E-SHOP INC. IMPLEMENTATION REPORT

Business Intelligence and Business Analytics CA2

Isaac Umukoro – x23215640

Table of Contents

1. Development Process

The development process for E-Shop Inc. began with a thorough requirement analysis to understand the key business needs and data requirements. This phase involved meetings with stakeholders to gather insights into the operations, challenges, and goals of E-Shop Inc. The requirement analysis highlighted the need for efficient order processing, inventory management, customer insights, and sales analytics. The development process for E-Shop Inc.'s solution was done in a couple of steps that comprised data extraction, transformation, loading, data analysis, and data visualization. The following elaborates on each step taken:

1.1. Data Extraction and Preparation:
The Online Retail II dataset was used for this analysis. The data was downloaded from the UC Irvine Machine Learning Repository as CSV. The dataset contained 8 columns: InvoiceNo, StcokCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID and Country.

The data was then cleaned and transformed by handling missing values, correcting inconsistencies and standardized formats. Test data was generated using a Python script and was added to the customer in terms of their names, emails, addresses and phone numbers, which were added to the dataset.

1.2. Database Design:
An entity relationship diagram was designed in Lucidcharts showing how the entities relate to each other based on the system design. The ERD described the major entities like Customer, Invoice, Product and OrderItem and their association.

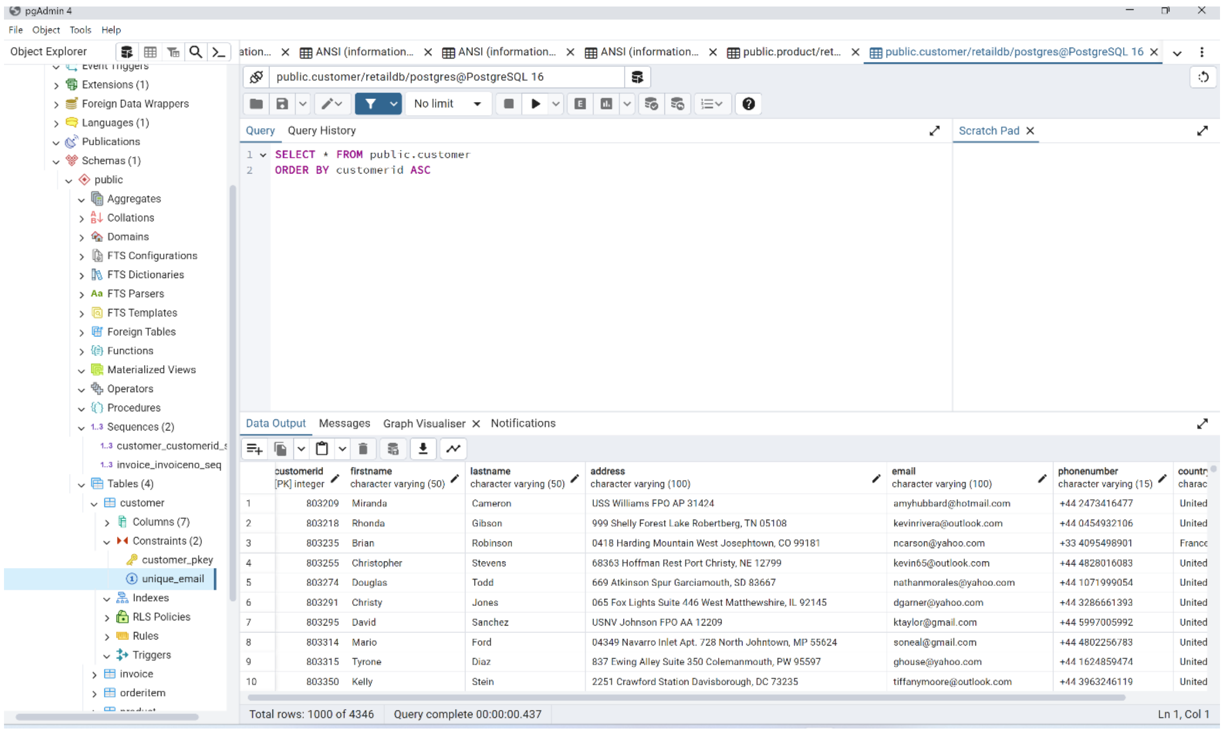
This design developed the database schema blueprint by considering all the data capture points and relationships.

1.3. Database Schema and Import:
PostgreSQL is used to implement the database. The database schema was created in PostgreSQL based on the ERD with four main tables: customer, product, invoice, and orderItem. Here are the actual implementations using SQL scripts in PostgreSQL:

- **Customer Table**
  CREATE TABLE public.customer (
  customerid SERIAL PRIMARY KEY,
  firstname VARCHAR(50),
  lastname VARCHAR(50),
  email VARCHAR(100) UNIQUE,
  phonenumber VARCHAR(15),
  address VARCHAR(255),
  country VARCHAR(50)
  );

  After the Customer table is created, an SQL script is used to alter the table to set email as a unique constraint for identifying customers as shown below.

