

# 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.

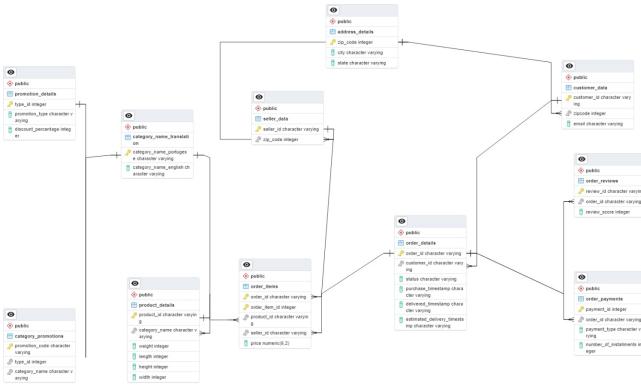


Fig. 1. er\_diagram

**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_portuguese	varchar	Name of category in Portuguese
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
delivered_timestamp	varchar	Time and date of delivery of the order
estimated_delivery_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_installments	integer	Number of instalments 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 portuguese
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 portuguese

## 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\_portuguese 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

payment\_id → order\_id payment\_id → payment\_type

payment\_id → number\_of\_installments

**order\_reviews**

review\_id → order\_id, review\_id → review\_score

**product\_details**

product\_id → category\_name product\_id → weight

product\_id → length product\_id → height

product\_id → width

**category\_name\_translation**

category\_name\_portuguese → category\_name\_english

**category\_promotions**

promotion\_code → type\_id promotion\_code →

category\_name

**promotion\_details**

type\_id → promotion\_type type\_id →

discount\_percentage

### 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 → zipcode, customer\_id → email

**address\_details**

zipcode → city, zipcode → state

**seller\_data**

seller\_id → zipcode

**order\_details**

order\_id → customer\_id, order\_id → states, order\_id → purchase\_timestamp, order\_id → delivered\_timestamp, order\_id → estimated\_delivery\_timestamp

**order\_items**

order\_id, order\_item\_id → product\_id

order\_id, order\_item\_id → seller\_id

order\_id, order\_item\_id → price

**order\_payments**

Query    Query History

1 **INSERT INTO** promotion\_details  
2 **VALUES**(7, 'quarterly', 30);

Data Output    Messages    Notifications

INSERT 0 1

Query returned successfully in 38 msec.

Fig. 2. insert

Query    Query History

```

1 SELECT * FROM public.promotion_details
2 ORDER BY type_id ASC

```

Data Output    Messages    Notifications

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

Fig. 3. insert\_after

Query    Query History

```

1 INSERT INTO customer_data(customer_id,zipcode,email)
2 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

```

1 SELECT * FROM public.customer_data
2 ORDER BY customer_id ASC

```

Data Output    Messages    Notifications

customer_id [PK] character varying	zipcode integer	email character varying
00000000000000000000000000000001	4841	abcd@gmail.com
00012a2ce6f8dcda20d059ce98491703	6273	annoyingMelanie@aol.com
000161a058600d5901f007fab4c27140	35550	oddClaudia3@yahoo.in
0001fd6190edaaf884bcacf3d49edf079	29830	lightTiffany@terra.com.br
0002414f95344307404f0ace7a26f1d5	39664	Evelyncrazy@live.co.uk

Fig. 5. insert\_after\_1

Query    Query History

```

1 DELETE FROM customer_data
2 WHERE customer_id='00000000000000000000000000000000abcd1';

```

Data Output    Messages    Notifications

DELETE 1

Query returned successfully in 66 msec.

Fig. 6. delete

Query    Query History

