

Winter Semester 2022

CSE250 Database Management System

Project Title:

E - Commerce Management System

**Group Details**

| **Name** | **Enrollment No.** |
| --- | --- |
| Tirth Patel | AU2040020 |
| Rushali Moteria | AU2040210 |
| Akshil Thumar | AU2040258 |
| Ushmay Patel | AU2040253 |

### **Project Description:**

The project is on an **E-commerce management system** coded in Postgres and Nodejs. With the increase in demand for online shopping there is a need to manage and store the huge e-commerce data that is generated everyday. The project develops a system that keeps track of information about different products, its categories, information about customers and shippers. It organizes the data into different interrelated tables which make it easy to store the data without redundancy and access it based on the requirement.

**Requirement Analysis:**

There are major two role in the system:

1. Admin user: The admin will have access to the details of every product, can add products and edit all details. Admin can also access the details of all customers.
2. Customer user: A customer can sign up by entering his details and then will be able to search products, can see the product details, add products to his/her cart and order it.

**Entities:**

The data has been stored in the following entities:

1. Admin
2. User
3. Customer
4. Address
5. category
6. Product
7. Cart
8. Order
9. Payment
10. Shipper
11. Canceled order

**Relationships:**

user 1 : M Address

Customer 1 : 1 Cart

Category 1 : M Product

Product M : M Cart

Cart 1 : M Order

Order 1:1 Payment

Shipper 1:M Order

**Normalization of Tables:**

While creating the database, we have taken in consideration the normalization of tables.

* The attribute of the phone number is multivalued so it has been divided into two different columns.
* A customer can have multiple addresses (multivalued attribute) to which the order can be shipped, so a separate table has been created which stores different addresses of a particular customer by referencing the user id of users table.
* Product and Cart have many to many relation which has been converted to one to many by introducing a bridge entity (Cart product)

**Functionalities:**

**Basic functionalities:**

1. **Managing products:** Admin can add and delete products and can also manage its quantity, price etc.
2. **Accessing customer details:** The admin can view the details of customers including the cart details and the order details
3. **Viewing products:** the customers can view the products based on a particular category or price.
4. **Adding Products to cart**: Customers can add products to cart by adding it as a cart product. Two different entities, cart product and cart, are created. Customers can add or add or delete products from the cart product table, whereas the cart table shows the total amount and quantity of a particular cart.
5. **Placing an order:** Customers can place an order followed by the payment of the same.
6. **Payment:** Customers can pay the amount of products during checkout through different modes of payment like cash on delivery, online payment via credit card or debit card.
7. **Order cancellation:** customers can cancel their order by providing order id.

**Other functionalities:**

1. The admin can search the number of orders that have been canceled.
2. The admin can view the orders placed between two dates.
3. The admin can view the number of products at sale of a particular brand.
4. Customers can view the number of products of a particular category.
5. Update customer details such as phone number and email id (customer name can not be changed)

**System Requirements and Implemented Functions**:

**Backend:**

We have created a backend database in Postgres.

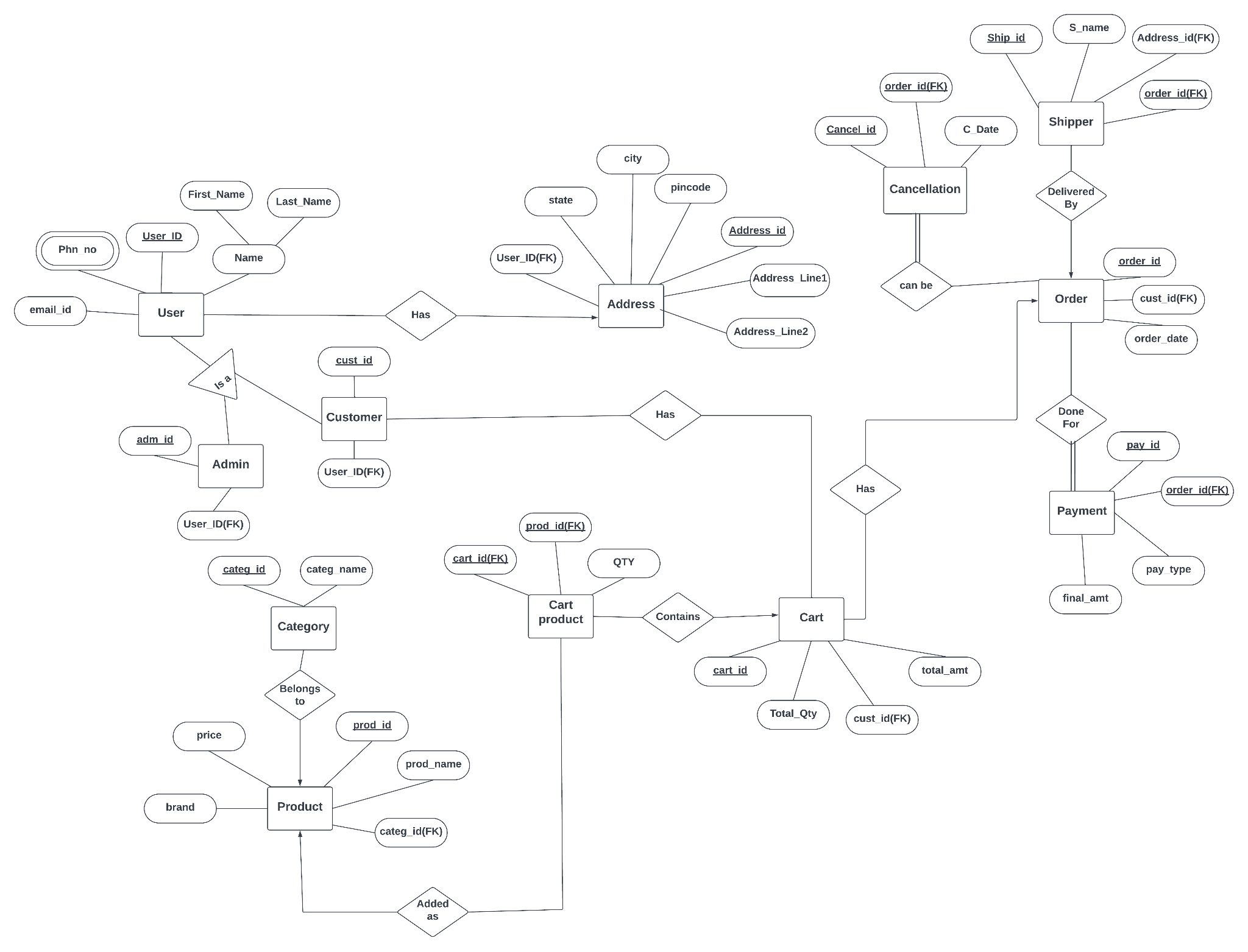
**Frontend:**

We have connected the backend database with the frontend using React(Node JS).

### **B. ER Diagram**

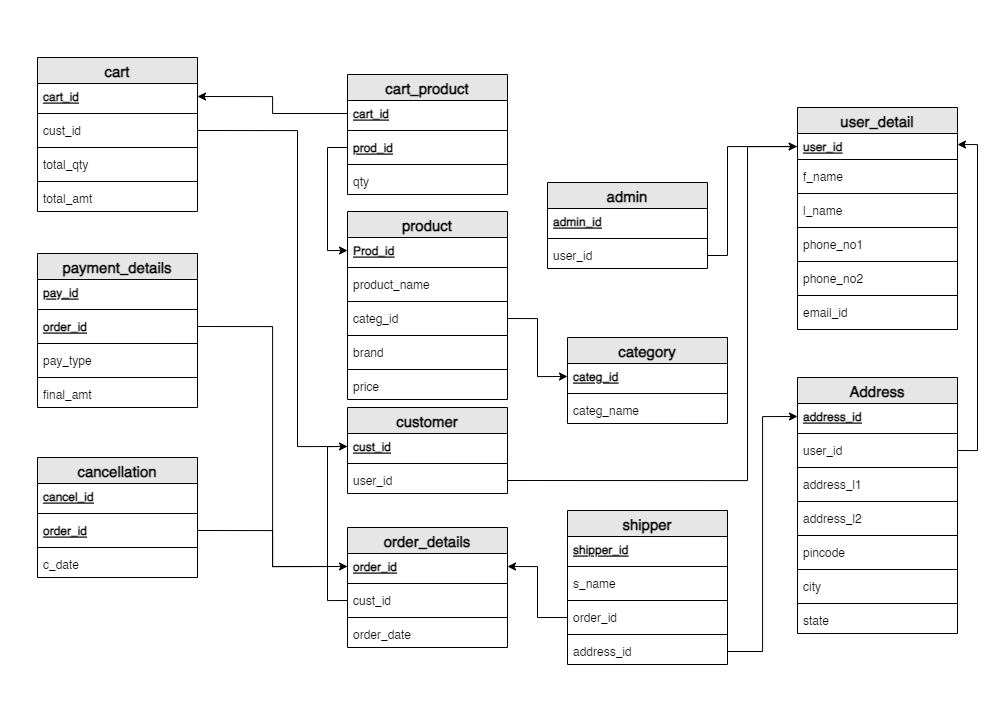
( The following ER Diagram has been made in Lucid Chart)

