DawnDash™

EARLY MORNING DELIVERY SUBSCRIPTION

Nixon Lobo
Aliya
DMA Group 5

Northeastern University

IE6700 Data Management for Analytics

Final Project Report

Group 5

Student 1: Aliya LNU

Student 2: Nixon Lobo

Lnu.ali@northeastern.edu

Lobo.ni@northeastern.edu

Submission Date: 02/18/2024

Table of Contents

1 Executive Summary

DawnDash, a leading early morning delivery service, recognized the need for efficient data management to maintain operational excellence and customer satisfaction. This project developed a comprehensive data management solution leveraging relational (MySQL) and NoSQL (MongoDB) databases.

The relational database schema stores data related to users, vehicles, drivers, routes, deliveries, warehouses, products, inventory, suppliers, orders, promotions, subscriptions, additional items, feedback, and payments. Appropriate constraints ensure data integrity and efficient querying.

Data was randomly generated through Python to facilitate future testing and ensure consistency across tables using queries. The data was then implemented in MySQL using DDL statements.

SQL queries were developed for various operations, including simple queries, aggregate functions, joins, nested queries, correlated queries, and the use of EXISTS and ANY. These queries enable effective data retrieval, calculations, and information combination from multiple tables.

A NoSQL solution was implemented in MongoDB for large-scale data processing, real-time analytics, and performance monitoring.

Finally, an application demo was created using Python, establishing a connection between the databases and generating charts (histogram, pie chart, scatter plot, and box plot) for data visualization and analysis.

The combined relational models and databases, along with the Python application, provide DawnDash with a robust, scalable data management solution, ensuring efficient operations, accurate data processing, and valuable insights for continuous improvement in the early morning delivery market.

2 Introduction

In the fast-paced world of early morning transportation, businesses and individuals alike require a reliable and efficient delivery service to meet their urgent needs before daybreak. DawnDash steps in to fill this crucial gap, specializing in swift and dependable deliveries during the early hours. However, with the increasing demand for such services comes the challenge of managing vast amounts of data efficiently.

Theory for Early Morning Delivery:

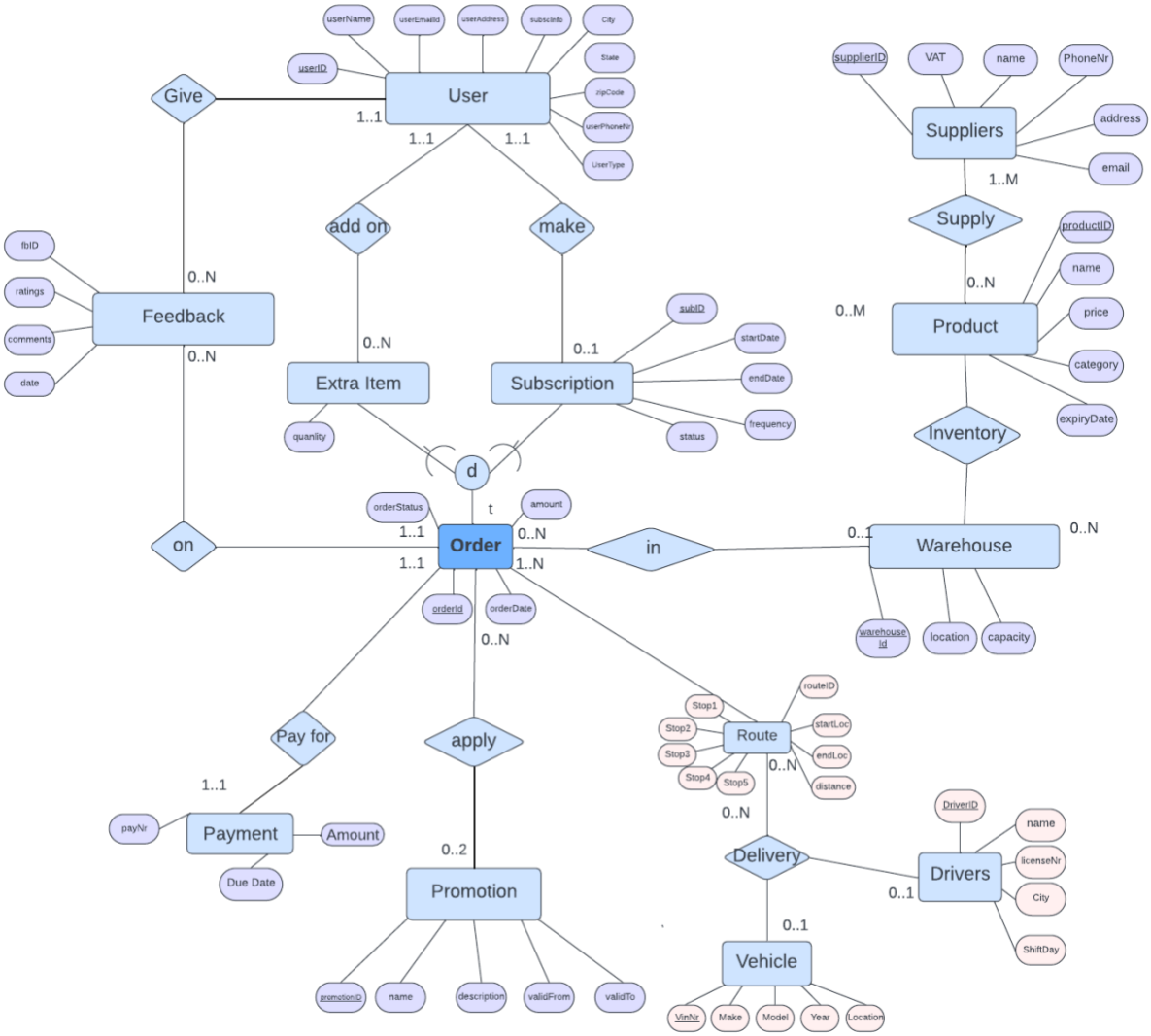
- **Enhanced Customer Experience**: Create a user-friendly platform where customers can subscribe to regular deliveries while having the flexibility to add extra items as per their needs.
- **Efficient Inventory Management**: Implement a robust data management system to track real-time inventory, ensuring accurate availability information and preventing order discrepancies.
- **Optimized Order Processing**: Develop an efficient order processing system that incorporates user preferences, processes orders in a timely manner, and schedules deliveries for the next morning.
- **Effective Communication**: Implement a notification system to keep users informed about order confirmations, delivery status, and reminders to add extra items before the cutoff time.