```

1 SELECT * FROM public.customer_data
2 ORDER BY customer_id ASC

```

Data Output    Messages    Notifications

customer_id [PK] character varying	zipcode integer	email character varying
00012a2ce6f8dcda20d059ce98491703	6273	annoyingMelanie@aol.com
000161a058600d5901f007fab4c27140	35550	oddClaudia3@yahoo.in
0001fd6190edaaf884bcacf3d49edf079	29830	lightTiffany@terra.com.br
0002414f95344307404f0ace7a26f1d5	39664	Evelyncrazy@live.co.uk
000379cdec625522490c315e70c7a9fb	4841	Marvinuptight@mail.ru

Fig. 7. delete\_after

Query    Query History

```

1 DELETE FROM promotion_details
2 WHERE promotion_type='quarterly';

```

Data Output    Messages    Notifications

DELETE 1

Query returned successfully in 76 msec.

Fig. 8. delete\_1

Query    Query History

```
1 SELECT * FROM public.promotion_details
2 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

```
1 SELECT * FROM public.promotion_details
2 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

```
1 UPDATE promotion_details
2 SET discount_percentage=20
3 WHERE type_id=7;
```

Data Output    Messages    Notifications

UPDATE 1

Query returned successfully in 74 msec.

Query    Query History

```
1 UPDATE customer_data
2 SET customer_id='0000000000000000000000abcd1', zipcode=29830, email='abcdef@gmail.com'
3 WHERE customer_id='00000000000000000000000000000001';
```

Data Output    Messages    Notifications

UPDATE 1

Query returned successfully in 88 msec.

Fig. 13. update\_1

Fig. 10. update

Query    Query History

```
1 SELECT * FROM public.promotion_details
2 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

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

Data Output    Messages    Notifications

	customer_id [PK] character varying	zipcode integer	email character varying
1	000000000000000000000000abcd1	29830	abcdef@gmail.com
2	00012a2ce6f8dcd20d059ce98491703	6273	annoyingMelanie@aol.com
3	000161a058600d5901f007fab4c27140	35550	oddClaudia3@yahoo.in
4	0001fd6190edaaef884bcfa3d49edf079	29830	lightTiffany@terra.com.br
5	0002414f95344307404fae7a26f1d5	39664	Evelyncrazy@live.co.uk
6	000379cdec625522490c315e70c7a9fb	4841	Marvinuptight@mail.ru

Fig. 14. update\_after\_1

Query		Query History					
1	SELECT * FROM public.customer_data						
2	ORDER BY customer_id ASC						
Data Output		Messages		Notifications			
    							
customer_id	[PK] character varying	zipcode	email				
1	00000000000000000000000000000001	4841	abcd@gmail.com				
2	00012a2ce6f8dcda20d059ce98491703	6273	annoyingMelanie@aol.com				
3	000161a058600d5901f007fb4c27140	35550	oddClaudia3@yahoo.in				
4	0001fd6190edaaf884bcfa3df49edf079	29830	lightTiffany@terra.com.br				
5	000241f95344307404f0ace7a26f1d5	39664	Evelyncrazy@live.co.uk				
6	000379cdec625522490c315e70c7a9fb	4841	Marvinuptight@mail.ru				

Fig. 15. update\_before\_1

The screenshot shows the pgAdmin 4 interface with a query editor and a results table.

**Query Editor:**

```
1 select customer_id, count(*) as total_no_of_orders from order_details
2 group by customer_id
3 order by total_no_of_orders;
```

**Data Output Table:**

	customer_id	total_no_of_orders
1	09033cfedb9bab5c54a33f339fd94ad0	1
2	4849a3b3f4f73f108f904c0171ec9fe0	1
3	9f184cd2aa748ce963a8d5e075aff28e	1
4	60a13661d952a5128a7c81ea31e54727	1
5	f796cc9e19019673220efa160fe7e610	1
6	6347d009d824a697594134c029ef771d	1
7	08afef5441616f4175e4b05f3201f23b	1

Fig. 17. query\_2

## **IX. SOME SAMPLE SQL QUERIES:**

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

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

The screenshot shows the DBeaver interface connected to a PostgreSQL database named 'ECommerce/postgres'. The query editor contains the following SQL code:

```
1 select * from customer_data
2 where email like '%yahoo%' or email like '%hotmail%';
```

