ORDERS ON A GO - AN E COMMERCE APPLICATION

Raghu Vamsi Mullapudi
raghuvam
50469258
raghuvam@buffalo.edu

Vaishnavi Koyyada
vkoyyada
50468340
vkoyyada@buffalo.edu

Naveen Bhogavalli
nbhogava
50471010
nbhogava@buffalo.edu

I. Problem Statement

The purchase or selling of products by the use of a website or an application that collects information(data) on various aspects like payment details, shipping address details etc is known as an e-commerce platform. Our platform **'Orders on a GO'** can be reached through the internet and can be accessed easily by a user. This project is on an e-commerce database which holds all the information about their business i.e the database is designed to store all the data related to customers, sellers, orders, delivery addresses etc. The data can be easily stored into the database whenever any customer purchases a product, or if any new seller needs to be added or any details regarding any of the order needs to be stored instead of adding it manually on any paper or to an excel file which is very tedious and is also very time consuming to check any previous details and update every time. By using SQL we are storing this data in Database which can be used to provide any required statistics. With this the end users can also easily see the history of their orders,check the status of the order and all these can be done without contacting any external person. We also have a category name Translation feature also which allows us to use different languages so that every user can easily understand the category names.

II. Why do we need a database instead of an excel file?

We need a Database instead of an Excel file for the following reasons. In case of Data sharing, the Database provides concurrent access to its users with certain specifications by restricting users to perform certain operations like reading or updating the data from specific tables. It provides security, so it is hard to hack.The excel files are locally present on some device and in order to access data by a specific user we need to send them the whole file, but in the case of Database it can be accessed with an IP address. And, maintaining multiple duplicate copies of the same excel sheets may lead to inconsistent data. If a data is changed in one of the excel sheets then we have to manually change it in all its duplicates.In case of a Database, we can handle and change multiple copies of data through clustering. If we use excel sheets then all the excel file data is stored in the main memory which might not be sufficient to store huge amounts of data.Further, we can't always load the real time data where the data will be updated from time to time which makes it tough to use an excel sheet.The database also provides many additional functionalities like functions, views,triggers etc.

III. Target User

Our application 'Orders on a GO' can be used by any customer to browse the list of all the available products, place an order or to check the status of the previously placed order. The administrator or the owner of the application will administer the database and only they can add or remove any data from the database.

IV. Real-Life Scenario

The customer who wants to purchase a product online can visit the application and can check all the available products in each category. With the Language Translation feature they can also change the language so that they can view the category names in the selected language. Once they have chosen the products they can place an order on the website by selecting the one of payment methods available. Once their order is confirmed they get the estimated delivery date. After the receiving of the order they can also review the order. The administrator can add any new sellers that they have collaborated with for business or can update the details of any of the existing ones.

V. E/R Diagram

There are 11 schemas in our E-Commerce Database Schema.

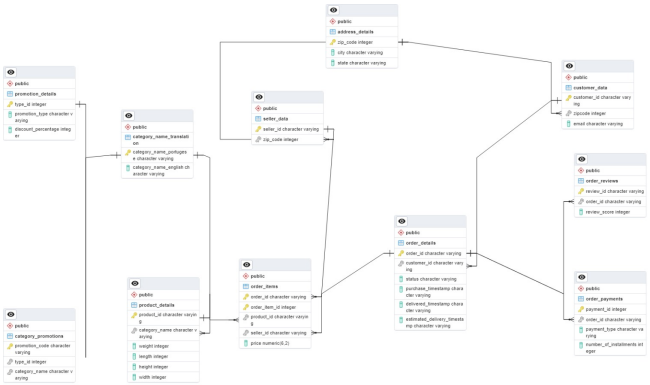

Fig. 1. er_diagram

address_detail : This table contains details about the zip code and the respective city and state details of the customers and seller.
category_name_translation : This table contains the category names in Portuguese and English.
customer_data : This table contains the details of all the customers which includes their ID's, email id's and zip code.
order_details : This table contains the details of all the orders which includes the ID of the order, the ID of the customer who placed the order, the status of the order and the timestamps of purchase, delivery etc.
order_items : This table contains the details of the ID of the order, order item id in the order,price of the item and details of the seller and the product.
order_payments : This table contains all the information related to the payment of a particular order. Payment details include the ID, type of payment, number of installments etc.
order_reviews : This table contains the details of the Review ID and the score for a particular order.
product_details : This table contains all the information related to the product which includes the product ID, category of the product, weight and the dimensions of the product.
seller_details : This table contains the ID of the seller and their zipcodes.

VI. Defining the List of Relations and their Attributes

TABLE I
ADDRESS_DETAILS

Name of the attribute	Datatype	Description of the Attribute
zip_code	integer	Zipcode of the customer and sellers
city	varchar	Name of the city
state	varchar	Name of the state

TABLE II
CATEGORY_NAME_TRANSLATION

Name of the attribute	Datatype	Description of the Attribute
category_name_portugese	varchar	Name of category in Portugese
category_name_english	varchar	Name of category in english

TABLE III
CUSTOMER_DATA

Name of the attribute	Datatype	Description of the Attribute
customer_id	varchar	unique ID for each customer
zipcode	integer	Zip code of the customer
email	varchar	Email ID of the customer

TABLE IV
ORDER_DETAILS

Name of the attribute	Datatype	Description of the Attribute
order_id	varchar	Unique ID of each order
customer_id	varchar	ID of the customer who placed the order
status	varchar	status of the order
purchase_timestamp	varchar	Time and date of purchase of the order
delievered_timestamp	varchar	Time and date of delivery of the order
estimated_delievery_timestamp	varchar	Time and date of estimated delivery of the order

TABLE V
ORDER_PAYMENTS

Name of the attribute	Datatype	Description of the Attribute
payment_id	integer	Unique payment ID for each Order
order_id	varchar	ID of the order
payment_type	varchar	Method of payment chosen by customer
number_of_installements	integer	Number of installements chosen by Customer

TABLE VI
ORDER_ITEMS

Name of the attribute	Datatype	Description of the Attribute
product_id	varchar	ID of product in the Order
order_id	varchar	ID of the order
order_item_id	integer	Sequential no of items in a particular order
seller_id	varchar	ID of seller of product in the order
price	numeric(6,2)	price of item in the order

TABLE VII
ORDER_REVIEWS

Name of the attribute	Datatype	Description of the Attribute
review_id	varchar	Unique Review ID of each Order
order_id	varchar	ID of the order
review_score	integer	review score given by each customer for each order

TABLE VIII
PRODUCT_DETAILS

Name of the attribute	Datatype	Description of the Attribute
product_id	varchar	Unique ID for each product
category_name	varchar	Name of the category of product in portugese
weight	integer	weight of product in grams
length	integer	length of the product in cms
height	integer	height of the product in cms
width	integer	width of the product in cms

TABLE IX
SELLER_DATA

Name of the attribute	Datatype	Description of the Attribute
seller_id	varchar	Unique ID for each seller
zip_code	integer	zip code of the seller

TABLE X
PROMOTION_DETAILS

Name of the attribute	Datatype	Description of the Attribute
type_id	integer	Unique ID for each promotion type
promotion_type	varchar	type of the promotion
discount_percentage	integer	percentage of discount

TABLE XI
CATEGORY_PROMOTIONS

Name of the attribute	Datatype	Description of the Attribute
promotion_code	varchar	Unique code for each promotion
type_Id	integer	Id for promotion type
category_name	varchar	name of the category in portugese

