USE CASE STUDY REPORT

Group No.: Group 17

Student Names: Maulik Shah and Ruchika Kheria

Executive Summary

Our goal was to implement payment gateway system that protects payment information by encrypting sensitive information, such as card details, to ensure that information is passed securely between a customer and the payment processor. Therefore, a database management system is required that will store all the information regarding the customer, merchant, orders and payment authorization to ensure streamlined flow of information. The generic flow followed is that a customer can create their account on the merchant website and decide whether they would like to make a purchase or not. For any order placed, our system keeps a track of the order ID, shipping address and price of the total transaction. The transaction is then proceeded towards the payment mode where customer can pay via card or digital wallet or partial payment using both. The information is then passed on to the payment authorization system where the merchant transfers the card transaction amount to the customer bank for payment initiation. The customer bank approves/declines the transaction and this status along with a payment receipt is sent to the merchant which is forwarded back to the customer.

To implement this process flow, initially EER and UML diagram was designed to understand the information flow and account for the design rules and constraints. We made use of Specialization to split the payment mode into Card and Digital Wallet and Aggregation to determine a relationship between card transaction and authorization which includes its own attributes and relationships. This conceptual model was then mapped to relational model to determine the tables, constraints, primary and foreign keys so that model can be implemented on real-time data.

MySQL and NoSQL implementation helped in understanding customer statistics and their buying habits from merchant's perspective. After creating the necessary tables and collections and inserting records into them, we ran a few queries to understand which merchants have most customers, recent order placed by each customer, average spending of each customer, etc. To implement the database access, we connected our schema from MySQL to Python in Jupyter Notebook. We created few visualizations by executing the necessary SQL queries and using extensive Python libraries such as matplotlib, pandas and NumPy to infer customer's purchasing behavior and perform analysis based on the merchant requirements.

In conclusion, we were able to successfully implement the complete payment gateway system. Few modifications and improvements can be made to enhance the model such as incorporating third party applications and process flow for digital wallet transactions, taking in-store/offline transactions into consideration, adding variety of payment modes such as Apple Pay, Google Pay, Cash, Gift Vouchers, etc. and considering various other aspects of system such as signup/register, shipping and tracking information, etc. Overall, this model can be improved and implemented on real-time data and scenarios to streamline the whole buying process and understanding customer preferences.

I. Introduction

A Payment Gateway is an e-commerce application service provider that provides tools to process a payment between a customer, merchant and banks over the World Wide Web (WWW). It helps secure a purchase and a customer's payment information in a transaction. A payment gateway protects payment information by encrypting sensitive information, such as credit/debit card details, to ensure that information is passed securely between a customer and the payment processor. Besides encrypting the payment information, a payment gateway also helps in authorizing payments and protect against financial frauds. Therefore, a database needs to be created which will store all the information about the customer, order details, mode of payment, merchant and customer bank details. A customer can pay with either his/her card or there is an added feature called "Digital Wallet". Customers have the flexibility to add money to the wallet and directly pay with the wallet.

Process Flow

1. The customer places the order on the online portal/in-store, and an order id is assigned for each order to be placed.
2. The consumer selects a payment method to pay for the order and a transaction is initiated.
3. An authorization request, containing the card payment details along with the transaction id, will be provided to the payment processing system.
4. The system redirects the authorization request to the relevant acquiring channel.
5. The acquirer then redirects the authorization request to the customer's card bank.
6. The card-issuing bank processes the transaction and returns authorization response back to the acquirer.
7. If the authorization is successful, then the card-issuing bank transfers the fund to the merchant's account.
8. Once, the transaction is made, the transaction status (Completed or Failure) is sent back to the system.
9. The system sends back the response to the customer and if it is completed, the order will be placed successfully.

II. Conceptual Data Modeling

Design and Rules

- Each customer can place one order at a time but can place multiple orders in any given time frame. Price, shipping address and date of order will be stored for each order placed. Multiple orders can have different shipping addresses.
- Payment mode can either be via card or digital wallet as recorded by our system or it can be via cash which is monitored separately by the merchant.
- In our system, we have allowed partial payment. A customer can pay partially via digital wallet and remaining amount due via card. Hence cardinality between order and payment mode is one to many as 1 order can have multiple payment modes.
- A separate payment authorization aggregation has been added to monitor payment authorization between the merchant and customer bank.