The 'Data Output' tab is selected, displaying the results of the query:

	customer_id [PK] character varying	zipcode integer	email character varying
1	000161a058600d5901f007fab4c27140	35550	oddClaudia3@yahoo.in
2	000419c5494106c306a97b5635748086	24220	brainyCraig@yahoo.es
3	00072d033fe2e59061ae5c3aff1a2be5	45026	cruelAntonio@yahoo.co.jp
4	0012a5c13793cf51e25f30967e740dd	20011	Benjamincreepy@yahoo.es
5	001c7f05398c45b42ee00d5a77783bca	6719	goodRachael@yahoo.com.sg
6	002236c4f333bc6f6fa593794eb7869	12330	clumsyRaquel@yahoo.com
7	002408f390f729598bbac1ef9421c6ae	89188	amusedJill64@yahoo.co.id
8	0028ff36263a86bf679df7c863a0ba0	28895	RebeccaGrumpy@yahoo.co.uk
9	002905287304e28c0218389269b4759b	37578	elatedAngela@hotmail.es
10	002937abdae1368e017dcdd3868b4825	45530	Gloriawild@yahoo.es
11	002b5342c72978fc0abaa6aae1f5d5293	22775	crazyAngelica@yahoo.co.jp

Fig. 16. query\_1

The screenshot shows the pgAdmin 4 interface. The top bar displays the connection name "ECommerce/postgres@PostgreSQL 15". The main area has tabs for "Query" and "Query History". A SQL query is written in the "Query" tab:

```
1 select email,zipcode,review_score-1 as new_review_score from
2 customer_data natural join order_details natural join order_reviews
3 where zipcode=20540;
```

Below the query, there are tabs for "Data Output", "Messages", and "Notifications". The "Data Output" tab is selected, showing a table structure and data:

	email	zipcode	new_review_score
1	crazyLauren39@neuf.fr	20540	2
2	modernLinda28abc@yahoo.de	20540	4
3	Franklonelyabc@yahoo.com.ar	20540	4
4	carefulJack17abc@hotmail.co.uk	20540	4
5	disturbedLaura1abc@facebook.com	20540	4
6	blue-eyedMarcus@live.co.uk	20540	4
7	Alanajoyous@outlook.com	20540	3
8	pricklyMark27@yahoo.com.br	20540	4

Fig. 18. query 3

**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.**

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

```

1 select category_name,product_id from product_details pd where exists
2 (select * from order_items oi where exists
3 (select * from order_reviews o rr where review_score=5 and
4 o rr.order_id=oi.order_id) and oi.product_id=pd.product_id)
5 order by product_id;

```

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

```

1 SELECT o.order_id, o.customer_id, o.purchase_timestamp, oo.product_id, p.category_name, c.promotion_code
2 FROM order_details o
3 JOIN order_items oo ON oo.order_id = o.order_id
4 JOIN product_details p ON oo.product_id = p.product_id
5 JOIN category_promotions c ON p.category_name = c.category_name
6 WHERE o.purchase_timestamp <= (SELECT MAX(purchase_timestamp) FROM order_details)
7 AND o.order_id IN (
8   SELECT order_id FROM order_details WHERE
9   purchase_timestamp <= (SELECT MAX(purchase_timestamp) FROM order_details)
10 ORDER BY purchase_timestamp DESC
11 LIMIT 10
12 )
13 ORDER BY o.purchase_timestamp DESC;