VII. DATABASE CONSTRAINTS

A. Primary Keys

address_details - zip_code
We have chosen zip_code as Primary Key as it is unique for each tuple in the table and for every city and state in the relation
category_name_translation - category_name_portugese
We have chosen category_name_portugese as Primary Key as it is unique. We can also chose category_name_english as Primary Key as it is unique and it doesn't contain any NULL values. But we have chosen the category_name_portugese as the Primary Key to establish the connection between this table and product_details table as we can consider category_name_portugese as the Foreign Key in the product_details table.
customer_data - customer_id
We have chosen customer_id as the Primary Key as it is unique for each tuple in the table. We can also chose email as Primary Key as it is also unique and it doesn't contain any NULL values. But we have chosen the customer_id as the Primary Key to establish the connection between this table and order_details table as we can consider customer_id as the Foreign Key in the order_details table.
order_details - order_id
We have chosen order_id as the Primary Key as it is unique for each of the order in the table.
order_items - (order_id,order_item_id)
We have chosen (order_id, order_item_id) as the primary key as they together uniquely identify each tuple in the table.]]
order_payments - payment_id
We have chosen payment_id as the Primary Key as it uniquely identifies every payment made for an order.
order_reviews - review_id
We have chosen review_id as the primary key as it uniquely identifies the review for every tuple in the table. Order_id can also be chosen as the Primary Key but we have chosen review_id as the Primary Key as we want to chose order_id as the Foreign Key to establish the relation between the order_reviews table and order_details table.
product_details - product_id
We have chosen product_id as the Primary Key as it uniquely identifies each product in the table.
seller_data - seller_id
We have chosen seller_id as the Primary Key as it uniquely identifies each tuple in the table

B. Foreign Keys

We have chosen the below as Foreign keys as they are the Primary Keys in another table and we can establish a relation between the two tables.
customer_data - zipcode
zipcode references zip_code in address_details table
order_details - customer_id
customer_id references customer_id in customer_data table.
order_items - product_id, order_id, seller_id
product_id references product_id in product_details relation.
order_id references order_id in order_details relation.
seller_id references seller_id in seller_data relation.
order_payments - order_id
order_id references order_id in order_details relation.
order_reviews - order_id
order_id references order_id in order_details relation.
product_details - category_name
category_name references category_name_portugese in category_name_translation table.
seller_data - zip_code zipcode references zip_code in address_details table

C. Default or NULL or NOT NULL Constraint

Category_name_translation - Category_name_english - Default
Default value is miscellaneous.
customer_data - email - NOT NULL
order_details - purchase_timestamp - NOT NULL

order_details - estimated_delivery_timestamp - NOT NULL

D. Actions taken on any foreign key when the primary key(that the foreign key refer to)is deleted (e.g., no action, delete cascade, set null, set default)

zipcode in customer_data references zip_code in address_details - set null
zipcode in seller_data references zip_code in address_details - set null
category_name in product_details references category_name_portugese in category_name_translation - set null
order_id in order_reviews references order_id in order_details relation.- delete cascade
order_id in order_payments references order_id in order_details relation - delete cascade
product_id in order_items references product_id in product_details relation - delete cascade
order_id in order_items references order_id in order_details relation. - delete cascade
seller_id in order_items references seller_id in seller_data relation - set null
customer_id in order_details references customer_id in customer_data table.- delete cascade

VIII. DESIGN THEORY

Below are the conditions for a schema to satisfy the respective normal forms
1NF:A table is in 1NF if all attributes of table has atomic values
2NF:A table is in 2NF if it is in 1NF and it should not have any partial dependencies
3NF:A table is in 3NF if it in 2NF and either $A \rightarrow B$, is a trivial FD or A is super key or each attribute A in Beta-alpha is contained in candidate key of R
BCNF: A table is in BCNF if it is in 3NF and for each FD, $A \rightarrow B$ either A has to be a superkey or $A \rightarrow B$ is a trivial Functional dependency

customer_data
$customer\_id \rightarrow zipcode$, $customer\_id \rightarrow email$
address_details
$zipcode \rightarrow city$, $zipcode \rightarrow state$
seller_data
$seller\_id \rightarrow zipcode$
order_details
$order\_id \rightarrow customer\_id$,$order\_id \rightarrow states$, $order\_id \rightarrow purchase\_timestamp$, $order\_id \rightarrow delivered\_timestamp$, $order\_id \rightarrow estimated\_delivery\_timestamp$
order_items
$order\_id, order\_item\_id \rightarrow product\_id$
$order\_id, order\_item\_id \rightarrow seller\_id$
$order\_id, order\_item\_id \rightarrow price$
order_payments
$payment\_id \rightarrow order\_id$ $payment\_id \rightarrow payment\_type$ $payment\_id \rightarrow number\_of\_installments$
order_reviews
$review\_id \rightarrow order\_id$, $review\_id \rightarrow review\_score$
product_details
$product\_id \rightarrow category\_name$ $product\_id \rightarrow weight$ $product\_id \rightarrow length$ $product\_id \rightarrow height$ $product\_id \rightarrow width$
category_name_translation
$category\_name\_portugese \rightarrow category\_name\_english$
category_promotions
$promotion\_code \rightarrow type\_id$ $promotion\_code \rightarrow category\_name$
promotion_details
$type\_id \rightarrow promotion\_type$ $type\_id \rightarrow discount\_percentage$

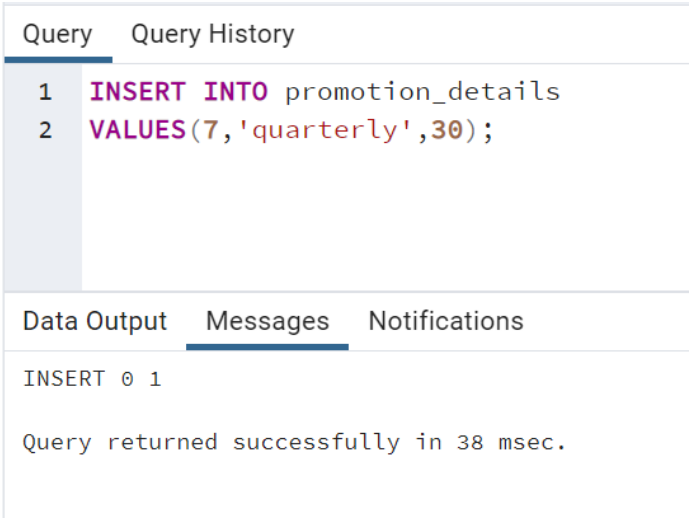

Fig. 2. insert

Query Query History

```
SELECT * FROM public.promotion_details
ORDER BY type_id ASC
```

Data Output Messages Notifications

	type_id [PK] integer	promotion_type character varying	discount_percentage integer
1	1	festive	60
2	2	seasonal	30
3	3	yearly	50
4	4	monthly	25
5	5	weekly	15
6	6	daily	10
7	7	quarterly	30

Fig. 3. insert_after

Query Query History

```
INSERT INTO customer_data(customer_id,zipcode,email)
VALUES ('00000000000000000000000000000001',4841,'abcd@gmail.com');
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 79 msec.
```

Fig. 4. insert_1

Query Query History

```
SELECT * FROM public.customer_data
ORDER BY customer_id ASC
```