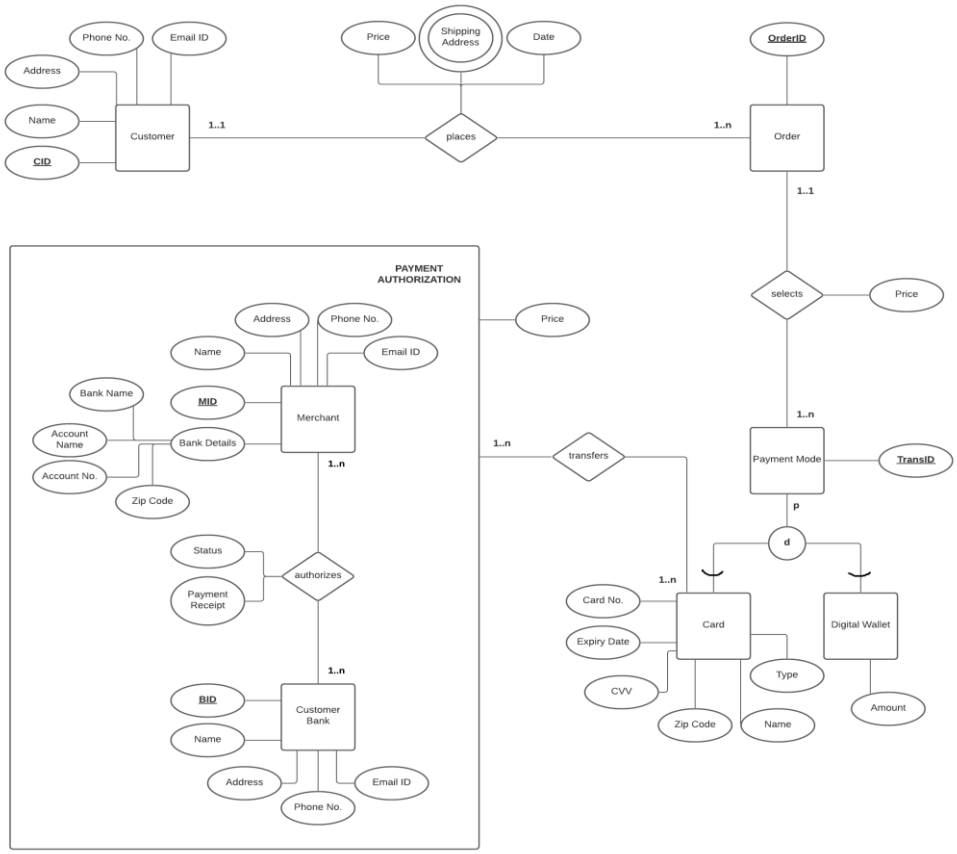
- A merchant can have multiple customer banks and multiple transactions happening at any given time. Similarly, a customer bank can have multiple merchants based in different locations. Status and payment receipt are recorded to keep a track of money sent between customer bank and merchant based on merchant's bank details.
- Multiple cards can be used for payment authorization as a customer can use someone else's card other than their own.
- Every relationship has a price attribute to it as it is important to keep track of the amount during the whole payment process.