  ALTER TABLE Customer
  ALTER COLUMN CustomerID SET DATA TYPE SERIAL,
  ADD CONSTRAINT unique_email UNIQUE (Email);



```
SELECT * FROM public.customer
ORDER BY customerid ASC
```

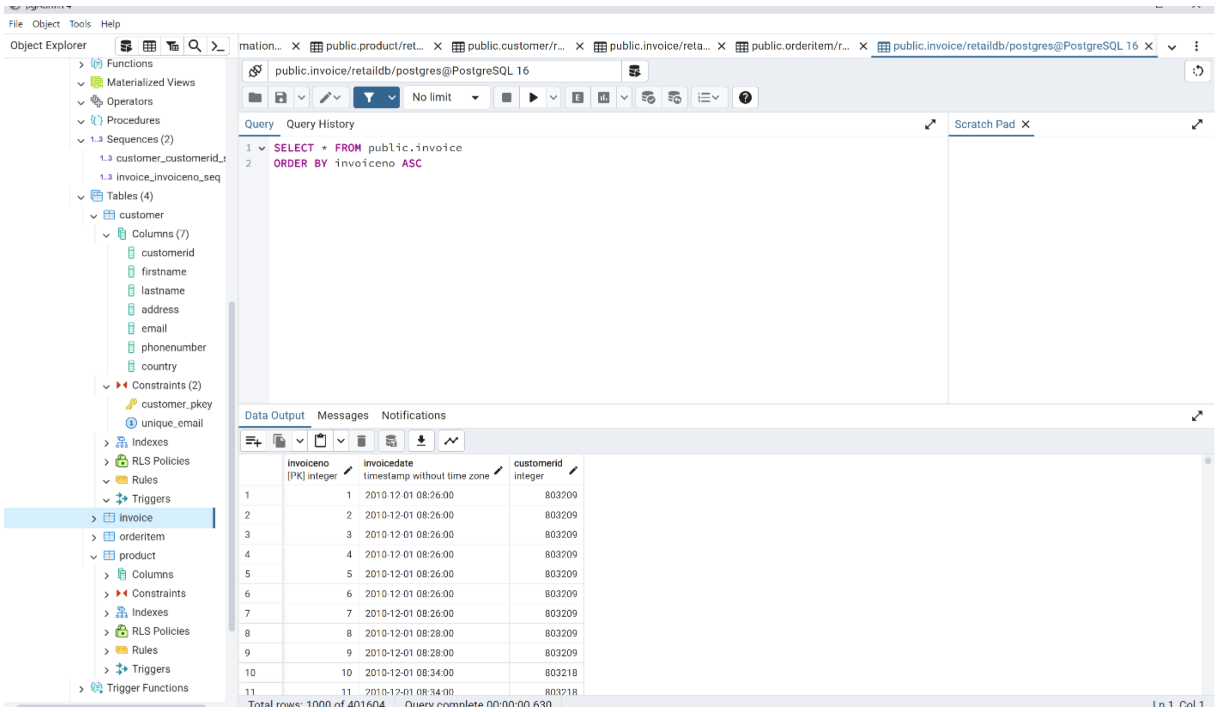
	customerid [PK] integer	firstname character varying (50)	lastname character varying (50)	address character varying (100)	email character varying (100)	phonenumber character varying (15)	countr charac
1	803209	Miranda	Cameron	USS Williams FPO AP 31424	amyhubbard@hotmail.com	+44 2473416477	United
2	803218	Rhonda	Gibson	999 Shelly Forest Lake Robertberg, TN 05108	kevinrivera@outlook.com	+44 0454932106	United
3	803235	Brian	Robinson	0418 Harding Mountain West Josephtown, CO 99181	ncarson@yahoo.com	+33 4095498901	France
4	803255	Christopher	Stevens	68363 Hoffman Rest Port Christy, NE 12799	kevin65@outlook.com	+44 4828016083	United
5	803274	Douglas	Todd	669 Atkinson Spur Garciamouth, SD 83667	nathanmorales@yahoo.com	+44 1071999054	United
6	803291	Christy	Jones	065 Fox Lights Suite 446 West Matthewshire, IL 92145	dgarner@yahoo.com	+44 3286661393	United
7	803295	David	Sanchez	USNV Johnson FPO AA 12209	ktaylor@gmail.com	+44 5997005992	United
8	803314	Mario	Ford	04349 Navarro Inlet Apt. 728 North Johntown, MP 55624	soneal@gmail.com	+44 4802256783	United
9	803315	Tyrone	Diaz	837 Ewing Alley Suite 350 Colemanmouth, PW 95597	ghouse@yahoo.com	+44 1624859474	United
10	803350	Kelly	Stein	2251 Crawford Station Davisborough, DC 73235	tiffanymoore@outlook.com	+44 3963246119	United

Total rows: 1000 of 4346 Query complete 00:00:00.437 Ln 1, Col 1

- **Product Table**
  CREATE TABLE public.product (
  stockcode VARCHAR(20) PRIMARY KEY,
  description VARCHAR(100),
  unitprice DECIMAL(10, 2)
  );

- **Invoice Table**
  CREATE TABLE public.invoice (
  invoiceno SERIAL PRIMARY KEY,
  invoicedate TIMESTAMP,
  customerid INT,
  FOREIGN KEY (customerid) REFERENCES public.customer(customerid)
  );

The Invoice table was altered to set the serial datatype for InvoiceNo for auto-incrementing a unique identifier.



```
SELECT * FROM public.invoice
ORDER BY invoiceno ASC
```

	invoiceno [PK] integer	invoicedate timestamp without time zone	customerid integer
1	1	2010-12-01 08:26:00	803209
2	2	2010-12-01 08:26:00	803209
3	3	2010-12-01 08:26:00	803209
4	4	2010-12-01 08:26:00	803209
5	5	2010-12-01 08:26:00	803209
6	6	2010-12-01 08:26:00	803209
7	7	2010-12-01 08:26:00	803209
8	8	2010-12-01 08:28:00	803209
9	9	2010-12-01 08:28:00	803209
10	10	2010-12-01 08:34:00	803218
11	11	2010-12-01 08:34:00	803218

