

Retail Store Inventory Management System

Milestone: Project Proposal

Group 7
ROHAN PRAKASH KRISHNA
PRAKASH
NISHCHAY LINGE GOWDA

857-746-9940
617-792-8408

krishnaprakash.r@northeaster.edu
lingegowda.n@northeastern.edu

**Percentage of Effort Contributed by
Student1: 50%
Percentage of Effort
Contributed by Student2: 50%**

**Signature of Student 1: Rohan Prakash
Krishna Prakash
Signature of Student 2: Nishchay Linge Gowda**

Retail Store Inventory Management System

Rohan Prakash Krishna Prakash and Nishchay Linge Gowda

Problem Statement:

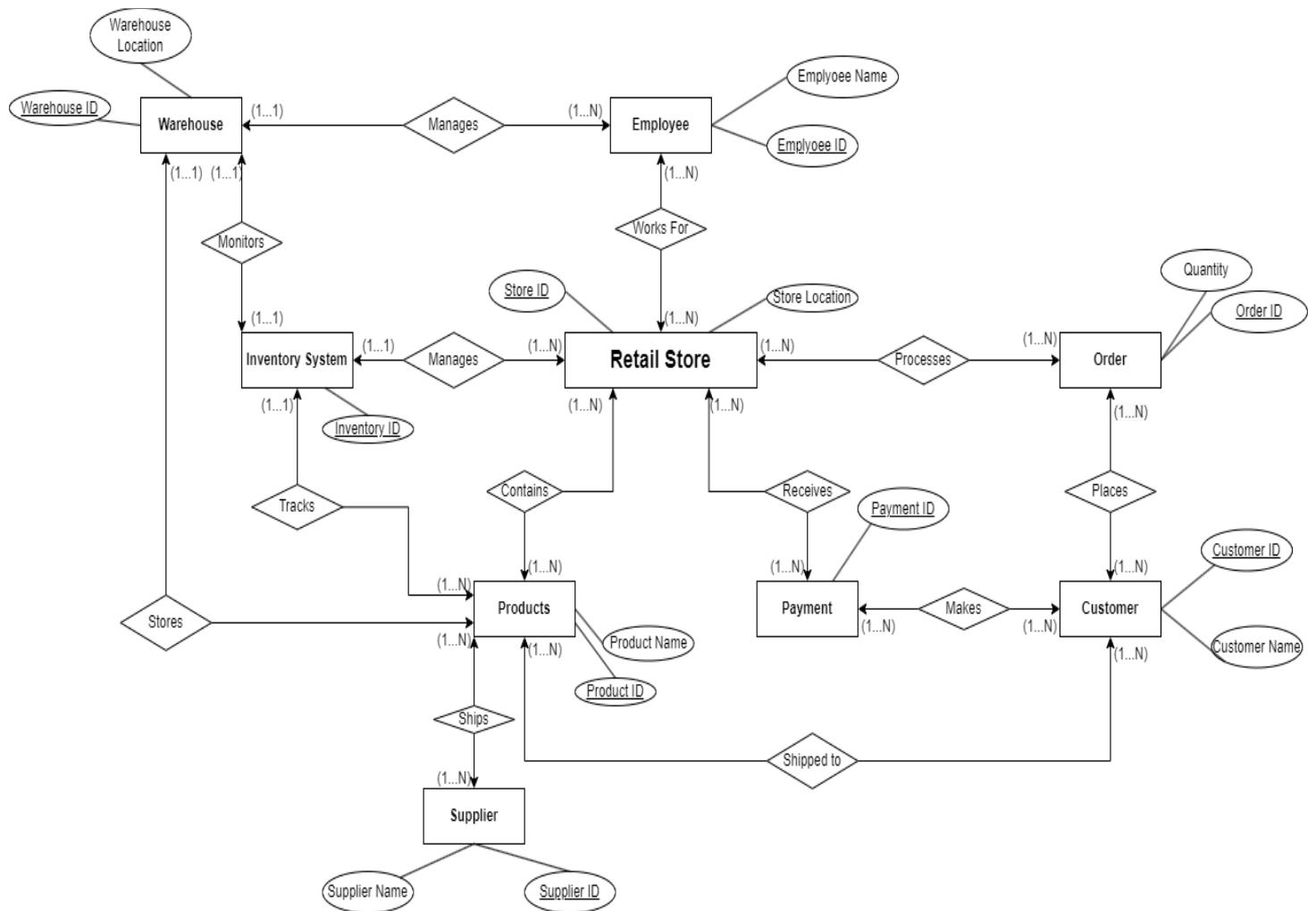
With the rapid growth of retail operations and the increasing need to manage stock efficiently, a well-structured Retail Store Inventory Management System has become essential. Retail stores must track inventory levels, orders, sales, and deliveries accurately, ensuring that the right products are available to meet customer demands. Poorly managed inventory systems can lead to overstocking, causing excess costs, or stockouts, resulting in lost sales opportunities and customer dissatisfaction. The aim of this system is to optimize inventory processes through automation, real-time data collection, and analytics. By analysing key factors such as purchase trends, sales velocity, and supply chain logistics, the system ensures a balanced stock flow and reduces human errors in tracking, ordering, and restocking inventory. The system must also enhance supply chain efficiency by automating the reordering process when stock reaches certain thresholds, ensuring that the retail store never runs out of essential products. Additionally, the system can provide real-time updates on stock movements, enabling store managers to make informed decisions based on accurate, up-to-date data. This avoids overstocking or stockouts, leading to increased customer satisfaction, smoother operations, and ultimately, higher profitability for the business.