The business problems we are going to solve by build the database management system:

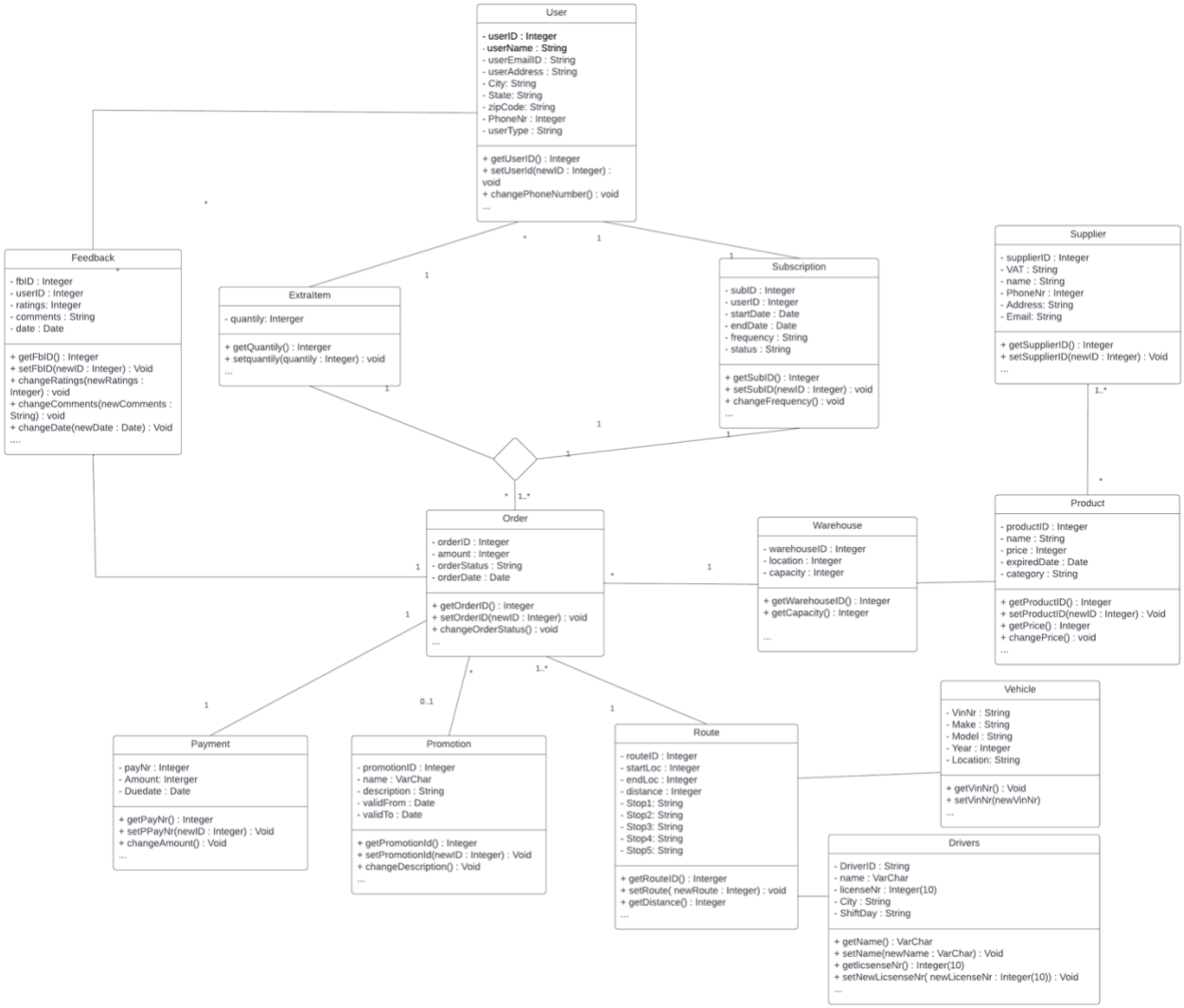
1. **Real-Time Order Processing**: Implement a system capable of processing delivery orders in real-time, ensuring prompt handling of customer requests as they come in.
2. **Optimized Route Planning**: Develop algorithms to optimize delivery routes based on factors such as distance, traffic conditions, and delivery deadlines, enabling efficient utilization of resources and timely deliveries.
3. **Customer Information Management**: Establish a secure database to store and manage customer information, including delivery preferences, contact details, and payment preferences, to streamline the ordering process and personalize the customer experience.
4. **Inventory Management**: Implement a system to track inventory levels of goods, food items, and groceries, ensuring adequate stock availability and preventing stockouts.
5. **Performance Monitoring**: Develop tools to monitor key performance metrics such as delivery times, customer's feedback/ratings, and driver efficiency, enabling continuous improvement and operational excellence.

3 Conceptual Data Modeling

3.1 EER Diagram



3.2 UML Diagram



4 Mapping Conceptual Model to Relational Model

Note: Primary Keys are underlined, and Foreign Keys are italic.