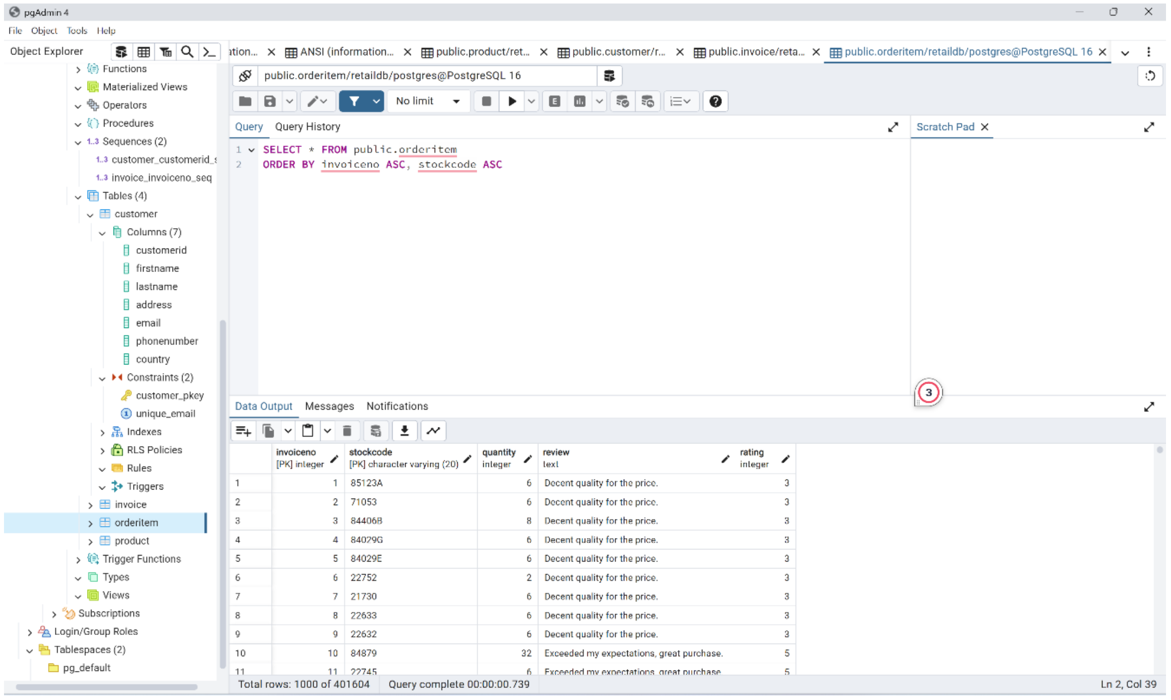
Total rows: 1000 of 401604 Query complete 00:00:00.630

ALTER TABLE Invoice

ALTER COLUMN InvoiceNo SET DATA TYPE SERIAL;

CREATE SEQUENCE IF NOT EXISTS invoice_invoiceno_seq;
ALTER TABLE Invoice ALTER COLUMN InvoiceNo SET DEFAULT nextval('invoice_invoiceno_seq');
ALTER SEQUENCE invoice_invoiceno_seq OWNED BY Invoice.InvoiceNo;
SELECT setval('invoice_invoiceno_seq', COALESCE((SELECT MAX(InvoiceNo) FROM Invoice), 1), false);

- **OrderItem Table**
  CREATE TABLE public.orderitem (
  invoiceno INT,
  stockcode VARCHAR(20),
  quantity INT,
  review TEXT,
  rating INT,
  FOREIGN KEY (invoiceno) REFERENCES public.invoice(invoiceno),
  FOREIGN KEY (stockcode) REFERENCES public.product(stockcode),
  PRIMARY KEY (invoiceno, stockcode)
  );,



```
SELECT * FROM public.orderitem
ORDER BY invoiceno ASC, stockcode ASC
```

	invoiceno [PK] integer	stockcode [PK] character varying (20)	quantity integer	review text	rating integer
1	1	85123A	6	Decent quality for the price.	3
2	2	71053	6	Decent quality for the price.	3
3	3	84406B	8	Decent quality for the price.	3
4	4	84029G	6	Decent quality for the price.	3
5	5	84029E	6	Decent quality for the price.	3
6	6	22752	2	Decent quality for the price.	3
7	7	21730	6	Decent quality for the price.	3
8	8	22633	6	Decent quality for the price.	3
9	9	22632	6	Decent quality for the price.	3
10	10	84879	32	Exceeded my expectations, great purchase.	5
11	11	22745	6	Exceeded my expectations, great purchase.	5

Total rows: 1000 of 401604   Query complete 00:00:00.739

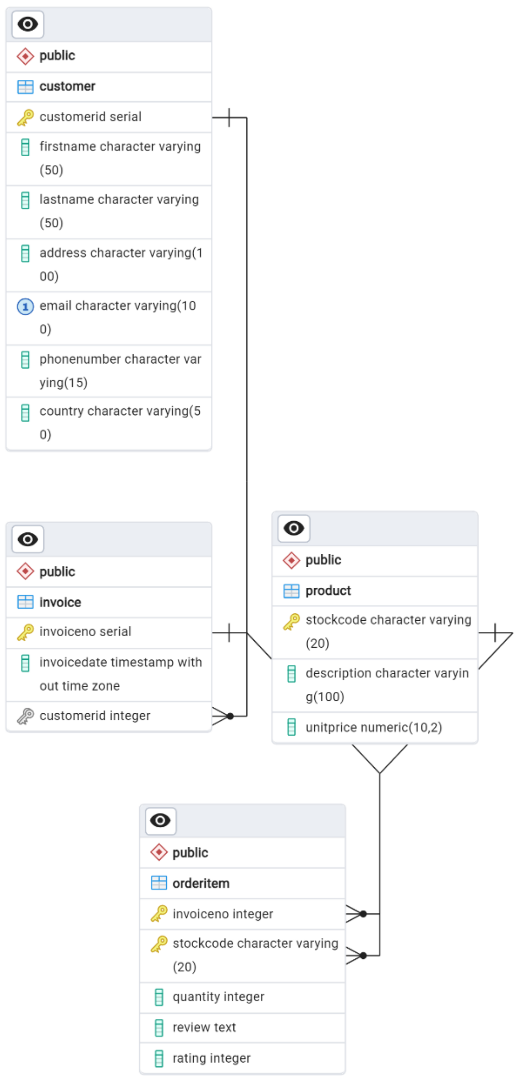
After creating the above tables, the cleaned and prepared datasets were imported into the various tables created in PostgreSQL using Python script. The relationship between each tables in the PostgreSQL database was derived from the ERD, and primary keys, foreign keys, and constraints were identified to enforce data integrity as shown below.

```
ALTER TABLE public.invoice
ADD CONSTRAINT fk_customer
FOREIGN KEY (customerid) REFERENCES public.customer (customerid);

ALTER TABLE public.orderitem
ADD CONSTRAINT fk_invoice
FOREIGN KEY (invoiceno) REFERENCES public.invoice (invoiceno);

ALTER TABLE public.orderitem
ADD CONSTRAINT fk_product
FOREIGN KEY (stockcode) REFERENCES public.product (stockcode);
```

