Superstore Data Management System Design and Analysis

Wenlan Shi*, Huan Gao†, Yifan Zhao‡, Yebin Zhong§, Zihao Wang¶
School of Data Science
The Chinese University of Hong Kong, Shen Zhen
Email: *119010265, †119010076, ‡119010462, §120090162, ¶119010324@link.cuhk.edu.cn

***Abstract*—An abstract should summarize the work in brief. Now follows some text that you can remove by writing the sign "%" in front of the command "blindtext", you can always comment out text you have written using the "%" command.**

***Index Terms*—Alphabetical, Be, In, Order, Should**

I. Introduction

A. Motivation

Generally, this section provides an introduction and literature overview, aim of the work and research questions.

B. Requirements of the organization

requirements of the organization

II. DATABASE STRUCTURE DESIGN

A. Overview of Database

Four main entities involved in our database are Customer, Product, Order and Address. The database store necessary information about entities and their relationships. And the entity Schema include

• Customer {Customer ID, Customer Name, Segment}

• Product {Product ID, Category, Sub-Category, Product Name}

• Order {Order ID, Customer ID, Product ID, Order Date, Ship Date, Ship Mode, Sales, Quantity, Discount, Profit, Postal Code}

• Address {Postal Code, Country, City, State, Region}

And the meaning of each attribute is as following:

• Order ID => Unique Order ID for each Customer.

• Order Date => Order Date of the product.

• Ship Date => Shipping Date of the Product.

• Ship Mode=> Shipping Mode specified by the Customer.

• Customer ID => Unique ID to identify each Customer.

• Customer Name => Name of the Customer.

• Segment => The segment where the Customer belongs.

• Country => Country of residence of the Customer.

• City => City of residence of of the Customer.

• State => State of residence of the Customer.

• Postal Code => Postal Code of every Customer.

• Region => Region where the Customer belong.

• Product ID => Unique ID of the Product.

• Category => Category of the product ordered.

• Sub-Category => Sub-Category of the product ordered.

• Product Name => Name of the Product.

• Sales => Sales of the Product.

• Quantity => Quantity of the Product.

• Discount => Discount provided.

• Profit => Profit/Loss incurred.

Order is a weak entity, whose existence is dependent on Customer and Product. It is because that customers have to buy products to form orders. The combination of Order ID, Customer ID and Product ID determines the unique order transaction.

Customer have a many-to-many relationship with Address, and every customer should have at least one address. This is because a customer must have at least one delivery address to make an order, and a customer can have multiple delivery addresses, while multiple customers can have the same delivery address.

B. E-R Diagram

The ER-Diagram(Fig.1) of our system contains four entities and two relations.

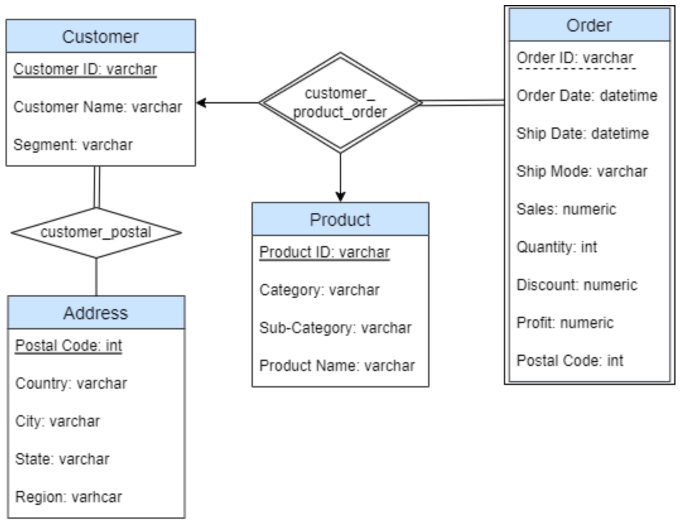


Figure 1. ER Diagram of Supermarket Management System.

C. Relational Schema

The relational schema generated from gieven ER diagram is as following:

- Customer {<u>CustomerID</u>, CustomerName, Segment}
- Product {<u>ProductID</u>, Category, SubCategory, ProductName}
- Order {<u>OrderID</u>, <u>CustomerID</u>, <u>ProductID</u>, OrderDate, ShipDate, ShipMode, Sales, Quantity, Discount, Profit, PostalCode}
- Address {<u>PostalCode</u>, segment, Country, City, State, Region}
- customer_postal {<u>CustomerID</u>, <u>PostalCode</u>}

$Customer$ contains the information of the customer except the address. $Product$ contains the information of the product. $Order$ contains the information of the order. $Address$ contains all possible address information of all customers. $customer\_postal$ contains the many to many relationship between $Customer$ and $Address$.

