Trendify Fashion Inc. - Solution

Designing a Relational Database.

After looking at each of the sections of information needed on the database, we will look at what needs to be included from the provided information, to make it applicable to a database. With this we will then use the site/application dbdiagram.io to design the relational database as we move across each section of information needed.

Provided Information:

1. Product Information

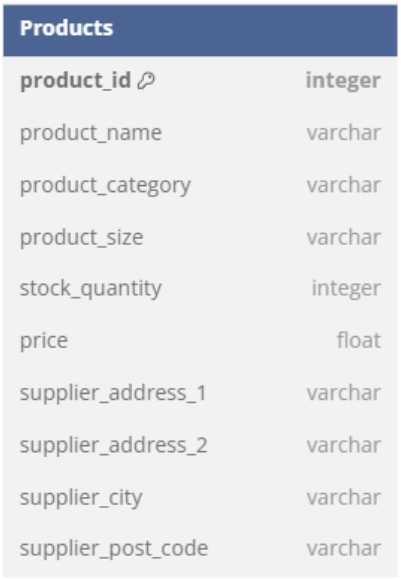
The company needs to track key details about each product in its inventory, including:

- Product Name
- Product Category
- Price
- Stock Quantity
- Product Description
- Supplier Information

Required Information:

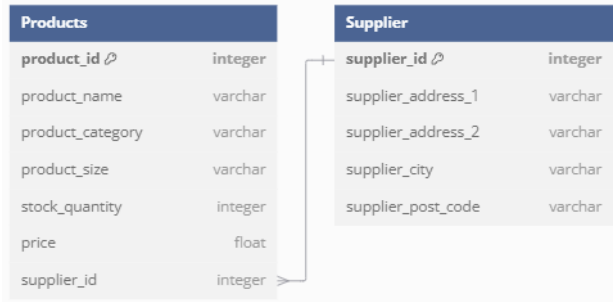
- Product ID
- Product Name
- Product Category
- Product Size
- Product Colour
- Stock Quantity
- Price
- Supplier Address 1
- Supplier Address 2
- Supplier City
- Supplier Post Code

We can then generate a table on the designing tool with the accumulated information above, and it would look like this:



From this, we can see that redundancy can occur with the supplier information that exists on this table. As a result, we must carry out normalisation, which will organise the data to improve efficiency and reduce redundancy. To do this, we can create a new table for the supplier for the supplier information and use references in the product table, through a unique identifier.

We will now remove the supplier attributes from the product table and replace it with the Supplier Id field that acts as a primary key for the Supplier table. After this, we establish a 1-to-Many relationship between the Supplier table and the Product table, respectively.



Provided Information :

2. Customer Information

The company needs to capture basic customer details for personalised marketing and order management:

- Customer Name
- Contact Information
- Shipping Address

3. Order Information

Each order placed by a customer should be tracked, including:

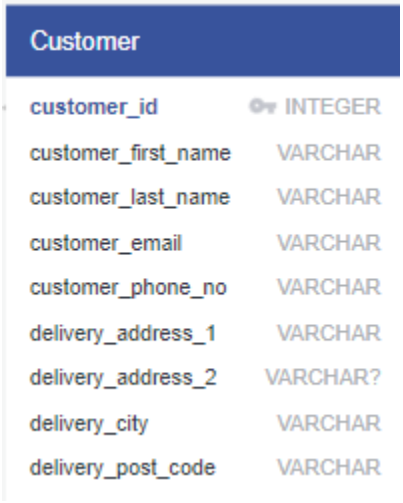
- Order Date
- Order Status
- Order Total
- Payment Method
- Shipping Method

Required Information :

- Order Id
- Order Date
- Order Status
- Product Name
- Product Size
- Product Category
- Price
- Customer Email
- Customer Phone Number
- Customer First Name
- Customer Last Name
- Customer Payment Method
- Shipping Method
- Delivery Address
- Delivery Address
- Delivery City
- Delivery Post Code

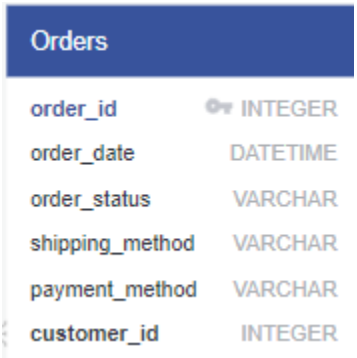
We should normalise this as redundancy can occur. To do this, we can already normalise the Product information with our Product Table. We should normalise the Customer information in a customer table.