Data Output Messages Notifications

	customer_id [PK] character varying	zipcode integer	email character varying
1	00000000000000000000000000000001	4841	abcd@gmail.com
2	00012a2ce6f8dcda20d059ce98491703	6273	annoyingMelanie@aol.com
3	000161a058600d5901f007fab4c27140	35550	oddClaudia3@yahoo.in
4	0001fd6190edaaf884bcaf3d49edf079	29830	lightTiffany@terra.com.br
5	0002414f95344307404f0ace7a26f1d5	39664	Evelyncrazy@live.co.uk
6	000379cdec625522490c315e70c7a9fb	4841	Marvinuptight@mail.ru
7	0004164d20a9e969af783496f3408652	13272	soreJacquelyn41@skynet.be

Fig. 5. insert_after_1

Query Query History

```
DELETE FROM customer_data
WHERE customer_id='00000000000000000000000000abcd1';
```

Data Output Messages Notifications

```
DELETE 1

Query returned successfully in 66 msec.
```

Fig. 6. delete

Query Query History

```
SELECT * FROM public.customer_data
ORDER BY customer_id ASC
```

Data Output Messages Notifications

	customer_id [PK] character varying	zipcode integer	email character varying
1	00012a2ce6f8dcda20d059ce98491703	6273	annoyingMelanie@aol.com
2	000161a058600d5901f007fab4c27140	35550	oddClaudia3@yahoo.in
3	0001fd6190edaaf884bcaf3d49edf079	29830	lightTiffany@terra.com.br
4	0002414f95344307404f0ace7a26f1d5	39664	Evelyncrazy@live.co.uk
5	000379cdec625522490c315e70c7a9fb	4841	Marvinuptight@mail.ru

Fig. 7. delete_after

Query Query History

```
DELETE FROM promotion_details
WHERE promotion_type='quarterly';
```

Data Output Messages Notifications

```
DELETE 1

Query returned successfully in 76 msec.
```

Fig. 8. delete_1

Query Query History

```
SELECT * FROM public.promotion_details
ORDER BY type_id ASC
```

Data Output Messages Notifications

	type_id [PK] integer	promotion_type character varying	discount_percentage integer
1	1	festive	60
2	2	seasonal	30
3	3	yearly	50
4	4	monthly	25
5	5	weekly	15
6	6	daily	10

Fig. 9. delete_after_1

Query Query History

```
SELECT * FROM public.promotion_details
ORDER BY type_id ASC
```

Data Output Messages Notifications

	type_id [PK] integer	promotion_type character varying	discount_percentage integer
1	1	festive	60
2	2	seasonal	30
3	3	yearly	50
4	4	monthly	25
5	5	weekly	15
6	6	daily	10
7	7	quarterly	30

Fig. 12. update_before

Query Query History

```
UPDATE promotion_details
SET discount_percentage=20
WHERE type_id=7;
```

Data Output Messages Notifications

```
UPDATE 1

Query returned successfully in 74 msec.
```

Fig. 10. update

Query Query History

```
UPDATE customer_data
SET customer_id='00000000000000000000000000abcd1',zipcode=29830,email='abcdef@gmail.com'
WHERE customer_id='00000000000000000000000000000001';
```

Data Output Messages Notifications

```
UPDATE 1

Query returned successfully in 88 msec.
```

Fig. 13. update_1

Query Query History

```
SELECT * FROM public.promotion_details
ORDER BY type_id ASC
```

Data Output Messages Notifications

	type_id [PK] integer	promotion_type character varying	discount_percentage integer
1	1	festive	60
2	2	seasonal	30
3	3	yearly	50
4	4	monthly	25
5	5	weekly	15
6	6	daily	10
7	7	quarterly	20

Fig. 11. update_after

Query Query History

```
SELECT * FROM public.customer_data
ORDER BY customer_id ASC
```

Data Output Messages Notifications

	customer_id [PK] character varying	zipcode integer	email character varying
1	00000000000000000000000000abcd1	29830	abcdef@gmail.com
2	00012a2ce6f8dcda20d059ce98491703	6273	annoyingMelanie@aol.com
3	000161a058600d5901f007fab4c27140	35550	oddClaudia3@yahoo.in
4	0001fd6190edaaf884bcaf3d49edf079	29830	lightTiffany@terra.com.br
5	0002414f95344307404f0ace7a26f1d5	39664	Evelyncrazy@live.co.uk
6	000379cdec625522490c315e70c7a9fb	4841	Marvinuptight@mail.ru

Fig. 14. update_after_1

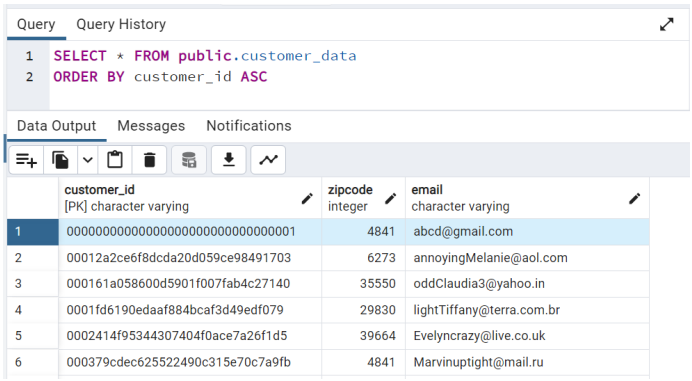

Fig. 15. update_before_1

IX. Some Sample SQL Queries:

1.SQL Query to find the email id's of all the customers who have yahoo or hotmail emails.

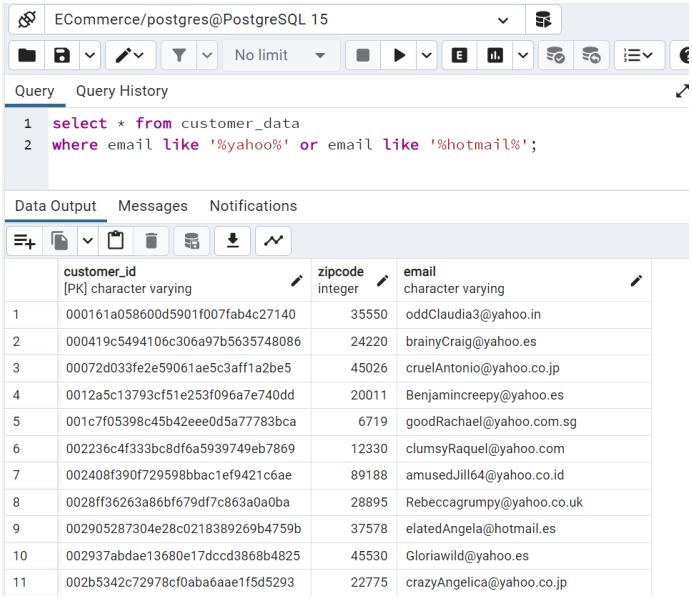

Fig. 16. query_1

2.SQL Query to find the count of the number of orders placed by each customer in ascending order of their count of number of order

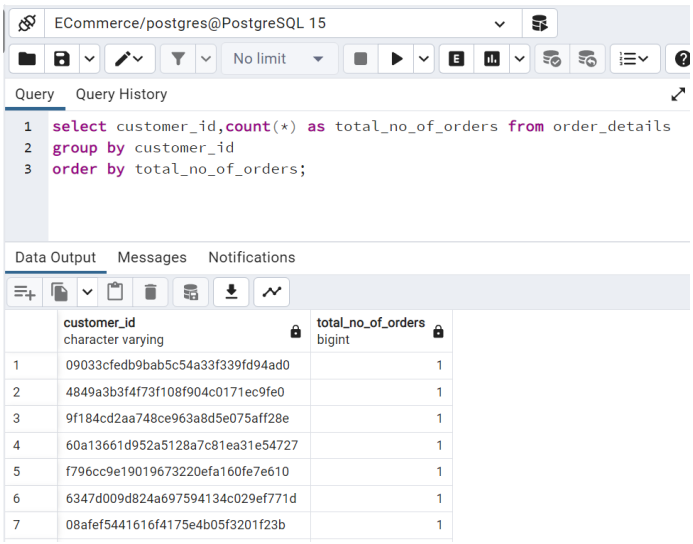

Fig. 17. query_2

3.SQL Query to decrease the review score by 1 for the orders of all the customers having zip code 20540.