Constraints / Domain

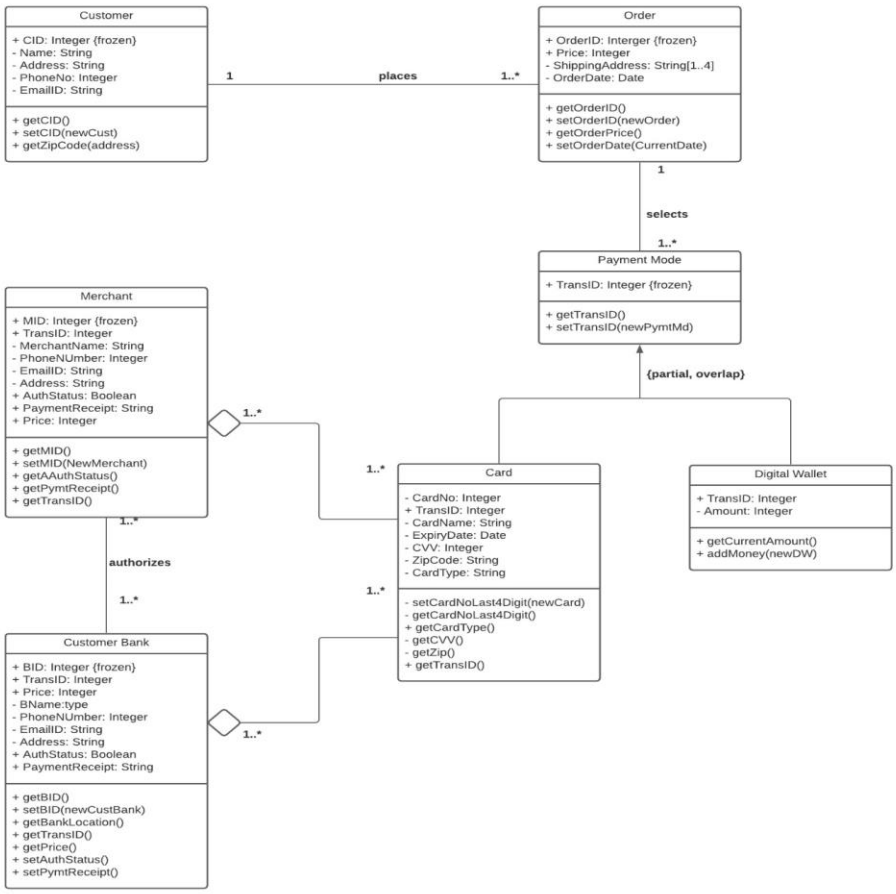
- Card number (can store only last 4 digits)
- Partial payment only via digital wallet and card
- Cannot demonstrate the continuous flow of the data within the system

EER Model

Shipping Address is multivalued as customers can place multiple orders with different shipping address. Specialization is used to categorize Payment Mode as Card and Digital Wallet. Our model is limit to payment system and transactions pertaining to card; hence, digital wallet hasn't been taken into consideration. Aggregation is also used to determine a relationship between card transaction and authorization of the card which includes its own attributes and relationships.



UML Diagram



III. Mapping Conceptual Model to Relational Model

CUSTOMER (CID, Name, Address, Phone No., Email ID)

ORDER (OrderID, CID, Price, Date)

- Foreign Key CID refers to CID in CUSTOMER, NOT NULL

ORDER_SHIPPING (Shipping Address, OrderID)

- Foreign Key OrderID refers to OrderID in ORDER, NOT NULL

PAYMENT_MODE (TransID, OrderID, Price)

- Foreign Key OrderID refers to OrderID in ORDER, NOT NULL

CARD (TransID, Card No., Expiry Date, CVV, Zip Code, CardHolderName, CardType, Amount)

- Foreign Key TransID refers to TransID in PAYMENT_MODE, NOT NULL

DIGITAL WALLET (TransID, Amount)

- Foreign Key TransID refers to TransID in PAYMENT_MODE, NOT NULL

MERCHANT (MID, Name, Address, Phone No., Email ID, Bank Name, Account Name, Account No., Zip Code))

CUSTOMER_BANK (BID, CID, Bank Name, Address, Phone No., Email ID)

- Foreign Key CID refers to CID in CUSTOMER, NOT NULL

AUTHORIZES (MID, BID, Confirm Status, Payment Receipt)

- Foreign Key MID refers to MID in MERCHANT, NOT NULL
- Foreign Key BID refers to BID in CUSTOMER_BANK, NOT NULL

PAYMENT_AUTHORIZATION (MID, BID, TransID, Price)

- Foreign Key MID refers to MID in MERCHANT, NOT NULL
- Foreign Key BID refers to BID in CUSTOMER_BANK, NOT NULL
- Foreign Key TransID refers to TransID in PAYMENT_MODE, NOT NULL

IV. Implementation of Relational Model via MySQL and NoSQL

Implementation of Relational Model in MySQL

Based on the Relational Mapping, 10 tables as mentioned above were created in MySQL implementation and 40-50 records were inserted in each table. The create and insert queries can be referenced in Appendix A.

