E-commerce Website

CSE4/560 Data Models and Query Language Semester Project
Milestone-2 Report

Aman Prakash
Engineering and Applied Sciences
M.S in Data Science
University at buffalo
email address

Serath Chandra Nutakki
Engineering and Applied Sciences
M.S in Data Science
University at buffalo
serathch@buffalo.edu

Prashant Upadhyay
Engineering and Applied Sciences
M.S in Data Science
University at buffalo
pupadhya@buffalo.edu

***Abstract*— In the present environment of business shopping manually at the store is a hassle, especially with multiple products, discounts, and variations. Considering all the parameters we would like to build an e-commerce website to serve the customer with efficiency, time management and provide insights for which a database is required.**

I. INTRODUCTION

E-commerce is quickly becoming a common business model. Web sites with the capability for conducting commercial transactions over the internet are being implemented by an increasing number of companies. Shopping via the internet may be described as a simple norm.

In this existing modern world, there is a need for modern shopping which can be satisfied by online shopping through an e-commerce website mainly due to its simplicity and easiest way to communicate and transition between buyer and seller.

The main objective of our project is to implement an e-commerce store website where multiple products like clothes, electronics, and many more can access through the internet from our own comfort by connecting to a database.

II. IMPLEMENTATION

We are going to implement an e-commerce website, where we want to store multiple users, products, discount coupons, etc. and for that, we will need a database. An e-commerce website has multiple products, and the end-user is not going to buy any product without having a view of the item, so we need to insert images of the product into our database. There will be multiple users buying the same items and this creates data redundancy which will hamper the data processing.

Taking the above points in mind we need a database instead of an excel file because:

Data Purpose

What type of data are we collecting? Excel files are great for text and numeric values in relatively low volume, but databases can not only handle numeric and text values but can easily handle other types of information, such as documents and images. Moreover, in our project, we will store multiple images which are not feasible for the database to capture.

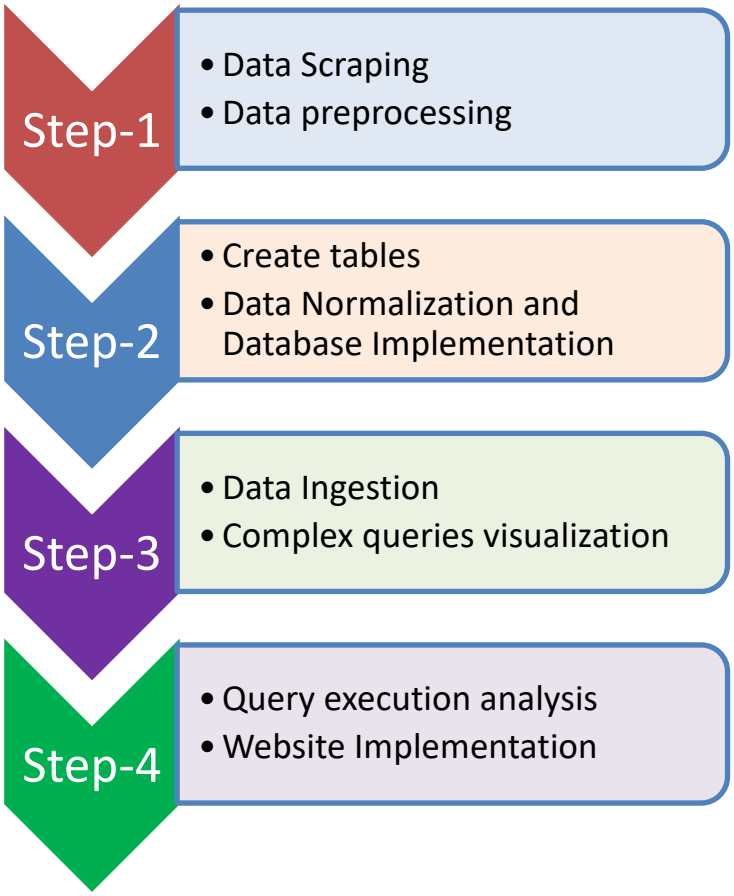
Data Volume

Spreadsheets have record limitations whereas databases do not. Spreadsheets require a large amount of hard-drive space for data storage than databases. When a spreadsheet has many fields, the spreadsheet can be hard to read, and finding specific data can be challenging. Relational databases on the other hand uses querying tools to overcome these issues.

