CIS550 Final Project: Oeda Platform

Huifang Ye, Joyce An-jie Wang, Zhihui Zhang, Zhuoran Hu

Project Overview

1

Introduction

- Target Problem
- Project Goal
- Application Functionality

Project goals and target problems

Create a user-friendly, intuitive Business Intelligence or Business Analytics tool for e-commerce:

- People’s shopping preferences
- Generate graphs and charts
- Analytics for sales and marketing
- Geographic insights

Application Functionality

Display trends

Display trending/featured visual analysis

Search

Search product, order, or payment information using multiple different search keywords and filters

Create reports

Users can generate a variety of charts, widgets, pivot, summary or tabular views on sales and geographical data based on their search input

Export

Users can export queried results as .csv files to their local device

Visualization

Provides visualization for geographic distribution of customer/sellers/sales data and display Brazilian city demographics

Market Analysis

Users can search and generate sales results and product analysis based on Brazilian city demographics

2

Overview of Datasets

- Brazilian E-Commerce Dataset
- Brazilian Cities

Overview of Datasets

Brazilian E-Commerce Dataset by Olist:

- Item
- Product
- Customer
- Payment
- OrderInfo
- Category
- Seller
- Review

A Brazilian ecommerce public dataset of orders made at the Olist store. The dataset has information of 100k orders from 2016 to 2018 made at multiple marketplaces in Brazil.

Overview of Datasets

Brazilian Cities

- This dataset is a compilation of several publicly available demographic information about Brazilian Municipalities.
- There are in total 79 fields for each city, which includes city, state, resident population, Human Development Index (HDI), number of Pay TV users, GDP, number of companies by industries, Walmart stores, and etc.

3

Data Cleaning

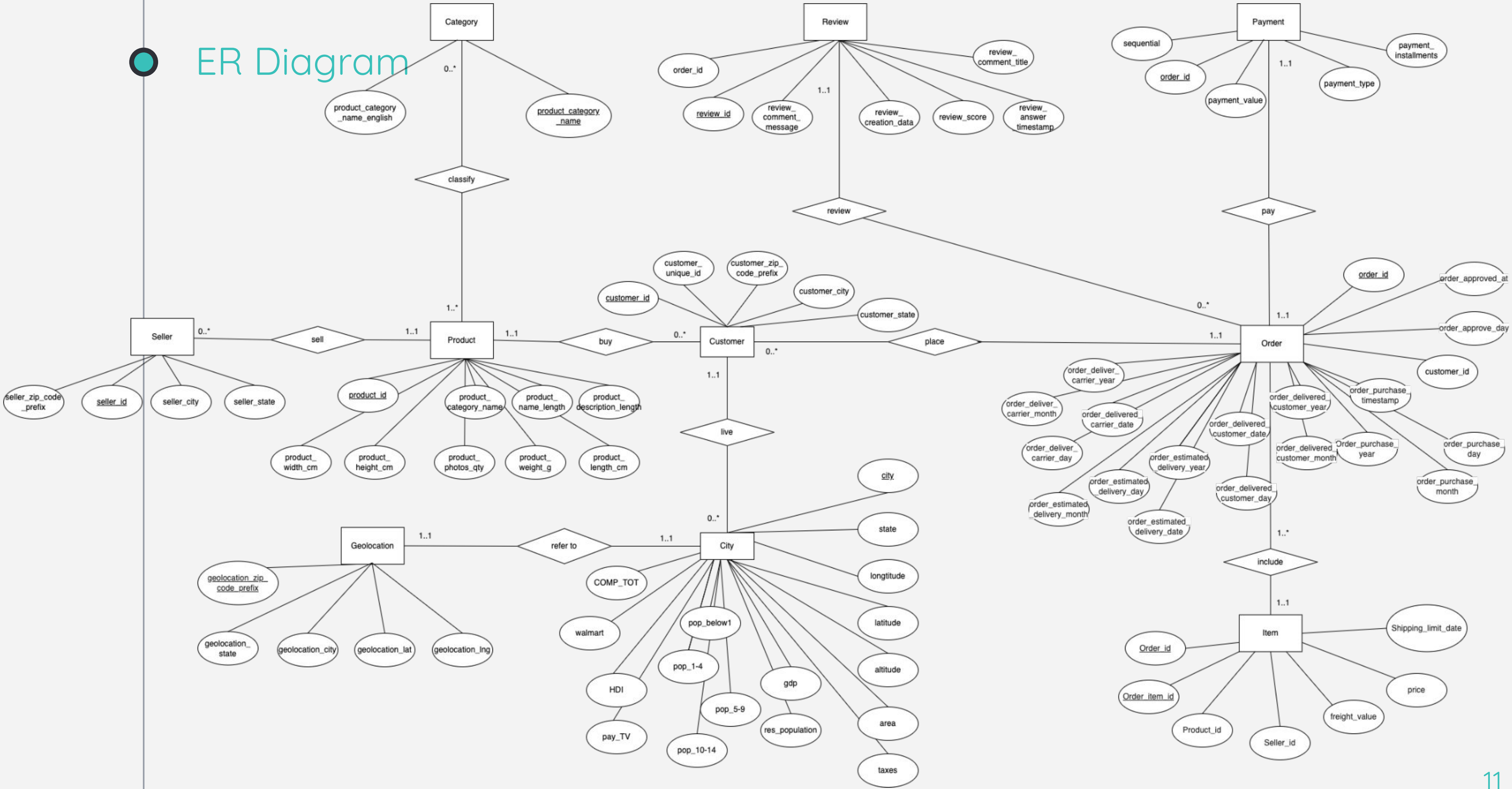
- Use Python (Pandas, PandaSQL) and Colab
- Problems with the datasets
  - Null values, irrelevant columns, incorrect column names, information scatter in various tables
- How we address the problems
  - Dropna, select useful columns, rename columns, merge and natural join cleaned datasets