```

Fig. 20. complex

order_id	customer_id	purchase_timestamp	product_id	category_name	promotion_code
1	2b2f538c598b73a9bd18e0d54592...	9/9/2017 9:54	3fc00a0f07084d4ec2933d3900416c977	climatizadores	YWFM7M
2	382a2285603c575241b44601ade85c3	1449d4f564274b18944678816c...	a237de12bdf0f64fb220bae65a9731	moveis_decoracao	QEDEHR
3	216237dbbe1af6a13a749e580c469	ae0a55c8402ba55165c04966b...	c39c5a3a88d5cd18800b86574d9ad	cama_mesa_banho	WNKBVH
4	634aff0f5cb0b3da79372b58458d...	30a8585de0bea3f588e49844...	4f6551ace08c388c09e5f4584b3309e5a	cool_atruff	VGVJN
5	634aff0f5cb0b3da79372b58458d...	30a8585de0bea3f588e49844...	4f6551ace08c388c09e5f4584b3309e5a	cool_atruff	FFSKVM
6	785606fae79b0dd51399521a70432ca...	708557214955455ced8de378e6e7...	d51e29080f0c03584477b75d5d353...	fashion_bolts_e_accessories	HFDDEW
7	785606fae79b0dd51399521a70432ca...	708557214955455ced8de378e6e7...	d51e29080f0c03584477b75d5d353...	fashion_bolts_e_accessories	PWTCDC
8	785606fae79b0dd51399521a70432ca...	708557214955455ced8de378e6e7...	d51e29080f0c03584477b75d5d353...	fashion_bolts_e_accessories	SDWIRJ
9	34084a2020122bb84edc515850581...	22aa2541ed4704ac45151a753792...	154e731ef0f0220379572580446	beltza_sauda	MNWIFT
10	34084a2020122bb84edc515850581...	22aa2541ed4704ac45151a753792...	154e731ef0f0220379572580446	beltza_sauda	JOWICE
11	194edc136ca5493f7a8d2374648d...	ee201f148f7a6293287648d...	2b460f9189408e10874484204946b3...	beltza_sauda	JOWICE
12	194edc136ca5493f7a8d2374648d...	ee201f148f7a6293287648d...	2b460f9189408e10874484204946b3...	beltza_sauda	MNWIFT
13	8990a2c1eeef015b5b5209d5e...	9e699e99d1f5b5b5209d5e...	442686e2034724c2e02d297395...	pet_shop	FHKPAP
14	1aa46433170a457014516874822...	02da9f205e47c08d7fe4349992...	7cf1bd206d274766b06e7563d3cf	beltza_sauda	MNWIFT
15	471d17c72833d99058c4d19b5942...	de0202510888e7594b8b4467...	154e731ef0f0220379572580446	beltza_sauda	MNWIFT

Total rows: 17 of 17 - Query complete 00:00:00.122

```

1 SELECT o.order_id, ordr.review_score, c.email,c.zipcode
2 FROM customer_data c
3 JOIN (
4   SELECT zip_code
5   FROM address_details
6   WHERE zip_code=29830
7 ) AS a ON a.zip_code = c.zipcode
8 JOIN order_details o ON c.customer_id = o.customer_id
9 JOIN order_reviews ordr ON ordr.order_id = o.order_id
10 WHERE ordr.review_score=5;
11

```

Fig. 22. complex\_2

```

1 SELECT o.order_id, ordr.review_score, c.email,c.zipcode
2 FROM customer_data c
3 JOIN (
4   SELECT zip_code
5   FROM address_details
6   WHERE zip_code=29830
7 ) AS a ON a.zip_code = c.zipcode
8 JOIN order_details o ON c.customer_id = o.customer_id
9 JOIN order_reviews ordr ON ordr.order_id = o.order_id
10 WHERE ordr.review_score=5;
11

```

order_id	review_score	email	zipcode
316a104623542e4d75189bb372bc5f...	5	lightTiffany@terra.com.br	29830
53d617654db4fc6f699b09a53c7ca	5	braveCharlotte10@qq.com	29830
6ebc7d36b09d160bebfb7e170be9	5	handsomeSamanthaabc@optusnet.com.au	29830
9ecc7884797c901042fe3114fe591f5	5	Krystalloughabc@rocketmail.com	29830
9f67054716b5806d7ecb46b0f309e5c3	5	cuteCalvin95abc@virgilio.it	29830

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

```

1 EXPLAIN ANALYZE
2 select email,count(email) from customer_data
3 group by email having (count(email)>0);