It would produce the following table below:



The customer Id uniquely recognises each customer and the fields of the table taken from the information we identified that is associated with the customer. These include: first name, last name, email, and others, which are placed into this table.

The next set of information we should normalise would be the order information. This would be the order id, the relevant customer, the order items, the shipping and payment method desired for that specific order. This information is requested by the client.



Provided Information :

4. Order Item Information

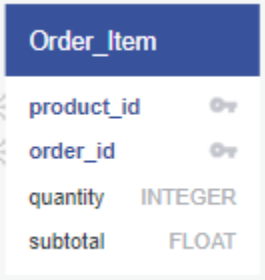
For each product purchased in an order, the following data should be tracked:

- Product Name
- Quantity Ordered
- Price
- Subtotal

Required Information:

- Product ID
- Order ID
- Quantity
- Subtotal

In line with the Product and Order tables, we should create another table that will hold the references to products that are selected for orders called 'Order Item'. As multiple products can be used for multiple orders, we should have a composite key consisting of both product id and order id. It will also store the quantity required for the order, the price, and the consequent subtotal.



Provided Information:

5. Inventory Information

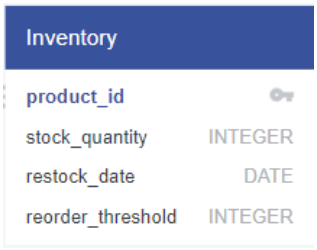
The company needs to manage and track product stock levels:

- Stock Quantity
- Restock Date
- Reorder Threshold

Required Information :

- Product ID
- Stock Quantity
- Restock Date
- Reorder Threshold

As a result, I designed a table where the product_id is used as the primary key with the stock quantity, restock date and restock threshold as fields.



Provided Information

6. Sales Data

To analyse sales performance, the company needs to track the following:

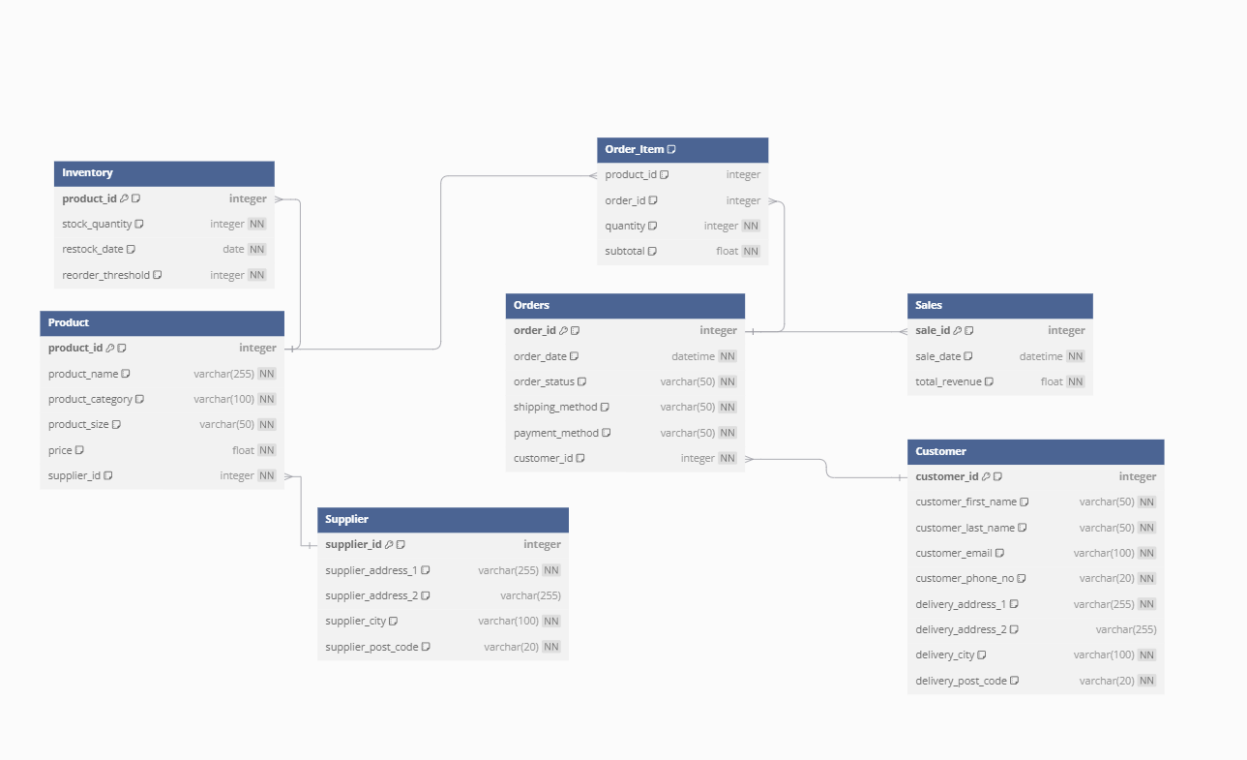
- Product Name
- Sale Date
- Quantity Sold
- Sales Revenue

Required Information :

- Sale ID
- Order ID
- Sale Date
- Total Revenue

A sales table is used to track the sales of the company, so transactions should be identified uniquely with a sale id. An order id should be used to link orders to the sale, the sale date, and the revenue accumulated should be held.