Query 1: How many customers are present for each merchant?

```
select m.mid, m.mname, count(*) as customercount
from customer c
join customerbank cb
on c.cid = cb.cid
join authorizes a
on cb.bid = a.bid
join merchant m
on a.mid = m.mid
group by mid, mname
order by mid;
```

mid	mname	customercount
1	Campus Chai	7
2	Wollaston	5
3	Qdoba	6
4	UHop	4
5	Boston Shawarma	6
6	Panera	5
7	Dunkins	2

Query 2: Retrieve customer details whose transactions declined.

```
select c.cid, c.name, p.transid, a.confirmstatus, a.paymentreceipt from customer c
join customerbank cb
on c.cid = cb.cid
join authorizes a
on cb.bid = a.bid
join paymentauthorization p
on a.mid = p.mid and a.bid = p.bid
where a.confirmstatus = 'Declined'
order by c.cid;
```

cid	name	transid	confirmstatus	paymentreceipt
5	Francklyn MacShirie	8	Declined	1230457887
8	Poul Lindenboim	25	Declined	7814282659
12	Thadeus Nyssen	3	Declined	9043819778

Query 3: Count of transactions for each payment mode type (using recursive query).

```
with cte as
(
select p.transid as payment_transid, c.transid as card_transid, d.transid as wallet_transid
from paymentmode p
left join digitalwallet d
on p.transid = d.transid
left join card c
on p.transid = c.transid
)
select 'Card' as PaymentMode, count(*) as 'No of Transactions' from cte
where wallet_transid is null
union
select 'Wallet' as PaymentMode, count(*) as 'No of Transactions' from cte
where card_transid is null
union
select 'Both Card & Wallet' as PaymentMode, count(*) as 'No of Transactions' from cte
where wallet_transid is not null and card_transid is not null;
```

PaymentMode	No of Transactions
Card	27
Wallet	25
Both Card & Wallet	8

Query 4: Total amount earned by each merchant based on card transactions, ordered by highest amount.

```
select a.mid, m.mname, m.bankname, sum(price) as TotalAmount
from paymentauthorization pa
join authorizes a
on pa.mid = a.mid and pa.bid = a.bid
join merchant m on a.mid = m.mid
where a.confirmstatus = 'Approved'
group by a.mid
order by 4 desc;
```

mid	mname	bankname	TotalAmount
1	Campus Chai	Bank of America	337
2	Wollaston	Bank of America	306
5	Boston Shawarma	Chase	280
4	UHop	Santander	159
6	Panera	Santander	137
3	Qdoba	Chase	108
7	Dunkins	Bank of America	83

Query 5: Correlated SQL query that returns the top 2 most orders.

```
select c.cid, c.name, count(*) as num from orders o
join customer c on o.cid = c.cid
where c.cid in
(       select cid from customer
        where cid in
        (       select m1.cid from
                (select cid, count(*) as num from orders group by cid) as m1
                where 2 >
                (       select count(*) from
                        (select cid, count(*) as num from orders group by cid) as m2
                        where m1.num < m2.num
                )))
group by c.cid;
```

cid	name	num
1	Ly Shieber	9
8	Poul Lindenboim	5
12	Thadeus Nyssen	5

Query 6: Order and payment price for each customer.

```
select o.cid, o.orderid, p.price as totalprice, c.amount as
cardamount, d.amount as digitalwalletamount
from paymentmode p
left join card c
on p.transid = c.transid
left join digitalwallet d
on p.transid = d.transid
join orders o
on p.orderid = o.orderid
order by o.cid;
```

cid	orderid	totalprice	cardamount	digitalwalletamount
1	6	92	92	NULL
1	11	69	69	NULL
1	12	55	55	NULL
1	21	88	88	NULL
1	22	16	16	NULL
1	24	77	77	NULL
1	26	11	11	NULL
1	29	43	23	20
1	56	69	NULL	69
2	35	67	35	32
2	44	83	NULL	83
2	47	60	NULL	60
2	49	83	NULL	83
3	20	38	38	NULL
3	43	82	NULL	82
3	53	85	NULL	85
4	5	37	37	NULL