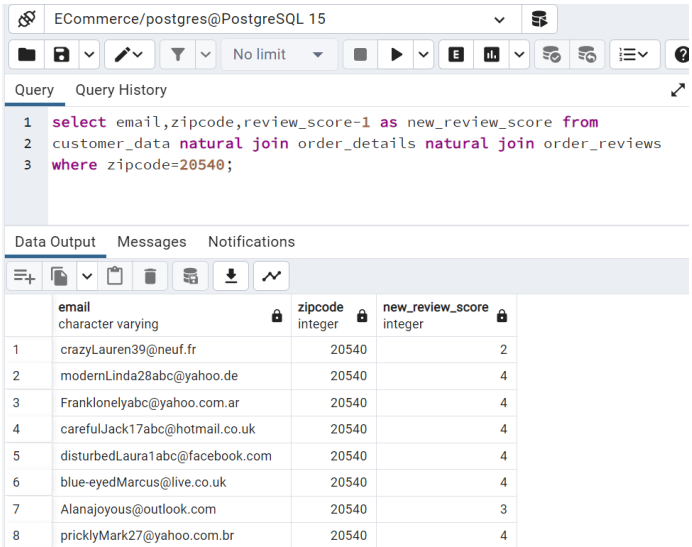

Fig. 18. query_3

4.SQL Query to find the category names and product id's of all the orders which have a review score of 5

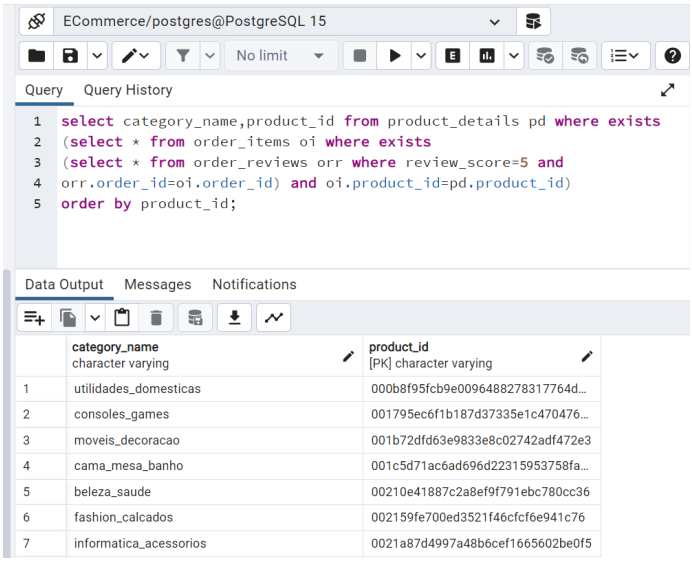

Fig. 19. query_4

complex Query

A query that gives the order id,customer id,purchase timestamp,product id,category name,promotion code of all the orders placed on the last recent day

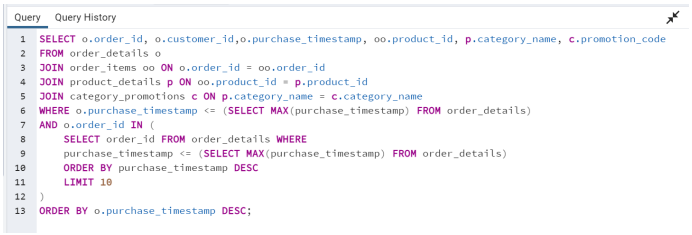

Fig. 20. complex

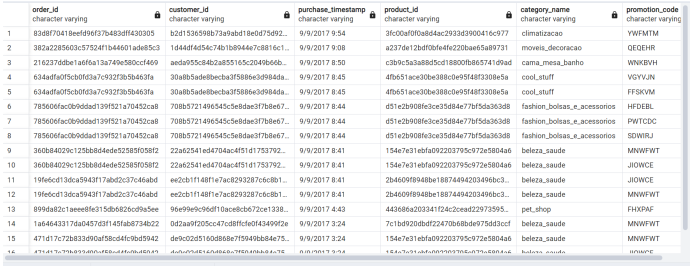

Fig. 21. complex_1

A Query to find the Order id,Review score,Email and Zipcode of all the customers having zip code 29830 having best review.

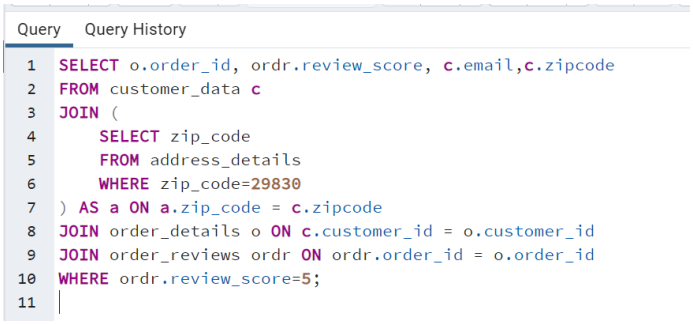

Fig. 22. complex_2

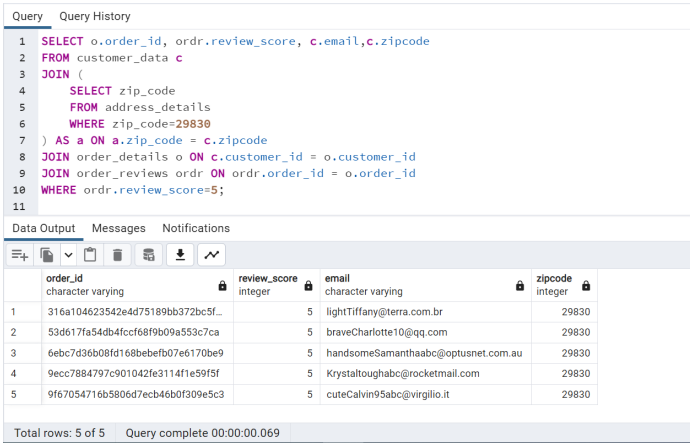

Fig. 23. complex_2_combined

Indexing queries

1. A Query to display the email and count of emails of all the customers who have an email

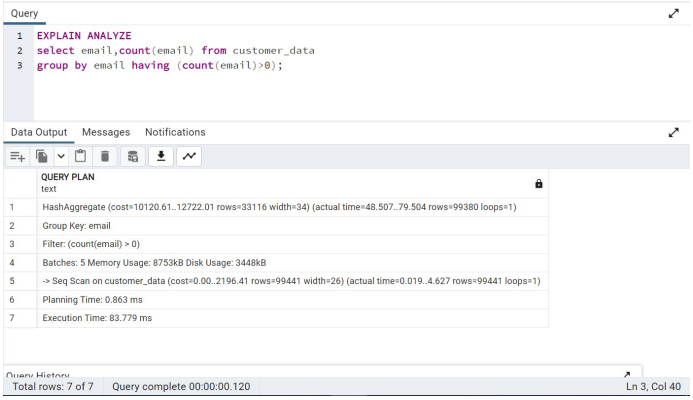

Fig. 24. before_indexing

```
CREATE INDEX i_index on customer_data(email);

EXPLAIN ANALYZE
select email,count(email) as unique_email from customer_data
group by email having (count(email)>0);
```