Relationships



Above shows the entire database diagram with each table and their relationship with each other. This section of the document will describe each relationship that has been established for the database.

Customer Table Relations

The customer table will maintain a one-to-many relationship with the Orders table. This is because customers can have multiple orders.

Order Table Relations

As previously stated, the order table maintains a many-to-one relationship with the Customer table. It also maintains one-to-many relationships with the Order Item table and a one-to-one relationship with the Sales table. The relationship established with the order item table is used to track what items are selected for each order and the sales table relationship is used to determine whether the order has been carried out, indicating revenue generated and when the sale occurred. It is a one-to-one because we should account one order being fulfilled to be one sale, so that financial analysis and tracking can be carried out accurately.

Sales Table Relations

The Sales Table only maintains a relationship with the Order table as a one-to-one relationship.

Order Item Table Relations

The order item table uses a composite key of both the product id and the order id to identify each item of an order. It has a many-to-one relationship with the orders table, as an order can have multiple items.

The Order Item Table has a many-to-one relationship with the Inventory table using the product Id. This makes it possible to track stock quantities when items have been selected for orders.

Inventory Table

The inventory table has a one-to-many relationship with the Order Item Table. It is used to track changes in stock for each product that has been placed for an order.

It also maintains a one-to-one relationship with the Product Table on the Product Id. This is to maintain stock tracking on each product available.

Product Table

The product table maintains a one-to-one relationship with the Inventory Table, stated previously.

It also maintains a many-to-one relationship with the Suppliers Table. This is because a supplier can supply several products to the company.

Creating Queries and Dashboards

In this section we will be writing queries that will discover insights for the user. Using the queries, we will develop dashboards so that the client can visualise the data. We will be joining data from different tables to simplify querying.

Section 1 – Order Activity

The first dashboard should display the following data:

- Total Orders
- Total Sales
- Total Items
- Average Order Value
- Sales by Category
- Top Selling Items
- Orders/Sales by hour
- Orders by Address
- Orders by delivery choice

Section 2 – Inventory Activity

The second dashboard should display the following data:

- Total Quantity by products/category
- Total cost of Stock
- Percentage Stock remaining by products

Order Dashboard

For this, we load the data into the PowerBi by connecting to the MySQL database; this is done using our credentials. Before getting the data, I will copy my SQL script for the order dashboard into the customer query tab, so that it can be run to fetch the required data and columns needed for the order dashboard. With this, we can go step-by-step down each of the requirements that the client required to be on the dashboard.

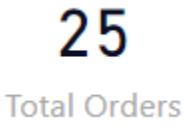
```
-- Order Dashboard Query
WITH CTE AS (
SELECT product.product_name,product.product_category,product.price, order_item.quantity,order_item.order_id
FROM product
LEFT JOIN order_item
ON product.product_id = order_item.product_id
)

SELECT
o.order_id,
s.total_revenue,
CTE.quantity,
CTE.product_category,
CTE.product_name,
CTE.price,
o.order_date,
c.delivery_address_1,
c.delivery_address_2,
c.delivery_city,
c.delivery_post_code,
o.shipping_method
FROM orders o
LEFT JOIN customer c on o.customer_id = c.customer_id
LEFT JOIN sales s on o.order_id = s.sale_id
LEFT JOIN CTE on o.order_id = CTE.order_id
;
```

Total Orders

To display total orders, we will do a Distinct Count of the Order IDs from this table. AS this table displays each item of each order naturally, we require a distinct count of unique order IDs to properly show how many orders have been placed.

For the dashboard, we will be using a Card to visualise this data, as it is a single number.



