

Leaf Groceries: A customer-oriented e-commerce model (B2C)

Chisako Tani

Xiaomin Lin

Sairaj Rege

Hua Lu

Our organization

Leaf Grocery Inc. is a grocery store based in the bay area. We advocate importing pure natural and organic vegetables and food to our customers. Our main sales method is online to offline, the model is B2C (Business to Customer), that is, selling products and services directly to consumers. Customers can order their favorite vegetables and food through their mobile phones, and delivery service will be provided for Leaf Grocery.

Online grocery shopping is a key player in the game of household efficiency. As lifestyle changes, the pace of life accelerates. More and more workers do not have time to purchase in supermarkets. Similarly, due to the impact of the COV19 in 2020, more people have to work at home. Such an environment has accelerated the development of e-commerce. Nowadays, people can generate their orders through their mobile phones. Buying alternatives online saves time and increases sales: because some people go to the supermarket to buy goods and need to ship them home, most people will choose goods according to their load-bearing capacity (especially people living in the city). However, e-commerce can ship goods directly at the door, so people's purchasing power also increases. Leaf Groceries mainly sells food, vegetables, fruits and daily necessities. Establishing e-commerce requires a complete data platform, so we wanted to design a database system for a retail manufacturer (Leafy Grocery Inc.), in order to understand the business rules and design a simple and easy to understand database & some supply chain systems that work inside a retail organization.

Entities Description

Customer: Customer_id (PK), Customer_Name, Customer_Phone, Customer_Address, Customer_City, Customer_State, Customer_Postal_Code, Date_became_customer. It helps us understand our customers better.

Payment: Statement_Date (PK), Supplier_id (PK, FK), Due_date, Status, Comments, Amount. It helps us understand how much we have to pay for each supplier every month.

Supplier: Supplier_id (PK), Supplier_Name, Supplier_Address, Contact_Name, Supplier_Phone, Supplier_Email. It explains how our supplier is related to other entities in our schema, also providing important supplier information for our business.

Products: Product_id (PK), Supplier_ID (FK), Product_Name, Price, Quantity, Comments. It helps us understand trends in the products by suppliers and to track them. Customers can see the product name and price on the website and choose what they buy.

Order: Order_id (PK), Customer_id (FK), Order_Status , Order_date, Comments. It explains basic order information made by customers. Order_Status contains the current status of the order such as Ordered, Shipped, Delivered.

Order_details: Order_id (PK, FK), Product_id (PK, FK), Quantity ordered, Price (Each item), Comments. Order_details exists as a result of full dependency between orders and products and explains what product and how many a customer purchased.

Business Rules

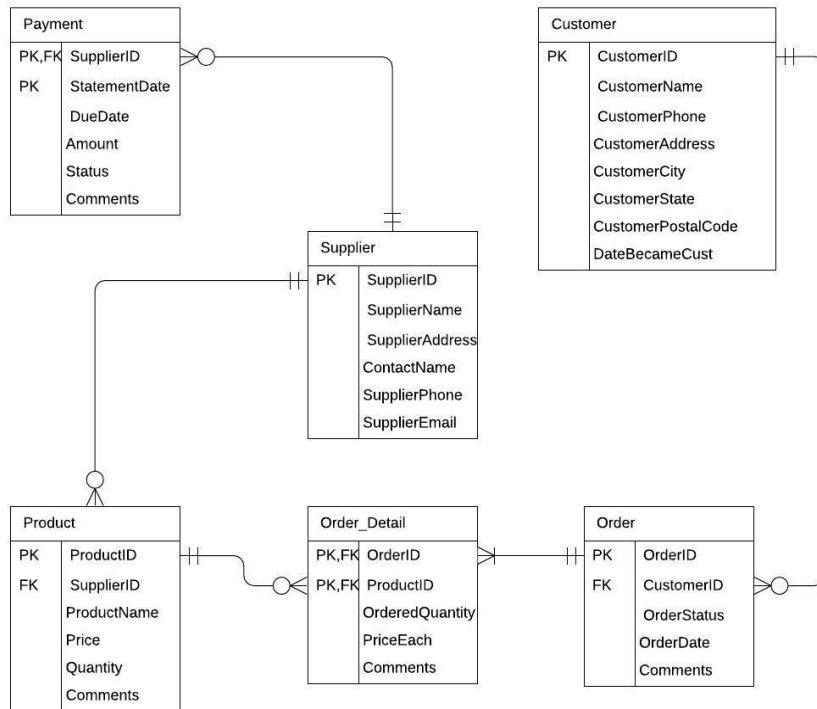
Business Side:

- Leaf Grocery has only one store.
- Leaf Grocery has several suppliers.
- Products data are stored in the Products table with id, name, price, quantity, and comments.
- Some products may not be contained in any order. (e.g. new products)
- Each supplier has a unique id, name, address, city, state, zip code, country, contact name, phone number, email.
- Payment information between Leaf Grocery and suppliers is managed by supplier id, statement date, due date, amount, status and comments.
- Payments are issued every month for each supplier.
- Some suppliers may have no product and no payment information. (e.g. new suppliers)

Customer Side:

- Each customer order only from the website
- Each customer has a unique id, name, phone number, address, city, state, zip code, note, registered date as a customer
- Each customer may have one or multiple orders
- Each order by a customer has id, customer id, order date, status, and comments
- Each order has at least one order detail, which has order id, product id, ordered quantity, price each, and comments.
- Some customers may have no online order (e.g. new customers)

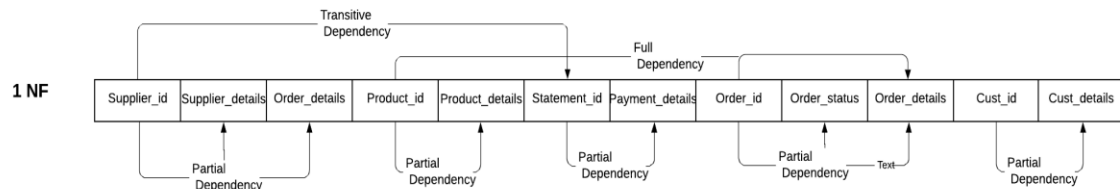
EER Diagram



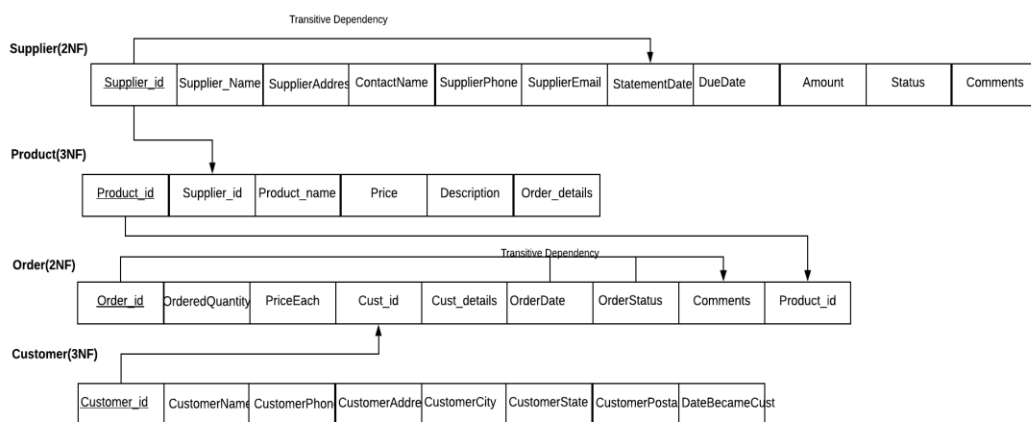
Relation Model (Normal Form 1, 2, and 3)