USER_DATA (UserID, userName, userEmailID, userAddress, City, State, zipCode, PhoneNr, UserType)

VEHICLE (VinNr, Make, Model, Year, Location)

DRIVERS (DriverID, name, LicenseNr, City, ShiftDay)

ROUTE (RouteID, StartLoc, EndLoc, Stop1, Stop2, Stop3, Stop4, Stop5, Distance)

DELIVERY (*RouteID*, *VinNr*, *DriverID*)

- *RouteID* foreign key refers to RouteID in ROUTE, NULL NOT ALLOWED
- *VinNr* foreign key refers to VinNr in VEHICLE, NULL NOT ALLOWED
- *DriverID* foreign key refers to DriverID in DRIVER, NULL NOT ALLOWED

WAREHOUSE (WarehosueID, location, capacity)

PRODUCT (ProductID, name, price, category, expireDate)

INVENTORY (*WarehosueID*, *ProductID*)

- *WarehosueID* foreign key refers to WarehosueID in WAREHOUSE, NULL NOT ALLOWED
- *ProductID* foreign key refers to ProductID in PRODUCT, NULL NOT ALLOWED

SUPPLIERS (SupplierID, VAT, name, PhoneNr, Address, Email)

SUPPLY (*SupplierID*, *ProductID*)

- *SupplierID* foreign key refers to SupplierID in SUPPLIER, NULL ALLOWED
- *ProductID* foreign key refers to ProductID in PRODUCT, NULL ALLOWED

ORDERS (OrderID, *ProductID*, Amount, OrderStatus, OrderDate, *RouteID*, *WarehosueID*, *PromotionID*, *UserID*)

- RouteID foreign key refers to RouteID in ROUTE, NULL NOT ALLOWED
- WarehosueID foreign key refers to WarehosueID in WAREHOUSE, NULL NOT ALLOWED
- ProductID foreign key refers to ProductID in PRODUCT
- PromotionID foreign key refers to PromotionID in ALL_PROMOTIONS

ALL_PROMOTION (PromotionID, name, description, VaildFrom, VaildTo)

SUBSCRIPTION (SubID, startDate, endDate, frequency, status, *UserID* , *OrderID*)

- *UserID* foreign key refers to UserID in USER, NULL NOT ALLOWED
- *OrderID* foreign key refers to OrderID in ORDERS, NULL NOT ALLOWED

EXTRAITEM (*UserID*, *OrderID*, *productID*, quanlity)

- *UserID* foreign key refers to UserID in USER_DATA, NULL NOT ALLOWED
- *OrderID* foreign key refers to OrderID in ORDERS, NULL NOT ALLOWED
- *ProductID* foreign key refers to ProductID in PRODUCT, NULL NOT ALLOWED

FEEDBACK (fbID, rating, comments, date, *UserID*, *OrderID*)

- *UserID* foreign key refers to UserID in USER_DATA, NULL NOT ALLOWED
- *OrderID* foreign key refers to OrderID in ORDERS, NULL NOT ALLOWED

PAYMENT (PayNr, DueDate, amount, *OrderID*)

- *OrderID* foreign key refers to OrderID in ORDERS, NULL NOT ALLOWED

5 Implementation of Relation Model via MySQL & NoSQL

5.1 MySQL Implementation

5.1.1 Simple Query

1. Find the name, ID, city and state of user from USER_DATA.

```
select userName, userID, City, State
from user_data;
```

userName	userID	City	State
Kelli Bowman	F60638	New York	NY
Julie Bell	B79933	Los Angeles	CA
April Baker	B19439	Boston	MA
Kevin Kramer	F44564	Boston	MA
Angelica Rowe	F48701	Los Angeles	CA
Jacob Pineda	B45710	Boston	MA
Shannon Combs	B15153	Boston	MA
Kenneth Randall	B94401	New York	NY
Dennis Frye	B84602	New York	NY
Matthew Perez	F52104	Boston	MA
Wyatt Bullock	F48704	New York	NY
Brandon Solis	F62140	Los Angeles	CA
John Hendricks	F23522	New York	NY

5.1.2 Aggregate Functions

2. Find the total amount of all orders.

```
select sum(Amount) as TotalAmount
from Orders;
```

TotalAmount
5596

3. Find the average of rating score as AverageRating from feedback

```
select avg(rating) as AverageRating from feedback
```

AverageRating
2.9950

4. Find the Max and Min Price of Products.

```
select max(price) as MaxPrice,
       min(price) as Minprice
from products;
```

MaxPrice	Minprice
19.94	0.52

5.1.3 Join Tabels

5. Retrieve a list of all deliveries, including information about the drivers assigned to them

```
select d.routeId, d.vinnr, d.driverid,
       dr.name as DriverName,
       dr.licensenr,
       dr.city as DriverCity, dr.shiftDay
from delivery d
right join drivers dr
on d.driverid = dr.driverid;
```

routeId	vinnr	driverid	DriverName	licensenr	DriverCity	shiftDay
R867	AQU43894T2	D9232	Jessica Burton	462124470	New York, NY	Sat
R997	ADQ75725V0	D9059	Chase Bishop	890753334	New York, NY	Tue
R610	RVU11580A7	D5971	Wesley Griffin	671284320	Boston, MA	Wed
R147	ESN29654U1	D6187	Barry Aguilar	430222484	New York, NY	Thu
R927	DKN58297J9	D4485	Thomas Hudson	667723152	New York, NY	Thu
R802	TEY06042D6	D2058	David Waters	366916335	Boston, MA	Fri
R115	ZAS38974T3	D7360	Richard Barnett	297163084	Boston, MA	Thu
R649	RSY37523W4	D4161	Donald Sullivan	621246346	Los Angeles, CA	Sun
R316	CLI94671A8	D5796	Julie Gibson	219219019	Boston, MA	Fri
R234	WMX15599Y1	D9905	Deborah Oliver	142918708	New York, NY	Mon
R747	WMX15599Y1	D1426	Jeffery Green	116578485	Boston, MA	Wed
R810	WMX15599Y1	D9366	Lee Martin	480599822	New York, NY	Fri
R823	WMX15599Y1	D8378	Logan Jacobs	631781312	Los Angeles, CA	Mon
R320	WMX15599Y1	D8212	Shawn Frank	789183483	Boston, MA	Fri

6. Use self join to find pairs of routes with similar distances.

```
select r1.routeid as Route1ID,
        r1.distance as Distance1,
        r2.routeid as route2ID,
        r2.distance as distance2
from route r1
join route r2
on r1.routeid < r2.routeid
where abs(r1.distance - r2.distance) < 5;
```