Total Sales

To display total sales, we can create a new measure that sums all rows of the order_total_revenue field in PowerBI.

```
Total Sales = SUM(Query1[order_total_revenue])
```

The 'order_total_revenue' field was created from our query, which retrieves the total price of the order from the sales table. To calculate the total revenue/sales, I created a view of the Order Items and Product table called 'Product_Cost_Information' and used aggregate functions to sum up the price of all items in each order.

```
-- Order Total
SELECT
product_cost_info.order_id,
ROUND(SUM(product_cost_info.subtotal),2) AS order_total
FROM product_cost_info
GROUP BY 1
ORDER BY 1
;
```

Which then was set to the relevant field and row in the sales table using the following query:

```
UPDATE sales s
JOIN (
    SELECT
        order_id,
        ROUND(SUM(subtotal), 2) AS order_total
    FROM product_cost_info
    GROUP BY order_id
) pci ON s.sale_id = pci.order_id
SET s.total_revenue = pci.order_total;
```

We then apply the measure created in PowerBi to a card and format it to display as currency.

Total Quantity

To display total Items sold, we can create a new measure which sums each row of the 'quantity' field and display it on a card.

```
Total Quantity = SUM(Query1[quantity])
```

Using Common Table Expressions, we retrieved the quantity of each item for each order and appended the column onto the custom table used to generate the data for this dashboard.

```
-- Order Dashboard Query
WITH CTE AS (
SELECT product.product_name,product.product_category, order_item.quantity,order_item.order_id
FROM product
LEFT JOIN order_item
ON product.product_id = order_item.product_id
)
```

```
SELECT
o.order_id,
s.total_revenue,
CTE.quantity,
CTE.product_category,
CTE.product_name,
o.order_date,
c.delivery_address_1,
c.delivery_address_2,
c.delivery_city,
c.delivery_post_code,
o.shipping_method
FROM orders o
LEFT JOIN customer c on o.customer_id = c.customer_id
LEFT JOIN sales s on o.order_id = s.sale_id
LEFT JOIN CTE on o.order_id = CTE.order_id
;
```

This measure was also displayed on a card.

Average Order Value

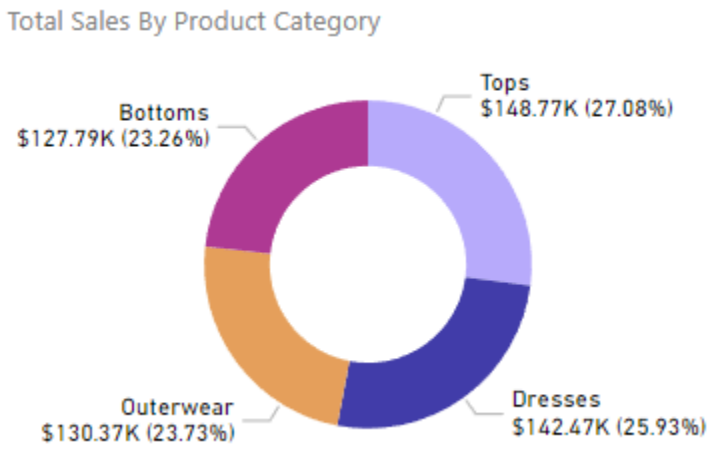
To find the Average Order Value, I created a measure on PowerBI that divides the Total Sales measure by the Total Quantity measure.

```
Average Order Value = Query1[Total Sales]/Query1[Total Quantity]
```

The measure was displayed on a card.

Total Sales By Product Category

To display the Total Sales by Product Category, I decided that using a donut graph would be acceptable, as there are only 4 categories of products available for sale. As a result, it would be a simple, yet insightful, visual that would be beneficial for the client. To make the visual, I used the measure that was created previously, 'Total Sales', as the values and used the product_category field to split the total sales, accordingly.

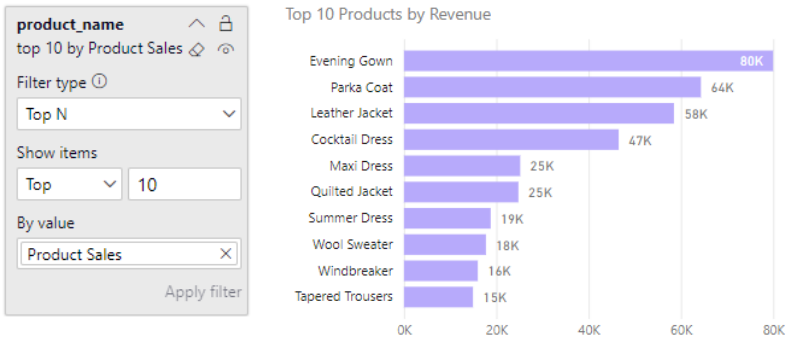


Top Products By Revenue

To display the top selling items, I decided to use a horizontal bar chart that displays the total sales for each product. With that, I would select the top ten of the products with the highest total sales. To do this, I created a measure called 'Product Sales' with the following query, where I multiply the sum of quantity with the sum aggregate of the price of a product.