- We perform Database Normalization, to eliminate redundant data, and to check the data dependencies. The attributes “details”, contain multivalued attributes of the related entity in our Schema. Here, we observe Transitive dependency between “Supplier_id” & “Payment_id”. For e.g. the entity “Product Details” contain information like Product price, Product description, comments and Order Details, that are related to the Entity Primary key “Product_id”. Due to full dependency between entities Order & Product, we get an additional entity named Order_details.
- The Primary keys help linking the entities, thus forming a Normalized Relationship between our entities.
- In 1 NF, we make sure that our entries are not duplicated and the anomalies are removed.
- In the 2NF form, remove the partial dependencies in our model.

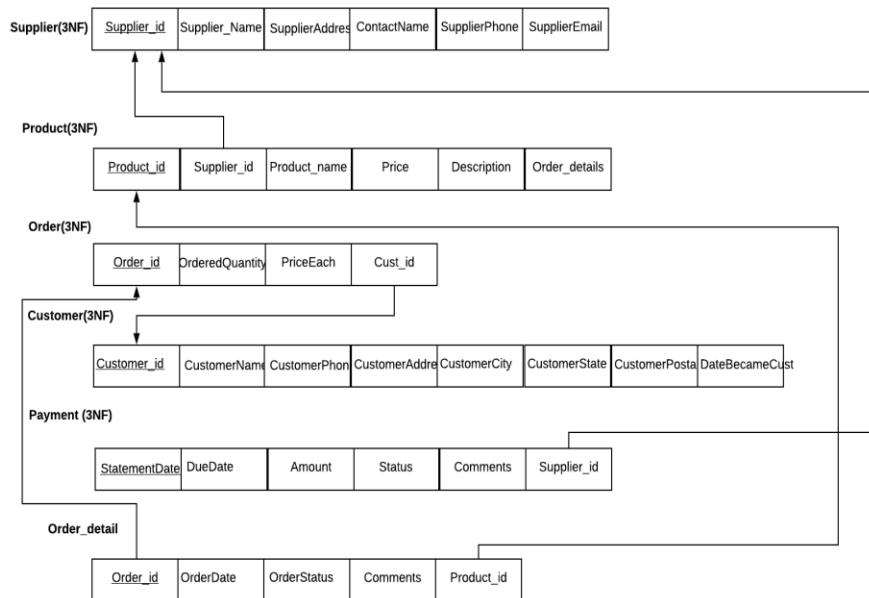
- Finally, in the 3rd NF, we check for any transitive dependencies. In our case, we have 2 transitive dependencies in our 2 NF form, so we remove them by creating new entities, i.e. "Order_details" & "Payment".



2 NF



3 NF



SQL Statements:

Data Definition Queries and Loading Data

Our data extracted from multiple excel documents which we re-loaded into 6 different key tables of Leaf Groceries: customers, payment, product, supplier, orders and order details. SQL queries that create all our tables are following:

```
CREATE TABLE `Customer` (  
  `CustomerID` INT(8) NOT NULL,  
  `CustomerName` VARCHAR(20) NOT NULL,  
  `CustomerPhone` VARCHAR(10) NOT NULL,  
  `CustomerAddress` VARCHAR(30) NOT NULL,  
  `CustomerCity` VARCHAR(20) NOT NULL,  
  `CustomerState` CHAR(2) NOT NULL,  
  `CustomerPostalCode` VARCHAR(5) NOT NULL,  
  `DateBecameCust` DATE NOT NULL,  
  CONSTRAINT `Customer_PK` PRIMARY KEY (`CustomerID`)  
);
```

```
CREATE TABLE `Supplier` (  
  `SupplierID` INT(8) NOT NULL,  
  `SupplierName` VARCHAR(45) NOT NULL,  
  `SupplierAddress` VARCHAR(100) NOT NULL,  
  `ContactName` VARCHAR(20) DEFAULT NULL,  
  `SupplierPhone` VARCHAR(10) NOT NULL,  
  `SupplierEmail` VARCHAR(45) NOT NULL,  
  CONSTRAINT `Supplier_PK` PRIMARY KEY (`SupplierID`)  
);
```

```
CREATE TABLE `Product` (  
  `ProductID` INT(8) NOT NULL,  
  `SupplierID` INT(8) NOT NULL,  
  `ProductName` VARCHAR(100) NOT NULL,  
  `Price` DECIMAL(5,2) NOT NULL,  
  `Quantity` INT(5) NOT NULL,  
  `Comments` VARCHAR(100) DEFAULT NULL,  
  CONSTRAINT `Product_PK` PRIMARY KEY (`ProductID`),  
  CONSTRAINT `Product_FK1` FOREIGN KEY (`SupplierID`) REFERENCES `Supplier`  
  (`SupplierID`)  
);
```

```
CREATE TABLE `Order` (  
  `OrderID` INT(8) NOT NULL,  
  `CustomerID` INT(8) NOT NULL,  
  `OrderStatus` VARCHAR(30) NOT NULL,  
  `OrderDate` DATE NOT NULL,  
  `Comments` VARCHAR(100) DEFAULT NULL,
```

```

CONSTRAINT `Order_PK` PRIMARY KEY (`OrderID`),
CONSTRAINT `Order_FK1` FOREIGN KEY (`CustomerID`) REFERENCES `Customer`
(`CustomerID`)
);

```

```

CREATE TABLE `Order_Detail` (
  `OrderID` INT(8) NOT NULL,
  `ProductID` INT(8) NOT NULL,
  `OrderedQuantity` INT(5) NOT NULL,
  `PriceEach` DECIMAL(5,2) NOT NULL,
  `Comments` VARCHAR(100) DEFAULT NULL,
  CONSTRAINT `OrderDetail_PK` PRIMARY KEY (`OrderID`,`ProductID`),
  CONSTRAINT `OrderDetail_FK1` FOREIGN KEY (`OrderID`) REFERENCES `Order`
(`OrderID`),
  CONSTRAINT `OrderDetail_FK2` FOREIGN KEY (`ProductID`) REFERENCES `Product`
(`ProductID`)
);

```

```

CREATE TABLE `Payment` (
  `SupplierID` INT(8) NOT NULL,
  `StatementDate` DATE NOT NULL,
  `DueDate` DATE NOT NULL,
  `Amount` DECIMAL(8,2) NOT NULL,
  `Status` VARCHAR(30) NOT NULL,
  `Comments` VARCHAR(100) DEFAULT NULL,
  CONSTRAINT `Payment_PK` PRIMARY KEY (`SupplierID`,`StatementDate`),
  CONSTRAINT `Payment_FK1` FOREIGN KEY (`SupplierID`) REFERENCES `Supplier`
(`SupplierID`)
);

```

Then, we collected and compiled data into the excel sheets to keep consistency and follow constraints we defined as business rules and EER diagram. Following Insert statements are the part of queries for each table. We had to be careful of the order to execute queries because Order, Order detail, and Payment table refer to other tables as foreign keys such as Customer, Product, Supplier. The excel and the full set of queries are available on the shared storage: https://drive.google.com/file/d/1a0vrm87JhIRUb9Pj-pB_44AEot0a4_V/view?usp=sharing

```

INSERT INTO `leaf_db`.`Customer` (
  `CustomerID`,`CustomerName`,`CustomerPhone`,`CustomerAddress`,`CustomerCity`,`Customer
State`,`CustomerPostalCode`,`DateBecameCust` ) VALUES ("10000001","Krystal
Elliott","8832990792","87 Harrison St.,""Thomasville","NC","27360","2015/05/25");