Link: <https://lucid.app/lucidchart/df779538-a122-4efc-99d6-5cc66738a599/edit?beaconFlowId=9D3FB2B755AAD81B&invitationId=inv_310cd7d8-cf55-4417-9cb7-f51d18b64afb&page=0_0#>



### **C. Table Design**

<https://drive.google.com/file/d/1Tlv449XPx7PA-FolKQDDoFST16Kn5M6-/view?usp=sharing>

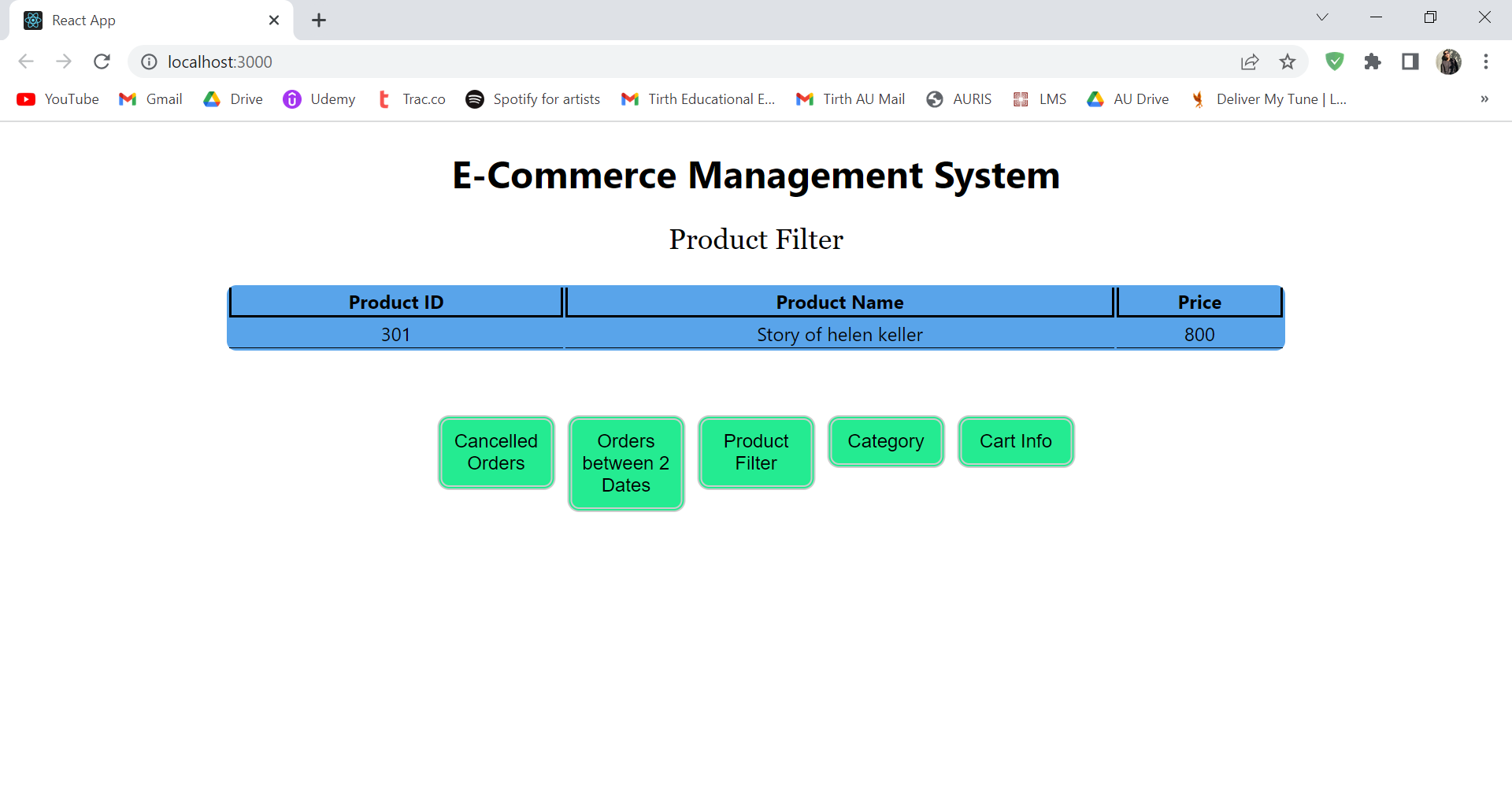


### **D. Functions**

1. **Function to filter products based on category and cost**

| create or replace function product\_filter(costitem int, category varchar(20))  returns table (prod\_id integer,product\_name varchar(30),price decimal(10,2))  as $$  begin  return query  select pr.prod\_id,pr.product\_name,pr.price from product pr,category ca where pr.price<=costitem and ca.categ\_name = category and pr.categ\_id=ca.categ\_id;  end  $$  language plpgsql;  select product\_filter(14000, 'Books'); |
| --- |