```

QUERY PLAN
1 HashAggregate (cost=10120.61..12722.01 rows=33116 width=34) (actual time=48.507..79.504 rows=99380 loops=1)
2 Group Key:email
3 Filter:(count(email)>0)
4 Batches: 5 Memory Usage: 8753K Disk Usage: 3448kB
5 > Seq Scan on customer_data (cost=0.00..2196.41 rows=99441 width=26) (actual time=0.019..4.627 rows=99441 loops=1)
6 Planning Time: 0.863 ms
7 Execution Time: 83.779 ms

Query\_History  
Total rows: 7 of 7 - Query complete 00:00:00.120

Fig. 24. before\_indexing

```

Query
1 CREATE INDEX i_index on customer_data(email);
2
3 EXPLAIN ANALYZE
4 select email, count(email) as unique_email from customer_data
5 group by email having (count(email)>0);

Data Output Messages Notifications
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=1)
5   Heap Fetches: 336
6   Planning Time: 0.949 ms
7   Execution Time: 23.947 ms

```

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

Fig. 25. after\_indexing

Fig. 28. before\_indexing\_3

## 2. A query to find the order id and status of all the delivered orders.

```

Query
1 EXPLAIN ANALYZE SELECT od.order_id, od.status FROM order_details od
2 WHERE od.order_id IN (SELECT order_id FROM order_items) AND od.status = 'delivered';
3
4
5
6
7
8
9
10
11
12
13

Data Output Messages Notifications
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')
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

Fig. 29. before\_indexing\_3.1

```

Query
1 EXPLAIN ANALYZE
2 SELECT od.order_id, od.status FROM
3 order_details od NATURAL JOIN order_items oi
4 WHERE od.status = 'delivered';
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

Data Output Messages Notifications
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: ((od.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')
8  Rows Removed by Filter: 1845
9  Planning Time: 0.334 ms
10 Execution Time: 29.041 ms

```

Fig. 27. after\_2\_indexing

Fig. 30. before\_indexing\_3.2

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

```

Query
1 EXPLAIN ANALYZE SELECT ad.state, pd.category_name, COUNT(DISTINCT od.order_id) AS total_orders,
2 COUNT(DISTINCT pd.zip_code) AS total_customers, AVG(ord.review_score) AS average_review_score
3 FROM address_details ad JOIN customer_data cd ON ad.zip_code = cd.zip_code
4 JOIN order_details od ON od.customer_id = cd.customer_id JOIN order_items oi ON od.order_id = oi.order_id
5 JOIN product_details pd ON oi.product_id = pd.product_id JOIN order_reviews ord ON od.order_id = ord.order_id
6 WHERE od.status = 'delivered' GROUP BY ad.state, pd.category_name ORDER BY total_orders DESC, average_review_score DESC;
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38

Data Output Messages Notifications
QUERY PLAN
text
1 -> 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)
2 -> Hash (cost=243.78..243.78 rows=15078 width=7) (actual time=0.249..0.249 rows=15078 loops=1)
3 Buckets: 16384 Batches: 1 Memory Usage: 717kB
4 -> 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)
5 -> Hash (cost=720.52..720.52 rows=32952 width=48) (actual time=7.049..7.049 rows=32952 loops=1)
6 Buckets: 65536 Batches: 1 Memory Usage: 3109kB
7 -> 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)
8 Planning Time: 2.129 ms
9 Execution Time: 529.898 ms

```

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

Fig. 31. before\_indexing\_3.3