	QUERY PLAN text
1	GroupAggregate (cost=0.42..5571.09 rows=33116 width=34) (actual time=0.469..22.424 rows=99380 loops=1)
2	Group Key: email
3	Filter: (count(email) > 0)
4	-> Index Only Scan using i_index on customer_data (cost=0.42..3832.03 rows=99441 width=26) (actual time=0.462..7.602 rows=99441 loops=…
5	Heap Fetches: 336
6	Planning Time: 0.949 ms
7	Execution Time: 23.947 ms

Total rows: 7 of 7 | Query complete 00:00:00.447 | Ln 5, Col 40

Fig. 25. after_indexing

2. *A query to find to the order id and status of all the delivered on*

```
EXPLAIN ANALYZE SELECT od.order_id,od.status FROM order_details od
WHERE od.order_id IN (SELECT order_id FROM order_items) AND od.status = 'delivered';
```

	QUERY PLAN text
1	Hash Join (cost=3412.89..6144.30 rows=55006 width=42) (actual time=39.525..55.853 rows=59997 loops=1)
2	Hash Cond: ((od.order_id)::text = (order_items.order_id)::text)
3	-> Seq Scan on order_details od (cost=0.00..1962.05 rows=59968 width=42) (actual time=0.014..4.892 rows=59999 loops=1)
4	Filter: ((status)::text = 'delivered'::text)
5	Rows Removed by Filter: 1845
6	-> Hash (cost=2703.81..2703.81 rows=56727 width=33) (actual time=39.327..39.329 rows=61345 loops=1)
7	Buckets: 65536 Batches: 1 Memory Usage: 4406kB
8	-> HashAggregate (cost=2136.54..2703.81 rows=56727 width=33) (actual time=21.065..32.826 rows=61345 loops=1)
9	Group Key: (order_items.order_id)::text
10	Batches: 5 Memory Usage: 8241kB Disk Usage: 1720kB
11	-> Seq Scan on order_items (cost=0.00..1961.23 rows=70123 width=33) (actual time=0.006..5.818 rows=70123 loops=1)
12	Planning Time: 0.342 ms
13	Execution Time: 59.544 ms

Total rows: 13 of 13 | Query complete 00:00:00.106 | Ln 1, Col 17

Fig. 26. before_2_indexing

```
EXPLAIN ANALYZE
SELECT od.order_id,od.status FROM
order_details od NATURAL JOIN order_items oi
WHERE od.status = 'delivered';
```

	QUERY PLAN text
1	Hash Join (cost=2711.65..4856.96 rows=67996 width=42) (actual time=12.305..27.175 rows=68593 loops=1)
2	Hash Cond: ((oi.order_id)::text = (od.order_id)::text)
3	-> Seq Scan on order_items oi (cost=0.00..1961.23 rows=70123 width=33) (actual time=0.009..2.551 rows=70123 loops=1)
4	-> Hash (cost=1962.05..1962.05 rows=59968 width=42) (actual time=12.110..12.111 rows=59999 loops=1)
5	Buckets: 65536 Batches: 1 Memory Usage: 4907kB
6	-> Seq Scan on order_details od (cost=0.00..1962.05 rows=59968 width=42) (actual time=0.007..5.763 rows=59999 loops=1)
7	Filter: ((status)::text = 'delivered'::text)
8	Rows Removed by Filter: 1845
9	Planning Time: 0.334 ms
10	Execution Time: 29.041 ms

Fig. 27. after_2_indexing

3. A query to calculate the total number of orders, total customers, and average review score for each category in each state.

```
EXPLAIN ANALYZE SELECT ad.state,pd.category_name,COUNT(DISTINCT od.order_id) AS total_orders,
COUNT(DISTINCT cd.customer_id) AS total_customers,AVG(ordr.review_score) AS average_review_score
FROM address_details ad JOIN customer_data cd ON ad.zip_code = cd.zipcode
JOIN order_details od ON cd.customer_id = od.customer_id JOIN order_items oi ON od.order_id = oi.order_id
JOIN product_details pd ON oi.product_id = pd.product_id JOIN order_reviews ordr ON od.order_id = ordr.order_id
WHERE od.status = 'delivered' GROUP BY ad.state, pd.category_name ORDER BY total_orders DESC, average_review_score DESC;
```

	QUERY PLAN text
30	-> Seq Scan on customer_data cd (cost=0.00..2196.44 rows=99444 width=37) (actual time=0.021..6.069 rows=99444 loops=1)
31	-> Hash (cost=243.78..243.78 rows=15078 width=7) (actual time=2.149..2.150 rows=15078 loops=1)
32	Buckets: 16384 Batches: 1 Memory Usage: 717kB
33	-> Seq Scan on address_details ad (cost=0.00..243.78 rows=15078 width=7) (actual time=0.038..0.928 rows=15078 loops=1)
34	-> Hash (cost=720.52..720.52 rows=32952 width=48) (actual time=7.049..7.049 rows=32952 loops=1)
35	Buckets: 65536 Batches: 1 Memory Usage: 3109kB
36	-> Seq Scan on product_details pd (cost=0.00..720.52 rows=32952 width=48) (actual time=0.013..2.512 rows=32952 loops=1)
37	Planning Time: 2.129 ms
38	Execution Time: 529.898 ms

Total rows: 38 of 38 | Query complete 00:00:00.546 | Ln 5, Col 81

Fig. 28. before_indexing_3

	QUERY PLAN text
1	Sort (cost=23252.15..23257.14 rows=1998 width=66) (actual time=525.528..525.568 rows=1277 loops=1)
2	Sort Key: (count(DISTINCT od.order_id)) DESC, (avg(ordr.review_score)) DESC
3	Sort Method: quicksort Memory: 163kB
4	-> GroupAggregate (cost=22105.34..23142.62 rows=1998 width=66) (actual time=320.241..524.869 rows=1277 loops=1)
5	Group Key: ad.state, pd.category_name
6	-> Sort (cost=22105.34..22274.05 rows=67487 width=88) (actual time=320.177..340.064 rows=68118 loops=1)
7	Sort Key: ad.state, pd.category_name
8	Sort Method: external merge Disk: 6664kB
9	-> Hash Join (cost=9854.89..13461.59 rows=67487 width=88) (actual time=56.520..191.645 rows=68118 loops=1)
10	Hash Cond: ((oi.product_id)::text = (pd.product_id)::text)
11	-> Hash Join (cost=8722.47..12152.00 rows=67487 width=106) (actual time=49.285..154.468 rows=68118 loops=1)
12	Hash Cond: (cd.zipcode = ad.zip_code)
13	-> Hash Join (cost=8290.21..11542.55 rows=67487 width=107) (actual time=47.067..135.002 rows=68118 loops=1)
14	Hash Cond: ((od.customer_id)::text = (cd.customer_id)::text)
15	-> Hash Join (cost=4850.72..7925.89 rows=67487 width=103) (actual time=23.786..84.615 rows=68118 loops=1)
16	Hash Cond: ((od.order_id)::text = (ordr.order_id)::text)