Route1ID	Distance1	route2ID	distance2
R342	40	R867	37
R811	35	R867	37
R250	41	R867	37
R772	33	R867	37
R119	41	R867	37
R287	39	R867	37
R212	38	R867	37
R409	34	R867	37
R605	35	R867	37
R687	35	R867	37
R215	39	R867	37
R709	40	R867	37
R601	37	R867	37
R376	36	R867	37

5.1.4 Nested Query

7. Finding routes with distance grater than the average distance.

```
select routeid, distance
from route
where distance > (select avg(distance) from route);
```

routeid	distance
R997	100
R147	95
R927	73
R115	69
R316	77
R823	93
R320	66
R910	90
R944	82
R788	68

8. Retrieve the names of drivers along with the total number of deliveries made by each driver who has received positive feedback.

```
select d.name,
        ( select count(*) from delivery
        where delivery.driverid = d.driverid
        ) as total_deliveries
from drivers d
where d.driverid in
        ( select driverid from feedback
        where rating > 3);
```

name	total_deliveries
Jessica Burton	1
Chase Bishop	1
Wesley Griffin	1
Barry Aguilar	1
Thomas Hudson	1
David Waters	1
Richard Barnett	1
Donald Sullivan	1
Julie Gibson	1
Deborah Oliver	1
Jeffery Green	1
Lee Martin	1
Logan Jacobs	1
Shawn Frank	1

5.1.5 Correlated Query

9. Find the products supplied by each supplier.

```
select s.supplierid, s.name as suppliername,
    (select group_concat(p.name separator ', ')
     from products p
     where exists
         (select *
         from supply
         where supplierid = s.supplierid and productid = p.productid)
         ) as suppliedproducts
from suppliers s;
```

supplierid	suppliername	suppliedproducts
S33628	Porter, Smith and White	Cheese, Turkey, Cream, Grape, Apple, Waterm...
S31782	Gonzales, Johnson and Mueller	Pasta, Pancake, Watermelon, Quinoa, Grape, B...
S11069	Durham, Bush and Eaton	Milk, Rice, Buckwheat, Cereal
S74683	Patel LLC	Cucumber, Toast, Ice Cream, Pineapple, Spinac...
S15429	Rodriguez-Mejia	Brownie, Ice Cream, Bread, Quinoa, Rice, Yogu...
S70046	Willis LLC	Cheese, Candy, Sausage, Sausage, Bagel, Bac...
S13641	Farrell-Woods	Toast, Cake, Barley, Tomato, Cheese, Banana,...
S57630	Stanton PLC	Butter, Brownie, Ice Cream
S97976	Peterson-Smith	Banana, Couscous, Tomato, Cucumber, Potato,...
S73950	Carlson PLC	Buckwheat, Quinoa, Cream, Tomato, Egg, Ice ...
S32269	Schmidt-Harmon	Cream, Potato, Bread, Pasta, Butter, Bell Pepp...
S56070	Thomas, Valencia and Bell	Lamb, Cheese, Chicken, Bread, Butter, Yogurt, ...
S20148	Harris, Downs and Cobb	Oatmeal, Lamb, Barley, Cake, Bagel, Buckwhea...
S11807	Torres LLC	Couscous, Tomato, Cucumber, Potato, Cousco...

5.1.6 Using ALL/ANY/Exists/Not Exists

10. Retrieve Products where the product is not available in inventory.

```
    -- No result because all the products available in inventory
```

```
SELECT *
FROM Orders o
WHERE NOT EXISTS (
   SELECT *
   FROM Inventory i
   WHERE i.ProductID = o.ProductID
);
```

OrderID	ProductID	Amount	OrderStatus	OrderDate	RouteID	WarehouseID	

Toggle wrapping

11. Retrive results where a feedback is associated with a given orderID.

```
SELECT * FROM Orders
WHERE EXISTS (
        SELECT *
        FROM feedback
        WHERE feedback.OrderID = Orders.OrderID);
```

OrderID	ProductID	Amount	OrderStatus	OrderDate	RouteID	WarehouseID	PromotionID
O00003	P75624	6	shipped	2023-08-03	R0405	W120	PRM5362
O00005	P29002	5	processing	2023-02-16	R0995	W910	PRM7272
O00006	P62249	8	pending	2023-09-14	R0099	W344	PRM3551
O00007	P19398	6	processing	2023-01-23	R0023	W393	PRM4037
O00008	P42406	6	shipped	2023-09-03	R0714	W414	PRM8921
O00010	P25614	7	pending	2023-01-17	R0969	W671	PRM4826
O00012	P77631	3	pending	2023-06-28	R0363	W421	PRM1861
O00013	P38180	6	shipped	2023-06-21	R0316	W421	PRM1861
O00014	P20964	1	pending	2023-04-17	R0018	W120	PRM1728
O00015	P60070	10	shipped	2023-05-15	R0120	W351	PRM2068
O00016	P30538	10	processing	2023-03-31	R0759	W414	PRM8106
O00018	P51407	6	shipped	2023-10-31	R0053	W393	PRM8206
O00020	P72064	1	processing	2023-01-04	R0035	W604	PRM5147
O00021	P32016	9	pending	2023-02-24	R0277	W181	PRM6013

12. Retrieve amount is greater than any amount for the same product in the orders table.

```
SELECT *
FROM Orders o1
WHERE Amount > ANY (
        SELECT Amount
        FROM Orders o2
        WHERE o1.ProductID = o2.ProductID
        );
```

OrderID	ProductID	Amount	OrderStatus	OrderDate	RouteID	WarehouseID	PromotionID
O00001	P60906	5	shipped	2023-10-19	R0156	W344	PRM3053
O00005	P29002	5	processing	2023-02-16	R0995	W910	PRM7272
O00007	P19398	6	processing	2023-01-23	R0023	W393	PRM4037
O00008	P42406	6	shipped	2023-09-03	R0714	W414	PRM8921
O00010	P25614	7	pending	2023-01-17	R0969	W671	PRM4826
O00012	P77631	3	pending	2023-06-28	R0363	W421	PRM1861
O00013	P38180	6	pped	2023-06-21	R0316	W421	PRM1861
O00015	P60070	10	shipped	2023-05-15	R0120	W351	PRM2068
O00016	P30538	10	processing	2023-03-31	R0759	W414	PRM8106
O00018	P51407	6	shipped	2023-10-31	R0053	W393	PRM8206
O00019	P12112	2	processing	2023-10-23	R0155	W421	PRM1453
O00021	P32016	9	pending	2023-02-24	R0277	W181	PRM6013
O00024	P88681	3	processing	2023-12-07	R0662	W791	PRM6878
O00025	P37225	8	pending	2023-12-31	R0038	W393	PRM6013