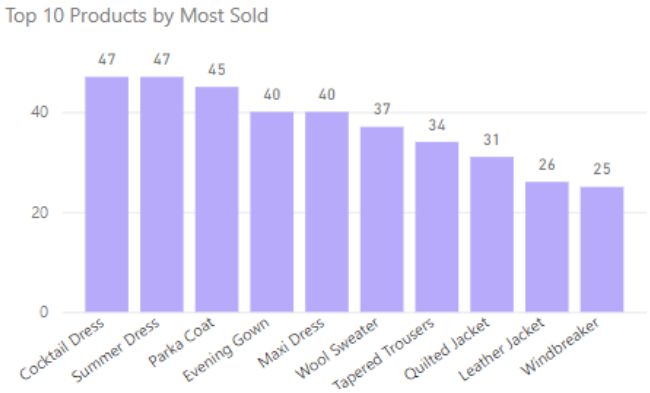
```
Product Sales = SUM(Query1[price]) * SUM(Query1[quantity])
```

With this, we select the product name to be the Y-axis of the horizontal bar chart, and the Product Sales will be the X-axis. The setting of the product name to be the Y axis of the chart will filter the Product Sales to individual products for each bar. We then set a filter on the chart, where we select the Top 10 Product names, based on the Product sales measure.



Top Products by Most Sold

This KPI is used to determine which products have been sold the most. To display this, I will be using a vertical bar chart of the Top 10 products by setting the X-axis to be the products, and the Y-axis being the Total Quantity. The Total Quantity measure that is being used on this visual will be split based on the product name, providing insight into how many of each product has been sold.

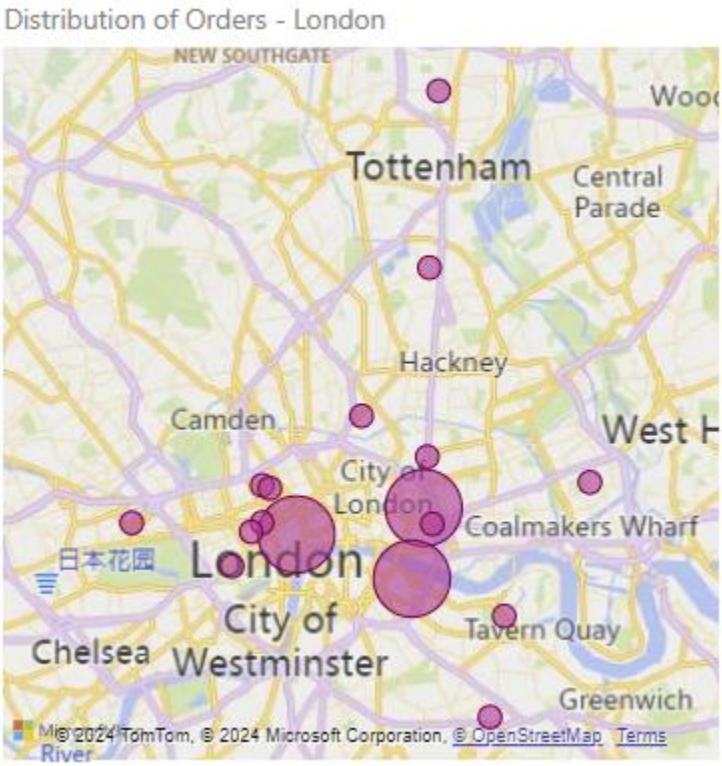


Order/Sales by Hour

To figure out sales/order by hour, we must transform our table extracting the hour of the 'order_date' field and placing it on a new column. After adding this column, it will update the data that is available to build visuals. With this, we can create a line graph indicating how many orders were placed during the hours of operation of the online store.

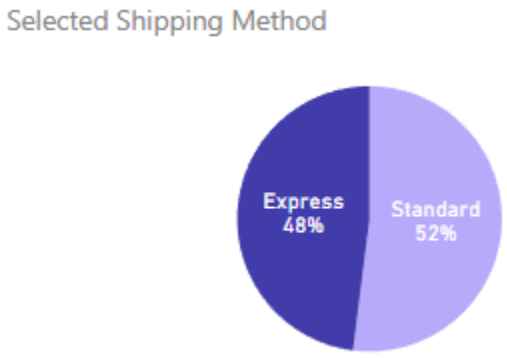
Distribution of Orders - London

As the data that we are using are based around London, we can use the map visual to see the distribution of customers that have placed orders. To do this, we simply set the location to be the city and the post code and adjust the bubble size based on the distinct count of orders.