4

Database

- Schema Design & ER Diagram
- Normal Form and Justification

ER Diagram



Category
0..*
product_category_name_english
product_category_name
classify
1..*
Review
order_id
review_id
review_comment_message
1..1
review_creation_data
review_score
review_answer_timestamp
review_comment_title
review
0..*
Payment
sequential
order_id
payment_value
1..1
payment_type
payment_installments
pay
Seller
0..*
sell
1..1
Product
1..1
buy
0..*
Customer
0..*
place
1..1
Order
1..1
seller_zip_code_prefix
seller_id
seller_city
seller_state
product_id
product_category_name
product_name_length
product_description_length
product_width_cm
product_height_cm
product_photos_qty
product_weight_g
product_length_cm
customer_id
customer_unique_id
customer_zip_code_prefix
customer_city
customer_state
1..1
live
0..*
order_id
order_approved_at
order_approve_day
customer_id
order_deliver_carrier_year
order_deliver_carrier_month
order_delivered_carrier_date
order_deliver_carrier_day
order_delivered_customer_year
order_purchase_timestamp
order_delivered_customer_date
order_delivered_customer_month
Order_purchase_year
order_purchase_day
order_purchase_month
order_estimated_delivery_year
order_estimated_delivery_day
order_estimated_delivery_month
order_estimated_delivery_date
order_delivered_customer_day
1..*
include
1..1
Item
Order_id
Order_item_id
Product_id
Seller_id
freight_value
price
Shipping_limit_date
Geolocation
1..1
refer to
1..1
City
geolocation_zip_code_prefix
geolocation_state
geolocation_city
geolocation_lat
geolocation_lng
COMP_TOT
walmart
HDI
pay_TV
pop_below1
pop_1-4
pop_5-9
pop_10-14
gdp
res_population
city
state
longtitude
latitude
altitude
area
taxes

Normal Form & Justification

Customer (customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state)

F = {customer_id → customer_unique_id, customer_id → customer_zip_code_prefix, customer_id → customer_city, customer_id →customer_state}

Proof:

customer_id is the primary key, therefore, a superkey of Customer. Since for every functional dependency, X → A holds over Customer, X is a superkey of Customer, we know this relation is in **BCNF**.

5

Complex Queries and Optimization

- Performance Evaluation
  - Runtime and Optimizations
- Example of Complex Query

Performance Evaluation

Query Description	Original	Optimized
Show the average rating and the number of review for each product category (type 1)	**769 ms**	**214 ms**
Show the average rating and the number of review for each product category (type 2)	**577 ms**	**225 ms**
Query the differences of total orders between 2016, 2017, and 2018 for each product category	**812 ms**	**409 ms**
Create a sales report for the top 5 cities with the most Walmart stores	**934 ms**	**551 ms**
Retrieve the top 10 products with highest average review scores	**148 ms**	**86 ms**
Get the differences in total, average, max, and min payment values by credit card users and bank ticket users from each state	**442 ms**	**280 ms**