```

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

Query History

```

1 EXPLAIN ANALYZE
2 WITH order_summary AS (SELECT od.order_id, cd.zip_code, pd.category_name, ord.review_score
3   FROM order_details od JOIN customer_data cd ON od.customer_id = cd.customer_id
4   JOIN order_items oi ON od.order_id = oi.order_id JOIN product_details pd ON oi.product_id = pd.product_id
5   JOIN order_reviews ord ON od.order_id = ord.order_id WHERE od.status = 'delivered')
6   state_category_summary AS (SELECT ad.state, os.category_name, COUNT(os.order_id) AS total_orders,
7   COUNT(DISTINCT os.zipcode) AS total_customers, AVG(os.review_score) AS average_review_score
8   FROM order_summary od JOIN address_details ad ON os.zipcode = ad.zip_code GROUP BY ad.state, os.category_name)
9   SELECT state, category_name, total_orders, total_customers, average_review_score
10  FROM state_category_summary ORDER BY total_orders DESC, average_review_score DESC;

```

Data Output Messages Notifications

QUERY PLAN

```

text
1 > 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)
2 > Hash (cost=243.78..243.78 rows=15078 width=7) (actual time=1.927..1.928 rows=15078 loops=1)
3 Buckets: 16384 Batches: 1 Memory Usage: 717kB
4 > 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)
5 Planning Time: 2.050 ms
6 Execution Time: 266.936 ms
Total rows: 38 of 38 Query complete 00:00:28.262
Ln 10, Col 83

```

Fig. 32. after\_indexing\_3

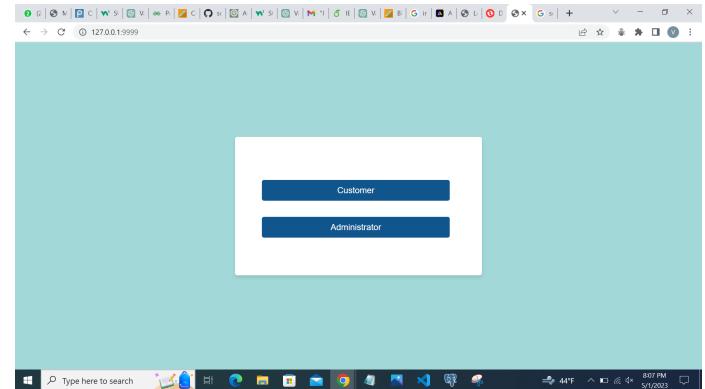


Fig. 37. initial login page

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(pd.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: ((pd.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

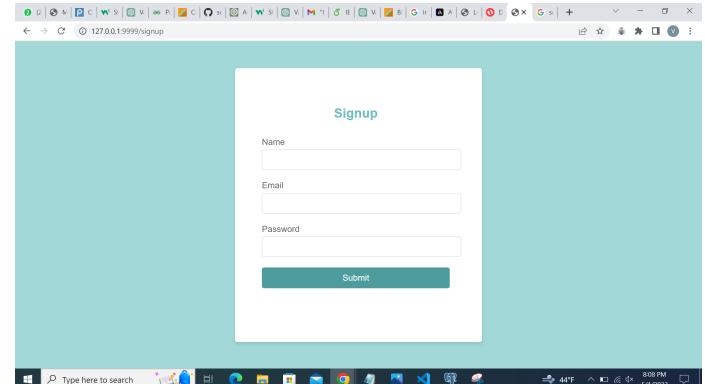


Fig. 38. Customer Signup Page

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 ord (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