Selected Shipping Method

This will be displayed through a pie chart, as there are only 2 shipping methods available. We easily perform a simple count of the shipping method and display it as a percentage.



Inventory Activity Dashboard

For this dashboard, we will use another query when we connect to our database to get the required fields to produce the required visualisations. In this query, we perform calculations and also create an additional column, so that when the stock of a product reaches below 60%, it will set the column to have the status 'Restock Req'. This is a good query to implement, as it allows the query to be re-run from the database when new data gets uploaded. With this, any dashboard that is connected to the database, will have accurate inventory information.

```
SELECT
i.product_id,
p.product_name,
p.product_category,
p.unit_cost,
SUM(oi.quantity) AS outgoing_units,
i.stock_quantity,
i.stock_quantity - SUM(oi.quantity) as units_remaining,
ROUND(1-SUM(oi.quantity)/i.stock_quantity,2) as percentage_remaining,
CASE WHEN ROUND(1-SUM(oi.quantity)/i.stock_quantity,2) <0.6 THEN 'Restock Req' ELSE 'Enough Stock' END Stock_Status
FROM order_item oi
LEFT JOIN inventory i
ON oi.product_id = i.product_id
LEFT JOIN product p
on oi.product_id = p.product_id
group by 1,2,3,4;
```

This dashboard will be relatively simple, as it will consist of major cards and tables.

Total Quantity By Product

To display the total quantity of stock that was purchased from suppliers for each product, we simply must create a table visualisation and select the stock quantity field. To separate the stock quantity from being a sum, we select the product name to essentially filter the stock quantity field by products. As a result, we can visualise the total quantity of each product.

Total Cost of Product

To display the total cost of each product in stock, we can use the table created prior and create a measure called Total Cost. The total cost is when you multiply the cost of a unit and the quantity of stock (this field is the original value before orders have been placed).

```
1  Total Cost = SUM('Inventory Data'[unit_cost]) * SUM('Inventory Data'[stock_quantity])
```

By selecting the product name to be a column, it will filter the costs by the product name.

Percentage Stock Remaining by Products

In the query that is being used for the data, a calculated column is already implemented which details the percentage of stock remaining. As a result, to display this visual we will just add it to the table above, which will filter the percentage of stock remaining by products.

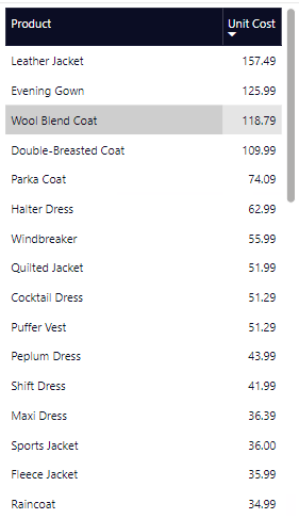
Stock Status

The client will also need to clearly be able to see whether certain products need a restock, so to indicate this a calculated field using CASES was carried out on the query. This is used so that if the stock of a product reaches below 60%, it will be indicated with 'Restock Req'. By doing this, we can add the field to the table above and apply conditional formatting, where the cells that have 'Restock req' are highlighted red.

Product	Total Quantity	Total Cost	Stock Status	Percentage Remaining
Sun Dress	50	$1,609.50	Restock Req	52.00%
Cocktail Dress	70	$3,590.30	Restock Req	33.00%
Lace Blouse	70	$1,176.00	Enough Stock	73.00%
Knitted Sweater	80	$1,919.20	Enough Stock	84.00%
Peplum Dress	80	$3,519.20	Enough Stock	91.00%
Summer Dress	80	$1,440.00	Restock Req	41.00%
Evening Gown	90	$11,339.10	Restock Req	56.00%
Henley Shirt	90	$935.10	Enough Stock	76.00%
Puffer Vest	90	$4,616.10	Enough Stock	98.00%
Casual T-Shirt	100	$900.00	Enough Stock	87.00%
Track Pants	100	$1,799.00	Enough Stock	96.00%
Wrap Dress	100	$3,419.00	Enough Stock	92.00%
Bootcut Jeans	110	$3,483.70	Enough Stock	99.00%
Flannel Shirt	110	$1,923.90	Enough Stock	97.00%
Graphic Tee	110	$1,731.40	Enough Stock	65.00%
Maxi Dress	110	$4,002.90	Enough Stock	64.00%

Unit Cost of Products

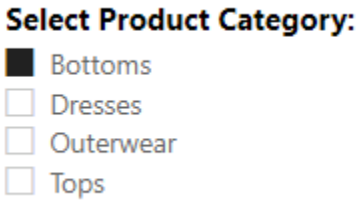
The client also wishes to see the Unit Cost of each product. To display this, we can simply create another table and select the Unit cost field and product name to be the columns of the table.