Theory for Inventory management for a retail store:

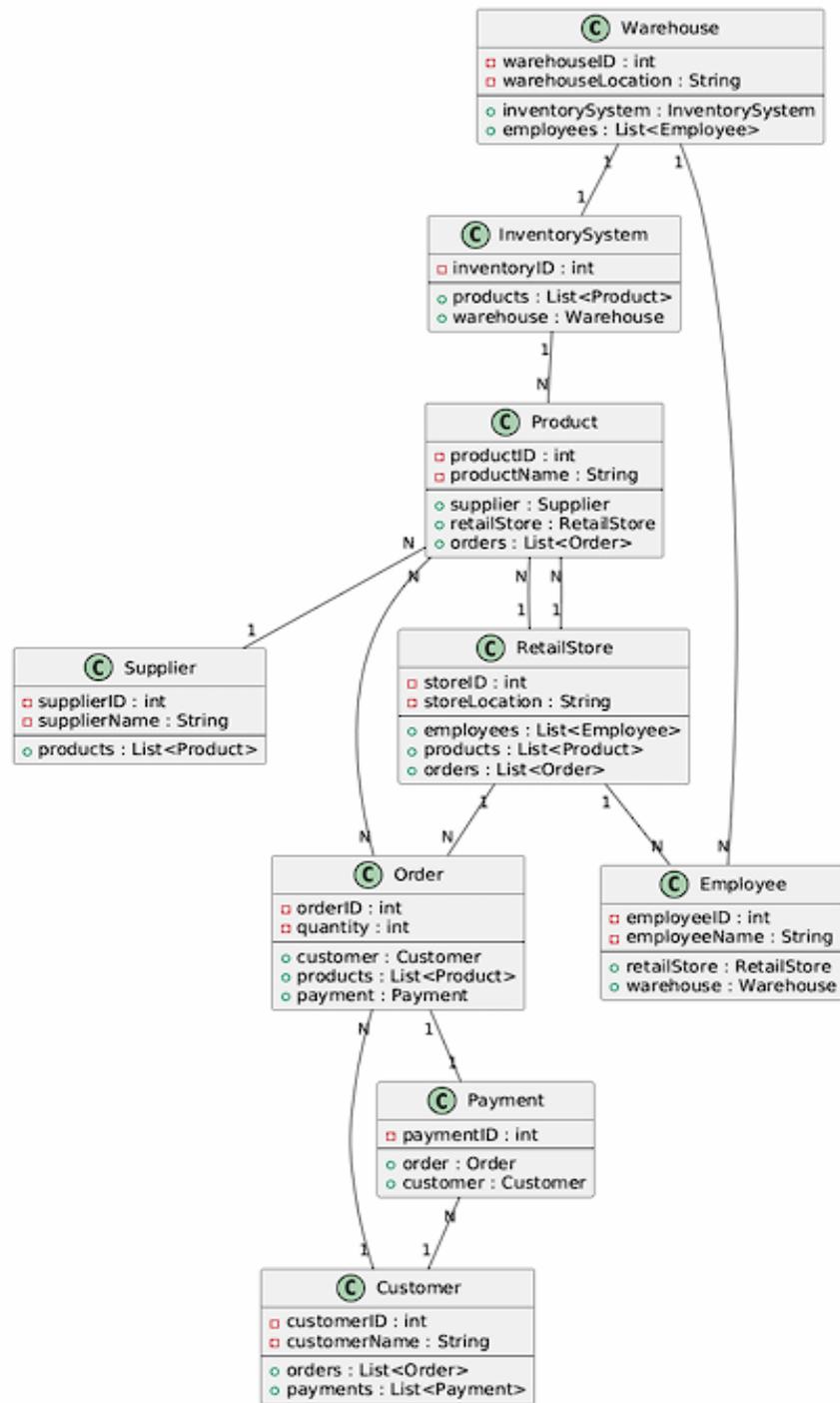
A well-structured Retail Store Inventory Management System is critical to meeting the dynamic demands of modern retail environments. As retail operations expand and customer expectations increase, it becomes essential to accurately track inventory levels, orders, sales, and deliveries. This system must prevent inefficiencies such as overstocking, which leads to increased holding costs, or stockouts, which result in lost sales and reduced customer satisfaction. To optimize these processes, the inventory management system should employ automation, real-time data analytics, and predictive modelling. This ensures seamless synchronization between the store's inventory levels and consumer demand, allowing store managers to manage stock more effectively. By collecting and analysing data on factors such as purchase trends, sales velocity, and supply chain logistics, the system can provide insights into inventory needs, reducing human errors in ordering, restocking, and managing stock levels. The system should also automate key processes such as reordering products when stock reaches pre-set thresholds, ensuring that essential items are never out of stock. Real-time updates on stock movements enable store managers to make quick and informed decisions, while an integrated dashboard offers insights into supply chain operations, helping to improve performance and reduce operational costs.