```

```

INSERT INTO `leaf_db`.`Supplier` (
  `SupplierID`,`SupplierName`,`SupplierAddress`,`ContactName`,`SupplierPhone`,`SupplierEmail` )
VALUES ("20000001","Davis, Hettinger and Veum","7034 Sage St. Pottstown, PA 19464","Lilia
Wilson","5362695964","jacques.brueen@schowalter.com");

```



```
INSERT INTO `leaf_db`.`Product` (
  `ProductID`,`SupplierID`,`ProductName`,`Price`,`Quantity`,`Comments`
)
VALUE ("30000001","20000001","Coca-Cola of Mexico","0.96","2","Best by 7/18");
```

```
INSERT INTO `leaf_db`.`Order` ( `OrderID`,`CustomerID`,`OrderStatus`,`OrderDate`,`Comments` )
VALUE ("80000001","10000001","Delivered","2019/08/16","Paypal");
```

```
INSERT INTO `leaf_db`.`Order_Detail` (
  `OrderID`,`ProductID`,`OrderedQuantity`,`PriceEach`,`Comments`
)
VALUE ("80000001","30000004","4","1.17","Coupon applied");
```

```
INSERT INTO `leaf_db`.`Payment` (
  `SupplierID`,`StatementDate`,`DueDate`,`Amount`,`Status`,`Comments`
)
VALUE ("20000001","2019/11/01","2019/12/01","21223.12","Issued","For Oct");
```

Data Retrieval Queries

Each table contains several hundred rows of data that are joined by primary key and foreign keys. I will show some SQL queries to explain the different uses of databases.

- 1. First, we want to find out which customers have bought milk products. There are many different kinds of milk products, such as chocolate milk or skim milk. We need to do a keyword search to get information. In this query, I search for the user id, order id, and product name. First, merge the order table and product table to obtain the data containing Milk% or %Milk%, and then connect with the user table to obtain the user id. Through the results, we found that there are two orders containing milk products.

```
1 use leaf_db;
2
3 Select
4   CustomerID, O.OrderID, ProductName
5 From
6   `Order` as O
7 Join
8   (
9     Select OrderID, Order_Detail.ProductID as ProductNum, ProductName
10    From Order_Detail
11   Left Join Product
12    On Order_Detail.ProductID = Product.ProductID
13   Where ProductName like 'Milk%'
14   ) tb
15 On
16   O.OrderID = tb.OrderID;
```

CustomerID	OrderID	ProductName
10000006	80000010	Milk, Daigold, heavy whipping cream
10000021	80000029	Milk, Kirkland Signature, 2% reduced-fat

Result 12: 2 row(s) returned

(1)

```
1 use leaf_db;
2
3 Select
4   Distinct S.SupplierName, Count(distinct ProductID) as Product_Cnt
5 From
6   Supplier as S, Product as P
7 Where
8   S.SupplierID = P.SupplierID
9 Group by
10  1
11 Order by
12  2 desc;
```

SupplierName	Product_Cnt
Schiller, Predovic and Lang	21
Anderson Ltd	21
Davis, Hattinger and Veum	21
Ernsler, Burton and Glover	21
Brokie-Witting	20
Bergstrom Group	20
Koslpin, Stiedemann and Hammes	20
Little, Ward and Hegmann	19