Product	Unit Cost
Leather Jacket	157.49
Evening Gown	125.99
Wool Blend Coat	118.79
Double-Breasted Coat	109.99
Parka Coat	74.09
Halter Dress	62.99
Windbreaker	55.99
Quilted Jacket	51.99
Cocktail Dress	51.29
Puffer Vest	51.29
Peplum Dress	43.99
Shift Dress	41.99
Maxi Dress	36.39
Sports Jacket	36.00
Fleece Jacket	35.99
Raincoat	34.99

Category Filter

To be able to gain insights based on product categories, I also built a slicer that allows the client to select between product categories, to see the inventory statuses and gain additional insights.



Total Quantity Of Stock

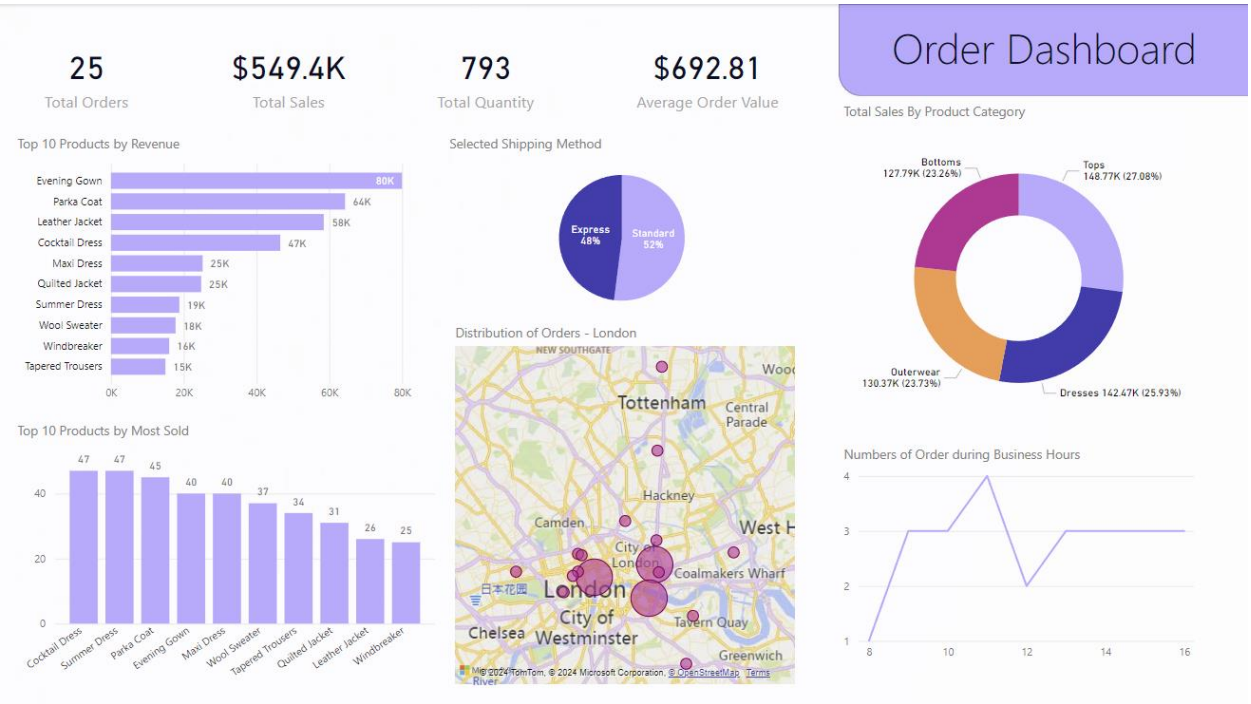
This KPI is meant to be a single value that tells the client how much stock the business currently holds. So, it would be appropriate to use a card visual and select the stock quantity field without a filter.