Total rows: 38 of 38 | Query complete 00:00:00.546

Fig. 29. before_indexing_3.1

	QUERY PLAN text
17	-> Hash Join (cost=2711.65..4856.96 rows=67996 width=132) (actual time=12.287..48.489 rows=68593 loops=1)
18	Hash Cond: ((oi.order_id)::text = (od.order_id)::text)
19	-> Seq Scan on order_items oi (cost=0.00..1961.23 rows=70123 width=66) (actual time=0.009..3.786 rows=70123 loops=1)
20	-> Hash (cost=1962.05..1962.05 rows=59968 width=66) (actual time=12.126..12.128 rows=59999 loops=1)
21	Buckets: 65536 Batches: 1 Memory Usage: 6255kB
22	-> Seq Scan on order_details od (cost=0.00..1962.05 rows=59968 width=66) (actual time=0.008..5.638 rows=59999 loops=1)
23	Filter: ((status)::text = 'delivered'::text)
24	Rows Removed by Filter: 1845
25	-> Hash (cost=1371.81..1371.81 rows=61381 width=37) (actual time=11.320..11.321 rows=61381 loops=1)
26	Buckets: 65536 Batches: 1 Memory Usage: 4649kB
27	-> Seq Scan on order_reviews ordr (cost=0.00..1371.81 rows=61381 width=37) (actual time=0.017..4.413 rows=61381 loops=1)
28	-> Hash (cost=2196.44..2196.44 rows=99444 width=37) (actual time=16.590..16.590 rows=99444 loops=1)
29	Buckets: 131072 Batches: 1 Memory Usage: 8017kB
30	-> Seq Scan on customer_data cd (cost=0.00..2196.44 rows=99444 width=37) (actual time=0.021..6.069 rows=99444 loops=1)
31	-> Hash (cost=243.78..243.78 rows=15078 width=7) (actual time=2.149..2.150 rows=15078 loops=1)
32	Buckets: 16384 Batches: 1 Memory Usage: 717kB

Total rows: 38 of 38 | Query complete 00:00:00.546

Fig. 30. before_indexing_3.2

32	Buckets: 16384 Batches: 1 Memory Usage: 717kB
33	-> Seq Scan on address_details ad (cost=0.00..243.78 rows=15078 width=7) (actual time=0.038..0.928 rows=15078 loops=1)
34	-> Hash (cost=720.52..720.52 rows=32952 width=48) (actual time=7.049..7.049 rows=32952 loops=1)
35	Buckets: 65536 Batches: 1 Memory Usage: 3109kB
36	-> Seq Scan on product_details pd (cost=0.00..720.52 rows=32952 width=48) (actual time=0.013..2.512 rows=32952 loops=1)
37	Planning Time: 2.129 ms
38	Execution Time: 529.898 ms

Total rows: 38 of 38 | Query complete 00:00:00.546

Fig. 31. before_indexing_3.3

```
EXPLAIN ANALYZE
WITH order_summary AS (SELECT od.order_id,cd.zipcode,pd.category_name,ordr.review_score
FROM order_details od JOIN customer_data cd ON od.customer_id = cd.customer_id
JOIN order_items oi ON od.order_id = oi.order_id JOIN product_details pd ON oi.product_id = pd.product_id
JOIN order_reviews ordr ON od.order_id = ordr.order_id WHERE od.status = 'delivered'),
state_category_summary AS (SELECT ad.state,os.category_name,COUNT(os.order_id) AS total_orders,
COUNT(DISTINCT os.zipcode) AS total_customers,AVG(os.review_score) AS average_review_score
FROM order_summary os JOIN address_details ad ON os.zipcode = ad.zip_code GROUP BY ad.state, os.category_name)
SELECT state,category_name,total_orders,total_customers,average_review_score
FROM state_category_summary ORDER BY total_orders DESC, average_review_score DESC;
```

Data Output | Messages | Notifications

	QUERY PLAN text
33	-> Seq Scan on product_details pd (cost=0.00..720.52 rows=32952 width=48) (actual time=0.011..2.099 rows=32952 loops=1)
34	-> Hash (cost=243.78..243.78 rows=15078 width=7) (actual time=1.927..1.928 rows=15078 loops=1)
35	Buckets: 16384 Batches: 1 Memory Usage: 717kB
36	-> Seq Scan on address_details ad (cost=0.00..243.78 rows=15078 width=7) (actual time=0.018..0.832 rows=15078 loops=1)
37	Planning Time: 2.050 ms
38	Execution Time: 266.936 ms

Total rows: 38 of 38 | Query complete 00:00:00.282 | Ln 10, Col 83

Fig. 32. after_indexing_3

	QUERY PLAN text
1	Sort (cost=22579.13..22584.12 rows=1998 width=66) (actual time=263.175..263.210 rows=1277 loops=1)
2	Sort Key: (count(od.order_id)) DESC, (avg(ordr.review_score)) DESC
3	Sort Method: quicksort Memory: 163kB
4	-> GroupAggregate (cost=21412.34..22449.62 rows=1998 width=66) (actual time=229.726..262.652 rows=1277 loops=1)
5	Group Key: ad.state, pd.category_name
6	-> Sort (cost=21412.34..21581.05 rows=67487 width=59) (actual time=229.702..248.647 rows=68118 loops=1)
7	Sort Key: ad.state, pd.category_name
8	Sort Method: external merge Disk: 4752kB
9	-> Hash Join (cost=9854.89..13461.59 rows=67487 width=59) (actual time=47.790..137.601 rows=68118 loops=1)
10	Hash Cond: (cd.zipcode = ad.zip_code)
11	-> Hash Join (cost=9422.63..12852.14 rows=67487 width=56) (actual time=45.818..124.147 rows=68118 loops=1)
12	Hash Cond: ((oi.product_id)::text = (pd.product_id)::text)
13	-> Hash Join (cost=8290.21..11542.55 rows=67487 width=74) (actual time=40.032..102.169 rows=68118 loops=1)
14	Hash Cond: ((od.customer_id)::text = (cd.customer_id)::text)
15	-> Hash Join (cost=4850.72..7925.89 rows=67487 width=103) (actual time=22.544..63.365 rows=68118 loops=1)
16	Hash Cond: ((od.order_id)::text = (ordr.order_id)::text)