D. Foreign key Constraint

The forerign key referencing in our schema is as following:

- $CustomerID$ in $Order$ is a foreign key from $Order$ referencing $Customer$. Explaination: $Order$ and $Customer$ are linked through $CustomerID$. In this way, we can refer to the customer who made the specific order. An order cannot exist without a customer.
- $ProductID$ in $Order$ is a foreign key from $Order$ referencing $Customer$. Explaination: $Order$ and $Product$ are linked through $ProductID$. In this way, we can refer to the product which was purchased in the specific order. An order cannot exist without a purchased product.
- $PostalCode$ in $Order$ is a foreign key from $Order$ referencing $Address$. Explaination: Explaination: In this way, we can refer to the deliver address where the purchased product should be delivered to with the specific order. And we can check the constraint that the combination of $CustomerID$ and $PostalCode$ should exist in $customer\_postal$. If not, we can reject the order, and ask whether the customer would like to update his or her address.
- $CustomerID$ in $customer\_postal$ is a foreign key from $customer\_postal$ referencing $Customer$. And $PostalCode$ in $customer\_postal$ is a foreign key from $customer\_postal$ referencing $Address$. Explaination: $Customer$ and $Address$ are linked through $customer\_postal$. In this way, we can refer to the address of a specific customer, and refer to the customer of a specific address, while saving a lot of memory.

E. Functional Dependency

The functional dependencies in our schema are as following:

- CustomerID $\rightarrow$ CustomerName, Segment
- ProductID $\rightarrow$ Category, SubCategory, ProductName
- {OrderID, CustomerID, ProductID} $\rightarrow$ OrderDate, ShipDate, ShipMode, Sales, Quantity, Discount, Profit, PostalCode
- PostalCode $\rightarrow$ Country, City, State, Region

F. Normalization of Schema

Since its domain is atomic, no partial dependency, no transitive dependency, so it meets Boyce-Codd normal form, so it meets Boyce-Codd normal form (BCNF). However, considering that in some datasets, the product name may become the candidate key (in our dataset, the product name is not the candidate key), we tend to declare third normal form (3NF) here.

G. Index and Hashing

a) CustomerID

In our dataset, $CustomerID$ contains some information about $CustormerName$: The combination of the first letter of the $CustormerName$ is the prefix of the $custormerID$. E.g. CG-12520 for Claire Gute.

In this way, when searching for customers with a specific customer name, non-target objects can be quickly filtered out by the two-letter prefix of CustomerID.

b) ProductID

$ProductID$ contains some information about $Category$ and $SubCategory$: The combination of the shortcut of the $Category$ and $SubCategory$ is the prefix of the $ProductID$. E.g. FUR-BO-10001798 for Category of Furniture, and Sub-Category of Bookcases.

In this way, when searching for products with a specific Category and Sub-Category, non-target objects can be quickly filtered out by the "3+2"-letter prefix of ProductID.

In our data, every transaction can be located using OrderID, CustomerID, ProductID, and those attributes are consist of Alphabetical prefixes and numeric suffixes, we can hash based on its letter prefix to speed up the search.

III. IMPLEMENTATION

A. Technology Stack

The major technology stack components used in our project are as following: Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript and Django (High-level Python web framework).

For more details, the data displayed on the interface is realized by HTML and the style of the page is written in CSS. The back-end database construction and application interfaces are implemented by Django.

B. Database Construction

a) Load Dataset

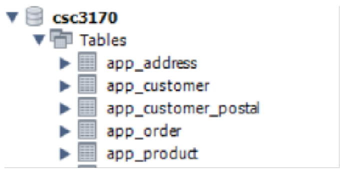
After running the back end database on the local server, we can load the data by accessing the url 127.0.0.1:8000/load/. When visiting the page, the self defined function $load(request)$ will be called and load the data to the local Mysql database.

b) Entities Construction

Model is Django's schema for representing data. It uses Python classes as the basis for setting data items and data formats in $Models.py$, basically a database table for each class.

With such a class, we can define a table of a database and operate on the results of the database at an abstract level through some Settings of the table and fields. The following classes are implemented:

- class Customer(models.Model)
- class Product(models.Model)
- class Order(models.Model)
- class Address(models.Model)
- class customer$_{postal}(models.Model)$



```
▼ csc3170
  ▼ Tables
    ▶ app_address
    ▶ app_customer
    ▶ app_customer_postal
    ▶ app_order
    ▶ app_product
```

Figure 2. Auto generated Mysql tables.