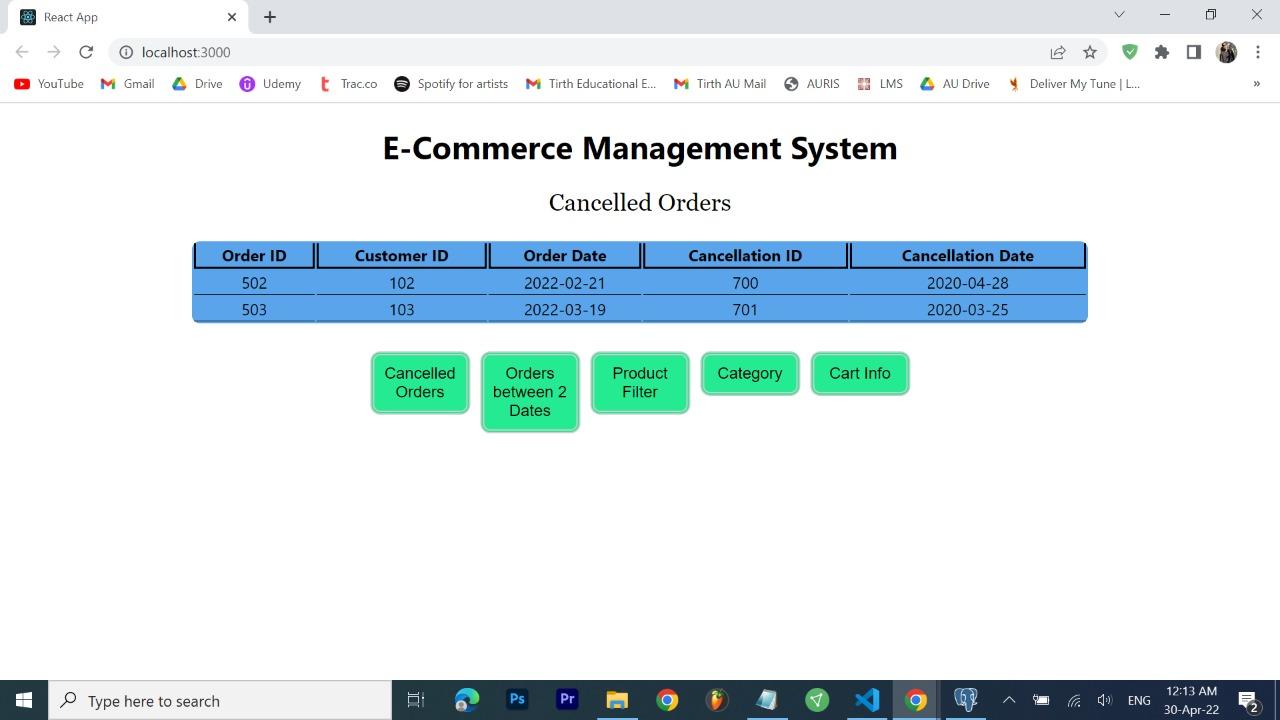
**Output:**



1. **Function to display the details of orders that have been canceled**

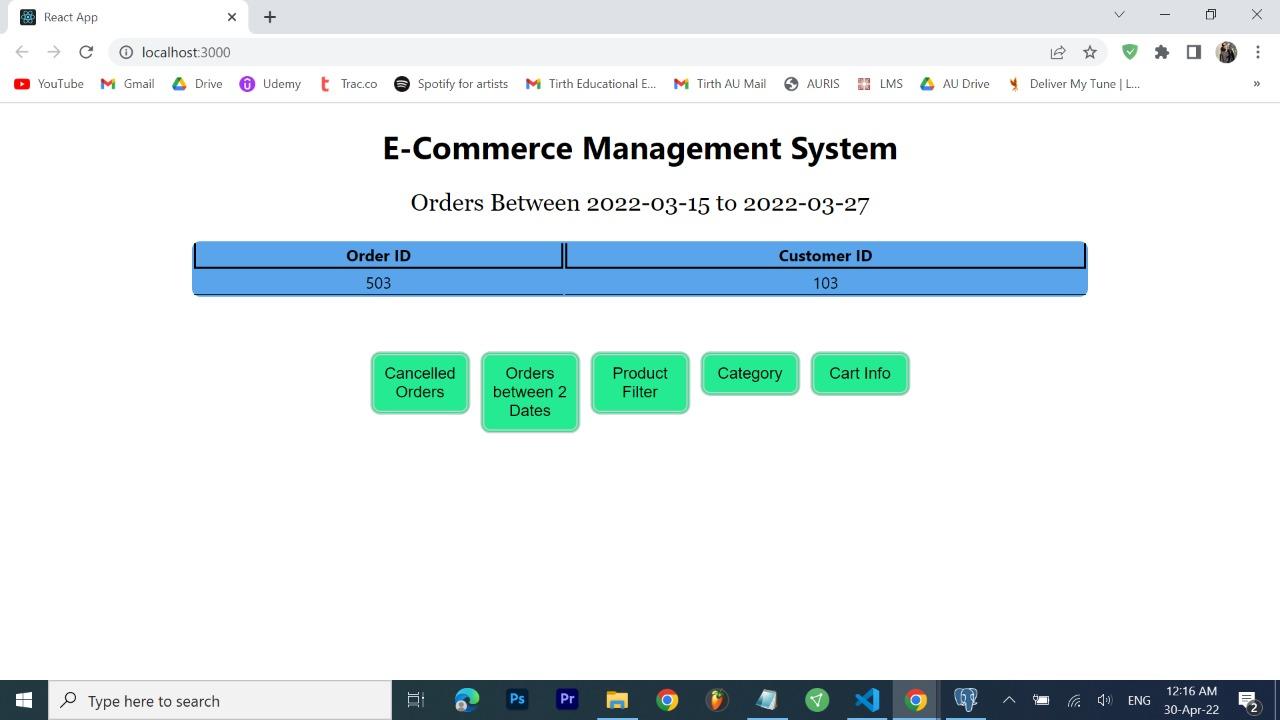
| create or replace function cancelled\_orders()  returns table(order\_id int,cust\_id int,order\_date date,cancel\_id int,c\_dt date)  as $$  begin  return query  select od.order\_id,od.cust\_id,od.order\_date,cd.cancel\_id,cd.c\_date from order\_details od,cancellation cd where od.order\_id=cd.order\_id;  end  $$  language plpgsql;  select cancelled\_orders(); |
| --- |

**Output:**



1. **Function to display the details of order that have been placed between two inputted dates.**

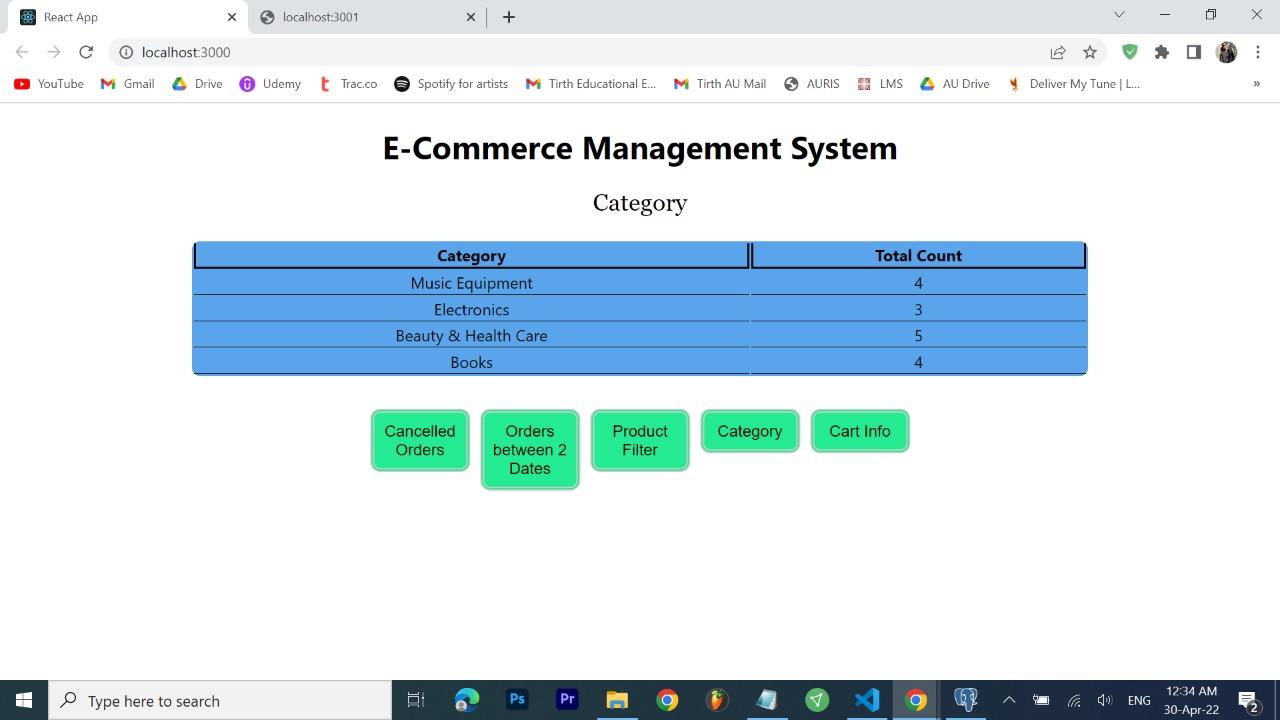
| create or replace function orders(date1 date,date2 date)  returns table(order\_id int,cust\_id int)  as $$  begin  return query  select od.order\_id,od.cust\_id from order\_details od where order\_date>=date1 and order\_date<=date2;  end  $$  language plpgsql;  select orders('2022-03-15','2022-03-27'); |
| --- |

**Output:**

1. **Function to display the number of products of each category**

| create or replace function products()  returns table(categ\_name varchar(20),product\_num bigint)  as $$  begin  return query  select category.categ\_name,count(prod\_id) from category,product group by category.categ\_name;  end  $$  language plpgsql;  select products(); |
| --- |