Conceptual Model (EER and UML Model)

- EER Model



- UML Model



Step 1: Identify Entities and Attributes In this step, we list each entity from the EER diagram and its corresponding attributes.

Entities and Their Attributes:

1. Customer

• **Attributes:**

- Customer_ID (Primary Key)
- Customer_Name

2. Order

• **Attributes:**

- Order_ID (Primary Key)
- Quantity

3. Retail Store

• **Attributes:**

- Store_ID (Primary Key)
- Store_Location

4. Employee

• **Attributes:**

- Employee_ID (Primary Key)
- Warehouse_ID (Foreign Key)
- Employee_Name

5. Warehouse

• **Attributes:**

- Warehouse_ID (Primary Key)
- Warehouse_Location

6. Products

• **Attributes:**

- Product_ID (Primary Key)
- Warehouse_ID (Foreign Key)
- Inventory_ID (Foreign Key)
- Product_Name

7. Supplier

• **Attributes:**

- Supplier_ID (Primary Key)
- Supplier_Name

8. Payment

• **Attributes:**

- Payment_ID (Primary Key)
- Payment_Type

9. Inventory System

• **Attributes:**

- Inventory_ID (Primary Key)

Step 2: Define Relationships Between Entities

The relationships connect entities according to the EER design. Below are descriptions of each relationship with cardinalities and explanations.

Relationships:

1. Warehouse to Inventory System (Manages):

- One warehouse manages one inventory system: (1..1)
- One inventory system is managed by one warehouse: (1..1)
- Final notation: 1:1

2. Warehouse to Employee (Manages):

- One warehouse is managed by many employees: (1..N)
- One employee manages one warehouse: (1..1)
- Final notation: 1:N

3. Warehouse to Inventory System (Monitors):

- One warehouse monitors one inventory system: (1..1)
- One inventory system is monitored by one warehouse: (1..1)
- Final notation: 1:1

4. Warehouse to Products (Stores):

- One warehouse stores many products: (1..N)
- One product is stored in one warehouse: (1..1)
- Final notation: 1:N

5. Retail Store to Warehouse (Manages):

- One retail store manages many warehouses: (1..N)
- One warehouse is managed by one retail store: (1..1)
- Final notation: 1:N

6. Retail Store to Order (Processes):

- One retail store processes many orders: (1..N)
- One order is processed by one retail store: (1..1)
- Final notation: 1:N

7. Retail Store to Products (Contains):

- One retail store contains many products: (1..N)
- One product can be contained in multiple retail stores: (1..N)
- Final notation: M:N

8. Retail Store to Payment (Receives):

- One retail store receives many payments: (1..N)
- One payment is received by one retail store: (1..1)
- Final notation: 1:N

9. Retail Store to Employee (Works For):

- One retail store employs many employees: (1..N)
- One employee works for one retail store: (1..1)
- Final notation: 1:N

10. Products to Inventory System (Tracks):

- One inventory system tracks many products: (1..N)
- One product can be tracked by multiple inventory systems: (1..N)
- Final notation: M:N

11. Products to Customer (Makes):

- One product is purchased by many customers: (1..N)
- One customer purchases many products: (1..N)
- Final notation: M:N

12. Products to Supplier (Ships):

- One supplier ships many products: (1..N)
- One product is shipped by one supplier: (1..1)
- Final notation: 1:N

13. Products to Retail Store (Contained in):

- One product can be contained in many retail stores: (1..N)
- One retail store contains many products: (1..N)
- Final notation: M:N

14. Order to Retail Store (Processes):

- One retail store processes many orders: (1..N)
- One order is processed by one retail store: (1..1)
- Final notation: 1:N

15. Order to Customer (Places):

- One customer places many orders: (1..N)
- One order is placed by one customer: (1..1)
- Final notation: 1:N

16. Payment to Retail Store (Receives):

- One retail store receives many payments: (1..N)
- One payment is linked to one retail store: (1..1)
- Final notation: 1:N

17. Supplier to Products (Ships):

- One supplier ships many products: (1..N)
- One product is shipped by one supplier: (1..1)
- Final notation: 1:N

Step 3: Convert Entities to Relational Tables

This is the relational model representation for each entity with all primary and foreign keys established.