Database Sche*ma*


public
customer
customerid serial
firstname character varying (50)
lastname character varying (50)
address character varying(100)
email character varying(100)
phonenumber character varying(15)
country character varying(50)
public
invoice
invoiceno serial
invoicedate timestamp without time zone
customerid integer
public
product
stockcode character varying (20)
description character varying(100)
unitprice numeric(10,2)
public
orderitem
invoiceno integer
stockcode character varying (20)
quantity integer
review text
rating integer

1.4. **Data Validation and Integrity:** The Python scripts used in loading the cleaned and prepared datasets into the PostgreSQL database play an important role in maintaining the integrity of the data as the transition from the CSV file into an organized database takes place. This script ensures that the final dataset is reliable and consistent. The following is how each element of the Python script enforces this:

A. Unique Constraints (On Conflicts)

- **Email for Customer:** According to a business rule, each customer can be uniquely identified by their email address. A clause was added ON CONFLICT(Email) DO NOTHING in the SQL INSERT statement. So, if the given file has duplicate records, it will not insert them for consumers based on their email addresses. If the email address already exists in the database, the script retrieves an existing CustomerID rather than creating a new entry. This will help to ensure that each customer is only entered into the database once, reducing redundancy and maintaining a clean set of information.

  Python Script:

  INSERT INTO Customer(FirstName, LastName, Address, Email, PhoneNumber, Country)
  VALUES (%s, %s, %s, %s, %s, %s)
  ON CONFLICT (Email) DO NOTHING
  RETURNING CustomerID

- **StockCode for Product:** Similarly, in the Product table, each product is individually recognized by its StockCode. The ON CONFLICT (StockCode) DO NOTHING clause in the Python script will check to see whether a product with the same 'StockCode' already exists and will not reinsert it in the table. This is because the database can only carry one entry for each product; thus, it is intended to ensure data consistency in the database and eliminate the occurrence of duplicate entries.

  Python Script

  INSERT INTO Product (StockCode, Description, UnitPrice)
  VALUES %s
  ON CONFLICT (StockCode) DO NOTHING

B. Ensuring Proper Format for InvoiceDate: The Python script below converts the InvoiceDate field from the CSV file into a proper datetime format using Pandas before inserting it into the database. This is important because it standardizes the date format on the Invoice table. This data consistency has to happen to do relevant and accurate data analysis and reporting. The date being converted into a datetime object It's essential to store the date in the correct format to make it uniform when doing any kind of time-based analysis or operations on the data.

Python Script:

```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], format='%Y-%m-%d
%H:%M:%S')
```

C. **Creating New Unique CustomerID and InvoiceNo for Relationships**

- **CustomerID:** When inserting a new customer into the Customer database, the Python script below attempts to insert the customer information. Hopefully, it will use the RETURNING clause to retrieve the new CustomerID. If the customer already exists due to the email's unique constraint, the existing CustomerID will be retrieved. In both cases, it ensures that all rows in the Invoice and OrderItem tables correctly refer to their respective customers, hence maintaining referential integrity.

  Python Script:

  ```
  INSERT INTO Customer (FirstName, LastName, Address, Email, PhoneNumber,
  Country)
  VALUES (%s, %s, %s, %s, %s, %s)
  ON CONFLICT (Email) DO NOTHING
  RETURNING CustomerID
  ```

- **InvoiceNo:** For each of the records in the dataset, an invoice for the customer is generated in the Invoice table; at the same time, a unique and unassigned invoice number is created and returned. This unique InvoiceNo is essential in associating all invoices to order items, which are placed into the OrderItem table later on. This Python script retrieves this InvoiceNo from the RETURNING clause and relates the same to the respective rows in the DataFrame. This helps to maintain the proper relationships between an invoice and its order items.

  Python Script:

  ```
  INSERT INTO Invoice (InvoiceDate, CustomerID)
  VALUES (%s, %s)
  RETURNING InvoiceNo
  ```

D. Bulk Operations with Execute_Values for PostgreSQL

The function execute_values from psycopg2.extras is used to insert numerous rows of data in a single operation. This is significantly more efficient than inserting one row at a time, particularly when the dataset is large. It reduces the number of transactions in the database, which improves its efficiency when loading data.

- **Insert Product:** This Python script loads unique products with the help of bulk insert in the Product table and inserts only new products, i.e., those whose StockCode is not already present.

  Python Script:
  ```
  products = df[['StockCode,' 'Description,'
  'UnitPrice']].drop_duplicates().values.tolist()
  execute_values(cursor, """
     INSERT INTO Product (StockCode, Description, UnitPrice)
     VALUES %s
     ON CONFLICT (StockCode) DO NOTHING
     """, products)
  ```

- **Inserting Order Items:** The order items are all added in bulk to the table OrderItem, which is pretty effective regarding speed and correctness.