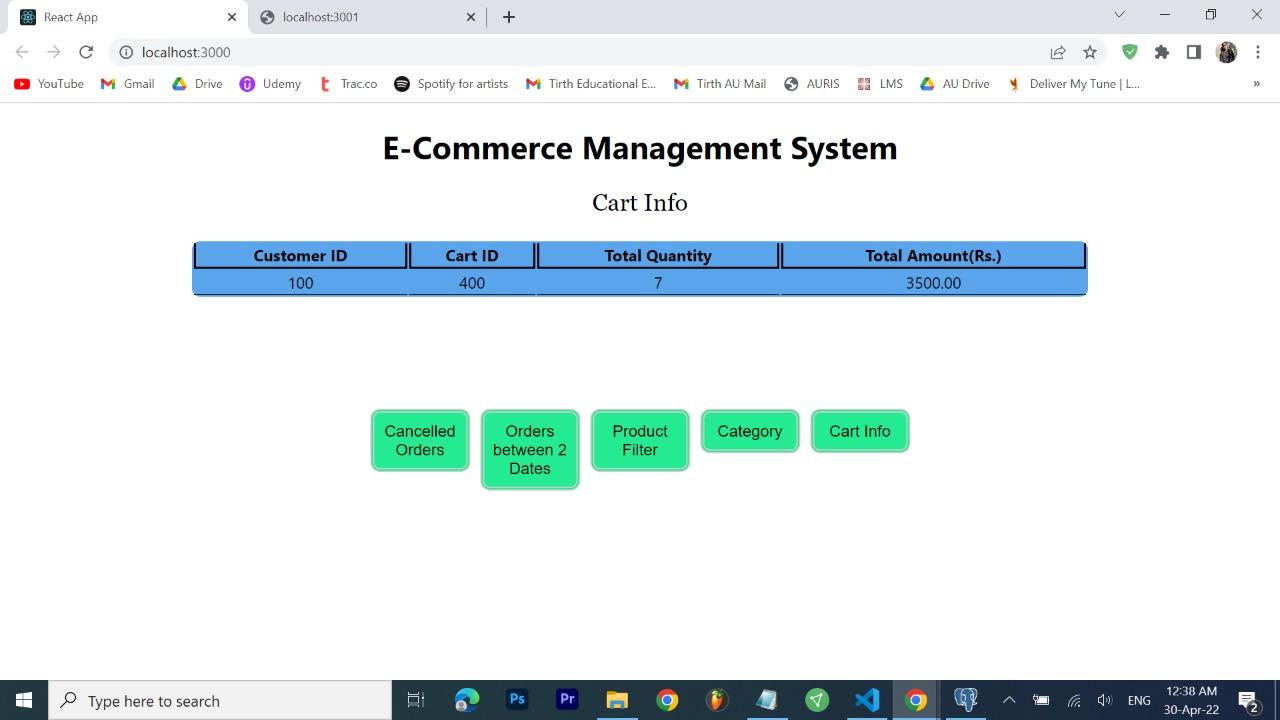
**Output:**



1. **Function to display the cart details of a particular customer based on the given customer id.**

| create or replace function cart\_info(custid int)  returns table(cust\_id int,cart\_id int,total\_qty decimal(3,0),total\_amt decimal(11,2))  as $$  begin return query  select cart.cust\_id,cart.cart\_id,cart.total\_qty,cart.total\_amt from cart where cart.cust\_id=custid;  end  $$  language plpgsql;  select cart\_info(100); |
| --- |

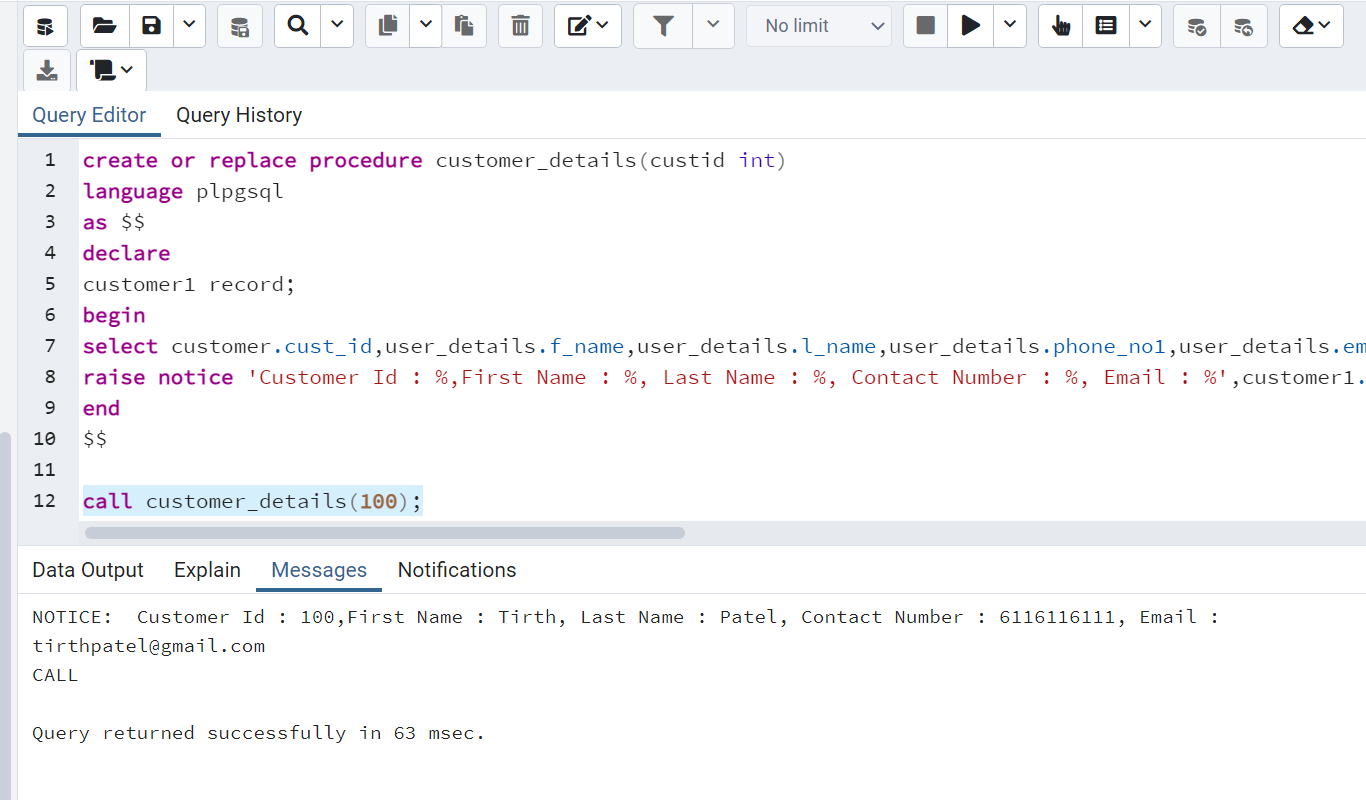
**Output:**



### **E. Procedures**

1. **Procedure to display the details of a customer based on an inputted customer id.**