Redundancy

The database structure avoids data redundancy. Since the data in different database tables are linked, there is little or no duplication of source data. On our project, one user can have multiple purchases of single items and of different items with the same quantity or different, so this information of duplicate items will make the excel sheet redundant.

***Design process*:**



Data Scraping:

The process of extracting the data from the website into a well-maintained spreadsheet that can be used as our data for further implementation is called Data Scraping.

It allows us to scrap specific data like product details, inputs, user details, order details, categories, and many more. This way we can export large product data in CSV for our desired output.

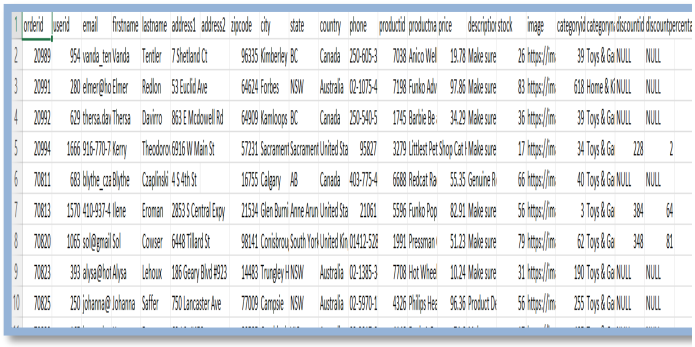
In this project, we extracted our essential data from the Amazon website through various sources and toolkits.

Data preprocessing:

Our sophisticated databases and websites cannot be implemented on poor data because they can yield insignificant and inefficient results. So, in order to avoid that we need to perfectly clean our data with special techniques, and this process is known as data preprocessing.

In this process, we have taken care of all the missing values product details, and order details. Filling the unknown prices with average mean values.

The sample representation of our dataset:



	orderid	userid	email	firstname	lastname	address1	address2	zipcode	city	state	country	phone	productid	productna	price	description	stock	image	categoryid	categorym	discountid	discountpercentage
2	20989	954	vanda.ten	Vanda	Tentler	7 Shetland Ct		96335	Kimberley	BC	Canada	250-605-3	7030	Aririco Wel	19.78	Make sure	26	https://m	39	Toys & Ga	NULL	NULL
3	20991	280	elmer@ho	Elmer	Redlon	53 Euclid Ave		64624	Forbes	NSW	Australia	02-1075-4	7186	Funko Adv	97.86	Make sure	83	https://m	618	Home & Ki	NULL	NULL
4	20992	628	thersa.dav	Thersa	Davirro	863 E Mcdowell Rd		64909	Kamloops	BC	Canada	250-540-5	1745	Barbie Be	34.29	Make sure	36	https://m	39	Toys & Ga	NULL	NULL
5	20994	1666	916-770-7	Kerry	Theodorou	6916 W Main St		57231	Sacrament	Sacrament	United Sta	95827	3279	Littlest Pet Shop Cat I	Make sure		17	https://m	34	Toys & Ga	228	2
6	20811	683	blythe_czz	Blythe	Czaplinski	4 S 4th St		16755	Calgary	AB	Canada	403-775-4	6688	Redcat Ra	55.35	Genuine R	66	https://m	40	Toys & Ga	NULL	NULL
7	20813	1570	410-937-4	Ilene	Eroman	2053 S Central Expy		21534	Glen Burni	Anne Arun	United Sta	21061	5586	Funko Pop	82.91	Make sure	56	https://m	3	Toys & Ga	384	64
8	20820	1065	sol@gmail	Sol	Cowser	6448 Tillard St		98141	Conisbrou	South York	United Kin	01412-526	1991	Pressman	51.23	Make sure	79	https://m	62	Toys & Ga	348	81
9	20823	393	alysa@hot	Alysa	Lehoux	186 Geary Blvd #323		14483	Trungley H	NSW	Australia	02-1385-3	7708	Hot Wheel	10.24	Make sure	31	https://m	190	Toys & Ga	NULL	NULL
10	20825	250	johanna@	Johanna	Soffer	750 Lancaster Ave		77009	Campsie	NSW	Australia	02-5970-1	4326	Philips Hec	96.36	Product De	56	https://m	255	Toys & Ga	NULL	NULL