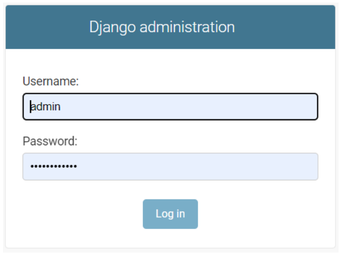
c) Administrator Page

Our project provides an administrator page for supermarket managers. So that they can view, modify, delete and other operations on commodity information, customer information and order information.

With command line

$python\ manage.py\ createsuperuser$,

the user can create an administrator account and log in at at url 127.0.0.1:8000/admin/.

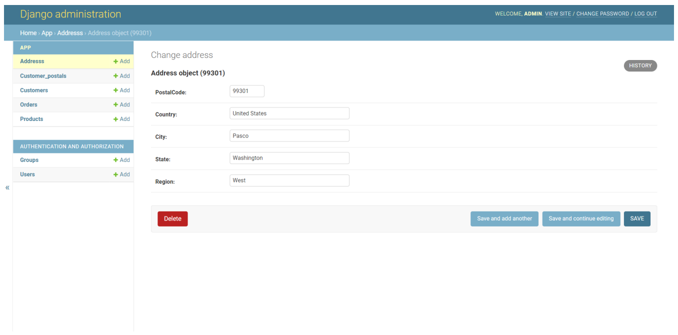


Django administration

Username: admin

Password: ••••••••••••

Figure 3. Admin login page.



Change address

Address object (99301)

PostalCode:	99301
Country:	United States
City:	Pasco
State:	Washington
Region:	West

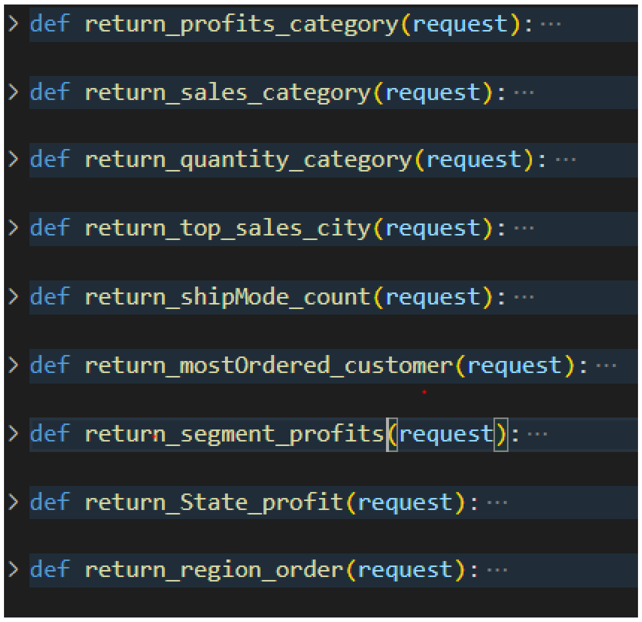
Figure 4. Administration Page.

The manager of the supermarket can change the entity to edit by clicking on the list of tables in the upper left corner of the page. And easily manipulate the page HTML input controls to make changes to the data. You can also add or remove data by clicking the Add or delete button.

d) Data analysis URLs

We set up multiple interfaces for data analysis with Django, and we can access them through the specified URL.

- $admin/$: Login to the administration page
- $load/$: Load data
- $productCount/$: Return the count of each product sold.
- $productProfit/$: Return the profit of each product.
- $categoryCount/$: Return the quantity of products from each category sold.
- $categoryProfit/$: Return the profit of each category.
- $categorySales/$: Return the sales of each category.
- $citySales/$: Return the sales of each city.
- $ShipModeCount/$: Return the count of each ship mode.
- $customerCount/$: Return the quantity of the products purchased by each customer.
- $regionCount/$: Return the quantity of the products purchased in each region.



```
> def return_profits_category(request): ···

> def return_sales_category(request): ···

> def return_quantity_category(request): ···

> def return_top_sales_city(request): ···

> def return_shipMode_count(request): ···

> def return_mostOrdered_customer(request): ···

> def return_segment_profits(request): ···

> def return_State_profit(request): ···

> def return_region_order(request): ···
```

Figure 5. Related query functions related to optional custom interfaces.

IV. DATA ANALYSIS AND DATA MINING

A. SQL function implementation

Produce sample SQL queries on these relations that are used for practical daily operations and activities

Produce sample SQL queries on these relations which are of an analytic or data mining nature

Suggest which data fields of the relational schema should be indexed or hashed, and explain your decision

B. Data Mining

Here, with the tools we build and the data visualization tool, Flourish, we take a look what data analysis work we can potentially using our tools.

