS with `RankedItems` as
(select `i`.`Category` AS `Category`,`i`.`ItemName` AS
`ItemName`,count(`sf`.`Customer_ID`) AS `PurchaseCount`,rank() OVER (PARTITION BY
`i`.`Category` ORDER BY count(`sf`.`Customer_ID`) desc )  AS `RankInCategory` from
(`datadynamos_wh`.`Sales_Fct` `sf` join `datadynamos_wh`.`Item_dm` `i`
on((`sf`.`Item_ID` = `i`.`Item_ID`))) group by `i`.`Category`,`i`.`ItemName`) select
`RankedItems`.`Category` AS `Category`,`RankedItems`.`ItemName` AS
`TopSellingItem`,sum(`RankedItems`.`PurchaseCount`) AS `TotalPurchaseCount` from
`RankedItems` where (`RankedItems`.`RankInCategory` = 1) group by
`RankedItems`.`Category`,`RankedItems`.`ItemName`;
```

AveragePurchaseByGenderView

```
CREATE ALGORITHM=UNDEFINED DEFINER=`datadynamos_user`@`%` SQL SECURITY
DEFINER VIEW `datadynamos_wh`.`AveragePurchaseByGenderView` AS select
`c`.`Gender` AS `Gender`,avg(`sf`.`Purchase_Amount`) AS `AvgPurchaseAmount` from
(`datadynamos_wh`.`Sales_Fct` `sf` join `datadynamos_wh`.`Customer_dm` `c`
on((`sf`.`Customer_ID` = `c`.`Customer_ID`))) group by `c`.`Gender`;
```

Total_Sales

```
CREATE ALGORITHM=UNDEFINED DEFINER=`datadynamos_user`@`%` SQL SECURITY
DEFINER VIEW `datadynamos_wh`.`Total_Sales` AS select count(0) AS `count(*)` from
`datadynamos_wh`.`Sales_Fct`;
```

PurchaseFrequencyPercentage

```
CREATE ALGORITHM=UNDEFINED DEFINER=`datadynamos_user`@`%` SQL SECURITY
DEFINER VIEW `datadynamos_wh`.`PurchaseFrequencyPercentage` AS select
`c`.`Purchase_frequency` AS `Purchase_frequency`,((count(0) / (select count(0) from
`datadynamos_wh`.`Sales_Fct`)) * 100) AS `PurchasePercentage` from
(`datadynamos_wh`.`Customer_dm` `c` join `datadynamos_wh`.`Sales_Fct` `sf`
on((`c`.`Customer_ID` = `sf`.`Customer_ID`))) group by `c`.`Purchase_frequency` order
by ((count(0) / (select count(0) from `datadynamos_wh`.`Sales_Fct`)) * 100) desc;
```

SalesRollupViews

```
CREATE ALGORITHM=UNDEFINED DEFINER=`datadynamos_user`@`%` SQL SECURITY
DEFINER VIEW `datadynamos_wh`.`SalesRollupViews` AS select (case when
(`datadynamos_wh`.`Calender_dm`.`Year` is not null) then
`datadynamos_wh`.`Calender_dm`.`Year` else 'Grand Total' end) AS `Year`,(case when
(`datadynamos_wh`.`Calender_dm`.`Quarter` is not null) then
`datadynamos_wh`.`Calender_dm`.`Quarter` else 'Total' end) AS `Quarter`,(case when
(`datadynamos_wh`.`Calender_dm`.`Month` is not null) then
`datadynamos_wh`.`Calender_dm`.`Month` else 'Sub_Total' end) AS
`Month`,sum(`datadynamos_wh`.`Sales_Fct`.`Purchase_Amount`) AS `Total_Sales` from
(`datadynamos_wh`.`Sales_Fct` join `datadynamos_wh`.`Calender_dm`
on((`datadynamos_wh`.`Sales_Fct`.`date_id` = `datadynamos_wh`.`Calender_dm`.`date_id`)))
group by
`datadynamos_wh`.`Calender_dm`.`Year`,`datadynamos_wh`.`Calender_dm`.`Quarter`,`data
dynamos_wh`.`Calender_dm`.`Month` with rollup;
```

```sql
YEAR(cd.Purchase_Date) AS Period,
SUM(sf.Purchase_Amount) AS Revenue
FROM
Sales_Fct sf
JOIN Item_dm i ON sf.Item_ID = i.Item_ID
JOIN Calender_dm cd ON sf.date_id = cd.date_id
GROUP BY
i.Category, i.ItemName, RollupLevel, Period
WITH ROLLUP;

-- Overall Rollup
SELECT
'Overall' AS RollupLevel,
i.Category,
i.ItemName,
NULL AS Period,
SUM(sf.Purchase_Amount) AS Revenue
FROM
Sales_Fct sf
JOIN Item_dm i ON sf.Item_ID = i.Item_ID
GROUP BY
i.Category, i.ItemName, RollupLevel
```