Create tables:

After getting our data we generated tables for different features by specific scripts that we have written.

- Users – table contains all the user-specific data (email, userID, password, first and last name, zip code)
- Products – table created to store all the product details (productid, name, price, description, image, etc.)
- Orders – table created to store order details (ordered, userid, productid, amount, etc)
- Discount – table created to store discount details (discount Id, percentage, description, etc)
- Categories – contains all the product categories' specific details (name, category id, etc)
- Cart – To store the cart details that we have added (userid, product id, etc)
- Support – Table to store the support related details (supportId, user id, body, etc)

Each table was created and defined as follows:

```
try:
    cur.execute('''CREATE TABLE users
        (userId INTEGER PRIMARY KEY,
        password TEXT,
        email TEXT,
        firstName TEXT,
        lastName TEXT,
        address1 TEXT,
        address2 TEXT,
        zipcode TEXT,
        city TEXT,
        state TEXT,
        country TEXT,
        phone TEXT
        )''')
```

```
    cur.execute('''CREATE TABLE products
            (productId INTEGER PRIMARY KEY,
            name TEXT,
            price TEXT,
            description TEXT,
            image TEXT,
            stock INTEGER,
            categoryId INTEGER,
            FOREIGN KEY(categoryId) REFERENCES categories(categoryId)
            )''')

    cur.execute('''CREATE TABLE cart
            (userId INTEGER,
            productId INTEGER,
            FOREIGN KEY(userId) REFERENCES users(userId),
            FOREIGN KEY(productId) REFERENCES products(productId)
            )''')

    cur.execute('''CREATE TABLE IF NOT EXISTS discount (
            discountid integer NOT NULL,
            description text NULL DEFAULT NULL,
            discountPercentage integer NULL DEFAULT NULL,
            PRIMARY KEY (discountid))''')

    cur.execute('''CREATE TABLE IF NOT EXISTS orders (
            orderid integer NOT NULL,
            userid integer NULL DEFAULT NULL,
            productid integer NULL DEFAULT NULL,
            discountid integer NULL DEFAULT NULL,
            PRIMARY KEY (orderid),
```

Data Normalization:

Currently, in our dataset, we have 1,50,000 records of the products but we observed some undesirable characteristics features and data redundancies. Therefore, this creates a need to normalize the data by forming a relationship linkage between smaller tables that are been derived from the larger tables.

Removing all the partial dependencies and nonprime attributes and transforming the data to BCNF(Boyce- Codd Normal Form) which are having working functional dependencies.

Therefore, we transformed the large unstructured data by normalizing it to a structured data form manually.

Database Implementation:

We have normalized our data and we created all the tables. So, we implement this database and establish the connection.

```python
import psycopg2


def conn_create():
    conn = psycopg2.connect(
        host="localhost",
        database="ecom",
        user="postgres",
        port="8888",
        password="abc12345")

    cur = conn.cursor()

    return conn, cur
```

```python
import psycopg2
from connect_db import conn_create

conn, cur = conn_create()

try:
    cur.execute('''CREATE TABLE users
        (userId INTEGER PRIMARY KEY,
        password TEXT,
        email TEXT,
        firstName TEXT,
        lastName TEXT,
        address1 TEXT,
        address2 TEXT,
        zipcode TEXT,
        city TEXT,
        state TEXT,
        country TEXT,
        phone TEXT
        )''')

    cur.execute('''CREATE TABLE categories
            (categoryId INTEGER PRIMARY KEY,
            name TEXT
            )''')

except (Exception, psycopg2.DatabaseError) as error:
    print(error)
finally:
    if conn is not None:
        conn.close()
        print('Tables Created Successfully.')
        print('Database connection closed.')
```

Above is the code snippet of the sample execution of the table created. By compiling the create_database.py it automates the process of creating the tables by establishing a connection with the database.