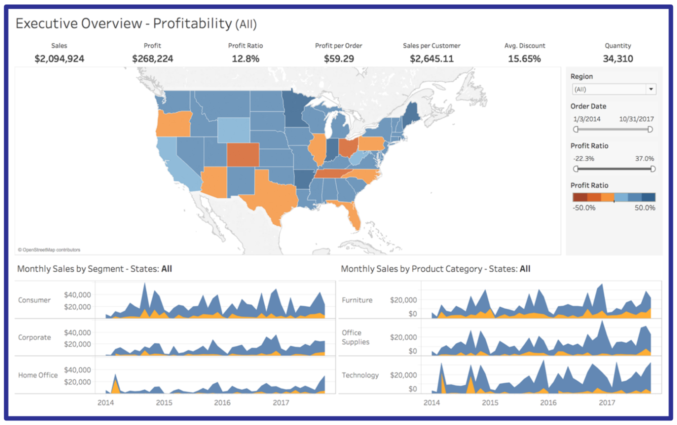

Executive Overview - Profitability (All)
Sales $2,094,924
Profit $268,224
Profit Ratio 12.8%
Profit per Order $59.29
Sales per Customer $2,645.11
Avg. Discount 15.65%
Quantity 34,310
Region (All)
Order Date 1/3/2014 10/31/2017
Profit Ratio -22.3% 37.0%
Profit Ratio -50.0% 50.0%
© OpenStreetMap contributors
Monthly Sales by Segment - States: All
Consumer
Corporate
Home Office
$40,000
$20,000
Monthly Sales by Product Category - States: All
Furniture
Office Supplies
Technology
$20,000
$0
2014
2015
2016
2017

Figure 6. Overview map of the data set

We can first build correlation map with our tools to give users a basic insight of the data set.

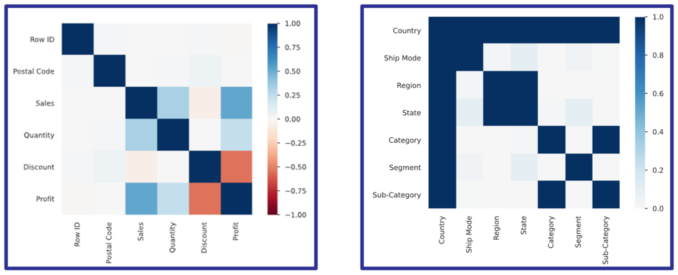


Figure 7. Correlation map of the data set using Spearman's and Cramér's methods

Then, we can select an attribute that we are interested in and analyze from there. Here, we select the "category" attribute. As shown in the image below, office supplies leads in sales in the three main categories.

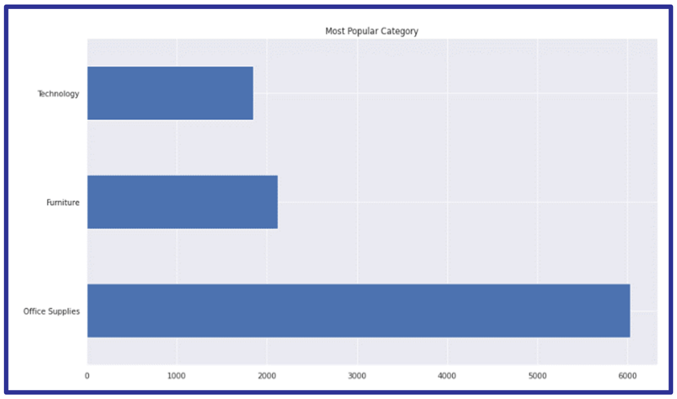


Figure 8. Category and Sales

We can add more attributes to give users a better understanding of the data. Here, we add profit to the mix. And we can see that though office supplies has the best sales, but technology has a much better profit margin, which makes it have the highest overall profit.

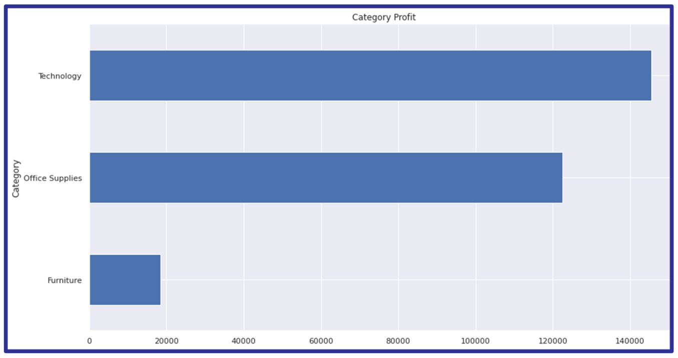


Figure 9. Category and Profit

Also, we add the "city" attribute for a deeper analysis. We can see that New York city as the economic center of the US, it has the highest profit when it comes economic related category like "technology" and "office supplies". And Seattle, though a much smaller city than New York, has a higher profit than New York with "furniture", showing its livable suburb property as a city. With more data like the population of each city, we can expand further and do a deeper analysis using our tools, showing great potential of our tools for data analysis and mining.