Result 11: 49 row(s) returned

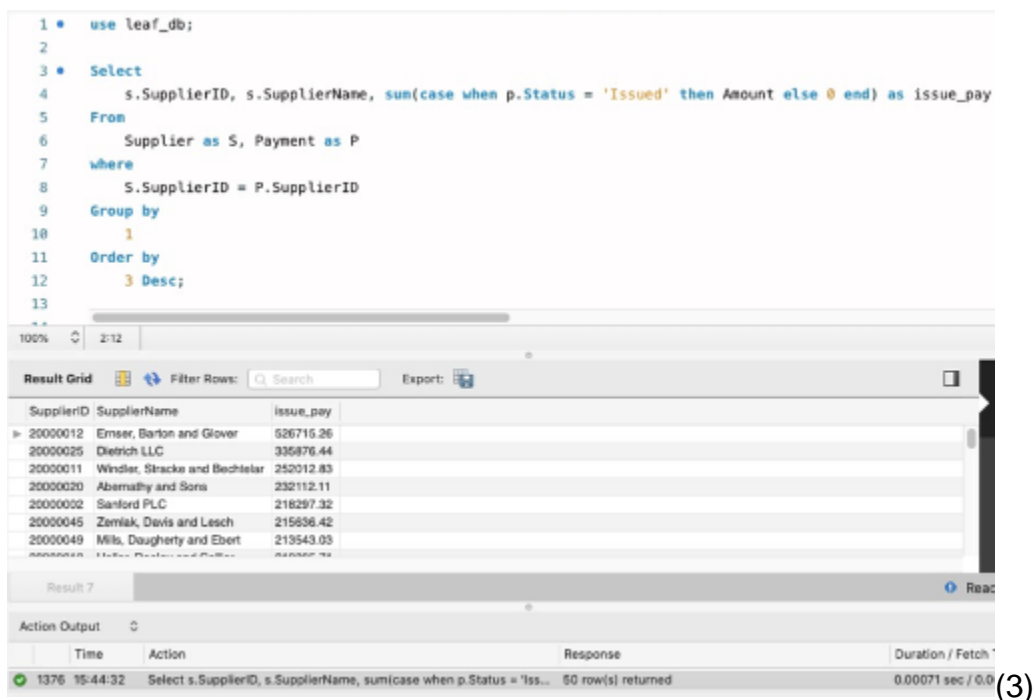
(2)

- 2. Second, we want to show which suppliers we import the most products from, and these suppliers will become our long-term major partners. Therefore, we reorder by selecting distinct supplier names, and count unique productID. By linking the supply table and the product table, we ended up with four suppliers providing us with the most merchandise (21 items).
- 3. In the third query, we want to get information about the supplier and the order. Since we place an order with a supplier, an invoice can be issued in one of three types -- "issued", "delayed", and "paid" -- representing three different states. When we want to get the "issued" payment information, we select supplierID, supplierName and create a flag column: when the state is issued, show amount, otherwise show 0, and sum all amounts by supplierID. The flag column eventually returns the sum of all amounts with status of issue.

```

1 * use leaf_db;
2
3 * Select
4   s.SupplierID, s.SupplierName, sum(case when p.Status = 'Issued' then Amount else 0 end) as issue_pay
5 From
6   Supplier as S, Payment as P
7 where
8   S.SupplierID = P.SupplierID
9 Group by
10  1
11 Order by
12  3 Desc;
13

```



SupplierID	SupplierName	issue_pay
20000012	Emser, Barton and Glover	526715.26
20000025	Dietrich LLC	336876.44
20000011	Windler, Stracks and Bechtelar	252012.83
20000020	Abernathy and Sons	232112.11
20000002	Sanford PLC	218297.32
20000045	Zemlak, Davis and Lesch	215856.42
20000049	Mills, Daugherty and Ebert	213543.03

Result 7

Action Output

Time	Action	Response	Duration / Fetch
1376 15:44:32	Select s.SupplierID, s.SupplierName, sum(case when p.Status = 'Iss...	80 row(s) returned	0.00071 sec / 0.0

(3)