Query 7: View months passed between today and latest order placed by each customer.

```
select c.cid, o.orderid, o.orderdate, min(timestampdiff(month,
o.orderdate, sysdate())) as monthspassedbeforelatestorder
from orders o
join customer c
on o.cid = c.cid
group by c.cid
order by c.cid;
```

cid	orderid	orderdate	monthspassedbeforelatestorder
1	6	2021-03-02	1
2	35	2021-05-12	0
3	20	2021-08-30	3
4	5	2021-06-27	5
5	8	2021-08-31	3
6	14	2021-10-16	1
7	10	2021-09-18	1
8	25	2021-03-13	6
9	28	2021-12-02	0
10	4	2021-07-27	1
11	9	2021-08-16	3
12	3	2021-09-25	2
13	19	2021-05-07	1
14	2	2021-09-03	3
15	1	2021-05-06	0

Query 8: Customers who haven't placed any order.

```
select * from customer
where cid not in
(select cid
from orders);
```

cid	name	address	phoneno	emailid
16	Lindsey Cornier	5515 Barnett Alley	962-848-7547	lcornierf@whitehouse.gov
17	Cher Appleford	67 Burrows Parkway	258-378-1166	capplefordg@de.vu
18	Haskell Baskerfield	958 Myrtle Road	203-503-2626	hbaskerfieldh@addtoany.com
19	Amber Brockie	6889 Ludington Terrace	741-353-7088	abrockiei@liveinternet.ru
20	Alvinia Hedney	580 Holy Cross Avenue	472-124-3481	ahedneyj@cnbc.com
21	Carroll Heymann	8 Loomis Park	170-784-5582	cheymannk@bigcartel.com
22	Bernie Shimmans	54 2nd Road	328-470-9038	bshimmansl@smh.com.au
23	Tabby Rozalski	72961 Ridgeway Center	694-702-4472	trozalskim@dyndns.org
24	Hirsch Twelves	7159 Kedzie Street	784-394-4832	htwelvesn@cdbaby.com
25	Nesta Worham	40881 Esker Street	308-432-2663	nworhamo@thetimes.co.uk
26	Fletcher Hellyar	81221 Scott Trail	957-872-1715	fhellyarp@indiatimes.com
27	Harwill Pulman	84824 Badeau Center	134-603-4997	hpulmanq@surveymonkey.com
28	Lewiss Laycock	7 Dahle Way	853-488-8709	llaycockr@alexa.com
29	Gregoor Oldroyd	64872 Delaware Center	609-591-9931	goldroyds@google.nl
30	Andras Mollett	38970 Pennsylvania Drive	236-300-8415	amollettt@disqus.com
31	Jo-ann O' Donohoe	5271 Eliot Terrace	972-168-1162	jou@springer.com
32	Elvis Elsay	97171 Warbler Hill	179-204-7731	eelsayv@jugem.jp

Query 9: Name and count of banks that merchant deals with for transactions and payment authorization.

	custbname	customerbankcount
▶	Chase	5
	Bank of America	4
	Santander	3
	Wells Fargo Bank	2
	Citi Bank	2
	TD Bank	1

```
select custbname, count(*) as customerbankcount
from customerbank
group by custbname
order by customerbankcount desc;
```

Implementation of Relational Model in NoSQL (MongoDB)

NoSQL has been implemented in MongoDB. We created 4 collections namely customer, orders, customerbank and card with 5-10 records in each collection. These queries can be referenced in Appendix B.

Query 1: Average price of orders placed by each customer without using a map-reduce pipeline.

```
db.orders.aggregate([
 { $group: { _id: "$cid", averageprice: { $avg: "$price" } } },
 { $out: "avg_price" }])
db.avg_price.find().sort( { _id: 1 } );
```

Result

```
{ "_id" : 1, "averageprice" : 41 }
{ "_id" : 2, "averageprice" : 65 }
{ "_id" : 3, "averageprice" : 84 }
{ "_id" : 4, "averageprice" : 50.666666666666664 }
{ "_id" : 5, "averageprice" : 40 }
```