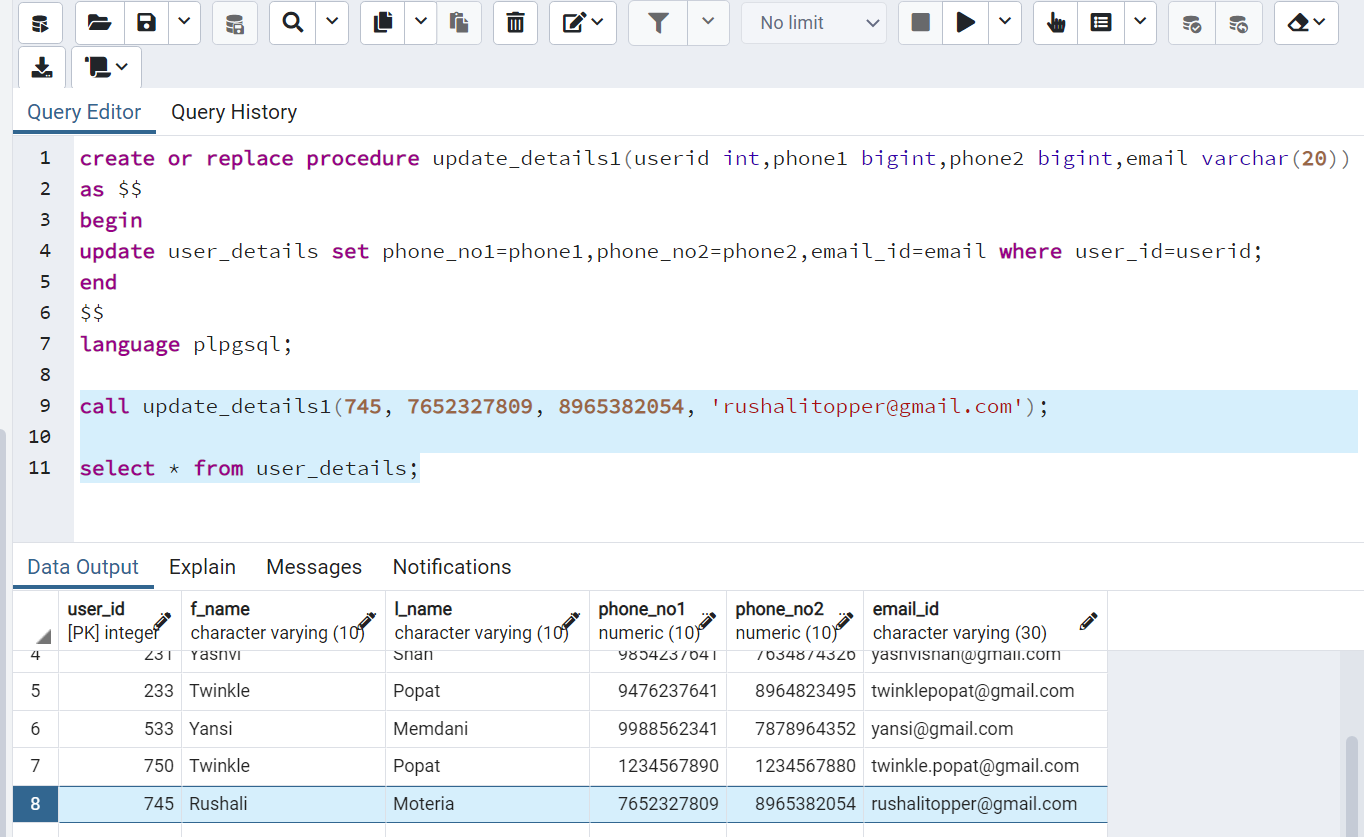
| create or replace procedure customer\_details(custid int)  language plpgsql  as $$  declare  customer1 record;  begin  select customer.cust\_id,user\_details.f\_name,user\_details.l\_name,user\_details.phone\_no1,user\_details.email\_id into customer1 from customer, user\_details where customer.user\_id=user\_details.user\_id and customer.cust\_id=custid;  raise exception 'Customer Id : %,First Name : %, Last Name : %, Contact Number : %, Email : %',customer1.cust\_id,customer1.f\_name,customer1.l\_name,customer1.phone\_no1,customer1.email\_id;  end  $$  call customer\_details(100); |
| --- |

**Output:**

1. **Procedure to update the details (phone number or email of a customer)**

| create or replace procedure update\_details1(userid int,phone1 bigint,phone2 bigint,email varchar(20))  as $$  begin  update user\_details set phone\_no1=phone1,phone\_no2=phone2,email\_id=email where user\_id=userid;  end  $$  language plpgsql;  call update\_details1(745, 7652327809, 8965382054, 'rushalitopper@gmail.com'); |
| --- |

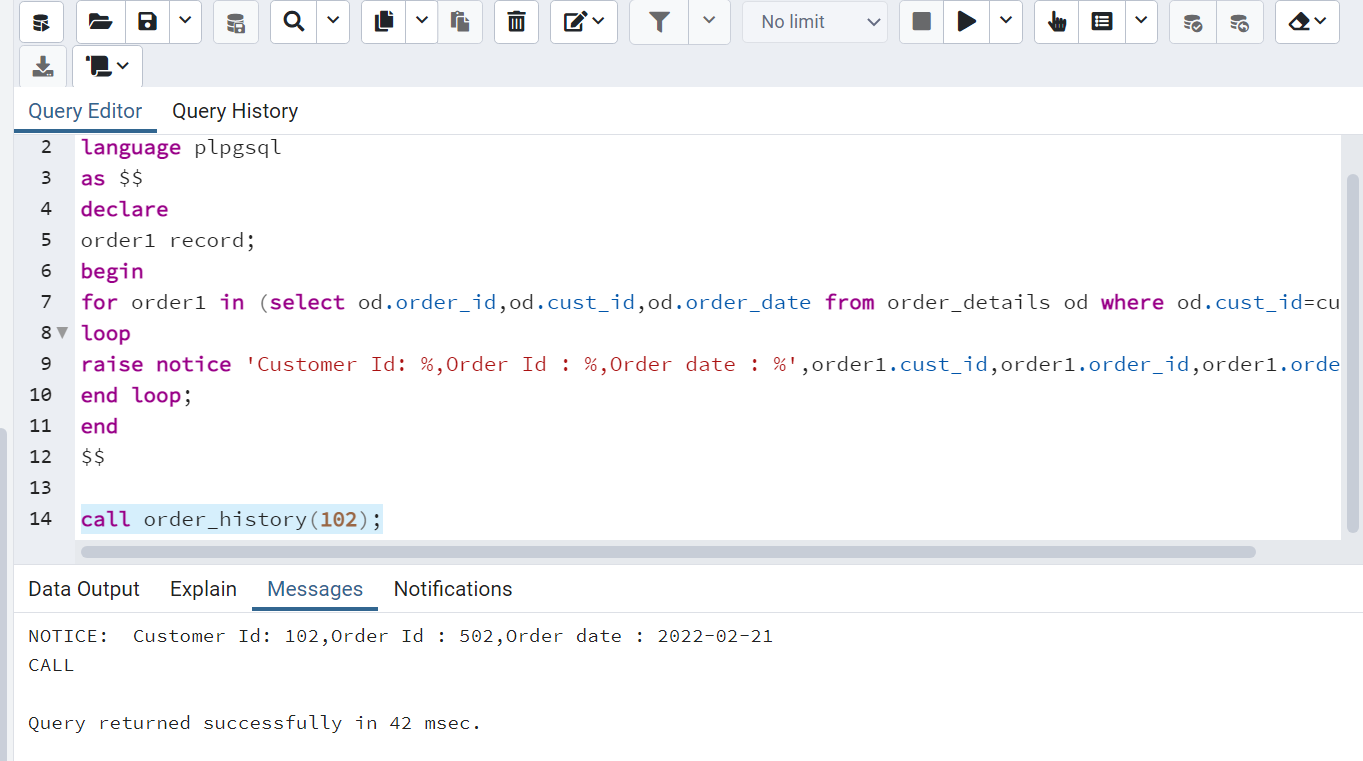
**Output:**



1. **Procedure to display the order history of a particular customer.**

| create or replace procedure order\_history(custid1 int)  language plpgsql  as $$  declare  order1 record;  begin  for order1 in (select od.order\_id,od.cust\_id,od.order\_date from order\_details od where od.cust\_id=custid1)  loop  raise exception 'Customer Id: %,Order Id : %,Order date : %',order1.cust\_id,order1.order\_id,order1.order\_date;  end loop;  end  $$  call order\_history(102); |
| --- |