5.1.7 Subquery in Select and From

14. Calculate the percentage of total orders that each user has placed.

```
        -- it will be 100 since each customer has only 1 order in our data
SELECT UserID,
        UserName,
        (SELECT COUNT(*)
        FROM Orders
```

```
        WHERE UserID = u.UserID)
        / (SELECT COUNT(*)
        FROM Orders) * 100 AS PercentageOfOrders
FROM user_data u;
```

UserID	UserName	PercentageOfOrders
F60638	Kelli Bowman	100.0000
B79933	Julie Bell	100.0000
B19439	April Baker	100.0000
F44564	Kevin Kramer	100.0000
F48701	Angelica Rowe	100.0000
B45710	Jacob Pineda	100.0000
B15153	Shannon Combs	100.0000
B94401	Kenneth Randall	100.0000
B84602	Dennis Frye	100.0000
F52104	Matthew Perez	100.0000
F48704	Wyatt Bullock	100.0000
F62140	Brandon Solis	100.0000
F23522	John Hendricks	100.0000
B62602	Tiffany Ramos	100.0000

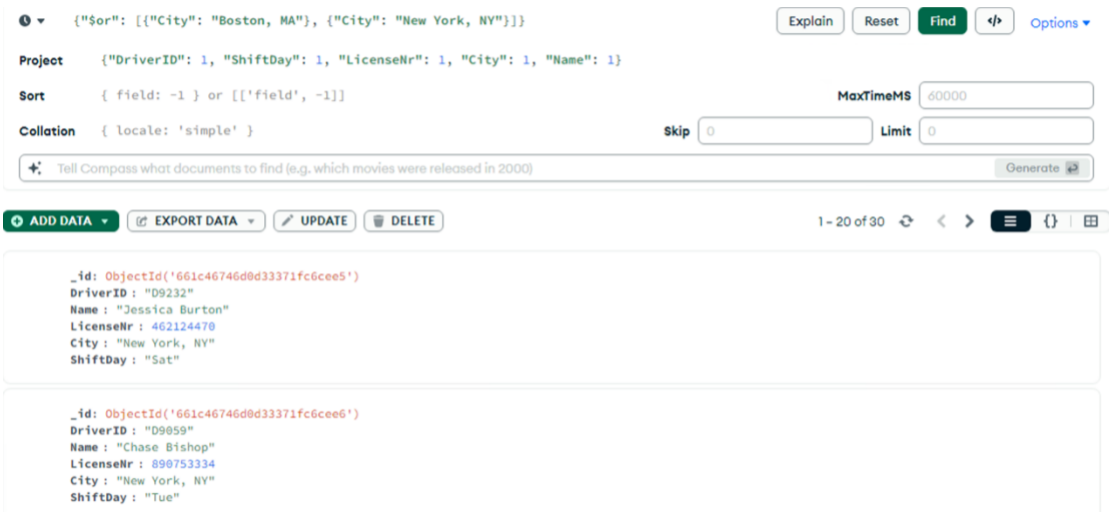
5.2 NoSQL Implementation

5.2.1 Simple Query

1. Finding Drivers from Boston, MA and New York, NY

```
{"$or": [{"City": "Boston, MA"}, {"City": "New York, NY"}]}
```

```
{"DriverID": 1, "ShiftDay": 1, "LicenseNr": 1, "City": 1, "Name": 1}
```



5.2.2 Complex Query

2. Finding the Total Amount of all Orders

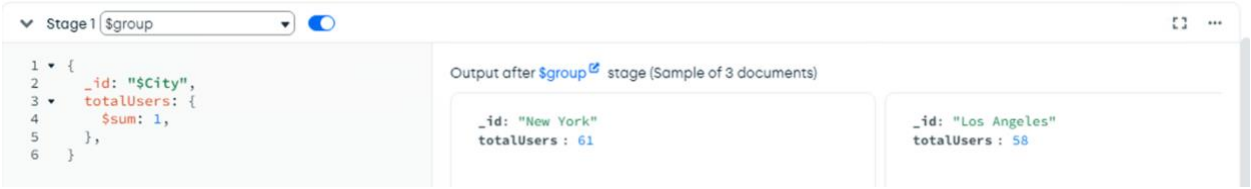
```
{
    _id: “$ProductID”,
    totalAmount: {
       $sum : “$Price”,
    },
}
```