Query 2: Name and count of banks that merchant deals with for transactions and payment authorization using map-reduce pipeline.

```
var mapFunction1 = function() {
 emit(this.custbname, {NoOfBanks:1});};
var reduceFunction1 = function(CustomerBank, val) {var value = {NoOfBanks:0}
 for (i=0; i<val.length; i++){ value.NoOfBanks += val[i].NoOfBanks; }
 return value;};
db.customerbank.mapReduce(
 mapFunction1,
 reduceFunction1,
 {out: "count_banks"})
db.count_banks.find().sort( { custbname: 1 } )
```

Result

```
{ "_id" : "Bank of America", "value" : { "NoOfBanks" : 2 } }
{ "_id" : "Chase", "value" : { "NoOfBanks" : 2 } }
{ "_id" : "Santander", "value" : { "NoOfBanks" : 1 } }
```

Query 3: Count of orders placed by each customer using map-reduce pipeline.

```
var mapFunction1 = function() {
 emit(this.cid, {NoOfOrders:1});};
var reduceFunction1 = function(CustID, val) {
 var value = {NoOfOrders:0}
 for (i=0; i<val.length; i++){value.NoOfOrders += val[i].NoOfOrders; }
 return value;};
db.orders.mapReduce(
```

Result

```
{ "_id" : 1, "value" : { "NoOfOrders" : 2 } }
{ "_id" : 2, "value" : { "NoOfOrders" : 1 } }
{ "_id" : 3, "value" : { "NoOfOrders" : 2 } }
{ "_id" : 4, "value" : { "NoOfOrders" : 3 } }
{ "_id" : 5, "value" : { "NoOfOrders" : 2 } }
```

```
 mapFunction1,
 reduceFunction1,
 {out: "count_orders"})
db.count_orders.find().sort( { cid: 1 } )
```

Query 4: Customer with highest average price without using a map-reduce pipeline.

```
db.orders.aggregate([
 { $group: { _id: "$cid", maxPrice: { $avg: "$price" } } },
 { $sort: {maxPrice: -1} },
 {$limit:1}]);
```

Result

```
{ "_id" : 3, "maxPrice" : 84 }
```

Query 5: View customer details for all orders placed (using join on orders and customer collection).

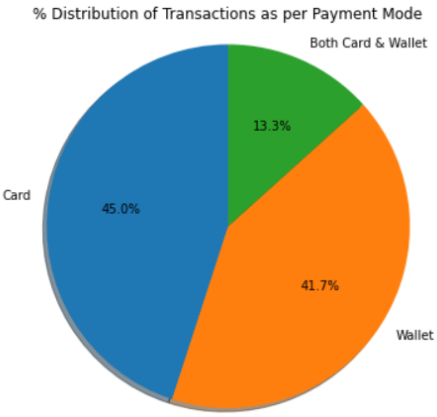
```
db.orders.aggregate([
{   "$lookup":
    {"from": "customer",
     "localField": "cid",
     "foreignField": "_id",
     "as": "customerinfo"}},
{$unwind:'$customerinfo'},
{$project:{customer_name:'$customerinfo.name',
        _id: 1,
        name: 1,
        price: 1,
        orderdate: 1    }}]);
```