- 4. The classification of orders is very important in e-commerce. When the order of the same area reaches a certain amount, it is cheaper to choose bulk logistics. When we want to search for orders from customers in the same area, we first create a new table that contains the customerID and the number of orders from customers. The status of these orders is stopped in "ordered" status and not yet

become "shipped". The customers who live in iowa are filtered through the where statement by merging the complete customer information table. The results showed that two customers living in iowa had unshipped orders.

```

1 * use leaf_db;
2
3 * Select
4   C.CustomerID, CustomerName, CustomerCity, CustomerState
5   From
6     "Customer" as C
7   Join
8     (
9       Select CustomerID, Count(distinct OrderID) as Cnt
10      From "Order"
11     Where OrderStatus = 'Ordered'
12     Group by CustomerID
13     Having Count(distinct OrderID) is NOT NULL
14     ) tb
15   On
16     C.CustomerID = tb.CustomerID
17   Where
18     CustomerState = 'IA';
19

```

CustomerID	CustomerName	CustomerCity	CustomerState
10000013	Wyatt Sosa	Iowa City	IA
10000009	Pedro Wallace	Des Moines	IA

Result 18

Action Output

Time Action Response

31 10:42:43 Select C.CustomerID, CustomerName, CustomerCity, CustomerState From "Cust... 2 row(s) returned

(4)

```

1 * use leaf_db;
2
3 * Select
4   CustomerID, CustomerName, CustomerAddress, CustomerCity, CustomerName, tbl.amount_rank
5   From
6     "Customer" C
7   Right Join
8     (
9       Select
10        distinct O.CustomerID as Cus_id, Sum(tb.sum_price) as amount_rank
11      From
12        "Order" AS O
13      Join (
14        Select OrderID, Sum(Round(OrderedQuantity * PriceEach, 2)) as sum_price
15        From Order_Detail
16        Group by OrderID
17      ) tb
18      On
19        O.OrderID = tb.OrderID
20      Group by
21        O.CustomerID
22      Order by
23        2 desc
24      ) tbl
25   On
26     C.CustomerID = tbl.Cus_id;
27

```

CustomerID	CustomerName	CustomerAddress	CustomerCity	CustomerName	amount_rank
10000042	Brynlee Bright	506 West Roosevelt Court	Litton	Brynlee Bright	89.32
10000003	Kalley Maddox	84 Kirkland St.	Moorhead	Kalley Maddox	53.00
10000053	Victor Kihl	389 Bear Hill St.	Zeland	Victor Kihl	51.50
10000029	Pedro Wallace	175 Shirley Street	Des Moines	Pedro Wallace	50.80
10000034	Lina Rocha	20 Bassett Lane	Corona	Lina Rocha	48.28
10000049	Jaycee Odum	7817 Addison Court	Greer	Jaycee Odum	47.89
10000028	Jeffery Love	48 South Meadow St	Muncie	Jeffery Love	47.68
10000088	Mauro Giral	930 Cemetery St	Carmel	Mauro Giral	44.71

Result 19

Action Output

Time Action Response

31 10:42:43 Select C.CustomerID, CustomerName, CustomerCity, CustomerState From "Cust... 2 row(s) returned

(5)

- 5. In this query, we want to find out which customer ordered the highest total amount from our website. We will reward this customer by providing coupons. By calculating the order id and the total order amount, we combined the customer information table and got the result that the customer named Bryn had the highest consumption from our website.
- 6. We also define day 1 retention of some date X to be the number of customers whose first order date is X and they logged back and placed order on the day right after X, divided by the number of customers whose first date is X, rounded to 2 decimal places. This is the method of calculating the retention rate in e-commerce, but since our data volume is only for presentation, there is no retained information.
- 7. I wrote an SQL query to report the total purchase amount per supplier per year, using the corresponding payment information as well as report_year. The purchase year period is 2019 (but we should make a three-year display, such as 2018, 2019 and 2020. Since the database information is only stored in 2019 at present, the main goal is to show the operation), and the ranking result table corresponding to the earliest order and the number of orders is returned.