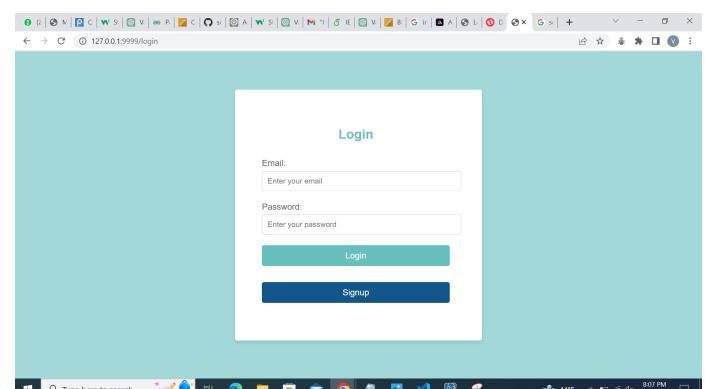


Fig. 39. Customer Login

```

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

The screenshot shows two browser windows side-by-side. The left window, titled 'Orders On a GO', displays a search form with a 'Category Name:' input field and a 'Search Products' button. Below it is a message: 'Just click on Search Products if you want to view all products'. The right window, also titled 'Orders On a GO', shows a table titled 'My Orders' with columns: Order ID, Customer ID, Status, Purchase\_Timestamp, Delivered\_Timestamp, and Estimated\_Delivery\_Timestamp. It lists three orders with details like status 'delivered' and timestamps from 2017 to 2011.

Fig. 40. Customer Home page



Fig. 43. Customers Orders

The screenshot shows two browser windows. The left window, 'Orders On a GO', displays a table of products under 'My Products' with columns: Product ID, Category Name, Weight, Length, Height, Width, and a 'Click to Order' button. The right window, also 'Orders On a GO', shows four buttons: View all Customers, Add Customer, Remove Customer, and Update Customer.

Fig. 41. Customer Order page



Fig. 44. Administrator Home Page

The screenshot shows two browser windows. The left window, 'Orders On a GO', displays 'My Details' with fields for Customer ID, Zipcode, and Email. The right window, also 'Orders On a GO', displays 'Order Data' with a table of order details. Both windows show the same timestamp of 8:10 PM 5/1/2023.

Fig. 42. Customer Details



Fig. 45. Viewing all orders

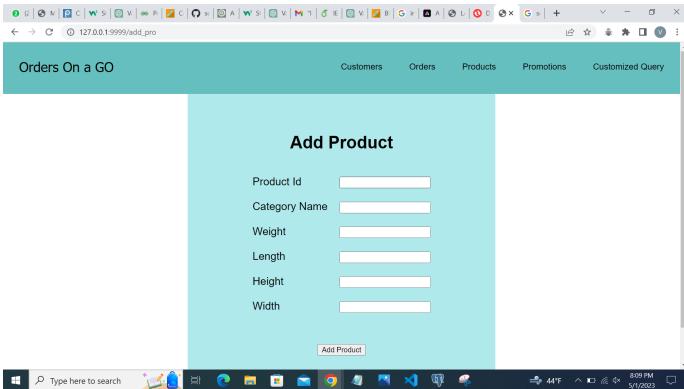


Fig. 46. Adding Products

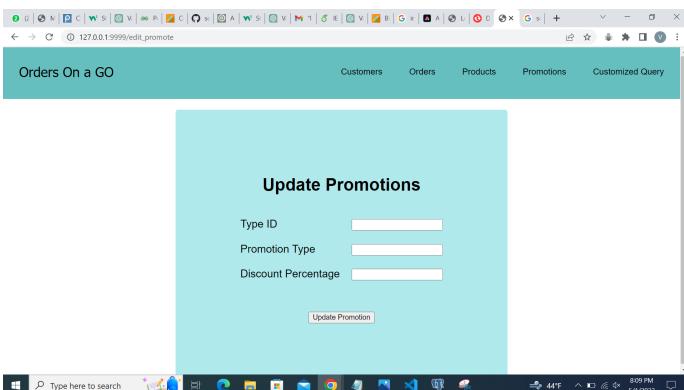


Fig. 47. Updating Promotions

customer_id	zipcode	email
000161a59800d5901f007fb4c27140	35550	oddClaudia33@yahoo.in
000168190eadaef884bcfa3349ed079	29830	lightTiffany@terra.com.br
0002414953443074040ace7a26f1d5	39664	EvelynCrazy@live.co.uk
000379cdec62522490c315e70c7a9fb	4841	Marinupright@gmail.ru
00041642009e969a7f334965408652	13272	soreJacquelyn14@skynet.be
000419c5494106c306897b5635748066	24220	brainyCraig@yahoo.es
00046a560d407e99b969756e010f282	20540	defiantVolanda81@sky.com
00059f6ea1a0d8efc7c419f1b7fb20c	98700	importedMavis@tin.it

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//