  Python Script:
  ```
  order_items = df[['InvoiceNo,' 'StockCode,' 'Quantity,' 'Review,'
  'Rating']].values.tolist()
  execute_values(cursor, """
     INSERT INTO OrderItem (InvoiceNo, StockCode, Quantity, Review, Rating)
     VALUES %s
     """)
  ```

1.5. Data Analysis and Visualization

This process consists of numerous processes, including creating relationships between tables, DAX formula development for computed fields, and the design of various representations. The following steps are involved in this process;

A. Importing Data into PowerBI

The cleaned and formatted data in the PostgreSQL database tables arr imported into PowerBi as public customer, public invoice, public orderitem and public product as shown in Fig 1.

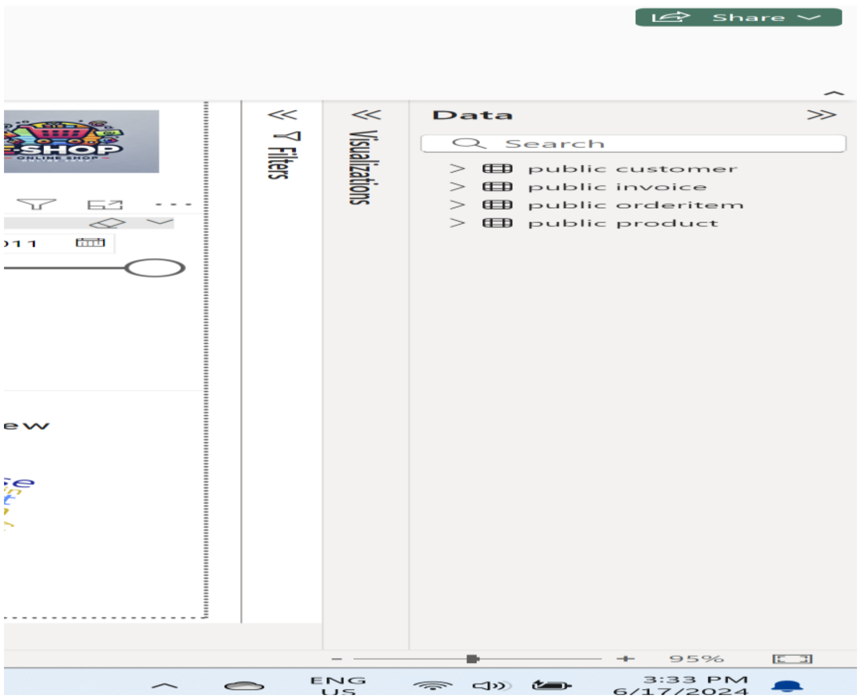


Fig. 1

B. Creating Relationships Between Tables

After importing data, the relationships between tables were automatically created in PowerBI as seen in Fig. 2 and 3. The following relationships were created:

i. **Customer Table:** CustomerID in public customer to CustomerID in public invoice
ii. **Invoice Table:** InvoiceNo in the public invoice to InvoiceNo in OrderItem
iii. **OrderItem Table:** StockeCode in public OrderItem to Stockcode in public product.

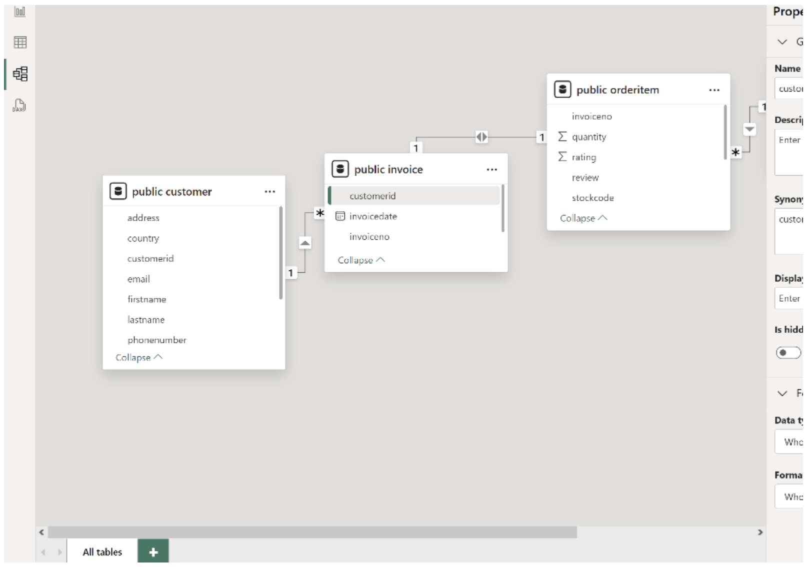
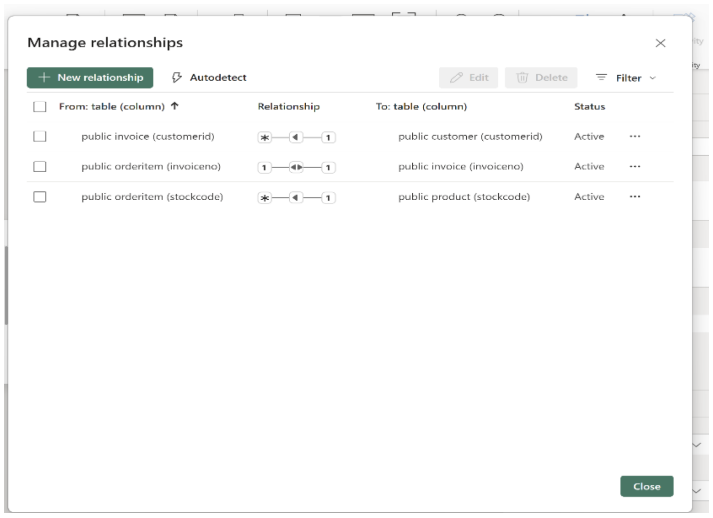


Fig. 2



From: table (column)	Relationship	To: table (column)	Status
public invoice (customerid)	* — 1	public customer (customerid)	Active
public orderitem (invoiceno)	1 — 1	public invoice (invoiceno)	Active
public orderitem (stockcode)	* — 1	public product (stockcode)	Active