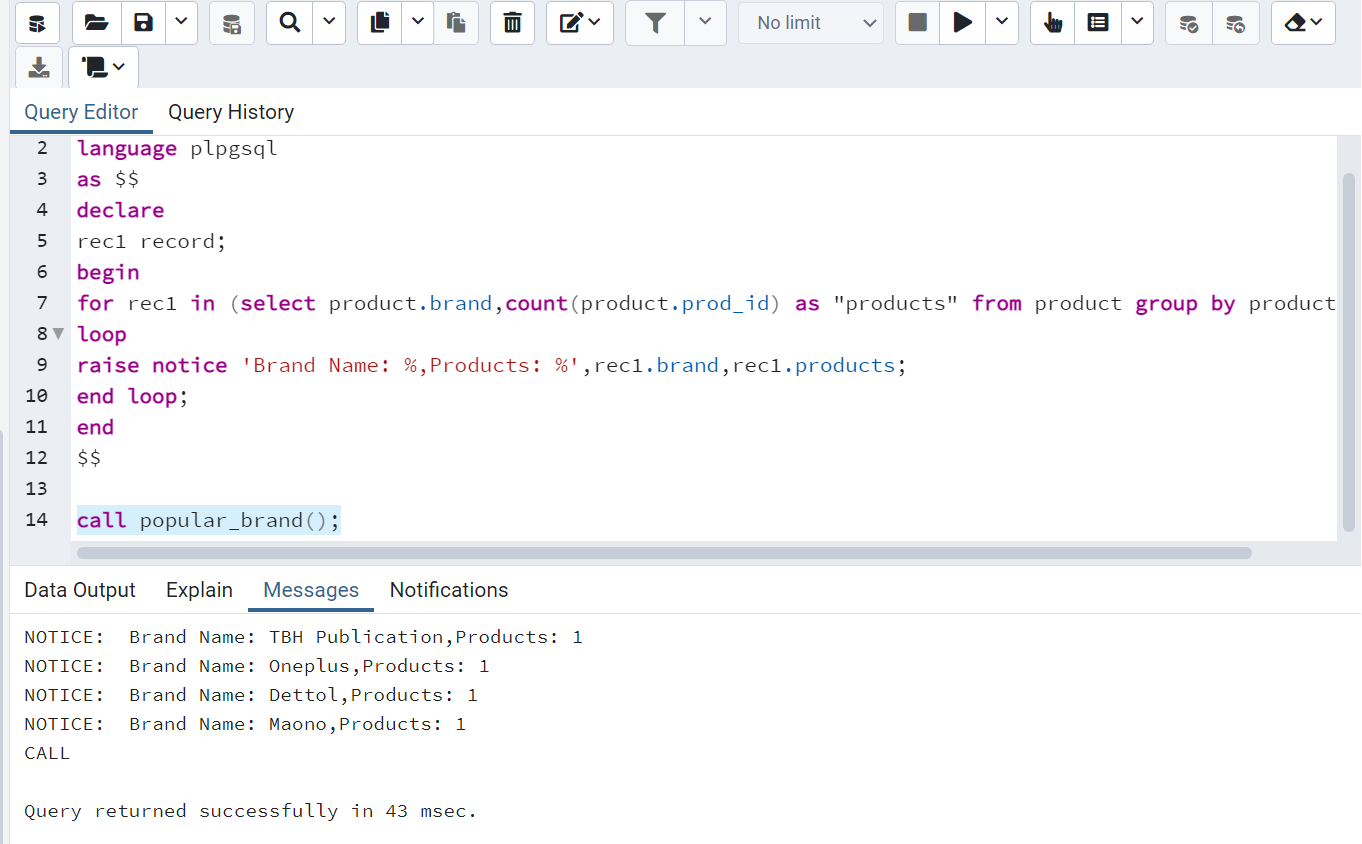
**Output:**

****

1. **Procedure to display the number of products of a particular brand.**

| create or replace procedure popular\_brand()  language plpgsql  as $$  declare  rec1 record;  begin  for rec1 in (select product.brand,count(product.prod\_id) as "products" from product group by product.brand)  loop  raise exception 'Brand Name: %,Products: %',rec1.brand,rec1.products;  end loop;  end  $$  call popular\_brand(); |
| --- |

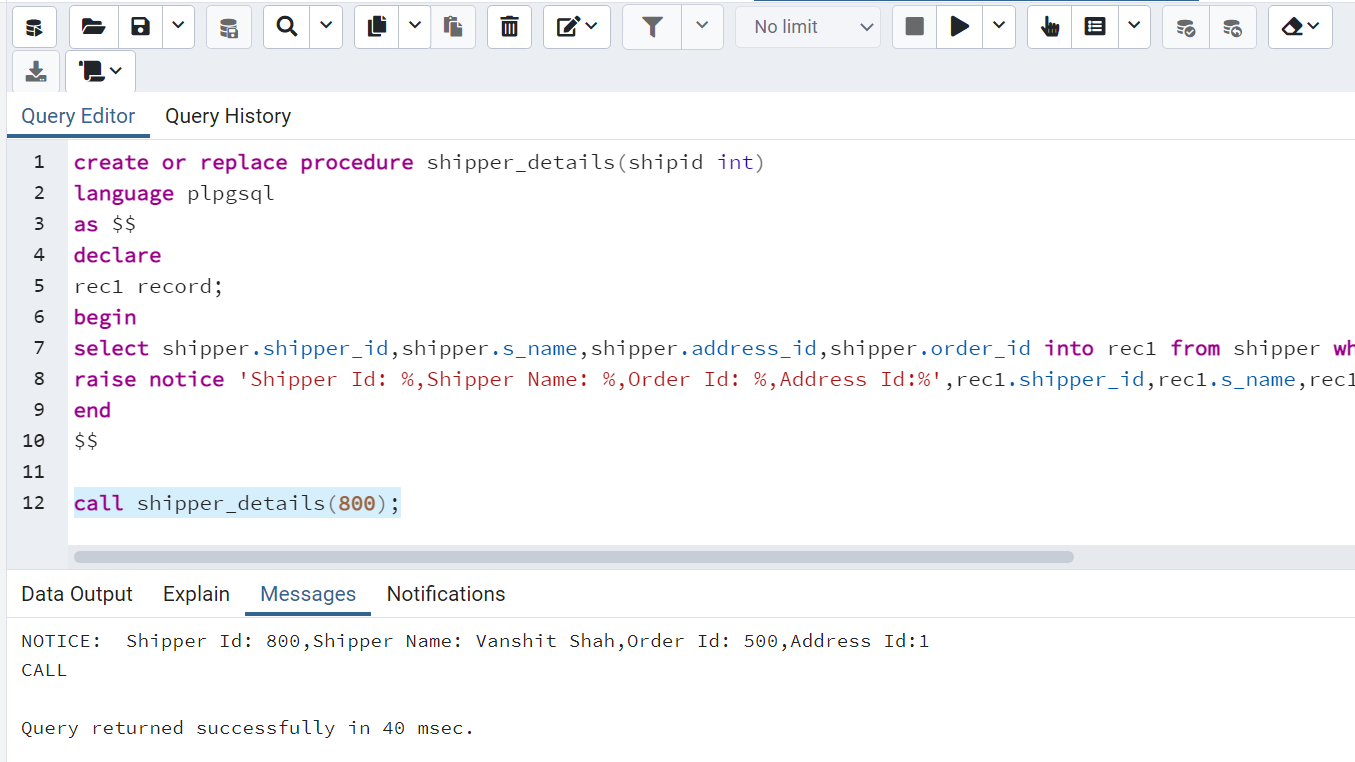
**Output:**



1. **Procedure to display the details of shippers.**

| create or replace procedure shipper\_details(shipid int)  language plpgsql  as $$  declare  rec1 record;  begin  select shipper.shipper\_id,shipper.s\_name,shipper.address\_id,shipper.order\_id into rec1 from shipper where shipper.shipper\_id=shipid;  raise exception 'Shipper Id: %,Shipper Name: %,Order Id: %,Address Id:%',rec1.shipper\_id,rec1.s\_name,rec1.order\_id,rec1.address\_id;  end  $$  call shipper\_details(800); |
| --- |