Data Ingestion:

The next step after creating the tables and database implementation is data ingestion. We implemented programs and scripts to import the data into our database.

Below is the sample code snippet for data ingestion of products data:

products_ingestion.py 1 X

C: > Users > serat > AppData > Local > Temp > Temp1_Archive (7).zip > ingestion_scripts > products_ingestion.py > ...

```python
import csv
import random


def category_id_fetcher(data, category):
    for i, item in enumerate(data):
        if category in item:
            # print('Index of 1 is [{}][{}]'.format(i, item.index(1)))
            return data[i][0]


conn, cur = conn_create()
cur = conn.cursor()

cur.execute('SELECT * FROM categories')
data = cur.fetchall()
print(data)
file = open('webscrap_data/amazon.csv')

print(type(file))

csvreader = csv.reader(file)
header = []
header = next(csvreader)
print(header)

insert_val = []
count = 1
for row in csvreader:
    if row[1] != '' and row[4] != '':
        insert_val.append({
            "productid": count,
            "name": row[1],
            "price": row[7].replace('$', '') if row[7] == '' else str(random.randint(1, 100)) + '.' + str(random.randint(10, 99)),
            "description": row[10] if row[10] != '' else "Product Description Yet To Be Updated By Seller.",
            "stock": row[8] if row[8] != '' else random.randint(0, 100),
```

In this way, we import all the data to the database to the respective schema.

The dependencies of each table are as follows:

User:

Type	Name	Restriction
1.3 Sequence	public.id_seq	auto
Function	nextval('id_seq'::regclass)	auto
Primary Key	public.users_pkey	auto
Foreign Key	public.cart.kart_userid_fkey	normal
Foreign Key	public.orders.orders_userid_fkey	normal
Foreign Key	public.support.support_userid_fkey	normal
Rule	_RETURN ON public.order_details	normal

Support:

Type	Name	Restriction
Foreign Key	public.support.support_userid_fkey	auto
Primary Key	public.support_pkey	auto

Products:

Type	Name	Restriction
Foreign Key	public.products.products_categoryid_fkey	auto
Primary Key	public.products_pkey	auto
Foreign Key	public.cart.kart_productid_fkey	normal
Foreign Key	public.orders.orders_productid_fkey	normal
Rule	_RETURN ON public.order_details	normal

Order:

Type	Name	Restriction
Foreign Key	public.orders.orders_discountid_fkey	auto
Foreign Key	public.orders.orders_productid_fkey	auto
Foreign Key	public.orders.orders_userid_fkey	auto
Primary Key	public.orders_pkey	auto
Rule	_RETURN ON public.order_details	normal

Discount:

Type	Name	Restriction
Primary Key	public.discount_pkey	auto
Foreign Key	public.orders.orders_discountid_fkey	normal
Rule	_RETURN ON public.order_details	normal

Categories:

Type	Name	Restriction
Primary Key	public.categories_pkey	auto
Foreign Key	public.products.products_categoryid_fkey	normal
Rule	_RETURN ON public.order_details	normal

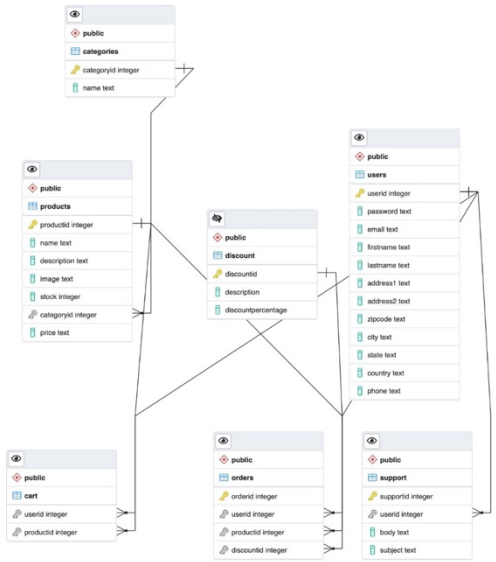
Cart:

Type	Name	Restriction
Foreign Key	public.cart.kart_productid_fkey	auto
Foreign Key	public.cart.kart_userid_fkey	auto

III. Issues Faced

We have not downloaded the data from any private repository rather we web scrapped the entire data from Amazon. We have used Beautiful Soup python package to extract the data from Amazon. It took us a lot of time to web scrape the data and clean it before implementing the database and running the query.

IV. ER Diagram



The ER diagram between all tables can be visualized

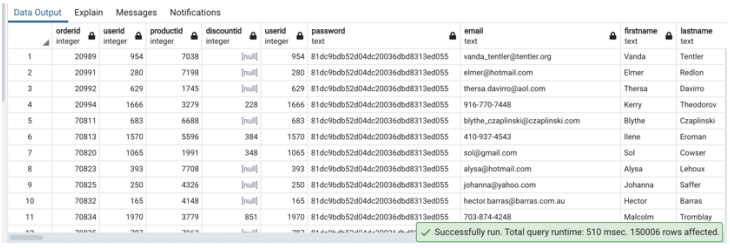
V. Query Implementation

Below are some of the queries that we implemented to get the overall scenario of the database and the relationship between all the tables.

We need to check the number of values in the dataset by connecting all the tables to each other and for that we implemented the below query which will give the total count of the number of cells in the database.

```
SELECT  * FROM orders o
FULL JOIN users u ON o.userid = u.userid
FULL JOIN products p ON o.productId = p.productId
FULL JOIN discount d ON o.discountid = d.discountid
FULL JOIN categories c ON c.categoryid = p.categoryid ;
```

Output: It took 510ms to load 150006 rows.



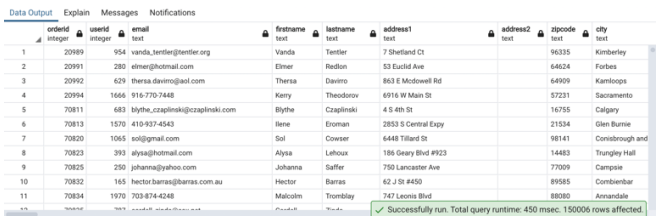
	orderid integer	userid integer	productid integer	discountid integer	userid integer	password text	email text	firstname text	lastname text
1	20989	954	7038	[null]	954	81dc9bdb52d04dc20036dbd8313ed055	vanda_tentler@tentler.org	Vanda	Tentler
2	20991	280	7198	[null]	280	81dc9bdb52d04dc20036dbd8313ed055	elmer@hotmail.com	Elmer	Redlon
3	20992	629	1745	[null]	629	81dc9bdb52d04dc20036dbd8313ed055	thersa.davirro@aol.com	Thersa	Davirro
4	20994	1666	3279	228	1666	81dc9bdb52d04dc20036dbd8313ed055	916-770-7448	Kerry	Theodorov
5	70811	683	6688	[null]	683	81dc9bdb52d04dc20036dbd8313ed055	blythe_czaplinski@czaplinski.com	Blythe	Czaplinski
6	70813	1570	5596	384	1570	81dc9bdb52d04dc20036dbd8313ed055	410-937-4543	Ilene	Eroman
7	70820	1065	1991	348	1065	81dc9bdb52d04dc20036dbd8313ed055	sol@gmail.com	Sol	Cowser
8	70823	393	7708	[null]	393	81dc9bdb52d04dc20036dbd8313ed055	alysa@hotmail.com	Alysa	Lehoux
9	70825	250	4326	[null]	250	81dc9bdb52d04dc20036dbd8313ed055	johanna@yahoo.com	Johanna	Saffer
10	70832	165	4148	[null]	165	81dc9bdb52d04dc20036dbd8313ed055	hector.barras@barras.com.au	Hector	Barras
11	70834	1970	3779	851	1970	81dc9bdb52d04dc20036dbd8313ed055	703-874-4248	Malcolm	Tromblay