Total Cost of All Stock

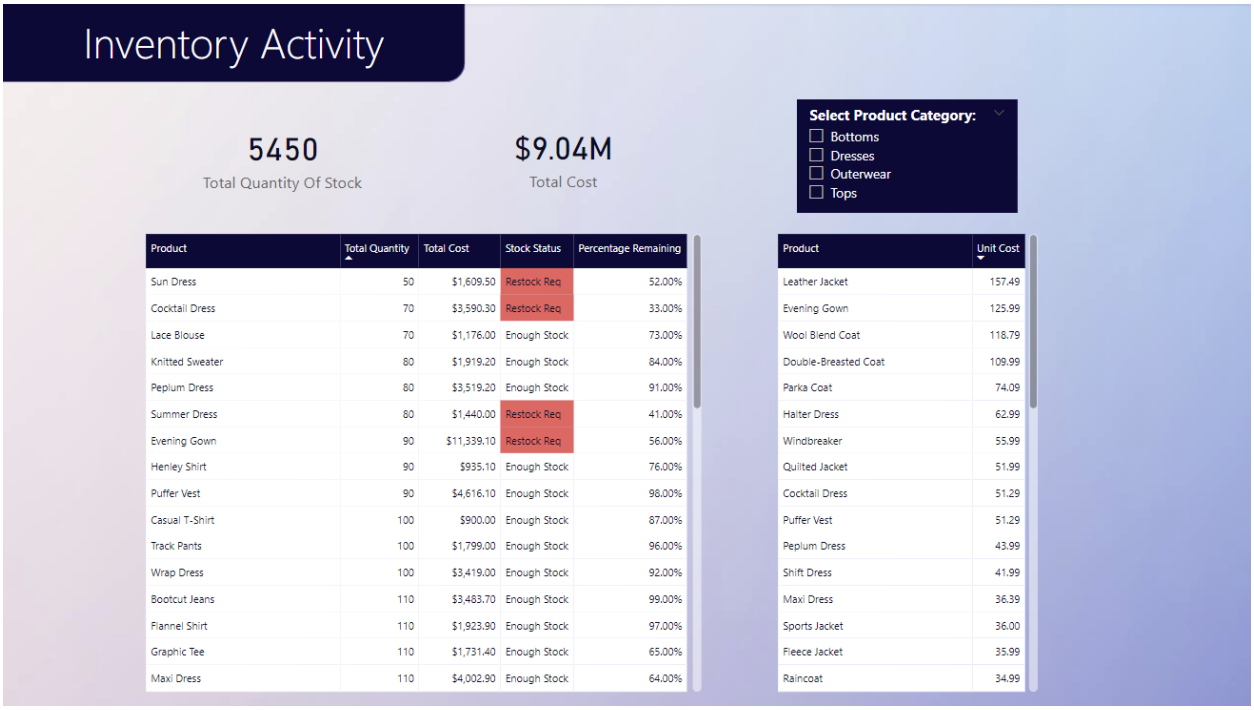
This KPI is also meant to be a single value that will tell the client how much has been spent on all the stock held by the business. So, we will also use a card visual for this and select the Total Cost Measure as the value to be displayed.

Final Dashboards

Order Dashboard



Inventory Dashboard



Product	Total Quantity	Total Cost	Stock Status	Percentage Remaining
Sun Dress	50	$1,609.50	Restock Req	52.00%
Cocktail Dress	70	$3,590.30	Restock Req	33.00%
Lace Blouse	70	$1,176.00	Enough Stock	73.00%
Knitted Sweater	80	$1,919.20	Enough Stock	84.00%
Peplum Dress	80	$3,519.20	Enough Stock	91.00%
Summer Dress	80	$1,440.00	Restock Req	41.00%
Evening Gown	90	$11,339.10	Restock Req	56.00%
Henley Shirt	90	$935.10	Enough Stock	76.00%
Puffer Vest	90	$4,616.10	Enough Stock	98.00%
Casual T-Shirt	100	$900.00	Enough Stock	87.00%
Track Pants	100	$1,799.00	Enough Stock	96.00%
Wrap Dress	100	$3,419.00	Enough Stock	92.00%
Bootcut Jeans	110	$3,483.70	Enough Stock	99.00%
Flannel Shirt	110	$1,923.90	Enough Stock	97.00%
Graphic Tee	110	$1,731.40	Enough Stock	65.00%
Maxi Dress	110	$4,002.90	Enough Stock	64.00%

Product	Unit Cost
Leather Jacket	157.49
Evening Gown	125.99
Wool Blend Coat	118.79
Double-Breasted Coat	109.99
Parka Coat	74.09
Halter Dress	62.99
Windbreaker	55.99
Quilted Jacket	51.99
Cocktail Dress	51.29
Puffer Vest	51.29
Peplum Dress	43.99
Shift Dress	41.99
Maxi Dress	36.39
Sports Jacket	36.00
Fleece Jacket	35.99
Raincoat	34.99