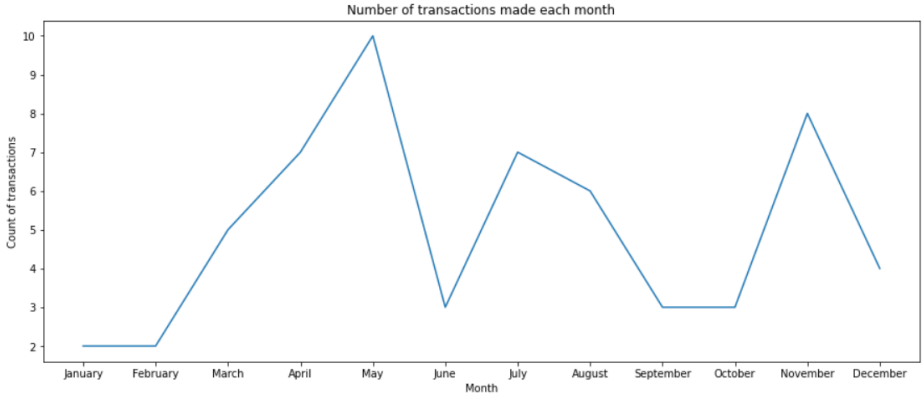
Result

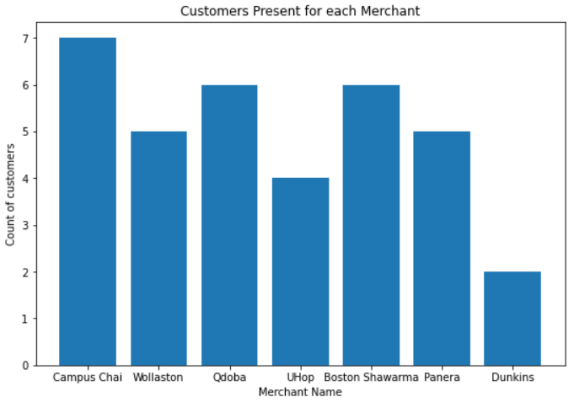
```
{ "_id" : 1, "price" : 45, "orderdate" : "2021-05-06", "customer_name" : "Francklyn MacShirie" }
{ "_id" : 2, "price" : 87, "orderdate" : "2021-03-02", "customer_name" : "Georgie Baudinet" }
{ "_id" : 3, "price" : 65, "orderdate" : "2020-12-16", "customer_name" : "Jacques Goscomb" }
{ "_id" : 4, "price" : 37, "orderdate" : "2021-04-26", "customer_name" : "Georgie Baudinet" }
{ "_id" : 5, "price" : 27, "orderdate" : "2021-11-15", "customer_name" : "Ly Shieber" }
{ "_id" : 6, "price" : 92, "orderdate" : "2021-12-02", "customer_name" : "Gilligan McCrillis" }
{ "_id" : 7, "price" : 35, "orderdate" : "2021-05-12", "customer_name" : "Francklyn MacShirie" }
{ "_id" : 8, "price" : 76, "orderdate" : "2021-03-02", "customer_name" : "Gilligan McCrillis" }
{ "_id" : 9, "price" : 28, "orderdate" : "2021-11-15", "customer_name" : "Georgie Baudinet" }
{ "_id" : 10, "price" : 55, "orderdate" : "2021-07-25", "customer_name" : "Ly Shieber" }
```

V. Database Access via R or Python

We implemented database access using Python and successfully established connection between our schema payment_gateway. Based on the dataset, we created following visualizations.







VII. Summary and recommendation

Our goal was to implement payment gateway system using MySQL, NoSQL and Python. The generic flow followed is that a customer can create their account on the merchant website and decide whether they would like to make a purchase or not. For any order placed, our system keeps a track of the order ID, shipping address and price of the total transaction. The transaction is then proceeded towards the payment mode where customer can pay via card or digital wallet or partial payment using both. The information is then passed on to the payment authorization system where the merchant transfers the card transaction amount to the customer bank for payment initiation. The customer bank approves / declines the transaction and this status along with a payment receipt is sent to the merchant which is forwarded back to the customer.

MySQL and NoSQL implementation helped in understanding customer statistics and their buying habits from merchant's perspective. We ran a few queries to understand which merchants have most customers, recent order placed by each customer, average spending of each customer, etc. We created few visualizations using Python to infer customer's purchasing behavior. Due to the limitation of the scope of the project, few shortcomings and recommendations are as follows:

- We only considered the payment system for card transactions. We didn't incorporate third party users for digital wallet. Adding separate entities for digital wallet payment method will make the system wholesome.
- Based on the information stored, only online transactions have been taken into consideration. We should look further into how offline (in-store) purchases can be accounted for.
- The payment modes can be improvised by introducing different payment modes such as Apple Pay, Google Pay, Cash, Gift Vouchers, etc.
- Lastly, our payment system only considers the payment and authorization stage of the whole buying process. The model can be elaborated further to include the whole start to end buying process such as signup/register, linking social media, shipping and tracking information, reviews, etc.

Overall, this model can be improved and implemented on real-time data and scenarios to streamline the whole buying process and understanding customer preferences.