**Output:**

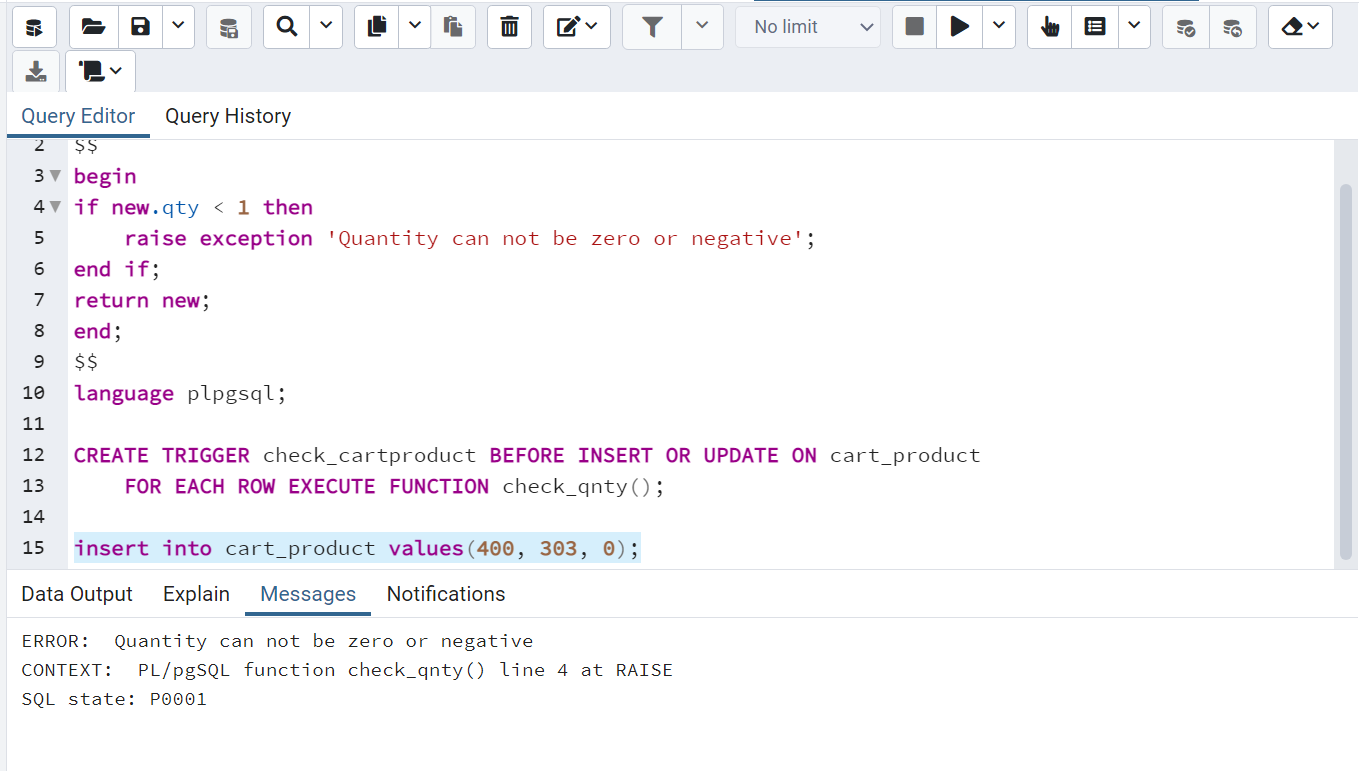


### **F. Triggers**

1. **Trigger to validate the quantity of the product. Quantity cannot be zero or negative.**

| create or replace function check\_qnty() returns trigger as  $$  begin  if new.qty < 1 then  raise exception 'Quantity can not be zero or negative';  end if;  return new;  end;  $$  language plpgsql;  CREATE TRIGGER check\_cartproduct BEFORE INSERT OR UPDATE ON cart\_products  FOR EACH ROW EXECUTE FUNCTION check\_qty;  insert into cart\_product values(400, 303, 0); |
| --- |

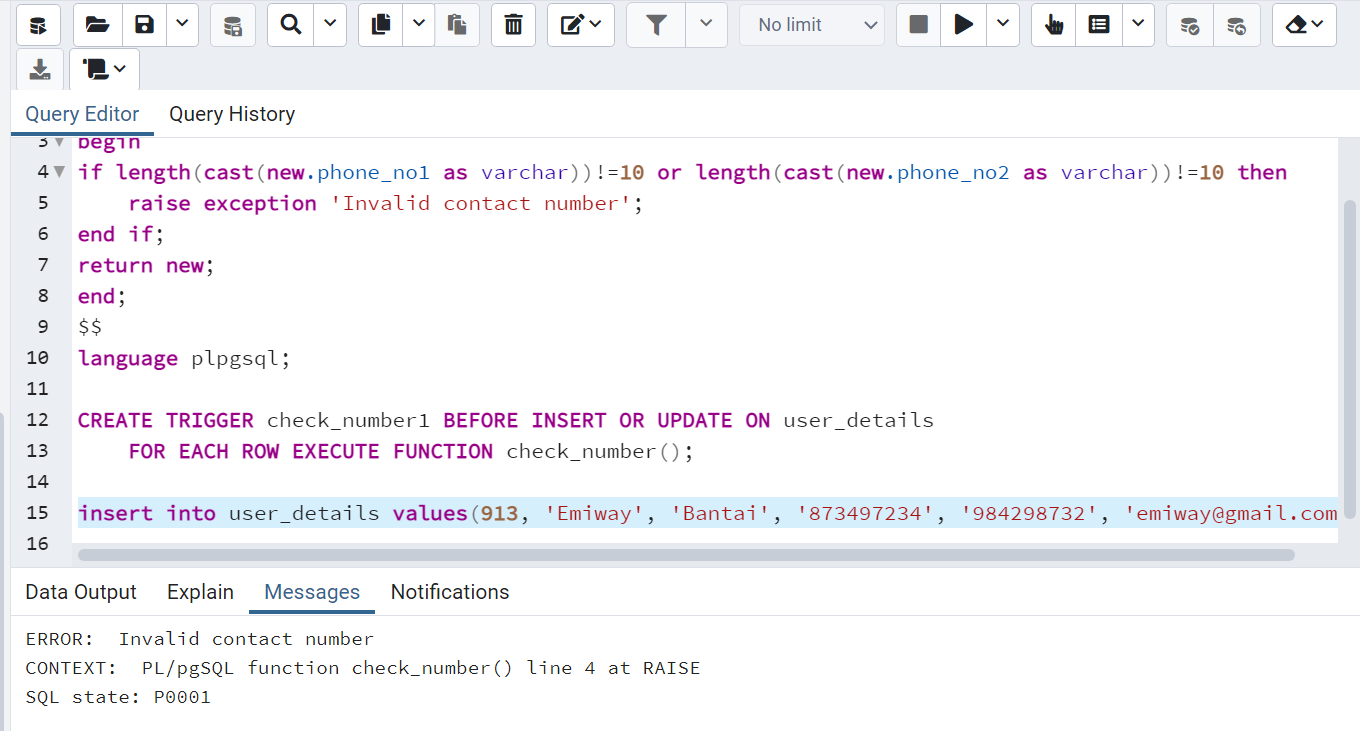
**Output:**



1. **Trigger to validate the inserted phone number.**

| create or replace function check\_number() returns trigger as  $$  begin  if length(cast(new.phone\_no1 as varchar))!=10 or length(cast(new.phone\_no2 as varchar))!=10 then  raise exception 'Invalid contact number';  end if;  return new;  end;  $$  language plpgsql;  CREATE TRIGGER check\_number1 BEFORE INSERT OR UPDATE ON user\_details  FOR EACH ROW EXECUTE FUNCTION check\_number();    insert into user\_details values(913, 'Emiway', 'Bantai', '873497234', '984298732', 'emiway@gmail.com'); |
| --- |