Example: Complex Query

For each of the five cities, list the total number of Walmart stores, total number of orders (based on customer location) by year, total sales (based on product price) by year, and the most popular product category based on sales (in English) by year. The result is ordered by the number of Walmart stores.

```
WITH top_cities
       AS (SELECT city, walmart
           FROM City
           ORDER BY walmart DESC
           LIMIT 5),
    orders_products
       AS (SELECT O.order_id,
                  O.order_deliver_customer_year AS year,
                  C.customer_id,
                  C.customer_city AS city,
                  P.product_id,
                  P.product_category_name,
                  I.price
           FROM OrderInfo O
           JOIN Item I ON O.order_id = I.order_id
           JOIN Product P ON I.product_id = P.product_id
           JOIN Customer C ON O.customer_id = C.customer_id),
   total_orders
       AS (SELECT city, year, COUNT(DISTINCT order_id) AS count
           FROM orders_products
           WHERE city IN (SELECT city FROM top_cities)
           GROUP BY city, year),
   total_sales
       AS (SELECT city, year, SUM(price) AS sales
           FROM orders_products
           WHERE city IN (SELECT city FROM top_cities)
           GROUP BY city, year),
   top_product
       AS (SELECT city, year, c.product_category_name_english, SUM(price) AS sales
           FROM orders_products op
           JOIN Category c ON c.product_category_name = op.product_category_name
           WHERE city IN (SELECT city FROM top_cities)
           GROUP BY city, year, c.product_category_name_english)
SELECT tc.city AS City,
       tc.walmart AS 'Number of Walmart Stores',
       tto.year AS Year,
       tto.count AS 'Number of Orders',
       ts.sales AS Sales,
       tp.product_category_name_english AS 'Top Selling Product'
FROM top_cities tc
NATURAL JOIN total_orders tto
NATURAL JOIN total_sales ts
JOIN top_product tp ON tc.city = tp.city
WHERE tto.year = tp.year AND tp.sales >= ALL (SELECT sales
                                              FROM top_product tp
                                              WHERE tp.city = tc.city
                                              AND tp.year = tto.year)
ORDER BY tc.walmart DESC, tto.year;
```

Optimization

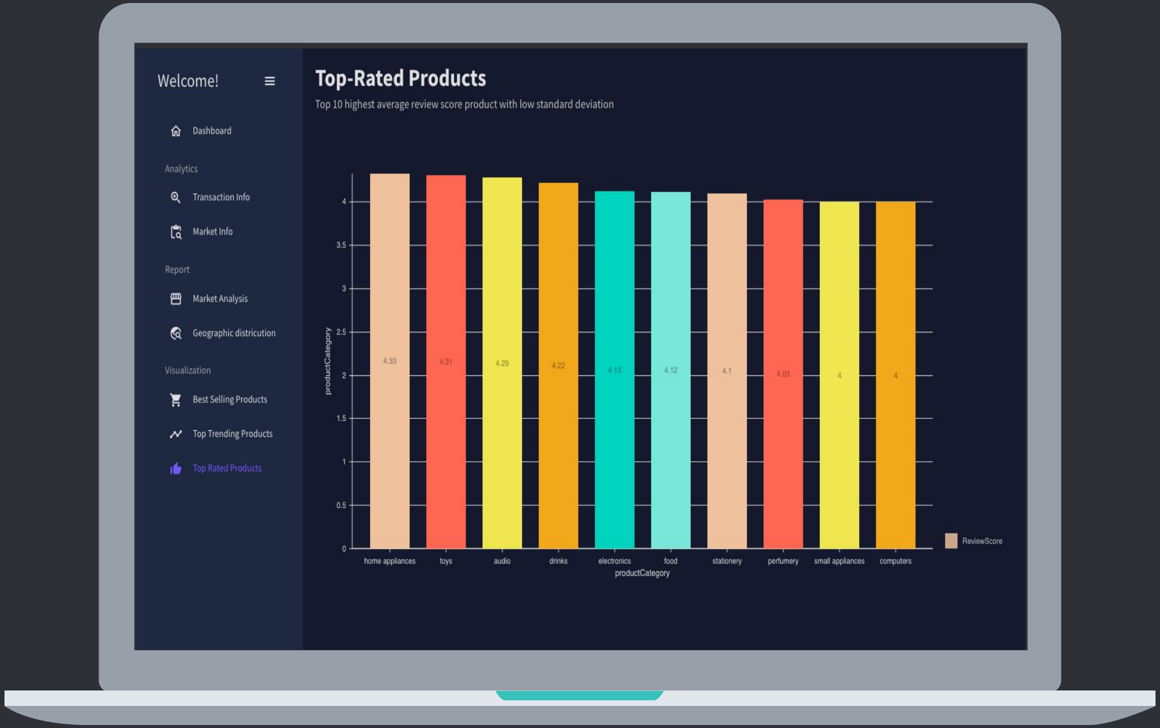
Strategy 1: Push selection and projection down to the base query. Specify necessary column names in the SQL query instead of selecting all columns (using SELECT * FROM).

Strategy 2: Use temporary tables to temporarily store frequently used data, which leads to improvement in query performance. We try to avoid correlated subqueries because these could decrease the speed of execution.

Strategy 3: Limit the results obtained by the query. In case only limited results are required, it is better to use the LIMIT statement. This statement limits the records and only displays the number of records specified.

Live Demo

Website Overview



6

Technical Challenges

- Set up Node.js in different systems
- Read Timeout Error
- Connecting data to the frontend

7

Extra Credit Features

- Code Coverage (unit testing > 80%)
- Integration with APIs and colab

Thank you!

Questions and Comments