Fig. 33. after_indexing_3.1

	QUERY PLAN text
17	-> Hash Join (cost=2711.65..4856.96 rows=67996 width=132) (actual time=12.338..34.060 rows=68593 loops=1)
18	Hash Cond: ((oi.order_id)::text = (od.order_id)::text)
19	-> Seq Scan on order_items oi (cost=0.00..1961.23 rows=70123 width=66) (actual time=0.011..3.042 rows=70123 loops=1)
20	-> Hash (cost=1962.05..1962.05 rows=59968 width=66) (actual time=12.177..12.178 rows=59999 loops=1)
21	Buckets: 65536 Batches: 1 Memory Usage: 6255kB
22	-> Seq Scan on order_details od (cost=0.00..1962.05 rows=59968 width=66) (actual time=0.010..5.622 rows=59999 loops=1)
23	Filter: ((status)::text = 'delivered'::text)
24	Rows Removed by Filter: 1845
25	-> Hash (cost=1371.81..1371.81 rows=61381 width=37) (actual time=10.039..10.040 rows=61381 loops=1)
26	Buckets: 65536 Batches: 1 Memory Usage: 4649kB
27	-> Seq Scan on order_reviews ordr (cost=0.00..1371.81 rows=61381 width=37) (actual time=0.015..4.176 rows=61381 loops=1)
28	-> Hash (cost=2196.44..2196.44 rows=99444 width=37) (actual time=17.163..17.164 rows=99444 loops=1)
29	Buckets: 131072 Batches: 1 Memory Usage: 7725kB
30	-> Seq Scan on customer_data cd (cost=0.00..2196.44 rows=99444 width=37) (actual time=0.023..6.275 rows=99444 loops=1)
31	-> Hash (cost=720.52..720.52 rows=32952 width=48) (actual time=5.630..5.630 rows=32952 loops=1)
32	Buckets: 65536 Batches: 1 Memory Usage: 3109kB

Fig. 34. after_indexing_3.2

33	-> Seq Scan on product_details pd (cost=0.00..720.52 rows=32952 width=48) (actual time=0.011..2.099 rows=32952 loops=1)
34	-> Hash (cost=243.78..243.78 rows=15078 width=7) (actual time=1.927..1.928 rows=15078 loops=1)
35	Buckets: 16384 Batches: 1 Memory Usage: 717kB
36	-> Seq Scan on address_details ad (cost=0.00..243.78 rows=15078 width=7) (actual time=0.018..0.832 rows=15078 loops=1)
37	Planning Time: 2.050 ms
38	Execution Time: 266.936 ms

Fig. 35. after_indexing_3.3

33	-> Seq Scan on product_details pd (cost=0.00..720.52 rows=32952 width=48) (actual time=0.011..2.099 rows=32952 loops=1)
34	-> Hash (cost=243.78..243.78 rows=15078 width=7) (actual time=1.927..1.928 rows=15078 loops=1)
35	Buckets: 16384 Batches: 1 Memory Usage: 717kB
36	-> Seq Scan on address_details ad (cost=0.00..243.78 rows=15078 width=7) (actual time=0.018..0.832 rows=15078 loops=1)
37	Planning Time: 2.050 ms
38	Execution Time: 266.936 ms