Relational Tables:

Customer (Customer ID, Customer Name)

Order (Order ID, Quantity)

Processes(*Order ID*, *Store ID*)

Order ID: foreign key refers to order ID in relation Order. NULL not allowed, on delete/update cascade.

Store ID: foreign key refers to *Store ID* in relation Retail store. NULL not allowed, on delete/update cascade.

Places(*Order ID*, *Customer ID*)

Customer ID: foreign key refers to Customer ID in relation Customer. NULL not allowed, on delete/update cascade.

Order ID: foreign key refers to order ID in relation Order. NULL not allowed, on delete/update cascade.

Retail(StoreID, Store location)

Employee(Employee ID, *warehouse ID*, Employee Name)

Warehouse ID: foreign key refers to Warehouse ID in relation Warehouse. NULL not allowed, on delete/update cascade.

Works For(*Employee ID*, *Store ID*)

StoreID: foreign key refers to storerID in relation Retail store. NULL not allowed, on delete/update cascade.

Employee ID: foreign key refers to *Employee ID* in relation *Employee*. NULL not allowed, on delete/update cascade.

Warehouse(Warehouse ID, Warehouse location)

Products(ProductID, *Warehouse ID* , *Inventory ID*, Product Name)

Warehouse ID: foreign key refers to Warehouse ID in relation Warehouse. NULL not allowed, on delete/update cascade.

Inventory ID: foreign key refers to Warehouse ID in relation Warehouse. NULL not allowed, on delete/update cascade.

Contains(*Product ID*, *Store ID*)

Store ID: foreign key refers to storerID in relation Retail store. NULL not allowed, on delete/update cascade.

Product ID: foreign key refers to Product ID in relation Product. NULL not allowed, on

delete/update cascade.

Ships(*Product ID, Supplier ID*)

Product ID: foreign key refers to Product ID in relation Product. NULL not allowed, on delete/update cascade.

Supplier ID: foreign key refers to Supplier ID in relation Supplier. NULL not allowed, on delete/update cascade.

Shipped To(*Product ID, customer ID*)

Product ID: foreign key refers to Product ID in relation Product. NULL not allowed, on delete/update cascade.

Customer ID: foreign key refers to Customer ID in relation Customer. NULL not allowed, on delete/update cascade.

Makes (*Payment ID, Customer ID*)

Customer ID: foreign key refers to Customer ID in relation Customer. NULL not allowed, on delete/update cascade.

Payment ID: foreign key refers to Payment ID in relation Payment. NULL not allowed, on delete/update cascade.

Payment(Payment ID,Payment Type)

Receive (*Store ID, Payment ID*)

StoreID: foreign key refers to storeID in relation Retail store. NULL not allowed, on delete/update cascade.

Payment ID: foreign key refers to Payment ID in relation Payment. NULL not allowed, on delete/update cascade.

Inventory System(InventoryID)

Supplier(Supplier ID, Supplier Name)

Implementation in MySQL Queries:

1. Simple Query

```
SELECT CustomerName  
FROM Customer
```

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query is:

```
1 SELECT CustomerName  
2 FROM Customer;
```

The result grid displays the following customer names:

CustomerName
Alice Williams
Robert Smith
Charles Brown
Diana Miller
Frank Wilson
Fiona Clark
George Adams
Helen Lewis
Ivan Martinez
Julia Lopez
Kevin Reed
Laura Garcia
Maria Gomez
RickardCSmb
Olivia King
Paul Scott
Quinn Evans
Rachel Brooks
Steve Hill
Tina Carter
Uma Watson
Victor Rivera
Wendy Collins
Xavier Young
Yvonne Perry
Zachary Hall
Avery Bennett
Blae Sanchez
Claire Mitchell
Dan Morris
Ella Phillips
Franklin Ward
Grace Cooper
Henry Foster
Isha Hughes
Jack Morgan
Katherine Diaz

Query Completed

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query is:

```
1 SELECT CustomerName  
2 FROM Customer;
```

The result grid displays the following customer names:

CustomerName
Paul Scott
Quinn Evans
Rachel Brooks
Steve Hill
Tina Carter
Uma Watson
Victor Rivera
Wendy Collins
Xavier Young
Yvonne Perry
Zachary Hall
Avery Bennett
Blae Sanchez
Claire Mitchell
Dan Morris
Ella Phillips
Franklin Ward
Grace Cooper
Henry Foster
Isha Hughes
Jack Morgan
Katherine Diaz
Liam O'Brien
Mia Bryant
Noah Gray
Oliver Butler
Penelope Cox
Quentin Howard
Rebecca Cole..
Samuel Richards
Taylor Johnson
Ulysses Bell
Victoria Chavez
William Watson
Zoe Peterson
New Customer