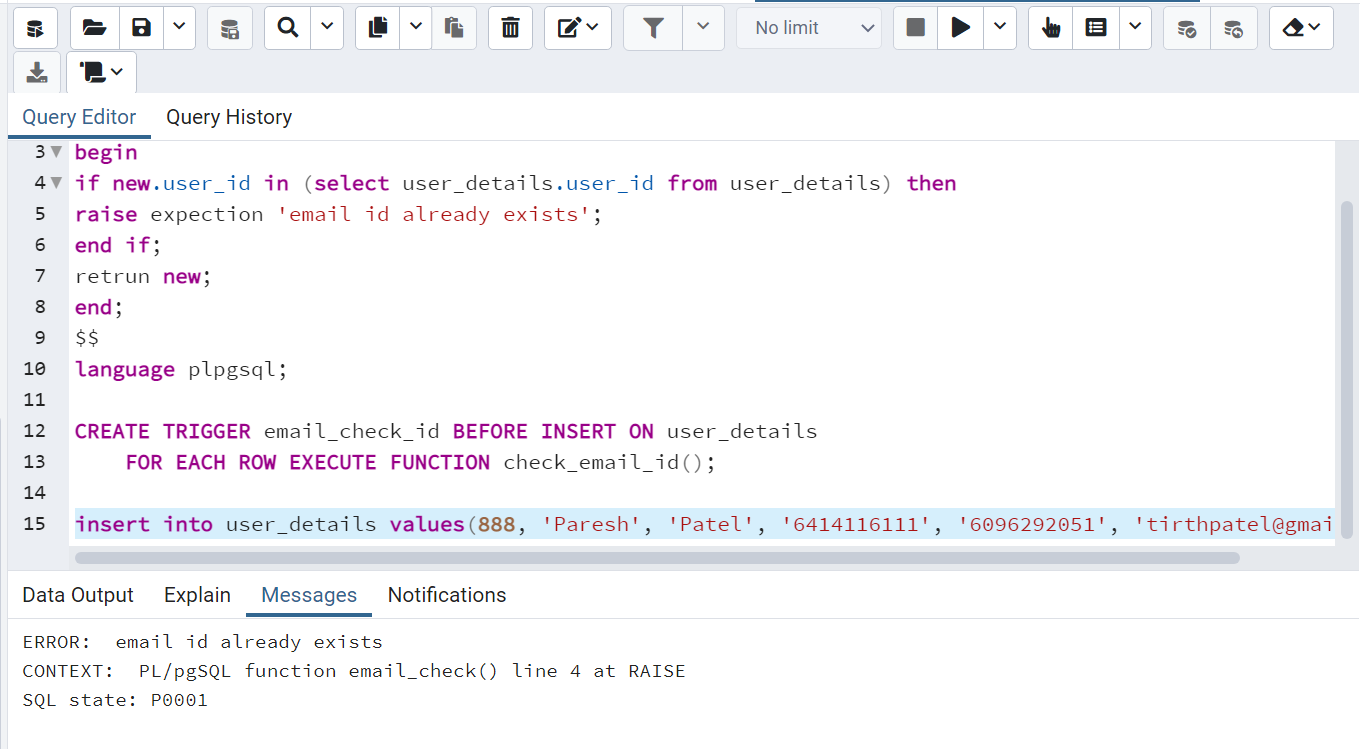
**Output:**



1. **Trigger to check whether the email id of the customer already exists or not. (A customer can not create multiple accounts using the same email id)**

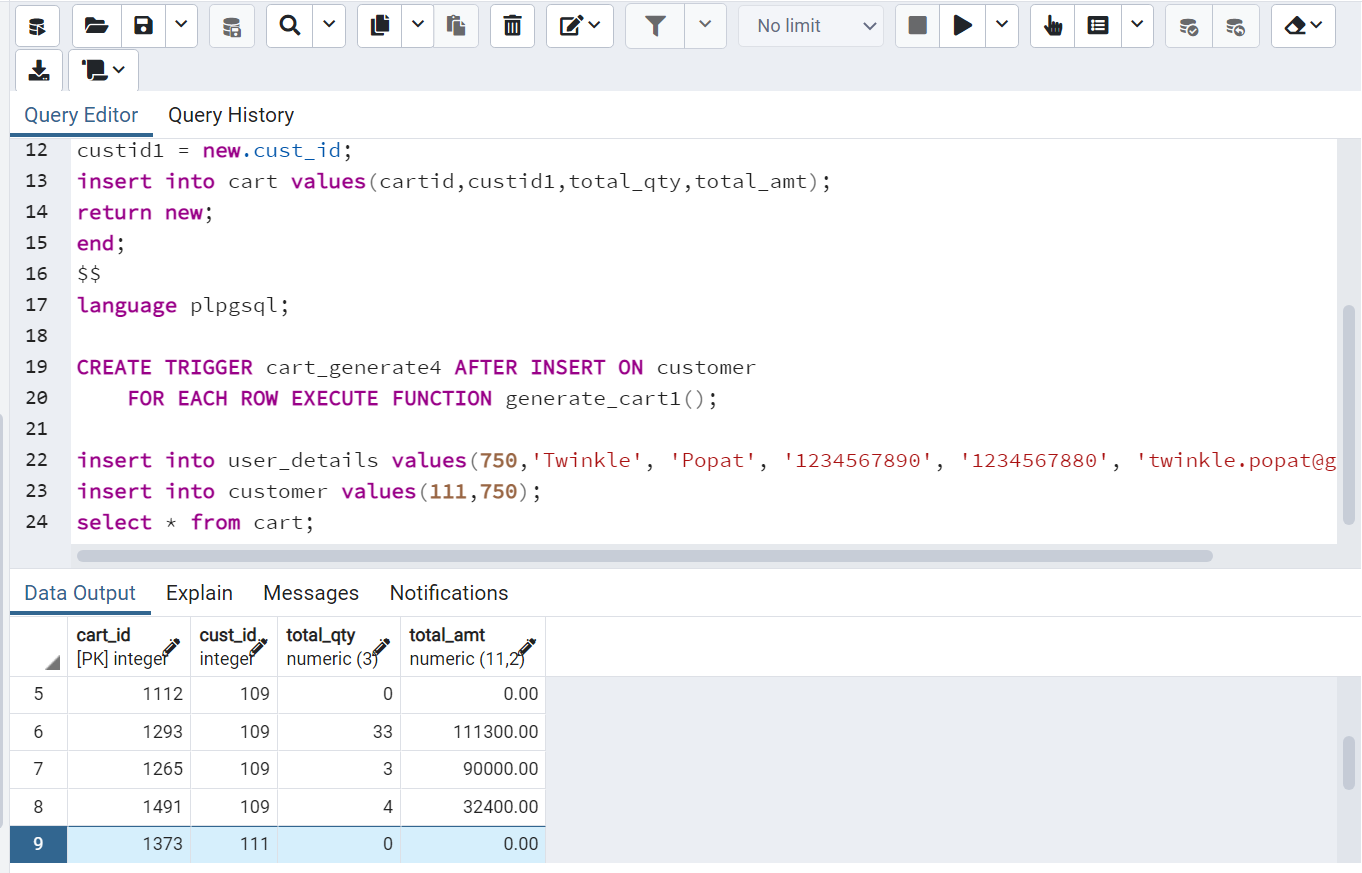
| create or replace function() email\_check returns trigger as  $$  begin  if new.user\_id in (select user\_details.user\_id from user\_details) then  raise expection 'email id already exists';  end if;  retrun new;  end;  $$  language plpgsql;  CREATE TRIGGER email\_check\_id BEFORE INSERT ON user\_details  FOR EACH ROW EXECUTE FUNCTION check\_email\_id();    insert into user\_details values(888, 'Paresh', 'Patel', '6414116111', '6096292051', 'tirthpatel@gmail.com'); |
| --- |

**Output:**



1. **Trigger to allot a cart to a customer whenever records are inserted into the customer table. We assign a cart id to the customer using a random function.**

| create or replace function generate\_cart1() returns trigger as  $$  declare  cartid int;  total\_amt int;  total\_qty int;  custid1 int;  begin  cartid = random()\*(1500-1100)+1100;  total\_amt = 0;  total\_qty = 0;  custid1 = new.cust\_id;  insert into cart values(cartid,custid1,total\_qty,total\_amt);  return new;  end;  $$  language plpgsql;  CREATE TRIGGER cart\_generate4 AFTER INSERT ON customer  FOR EACH ROW EXECUTE FUNCTION generate\_cart1();    insert into user\_details values(750,'Twinkle', 'Popat', '1234567890', '1234567880', 'twinkle.popat@gmail.com');  insert into customer values(111,750);  select \* from cart; |
| --- |

**Output**

1. **Trigger to update the values in the cart table whenever a product is added to cart product table.**

| create or replace function insert\_cart() returns trigger as  $$  declare  prod\_price decimal(10,2);  prod\_qnty int;  cart\_qnty int;  amt decimal(10,2);  begin  select product.price into prod\_price from product where product.prod\_id=new.prod\_id;  select cart.total\_amt into amt from cart where cart.cart\_id=new.cart\_id;  prod\_qnty=new.qty;  amt=amt+(prod\_price\*prod\_qnty);  select cart.total\_qty into cart\_qnty from cart where cart.cart\_id=new.cart\_id;  cart\_qnty=cart\_qnty+prod\_qnty;  update cart set total\_qty=cart\_qnty where cart.cart\_id=new.cart\_id;  update cart set total\_amt=amt where cart.cart\_id=new.cart\_id;  return new;  end;  $$  language plpgsql;  CREATE TRIGGER cart\_insert3 BEFORE INSERT ON cart\_product  FOR EACH ROW EXECUTE FUNCTION insert\_cart();  insert into cart\_product values(1373, 301, 7);  select \* from cart; |
| --- |

**Output:**