Fig. 3

i. Developing Dax Formular

The following DAX (Data Analysis Expressions) formulas were used in Power BI to create new calculated fields that help in detailed analysis:

- **Total Sales:**

  *SUMX('public orderitem', 'public orderitem'[quantity] * RELATED('public product'[unitprice]))*

- **Total Sales By Customer:**

  ```
  SUMX(

     'public orderitem',

     'public orderitem'[quantity] * RELATED('public product'[unitprice])

  )
  ```

- **Total Sales by Month:**

  ```
  CALCULATE(
     SUMX(
         'public orderitem',
         'public orderitem'[quantity] * RELATED('public product'[unitprice])
     ),
     ALLEXCEPT('public invoice', 'public invoice'[invoicedate].[Year], 'public
  invoice'[invoicedate].[Month])
  ```

- **Total Quantity Sold:**

  SUM('public orderitem'[quantity])

- **Rating Count:**

  COUNT('public orderitem'[rating])

- **Purchase Frequency:**

  COUNT('public invoice'[InvoiceNo])

- **Top Selling Product:**

```
VAR TopProduct =
   TOPN(
      1,
      ALL('public product'),
      [Total Sales],
      DESC
   )
RETURN
   MAXX(TopProduct, 'public product'[description])
```

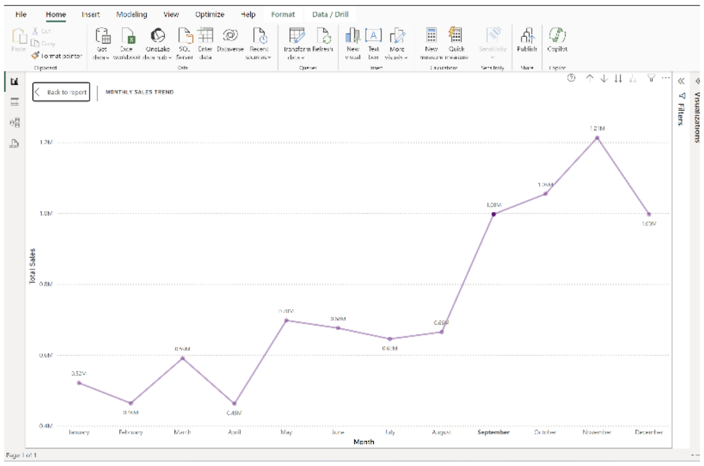
- **Top Customer:**

```
VAR TopCust =
   TOPN(
      1,
      SUMMARIZE('public invoice', 'public invoice'[customerid],
"TotalCustomerSales", [Total Sales]),
      [TotalCustomerSales],
      DESC
   )
RETURN
   MAXX(TopCust, 'public invoice'[customerid])
```

ii. Designing Visualization

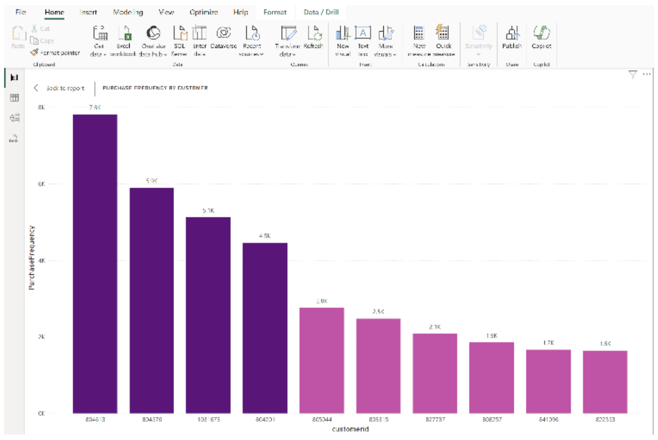
With the relationships and calculated fields set, several visualizations were created to show the following insights.

Monthly Sales Trend



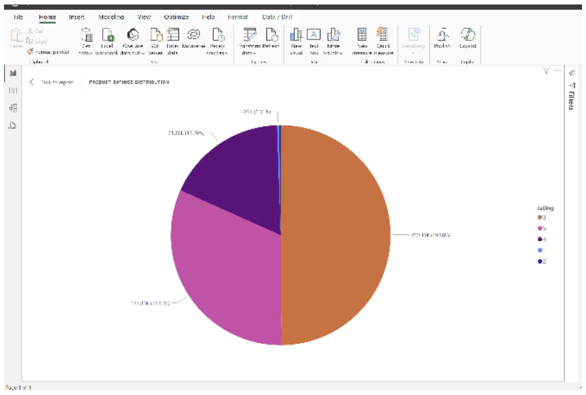
X-Axis: invoiceDate (Month), Y-Axis: TotalSalesByMonth,

Purchase Frequency by Customer



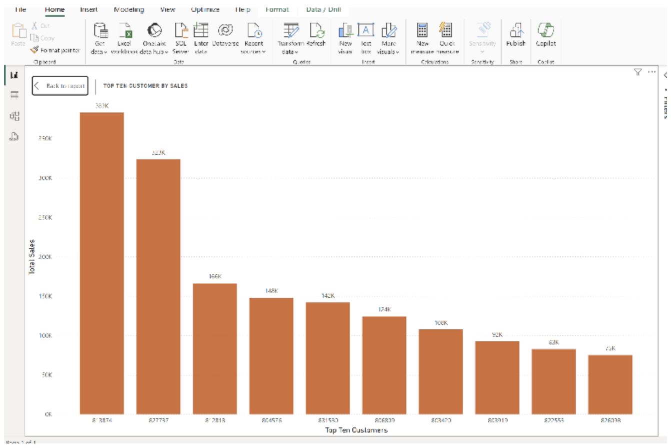
X-Axis: CustomerID, Y-Axis: PurchaseFrequency