5.2.3 Aggregate

3. Finding how many users are in which city

```
{
    _id: “$City”,
    totalUsers: {
       $sum : 1,
    },
```

}



4. Finding the Split of Business and Family USERS

{

_id: “$UserType”,

Count : {

$sum : 1,

},

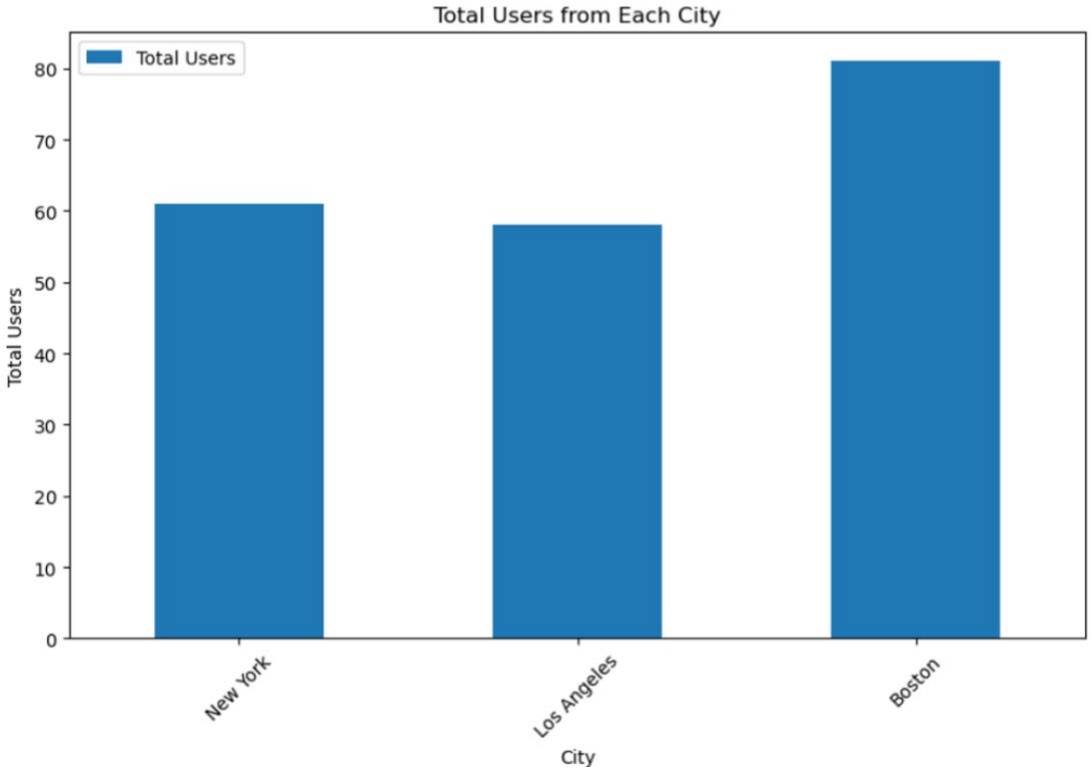
}

6 Database Access via Python

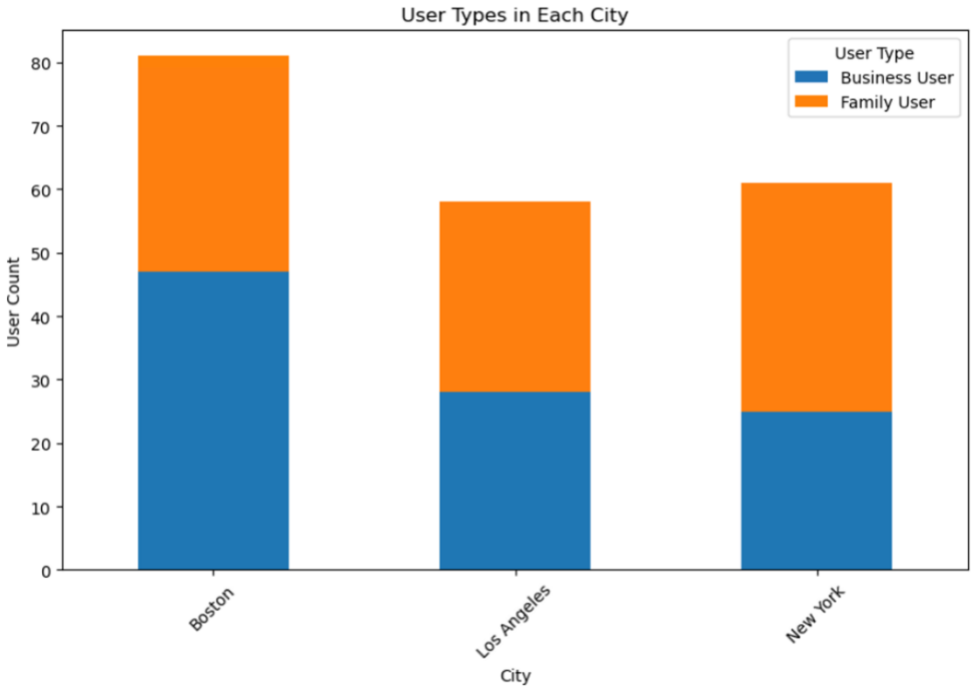
6.1 Histogram

1. Total Users from Each City

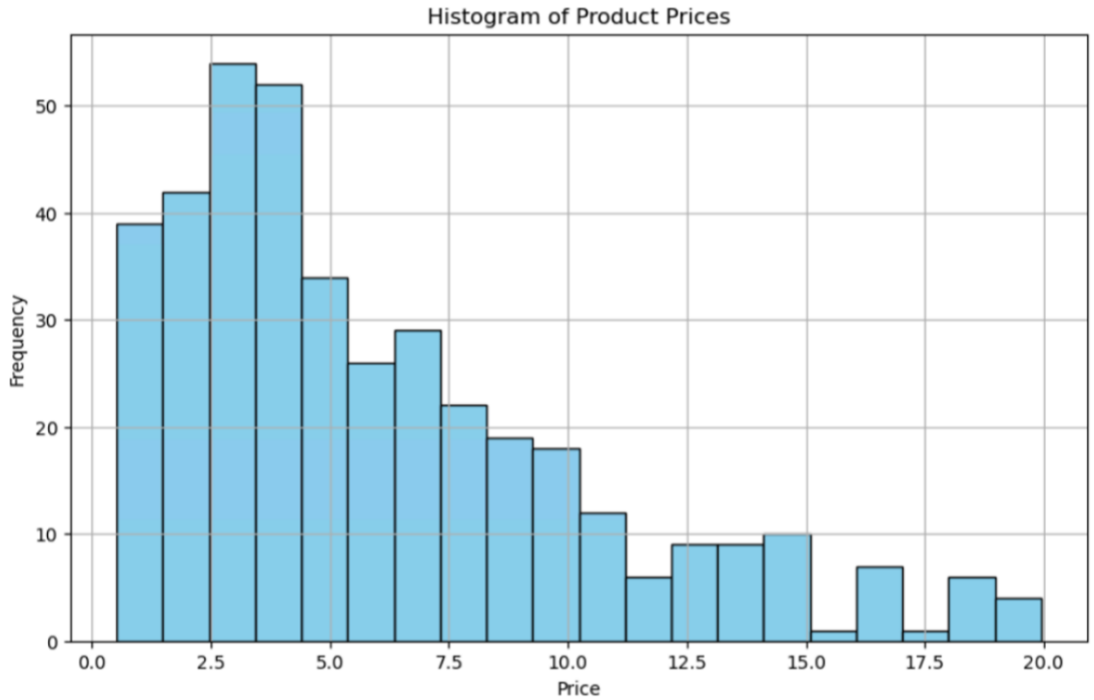
```
Connected to MySQL server
Total users from each city:
          City  Total Users
0     New York           61
1  Los Angeles           58
2       Boston           81
```