As our dataset is an online ecommerce website dataset. We wanted to have a look of all the customers with their orders and their details. For such we implemented view for that because view doesn't store data permanently and it also encapsulate the name of the table.
Below is the code for view in sql.

```
CREATE VIEW order_details AS
SELECT
o.orderid as orderid,
u.userid as userid,
u.email as email,
u.firstName as firstName,
u.lastname as lastname,
u.address1 as address1,
u.address2 as address2,
u.zipcode as zipcode,
u.city as city,
u.state as state,
u.country as country,
u.phone as phone,
p.productId as productId,
p.name as productname,
p.price as price,
p.description as description,
p.stock as stock,
p.image as image,
p.categoryid as categoryid,
c.name as categoryname,
d.discountid as discountid,
d.discountpercentage as discountpercentage
FROM orders o
FULL JOIN users u ON o.userid = u.userid
FULL JOIN products p ON o.productId = p.productId
FULL JOIN discount d ON o.discountid = d.discountid
FULL JOIN categories c ON c.categoryid = p.categoryid;
```

For running the above view query we have to write the below code:
SELECT * FROM order_details;

Output: It took 450 ms to load the database with the order details for all the users with their personal details.



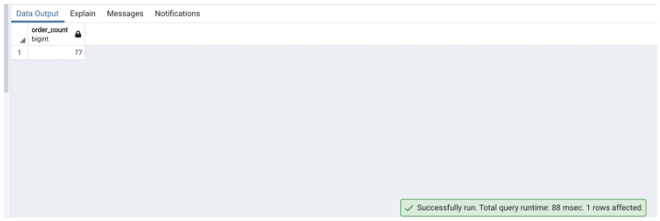
Now we wanted to see the stored orders of the user in the database and for that we have used stored procedure. Stored procedure has multiple advantages like better performance, higher productivity, scalability, easy to use and secure .Below is the code for stored procedure.

```
Stored Procedure 1 -

DROP FUNCTION order_count_per_user(text);
DROP TYPE order_details_type;
CREATE TYPE order_details_type AS
(
order_count bigint
);

CREATE OR REPLACE FUNCTION order_count_per_user(email_input text)
  RETURNS SETOF order_details_type
  AS
$func$
BEGIN
    RETURN QUERY
    SELECT count(*) as order_count FROM order_details where order_details.email=email_input;
END
$func$
LANGUAGE plpgsql;
```

To run the above code, we have used the below query:
SELECT * FROM order_count_per_user('amanprakash9597@gmail.com');

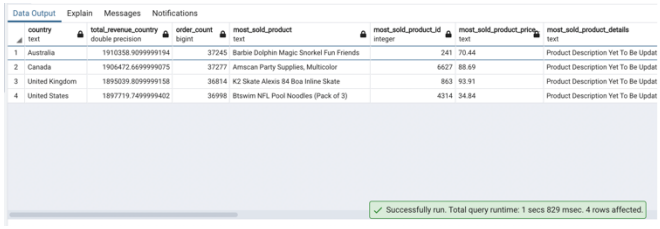


Output: It took 88ms to load 1 rows with all the orders of the user.