Query Completed

2. Aggregate Query

```
SELECT SUM(Quantity) AS TotalQuantity  
FROM RSOrder;
```

The screenshot shows a database management interface with the following details:

- Toolbar:** Includes icons for Home, Project, Schemas, Customer, Employee, InventorySystem, Payment, Products, RSOrder, Supplier, Warehouse, RetailStore, and a Help icon.
- Schemas:** A sidebar showing the schema structure with categories like Views, Stored Procedures, Functions, and sys.
- Query Editor:** Displays the SQL query:

```
1 SELECT SUM(Quantity) AS TotalQuantity  
2 FROM RSOrder;  
3
```
- Result Grid:** Shows the result of the query, which is a single row with the value 2786.
- Object Info:** Shows information for the "Warehouse" table, including columns: WarehouseID (int PK) and WarehouseLocation (varchar(255)).
- Right Panel:** Contains tabs for Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan.
- Status Bar:** Shows "Query Completed" at the bottom left and "Read Only" at the bottom right.

3. Inner Join Query

```

SELECT RSOrder.OrderID, RSOrder.Quantity, RetailStore.StoreLocation
FROM RSOrder
INNER JOIN Processes ON RSOrder.OrderID = Processes.OrderID
INNER JOIN RetailStore ON Processes.StoreID = RetailStore.StoreID;

```

The screenshot shows a database management interface with the following details:

- Toolbar:** Includes icons for Home, Project, Warning - not supported, and various database objects like Customer, Employee, InventorySystem, Payment, Products, RSOrder, Supplier, Warehouse, and RetailStore.
- Schemas:** A sidebar showing the current schema structure, including Views, Stored Procedures, Functions, Project, Retail, and sys.
- Query Editor:** Displays the SQL query:

```

1 • SELECT RSOrder.OrderID, RSOrder.Quantity, RetailStore.StoreLocation
2   FROM RSOrder
3   LEFT JOIN Processes ON RSOrder.OrderID = Processes.OrderID
4   LEFT JOIN RetailStore ON Processes.StoreID = RetailStore.StoreID;
5
6

```
- Result Grid:** Shows the results of the query, which are empty (0 rows).

OrderID	Quantity	StoreLocation
1	88	NULL
2	89	NULL
3	84	NULL
4	75	NULL
5	30	NULL
6	4	NULL
7	64	NULL
8	89	NULL
9	8	NULL
10	13	NULL
11	35	NULL
12	22	NULL
13	100	NULL
14	77	NULL
15	24	NULL
16	53	NULL
17	84	NULL
18	70	NULL
19	84	NULL
20	69	NULL
21	87	NULL
22	66	NULL
23	62	NULL
24	78	NULL
25	29	NULL
26	39	NULL
27	77	NULL
28	63	NULL
29	98	NULL
30	30	NULL
31	27	NULL
32	42	NULL
33	96	NULL
34	99	NULL
35	69	NULL
- Right Panel:** Contains links to other tools: Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan.
- Status Bar:** Shows "Result 6" and "Read Only".

Project - Warning - not supported

Administration Schemas

Customer Employee InventorySystem Payment Products RSOrder Supplier Warehouse RetailStore >

Filter objects Views Stored Procedures Functions Project Retail sys

Object Info Session

Table: Warehouse

Columns: WarehouseID int PK WarehouseLocation varchar(255)

```

1 • SELECT RSOrder.OrderID, RSOrder.Quantity, RetailStore.StoreLocation
2 FROM RSOrder
3 LEFT JOIN Processes ON RSOrder.OrderID = Processes.OrderID
4 LEFT JOIN RetailStore ON Processes.StoreID = RetailStore.StoreID;
5
6

```

100% 1/5

Result Grid Filter Rows: Search Export:

OrderID	Quantity	StoreLocation
10	33	N/A
11	64	N/A
18	70	N/A
19	94	N/A
20	69	N/A
21	87	N/A
22	66	N/A
23	62	N/A
24	79	N/A
25	2	N/A
26	39	N/A
27	77	N/A
28	63	N/A
29	98	N/A
30	30	N/A
31	27	N/A
32	82	N/A
33	95	N/A
34	29	N/A
35	69	N/A
36	8	N/A
37	65	N/A
38	64	N/A
39	45	N/A
40	82	N/A
41	65	N/A
42	27	N/A
43	53	N/A
44	58	N/A
45	73	N/A
46	2	N/A
47	63	N/A
48	19	N/A
49	50	N/A
50	14	N/A

Result 6

Execution Plan

Query Completed

4. Left Outer Join

```

SELECT Products.ProductName, Warehouse.WarehouseLocation
FROM Products
LEFT JOIN Warehouse ON Products.WarehouseID = Warehouse.WarehouseID;

```

Project - Warning - not supported

Administration Schemas

Customer Employee InventorySystem Payment Products RSOrder Supplier Warehouse RetailStore >