Limit to 10000 rows

```

1 use leaf_db;
2
3 SELECT first_order, COUNT(CustomerID) AS Order_Count,
4 ROUND(COUNT(next_day) / COUNT(CustomerID), 2) AS Day1_retention
5 FROM (
6 SELECT a1.CustomerID, a1.first_order, a2.OrderDate AS next_day
7 FROM
8 (
9 SELECT CustomerID, MIN(OrderDate) AS first_order
10 FROM `Order`
11 GROUP BY CustomerID
12 ) AS a1
13 LEFT JOIN `Order` AS a2
14 ON a1.CustomerID = a2.CustomerID
15 AND a2.OrderDate = a1.first_order + 1
16 ) AS t
17 GROUP BY first_order;
18

```

100% 22:17

Result Grid Filter Rows: Search Export:

first_order	Order_Count	Day1_retention
2019-08-16	1	0.00
2019-11-09	2	0.00
2019-08-15	2	0.00
2019-09-25	1	0.00
2019-12-11	2	0.00
2019-08-08	5	0.00
2019-09-23	1	0.00
2019-09-17	2	0.00
2020-01-04	1	0.00
2019-09-03	1	0.00
2020-01-19	1	0.00

(6)

Limit to 10000 rows

```

1 use leaf_db;
2
3 SELECT SupplierID, SupplierName, YEAR(StatementDate) as report_year,
4 (DATEDIFF(Concat(YEAR(StatementDate), '-12-31'), StatementDate)+1) as period
5 FROM Payment
6 JOIN Supplier USING(SupplierID)
7 WHERE YEAR(DueDate) - YEAR(StatementDate) >= 0
8
9 UNION ALL
10 SELECT SupplierID, SupplierName, YEAR(DueDate) as report_year,
11 (DATEDIFF(DueDate, Concat(YEAR(DueDate), '-01-01'))+1) as period
12 FROM Payment
13 JOIN Supplier USING(SupplierID)
14 WHERE YEAR(DueDate) - YEAR(StatementDate) >= 0
15
16 UNION ALL
17 SELECT SupplierID, SupplierName, YEAR(DueDate) as report_year,
18 (DATEDIFF(DueDate, StatementDate)+1) as period
19 FROM Payment
20 JOIN Supplier USING(SupplierID)
21 WHERE YEAR(DueDate) - YEAR(StatementDate) = 0
22
23 UNION ALL
24 SELECT SupplierID, SupplierName, '2019' as report_year,
25 >=5 as period
26 FROM Payment
27 JOIN Supplier USING(SupplierID)
28 WHERE YEAR(DueDate) - YEAR(StatementDate) = 2
29 ORDER by SupplierID, report_year;

```

100% 35:29

Result Grid Filter Rows: Search Export:

SupplierID	SupplierName	report_year	period
20000002	Santford PLC	2019	32
20000002	Santford PLC	2019	31
20000003	Schiller, Predovic and Lang	2019	32
20000003	Schiller, Predovic and Lang	2019	31

Result 3

Action Output

Time	Action	Response
49 19:57:59	SELECT SupplierID, SupplierName, YEAR(StatementDate) as report_year, (DATEDIFF(Concat(YEAR(StatementDate), '-12-31'), StatementDate)+1) as period	172 row(s) returned

(7)

Conclusion

We want our customers feel special feeling of having fresh groceries, while they sit and chill at home. So, in order to save time and letting you have a great shopping experience, we developed “Leafy Vegetables”. The database model behind such an application is the topic of our report. The idea of this model is that an application (web, mobile, or both) will allow registered customers to purchase an order (made up of products from our store) and the store to record the inventory. The model is not very complicated, but we did have a better concept of creating a completed database system with the application of various tools.

However, it is extremely hard to build one reliable, extensive and standardized grocery database. We will obviously store customer, product and inventory related data to support our database, but it seems like there is some other information we might need to concern in the reality as well which might be the delivery, rating and customer service. In this case, the challenge is to take the process in deliberate steps making certain that you understand each step before moving on to the next. Besides, it is essential for the brands and manufacturers to come together to make the information related to their products freely available since there are thousands of grocery and products selling in the store.