As we are going through only the users data and their orders, now let's also go through the country stats and check the country with the maximum sale and order.

```
CREATE VIEW get_country_stats AS
SELECT
country,
SUM(CAST(price AS DOUBLE PRECISION)) as total_revenue_country,
COUNT(*) AS order_count,
MODE() WITHIN GROUP (ORDER BY productname) as most_sold_product,
MODE() WITHIN GROUP (ORDER BY productId) as most_sold_product_id,
MODE() WITHIN GROUP (ORDER BY price) as most_sold_product_price,
MODE() WITHIN GROUP (ORDER BY description) as most_sold_product_details,
MODE() WITHIN GROUP (ORDER BY stock) as most_sold_product_stock,
MODE() WITHIN GROUP (ORDER BY image) as most_sold_product_image,
MODE() WITHIN GROUP (ORDER BY categoryname) as most_sold_category,
MODE() WITHIN GROUP (ORDER BY categoryid) as most_sold_category_id
FROM order_details
WHERE price<>''
GROUP BY country;
```

To run the above code, we have used the below query:
SELECT * from get_country_stats;



Output: It took 1 sec 829ms to load 4 rows with the country stats.

VI. Modules used

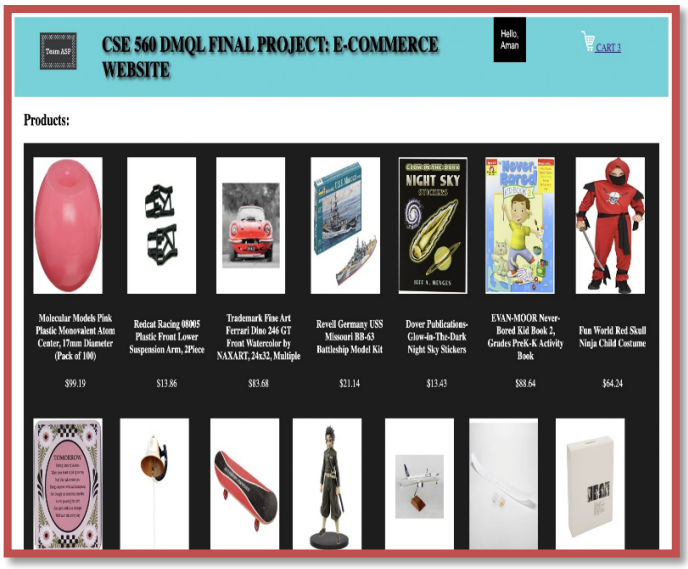
1. autopep8==1.6.0
2. click==8.1.3
3. Flask==2.1.2
4. importlib-metadata==4.11.3
5. itsdangerous==2.1.2
6. Jinja2==3.1.2
7. MarkupSafe==2.1.1
8. psycopg2==2.9.3
9. pycodestyle==2.8.0
10. toml==0.10.2
11. Werkzeug==2.1.2
12. zipp==3.8.0

VII. Website Design

Our website is designed for anyone who wants to shop a wide range of products, and different categories by login into our website with their credentials.

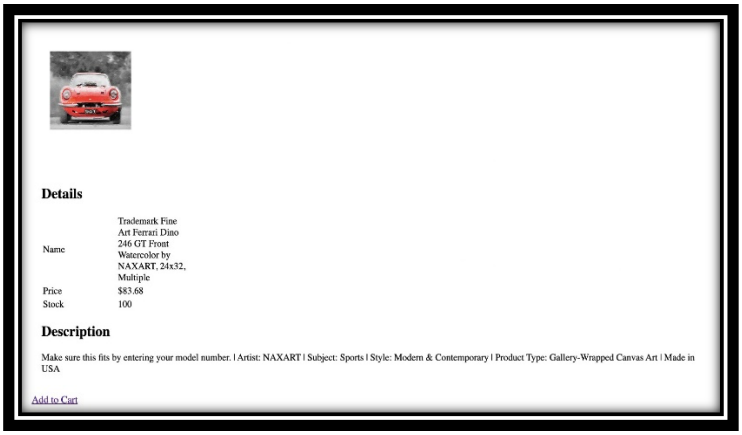
UI:

The homepage of our website linked to our database consists of multiple products that can be accessed by the user.