Products Ratings Distribution



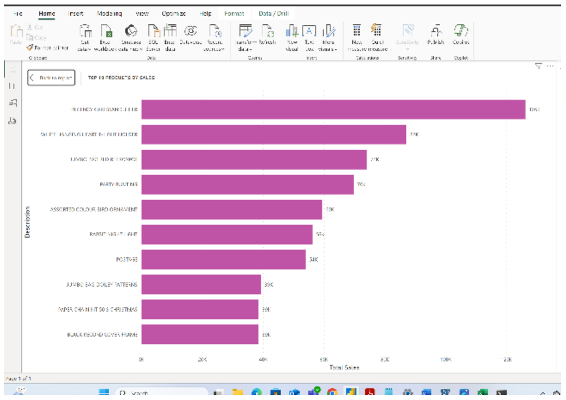
Legend: Rating, Values: Rating Count

Top Ten Customer by Sales



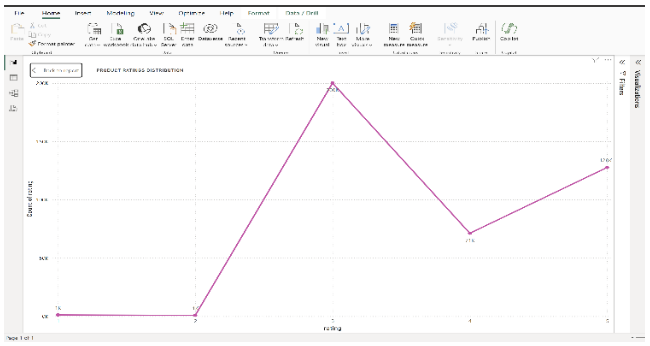
Y-Axis: Product Description, X-Axis: TotalSales, Filters: Top 10 products by TotalSales

Top Ten Products by Sales



Y-Axis: Product Description, X-Axis: TotalSales, Filter: Top 10 products by TotalSales

Product Ratings Distribution

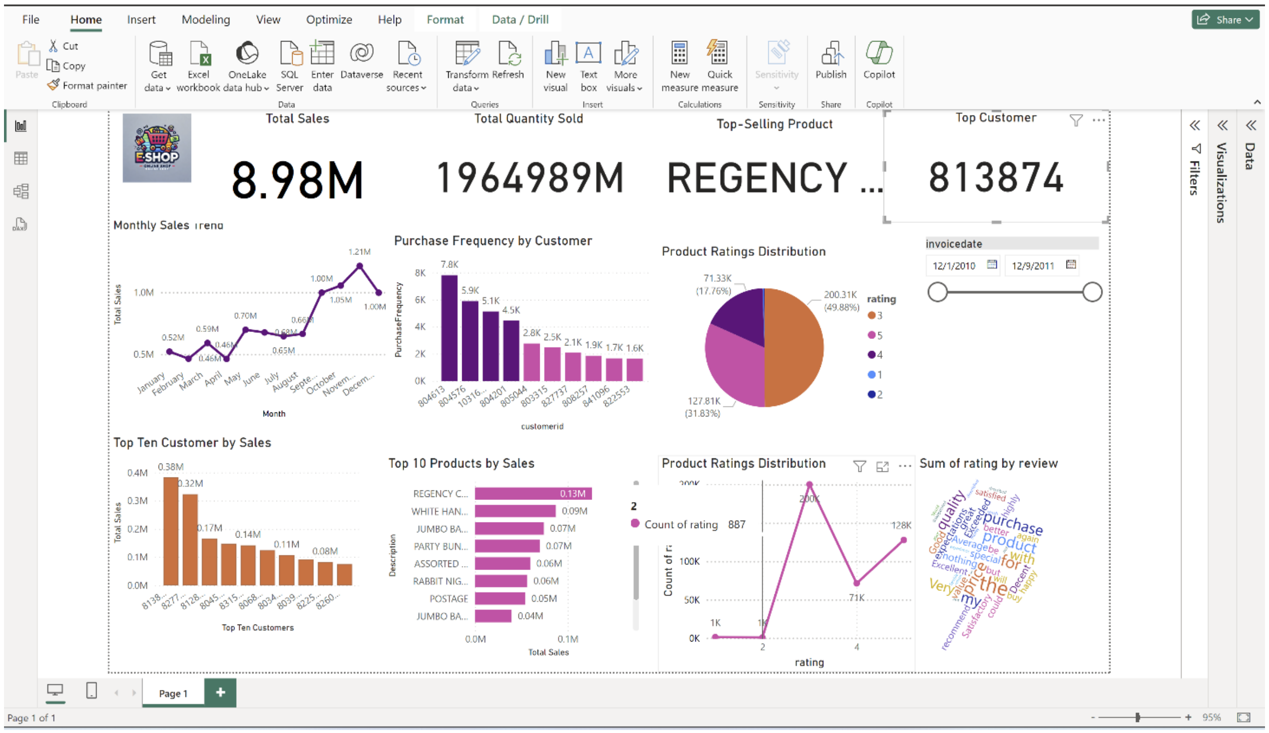


Y-Axis: Count of Rating, X-Axis: Rating,

iii. Creating and Formatting Reports

Each of the visualizations is formatted in detail for explicit readability. Titles, labels, and legends are added so that someone can understand the charts. Filters and slicers are added to give interactivity and dynamicity of the report. The data slicer was added to the report to filter out criteria based on the date range.