Fig. 36. after_indexing_3.3

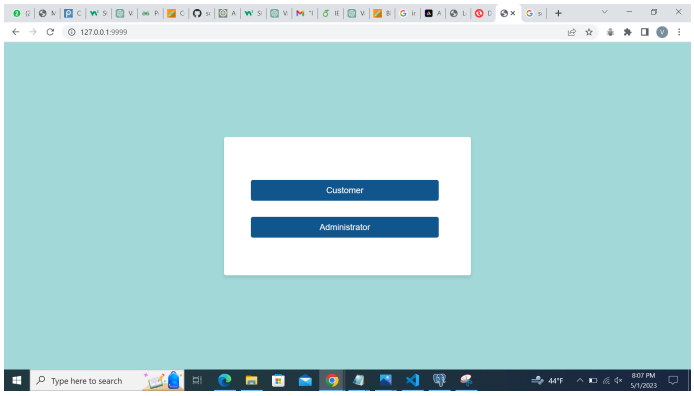

Fig. 37. initial login page

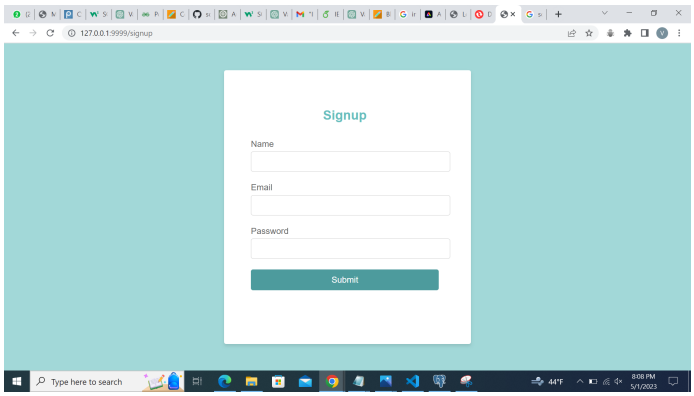

Fig. 38. Customer Signup Page

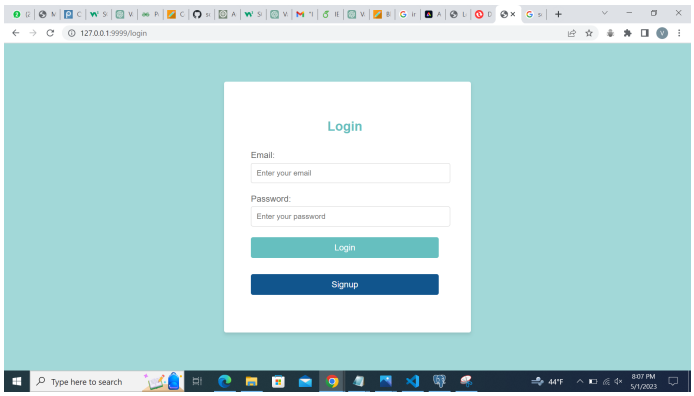

Fig. 39. Customer Login

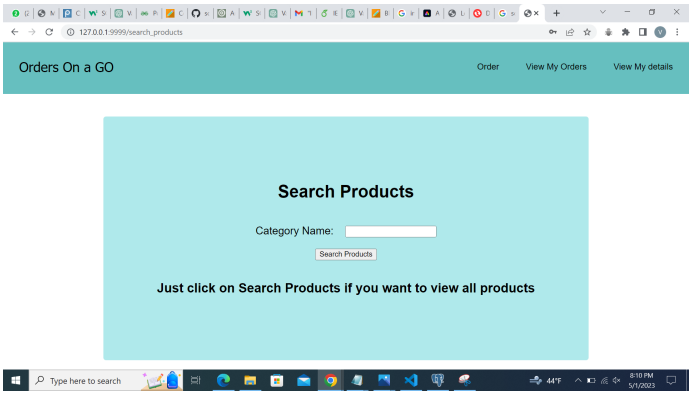

Fig. 40. Customer Home page

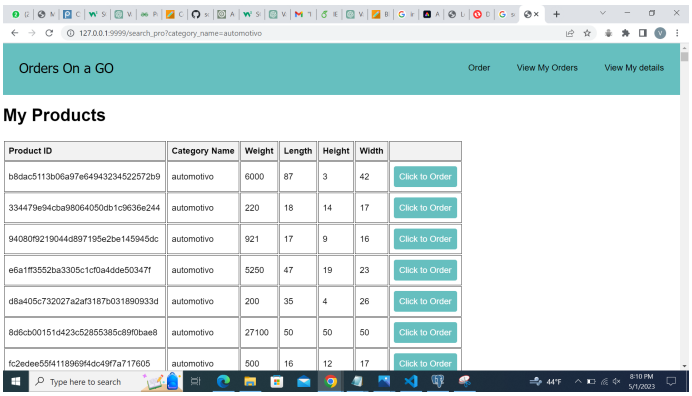

Fig. 41. Customer Order page

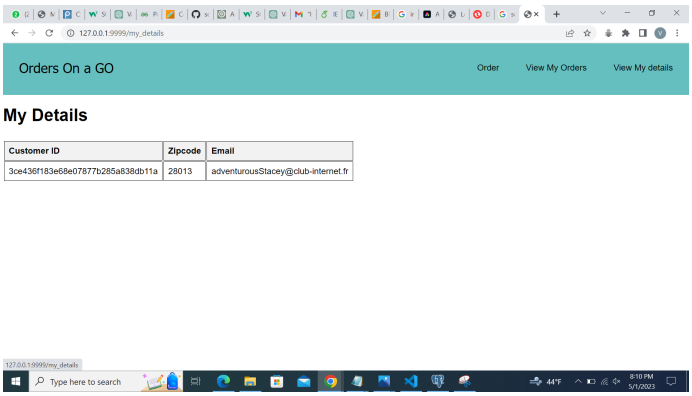

Fig. 42. Customer Details

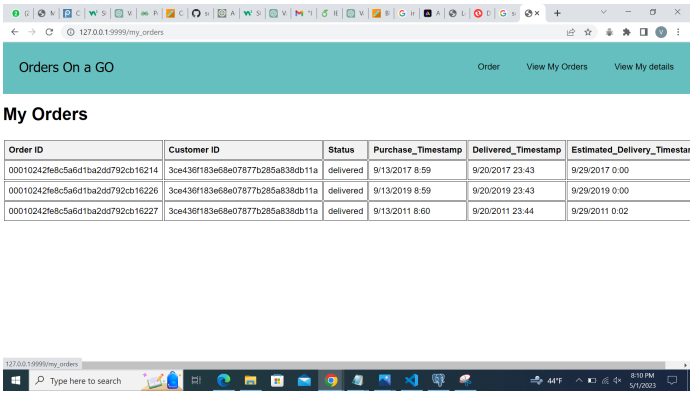

Fig. 43. Customers Orders

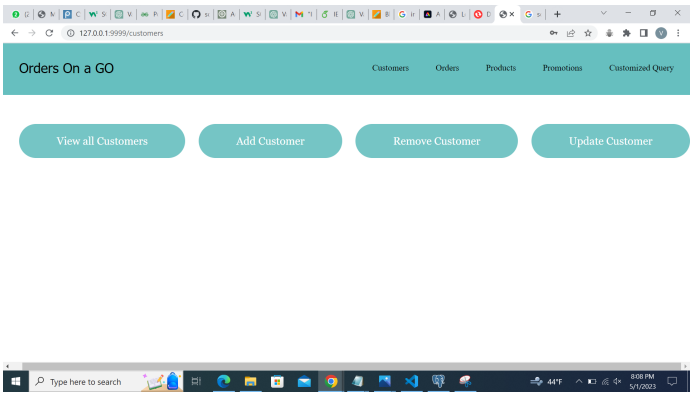

Fig. 44. Administrator Home Page

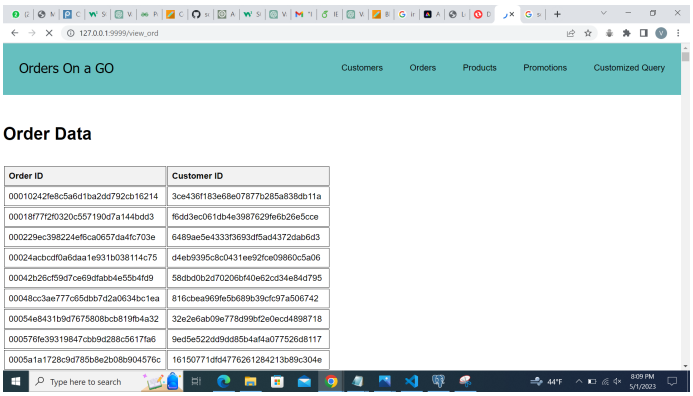

Fig. 45. Viewing all orders

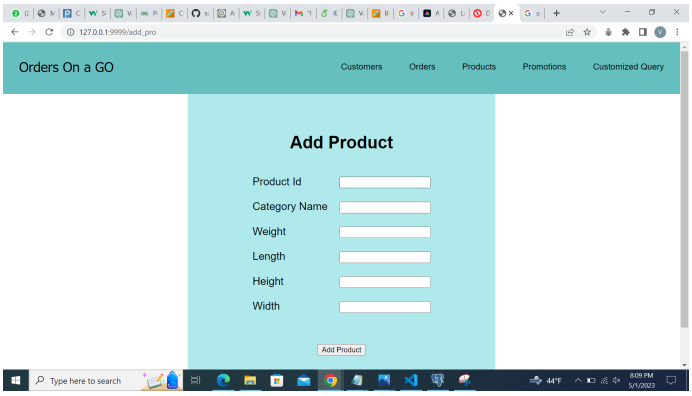

Fig. 46. Adding Products

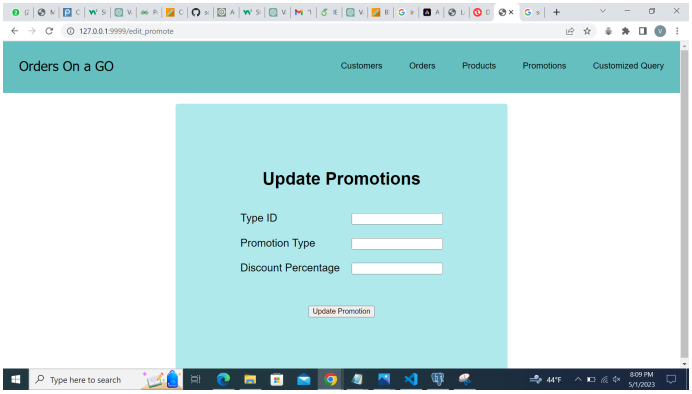

Fig. 47. Updating Promotions

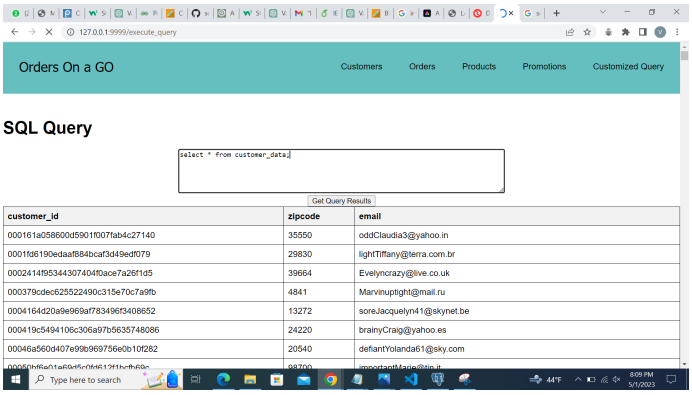

Fig. 48. Customized Query Execution

X. FUTURESCOPE

Many advanced modern ideas can also be implemented in this project due to vast amount of space provided by the E commerce platform, The major functionalities such as monitoring the shipment details, providing security features Moreover one can bid and sell the antique items, end user communication etc can be extended//