2. User Type in each city

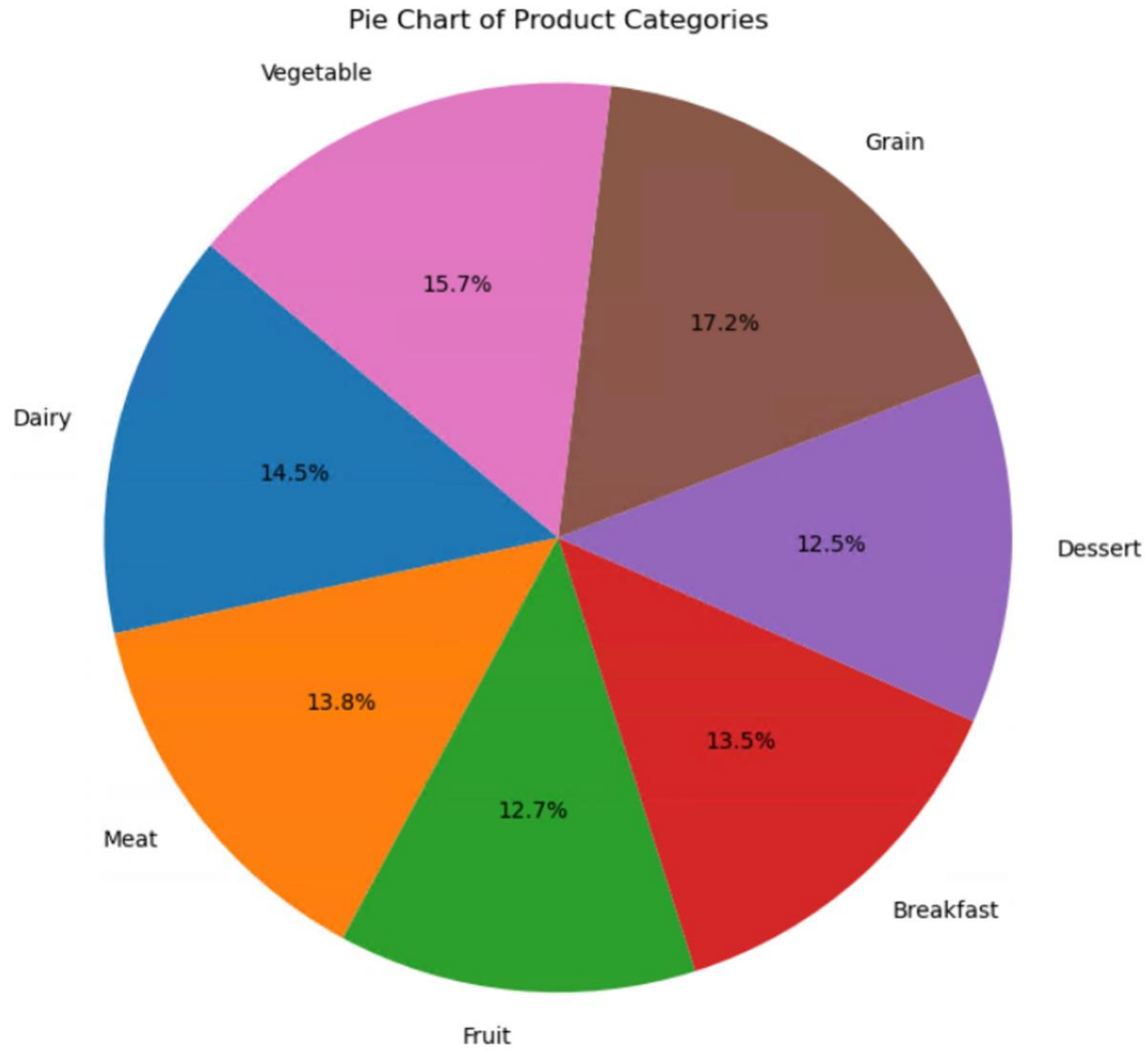


3. Histogram of Product prices



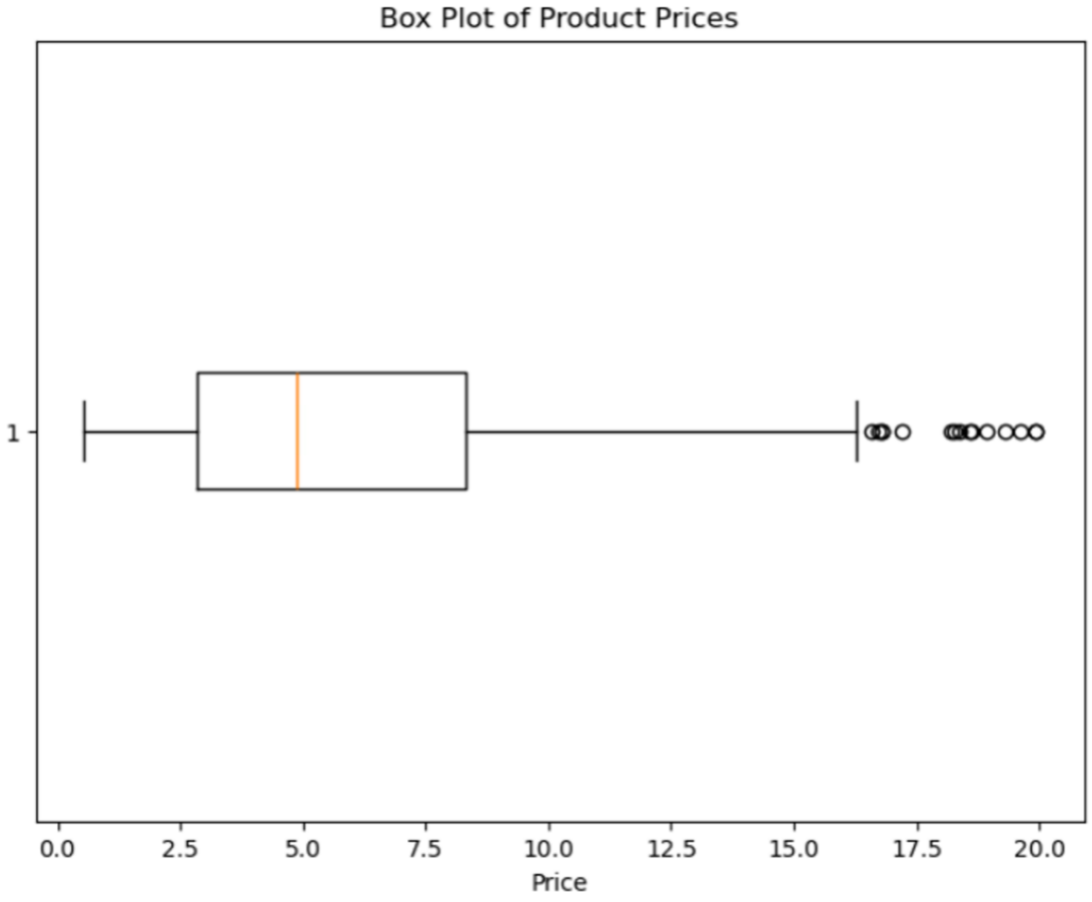
6.2 Pie Chart

4. Product Categories



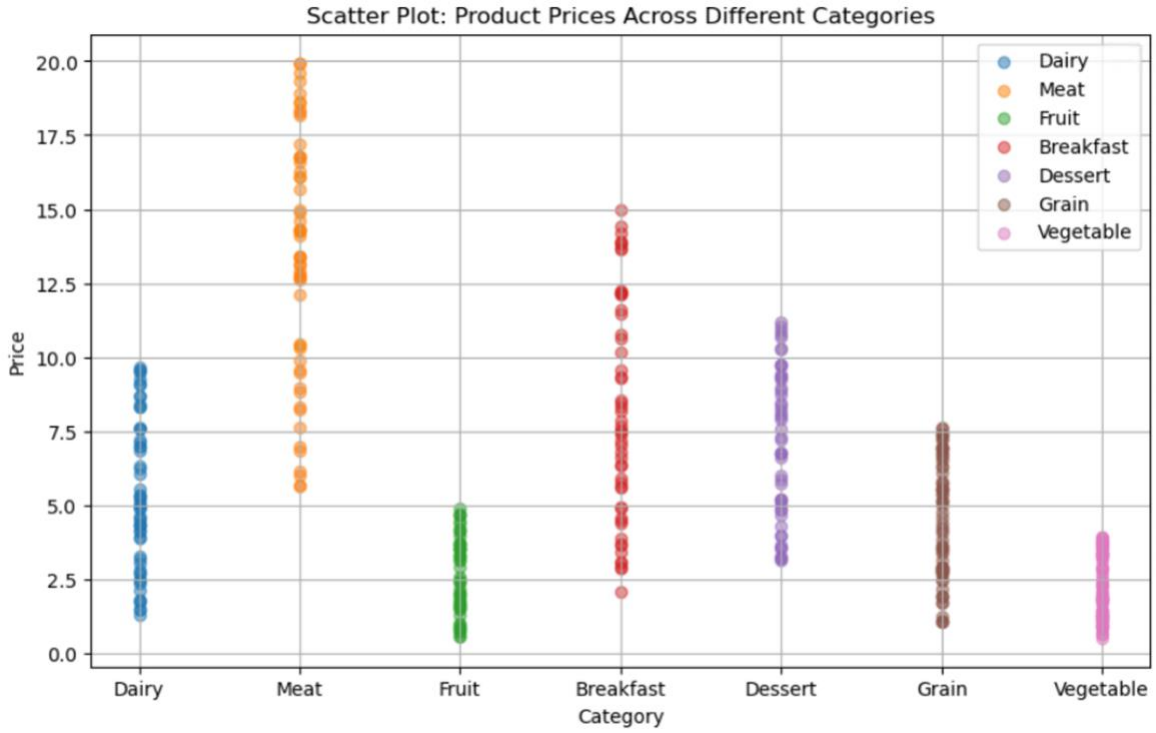
6.3 Box Plot

5. Product Prices



6.4 Scatter Plot

6.Product Prices across various categories



7 Summary and Recommendation

The DawnDash project has successfully delivered a comprehensive data management system that addresses the company's diverse data storage and processing requirements. The relational database model, implemented in MySQL, provides a robust foundation for storing and managing structured data, while the NoSQL solution, implemented in MongoDB, offers scalability and flexibility for handling large-scale data processing and real-time analytics.

The combination of these technologies, along with the Python application, equips DawnDash with a powerful toolset for efficient data management, accurate data processing, and valuable insights for continuous improvement in the early morning delivery market.

Some improvement can be made by Establishing a data governance framework to ensure data quality, consistency, and accountability across the organization. This framework should encompass data policies, standards, processes, and roles for managing data assets throughout their lifecycle. Additionally, explore the integration of Neo4j, a graph database, and its query language, Cypher, to handle complex relationships and interconnected data. This can be particularly useful for route optimization, network analysis, and recommendation systems, leveraging the strengths of graph databases in representing and querying highly connected data.

Consider leveraging big data technologies like Apache Spark and Hadoop to process and analyze large volumes of data generated by DawnDash's operations. These technologies can provide scalable and distributed computing capabilities, enabling efficient batch processing, real-time stream processing, and advanced analytics on massive datasets.

By implementing these recommendations, DawnDash can further enhance the capabilities of its data management solution, unlocking new opportunities for operational efficiency, data-driven decision-making, and delivering exceptional customer experiences.