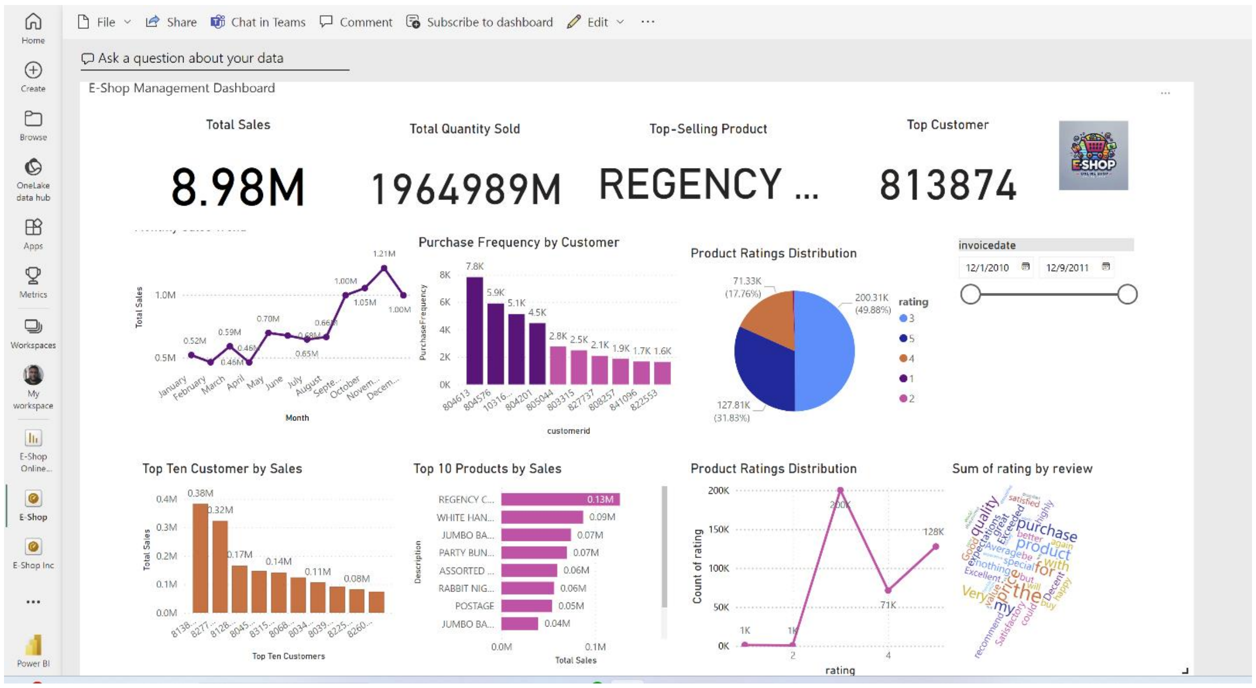
E-Shop Inc. Reports



iv. Building Dashboards

The reports created earlier on were then put into place to organize a full-fledged dashboard. The dashboard gave an outlook of the key metrics and insights derived from the analysis. Also, the dashboard is a repository for business stakeholders in E-Shop Inc. to track performance and support decision-making.

E-Shop Inc. Management Dashboard



2. Final Architect

The final architect for E-Shop Inc. consists of the following components already explained above:

A. **Data Capture Points:** A CSV file containing data on customer interactions and transactions was acquired from the UC Irvine Machine Learning Repository. Python scripts were used to produce synthetic data for customer names, email addresses, addresses, phone numbers, reviews, and ratings, resulting in a more complete dataset.
In a real-world scenario, this information is often collected via the E-Shop Inc. website and other digital platforms.

B. **Data Processing (Python Scripts):** The downloaded dataset that was captured is then processed to ensure data quality. Synthetic data was generated using Python script to add additional fields like Customer First and last name, address, email, phone number, review and rating to enable us to perform a complete analysis of the E-Shop Inc. The synthetic data generated is then merged with the main dataset using Python scripts. The complete dataset is then prepared for loading into the PostgreSQL database by checking if it is in the right format.

C. **Database (PostgreSQL):** PostgreSQL serves as the main repository for data used in the E-Shop Inc. analysis. Four tables were created in the PostgreSQL database: Customer, Invoice, Product and Order Item. The processed dataset is then loaded into the various tables in the PostgreSQL database using Python scripts ensuring information are stored correctly.

D. **Analytics and Visualization (PowerBI)**

The data in the 4 tables in the PostgreSQL are then imported into Power BI, The relationships between the different tables are established, and DAX formulas are used in creating calculated columns to enhance data analysis. Furthermore, interactive reports and dashboards were created to visualized key metrics and insights.

3. Operations

This involves the following:

- **Data Capture and Storage**: The initial dataset from the UC Irvine Machine Learning Repository was used to extract customer data, product details, and transaction records, which were supplemented with synthetic data. This data was saved in a PostgreSQL database.
- **Data Processing**: The raw data were cleaned, supplemented, and processed with Python scripts. The processed dataset was then uploaded to the PostgreSQL database. Regular data processing guarantees that the database is always populated with accurate and current information.
- **Data Analysis and Visualization:** PostgreSQL data is imported into Power BI for modelling and analysis. Different visualizations are created to give insights into sales trends, customer behaviour, product performance, and many more.

4. Benefits of the Solution

- **Improved Data Quality and Integrity**: It enhances the quality and integrity of data, assuring that the data used is correct, consistent, and up-to-date since it has gone through a vigorous process of validating and cleaning the data.
- **Enhanced Decision Making**: Detailed and interactive visualizations support the understanding of business metrics, leading to decision-making based on data.

- **Operational Efficiency:** This reduces manual effort and increases operational efficiencies. With scheduled updates of the data and its automated reporting, users have real-time access to information, saving time for the organization.
- **Operational Efficiency:** This reduces manual effort and increases operational efficiencies. With scheduled updates of the data and its automated reporting, users have real-time access to information, saving time for the organization.
- **Scalability:** E-Shop Inc's architecture is scalable to handle massive volumes of data and support future expansion
- **Flexibility:** It is flexible in incorporating other data sources and analytical tools if needed.
- **Customer Insights:** it provides valuable insights into customer behaviour and preferences, allowing E-Shop Inc. to customize its services and marketing initiatives.
- **Sales and Inventory Management:** it provides detailed reports on sales performance and inventory levels, facilitating effective management and sales forecasting.