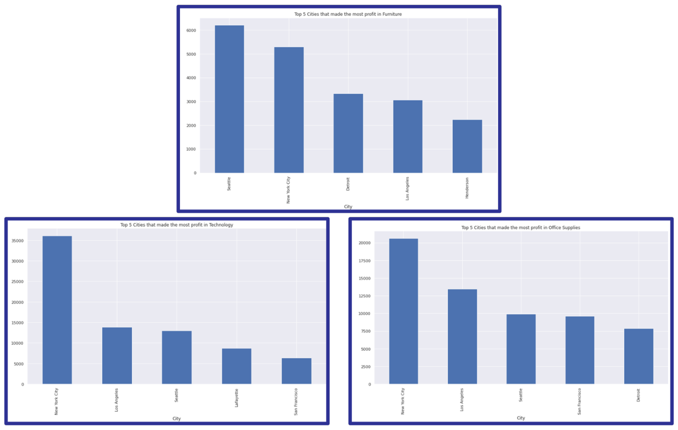


Figure 10. Category and Profit and City

Additionally, we can build other useful graph, such as pie chart. Here, we build a pie chart to show the percentage of different kind of customer's contribution to sales.

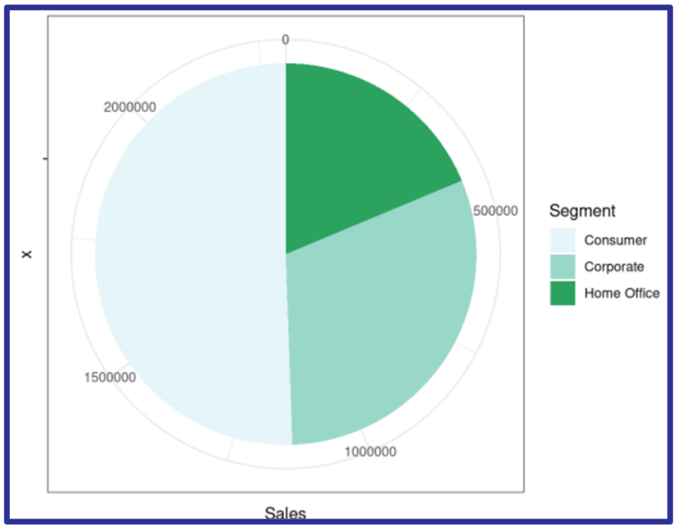


Figure 11. Pie chart of category of customers

V. Conclusion

a) Conclusion

We used a large amount of data to do the superstore value analysis. The structure of our database satisfies the 3NF to reduce the redundancy as much as possible. We recognized Customer, Address, Product, Order as main entities and studied the relation between them. Then, we designed the ER-diagram and tried to combine relations with entities to improve query efficiency. The relation customer_postal links the relationship between the Customer and Address. The relation customer_product_order links the rest of entities. We also reduce the refined ER-diagram to the relational schema.

Then, to makes it easy for users to query the results of our study. We designed a user-friendly website and had a lot of data analysis outcomes. Firstly, we studied correlations on the attributes of each entities. Then, we make comparisons of category by Sales and Profit. Finally, we get the most ordered category per region by considering the city on the basis of the previous two analysis.

b) Future

As for further enhancement, more realistic data and attributes should be added to extend the database. With the increase of data, the storage structure needs to be optimized to speed up data access.

In conclusion, we design and implement the Super Market management system. We expect it will help the superstore to allocate different commodities in different regions to maximize profits.

c) Self-evaluation

In this project, we designed and implemented this superstore database by in-depth analysis of the needs of supermarkets and consumers in various regions. In the process of project implementation, we have deepened our understanding of the database, such as the determination of entities, the relationship between attributes, how to manipulate the database, etc.

Wenlan Shi: ER-diagram, relational schema, database implementation and data loading.

Huan Gao:

Yifan Zhao: Data analysis, database framework and presentation slides.

Yebin Zhong:

Zihao Wang:

REFERENCES