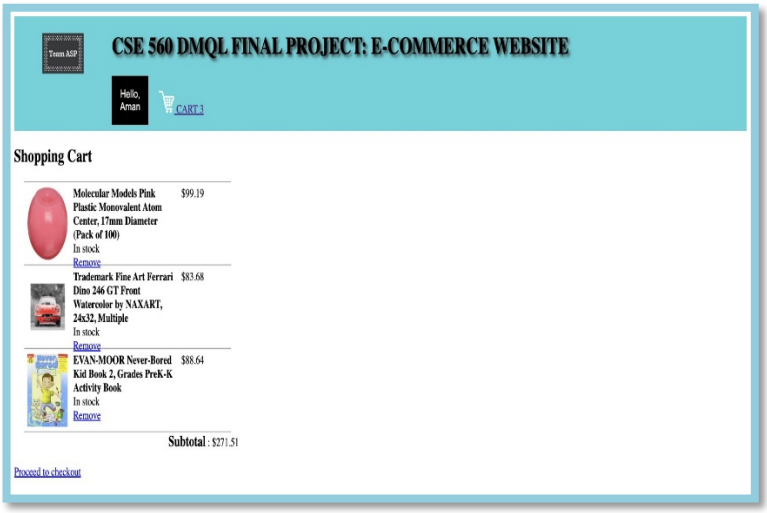


Products by Category: The user can view the products for each category and choose accordingly.

Product Details: This page consists of details of the product selected



Order page: This page consists of the items the user wanted to order and present on the card. It also displays the total amount of the products.

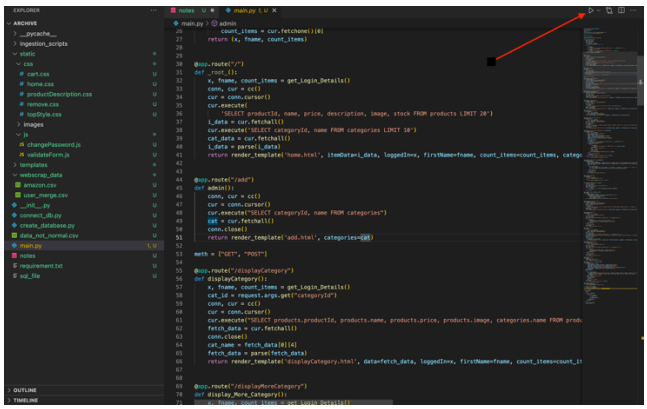


Register Page: The user can register himself with his credentials and login the next time he visits the website

Edit Profile: The user can edit his/her profile using the edit profile page and it will store the latest details of the user in the database.

Way to run the project:

1. Open VS code and open the project directory.
2. Click on main.py and click on the top left to run the code.



3. A terminal will pop in the below just run the below command to install all the requirements.

   pip install -r requirement.txt
   or
   pip3 install -r requirement.txt

4. After installing all the requirements, if you used pip just write in the terminal the below code:
   python main.py
   If you used pip3 while installing all the requirements, then use the below code in the terminal:
   python3 main.py

5. Click ctrl + left mouse on the link (http://127.0.0.1:5000/) to redirect to the website.

Team Members Contribution:

We have successfully completed the project with equal contribution from each teammate.

Team Member	Project + Report	Contribution
Aman Prakash	• Data scrapping • Data preprocessing • Create tables • Data Normalization and Database Implementation • Data Ingestion • Complex queries visualization • Query execution analysis • Website Implementation	• 33.3% • 33.3% • 33.3% • 33.3% • 33.3% • 33.3% • 33.3% • 33.3%
Serath Chandra Nutakki	• Data scrapping • Data preprocessing • Create tables • Data Normalization and Database Implementation • Data Ingestion • Complex queries visualization • Query execution analysis • Website Implementation	• 33.3% • 33.3% • 33.3% • 33.3% • 33.3% • 33.3% • 33.3% • 33.3%
Prashant Upadhyay	• Data scrapping • Data preprocessing • Create tables • Data Normalization and Database Implementation • Data Ingestion • Complex queries visualization • Query execution analysis • Website Implementation	• 33.3% • 33.3% • 33.3% • 33.3% • 33.3% • 33.3% • 33.3% • 33.3%

For convenient access of files please projects github repo by accessing the below link:-

https://github.com/amanprak/cse560_final_project