Filter objects Views Stored Procedures Functions Project Retail sys

Object Info Session

Table: Warehouse

Columns: WarehouseID int PK WarehouseLocation varchar(255)

```

1 • SELECT Products.ProductName, Warehouse.WarehouseLocation
2 FROM Products
3 LEFT JOIN Warehouse ON Products.WarehouseID = Warehouse.WarehouseID;
4

```

100% 1/4

Result Grid Filter Rows: Search Export:

ProductName	WarehouseLocati...
Laptop	New York
Smartphone	Los Angeles
Tablet	Chicago
Headphones	Houston
Monitor	Phoenix
Keyboard	Philadelphia
Mouse	San Antonio
Speaker	San Diego
Webcam	Dallas
Printer	San Jose
Scanner	Austin
Router	Jacksonville
Hard Drive	Fort Worth
Power Bank	Columbus
USB Cable	Charlotte
Mouse Pad	Indianapolis
Projector	San Francisco
Smartwatch	Seattle
VR Headset	Denver
Drone	Washington DC
Camera	Boston
Tripod	El Paso
Microphone	Detroit
Lighting Kit	Nashville
Game Controller	Portland
VR Glasses	New York
Graphic Card	Los Angeles
RAM	Chicago
Processor	Houston
Motherboard	Phoenix
Power Supply	Philadelphia
Computer Case	San Antonio
Computer C.	San Diego
Bluetooth A...	Dallas
Wi-Fi Extender	San Jose
Smart Bulb	Austin
Thermometer	Tennesseeville

Result 2

Read Only

Query Completed

The screenshot shows a database interface with a query editor and a results grid. The query is:

```

1  SELECT Products.ProductName, Warehouse.WarehouseLocation
2  FROM Products
3  LEFT JOIN Warehouse ON Products.WarehouseID = Warehouse.WarehouseID;

```

The results grid displays data from the Warehouse table:

ProductName	WarehouseLocati...
USB Cable	Charlotte
Charger	Indianapolis
Projector	San Francisco
Smartwatch	Seattle
VR Headset	Denver
Drone	Washington DC
Camera	Boston
Tripod	El Paso
Microphone	Detroit
Lighting Kit	Nashville
Gaming Cons.	Portland
VR Headset	New York
Graphic Card	Los Angeles
RAM	Chicago
Processor	Houston
Motherboard	Phoenix
Power Supply	Philadelphia
Cooling Fan	San Antonio
Computer A... C...	San Diego
Bluetooth A...	Dallas
Wi-Fi Extender	San Jose
Smart Bulb	Austin
Thermostat	Jacksonville
Doorbell Ca...	Fort Worth
Smart Lock	Columbus
Air Purifier	Charlotte
Smart Plug	Indianapolis
Robot Vacuum	San Francisco
Coffee Maker	Seattle
Blender	Denver
Toaster	Washington DC
Oven	Boston
Microwave	El Paso
Refrigerator	Detroit
Dishwasher	Nashville
Washing Ma...	Portland

Query Completed

5. Nested Query (Subquery in WHERE):

```

SELECT CustomerName
FROM Customer
WHERE CustomerID IN (
    SELECT CustomerID
    FROM Places
    WHERE OrderID IN (
        SELECT OrderID
        FROM RSOrder
        WHERE Quantity > 10
    )
);

```

Project - Warning - not supported

Administration Schemas

SCHEMAS

Object Info Session

Tables: RSOrder

```

1 -- Find the customers who ordered products with quantity greater than 10
2 • SELECT CustomerName
3   FROM Customer
4   WHERE CustomerID IN (
5     SELECT CustomerID
6       FROM Places
7     WHERE OrderID IN (
8       SELECT OrderID
9         FROM RSOrder
10        WHERE Quantity > 10
11      )
12    );
13

```

100% 1/13

Result Grid Filter Rows: Search Export:

CustomerName
Alice Williams
Robert Smith
Charles Brown
Diana Miller
Edwin Wilson
George Adams
Hannah Lewis
Jill Lopez
Karen Ward
Laura Garcia
Michael Turner
RogerXoCSMrb
Olivia King
Paul Scott
Quinn Evans
Rachel Brooks
Sally Green
Tina Carter
Uma Watson
Victor Rivera
Wendy Collins
Xavier Young

Customer 19

Query Completed

Project - Warning - not supported

Administration Schemas

SCHEMAS

Object Info Session

Tables: RSOrder

```

1 -- Find the customers who ordered products with quantity greater than 10
2 • SELECT CustomerName
3   FROM Customer
4   WHERE CustomerID IN (
5     SELECT CustomerID
6       FROM Places
7     WHERE OrderID IN (
8       SELECT OrderID
9         FROM RSOrder
10        WHERE Quantity > 10
11      )
12    );
13

```

100% 1/13

Result Grid Filter Rows: Search Export:

CustomerName
Avery Bennett
Bella Martinez
Chloe Mitchell
Derek Morris
Ella Phillips
Franklin Ward
Grace Cooper
Henry Foster
Isla Hughes
Julian Diaz
Leo Jenkins
Mia Bryant
Noah Gray
Oliver Butler
Penelope Cox
Quentin Howard
Rebecca Cole...
Samuel Johnson
Ulysses Bell
Victoria Chavez
William Watson
Zoe Peterson

Customer 19

Query Completed

6. correlated query

```
SELECT OrderID, Quantity
FROM RSOrder R1
WHERE Quantity > (
    SELECT AVG(R2.Quantity)
    FROM RSOrder R2
    WHERE R2.OrderID != R1.OrderID
);
```

The screenshot shows a database management system interface with a query editor. The query is:

```
1 • SELECT OrderID, Quantity
2   FROM RSOrder R1
3   WHERE Quantity > (
4       SELECT AVG(R2.Quantity)
5       FROM RSOrder R2
6       WHERE R2.OrderID != R1.OrderID
7   );
8
```

The results grid displays the following data:

OrderID	Quantity
1	88
2	89
3	84
4	75
7	64
13	100
14	77
17	64
18	70
19	94
20	69
21	87
22	66
23	62
24	78
27	77
28	63
29	98
32	82
33	85
34	69
35	69
37	65
38	64
40	82
41	69
44	58
45	73
47	63
NULL	NULL

7. ALL Operator

```
GROUP BY StoreID
HAVING COUNT(*) >= ALL (
    SELECT COUNT(*)
    FROM Processes p
    JOIN RetailStore rs ON p.StoreID = rs.StoreID
    WHERE rs.StoreLocation LIKE 'Location-1%'
    GROUP BY p.StoreID
);
```

Project - Warning - not supported

Administration Schemas

Filter objects

SCHEMAS

Customer Employee InventorySystem Payment Products RSOrder Supplier Warehouse RetailStore Customer

Limit to 50000 rows

```

3 GROUP BY StoreID
4 HAVING COUNT(*) >= ALL (
5   SELECT COUNT(*)
6   FROM Processes p
7   JOIN RetailStore rs ON p.StoreID = rs.StoreID
8   WHERE rs.StoreLocation LIKE 'Location-1%'
9   GROUP BY p.StoreID
10 )

```

Object Info Session

Table: Contains

Columns:

- ProductID int PK
- StoreID int PK

Result Grid Filter Rows: Search Export:

StoreID
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Processes 2

Action Output Response Duration / Fetch Time

110 22:23:59 SELECT StoreID FROM Processes GROUP BY StoreID HAVING COUNT(*) >= ALL (SELECT COUNT(*) FROM Processes... 15 row(s) returned 0.001 sec / 0.00000...

Query Completed

8. UNION:

```

SELECT WarehouseLocation as Location, 'Warehouse' as Type
FROM Warehouse
UNION
SELECT StoreLocation, 'Store'
FROM RetailStore;

```

Project - Warning - not supported

Administration Schemas

Filter objects

SCHEMAS

Warehouse WorksFor Views Stored Procedures Functions Project Retail

Object Info Session

Table: RSOrder

Columns:

- OrderID int PK
- Quantity int

Result Grid Filter Rows: Search Export:

Location	Type
New York	Warehouse
Los Angeles	Warehouse
Chicago	Warehouse
Houston	Warehouse
Dallas	Warehouse
Philadelphia	Warehouse
San Antonio	Warehouse
San Diego	Warehouse
Dallas	Warehouse
San Jose	Warehouse
Austin	Warehouse
Louisville	Warehouse
Fort Worth	Warehouse
Columbus	Warehouse
Charlotte	Warehouse
Indianapolis	Warehouse
San Francisco	Warehouse
Seattle	Warehouse
Denver	Warehouse
Washington	Warehouse
Baltimore	Warehouse
El Paso	Warehouse
Detroit	Warehouse
Nashville	Warehouse
Portland	Warehouse
Las Vegas	Warehouse
Memphis	Warehouse
Louisville	Warehouse
Baltimore	Warehouse
Milwaukee	Warehouse
Albuquerque	Warehouse
Tucson	Warehouse
Fresno	Warehouse
Sacramento	Warehouse
Kansas City	Warehouse

Result 2

Query Completed

The screenshot shows a database management interface with the following details:

- Toolbar:** Includes icons for Home, Project, Schemas, Administration, and various database objects like Tables, Views, Procedures, Functions, and Scripts.
- Schemas Panel:** Shows the current schema structure with objects like Warehouse, WorksFor, Views, Stored Procedures, and Functions.
- Query Editor:** Displays a T-SQL script:

```
1 SELECT WarehouseLocation as Location, 'Warehouse' as Type
2 FROM Warehouse
3 UNION
4 SELECT StoreLocation, 'Store'
5 FROM RetailStore;
```
- Result Grid:** Shows the execution results for the query, listing locations and types. The results are as follows:

Location	Type
Atlanta	Warehouse
Omaha	Warehouse
Colorado...	Warehouse
Raleigh	Warehouse
Miami	Warehouse
Virginia Be...	Warehouse
Oakland	Warehouse
Madisn, WI	Warehouse
Tulsa	Warehouse
Arlington	Warehouse
Tampa	Warehouse
New Orleans	Warehouse
Wichita	Warehouse
Cleveland	Warehouse
Brooklyn, NY	Store
St. Louis	Store
Oak Park, IL	Store
Sugar Lan...	Store
Scottsdale...	Store
Media, PA	Store
Burne, TX	Store
La Jolla, CA	Store
Plano, TX	Store
TSX	Store
Round Ro...	Store
Winter Par...	Store
Arlington, TX	Store
Dublin, OH	Store
Carmel, IN	Store
Hunters...	Store
Bellview, CO	Store
Boulder, CO	Store
Silver Spr...	Store
Cambridge	Store

- Session Panel:** Shows the session information for the current user.
- Right Sidebar:** Contains various tools and panels: Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan.
- Status Bar:** Shows "Query Completed".

9. Subquery

```
SELECT w.WarehouseLocation,
       (SELECT COUNT(*)
        FROM Products p
        WHERE p.WarehouseID = w.WarehouseID) as ProductCount
  FROM Warehouse w;
```

FROM Warehouse w;

The screenshot shows a database management system interface with the following components:

- Top Bar:** Project - Warning - not supported, Administration, Schemas, Query 1, Customer, Employee, InventorySystem, Payment, Products, RSOrder, Supplier, Warehouse, RetailStore, Customer, and a "Result" button.
- Schemas Panel:** Shows the schema structure with tables: Contains, Customer, Employee, InventorySystem, Makes, Payment, and others.
- Query Editor:** Displays the following SQL query:

```
1 SELECT w.WarehouseLocation,
2        (SELECT COUNT(*)
3         FROM Products p
4         WHERE p.WarehouseID = w.WarehouseID) as ProductCount
5        FROM Warehouse w;
```
- Result Grid:** Shows the results of the query, listing 30 warehouse locations and their product counts. The data is as follows:

WarehouseLocation	ProductCount
New York	2
Los Angeles	2
Chicago	2
Houston	2
Phoenix	2
Philadelphia	2
San Antonio	2
San Diego	2
Dallas	2
San Jose	2
Austin	2
Jacksonville	2
Ft. Worth	2
Columbus	2
Charlotte	2
Indianapolis	2
San Francisco	2
Seattle	2
Denver	2
Washington DC	2
Baltimore	2
El Paso	2
Detroit	2
Nashville	2
Portland	2
Las Vegas	0
Memphis	0
Louisville	0
Baltimore	0
Milwaukee	0
Albuquerque	0
Tucson	0
Fresno	0
Sacramento	0
Kansas City	0

- Object Info:** Shows the table structure for 'Contains'.
- Table Column Details:** Shows columns: ProductID (int PK) and StoreID (int PK).
- Right Sidebar:** Includes tabs for Result Grid, Form Editor, Field Types, and Query Stats.
- Bottom Status:** Query Completed and Read Only.

The screenshot shows a database query interface with the following details:

- Project - Warning - not supported** is displayed at the top.
- Administration Schemas** are listed on the left.
- Table: Contains** is selected in the center.
- Columns:** ProductID int PK, StoreID int PK are listed.
- Query 1:**

```

1   SELECT w.WarehouseLocation,
2       (SELECT COUNT(*)
3        FROM Products p
4        WHERE p.WarehouseID = w.WarehouseID) as ProductCount
5   FROM Warehouse w;
    
```
- Result Grid:** Shows a table with two columns: WarehouseLocation and ProductCount. The data includes rows for various cities like San Francisco, Seattle, Denver, Washington DC, Boston, El Paso, Detroit, Nashville, Portland, Las Vegas, Memphis, Louisville, Baltimore, Milwaukee, Albuquerque, Tucson, Fresno, Sacramento, Kansas City, Mesa, Atlanta, Omaha, Colorado Springs, Raleigh, Miami, Virginia Beach, Oakland, Minneapolis, Tulsa, Arlington, Tampa, New Orleans, Wichita, and Cleveland. Most cities have a ProductCount of 2, except for some like Las Vegas, Memphis, Louisville, Baltimore, Milwaukee, Albuquerque, Tucson, Fresno, Sacramento, Kansas City, Mesa, Atlanta, Omaha, Colorado Springs, Raleigh, Miami, Virginia Beach, Oakland, Minneapolis, Tulsa, Arlington, Tampa, New Orleans, Wichita, and Cleveland which have a ProductCount of 0.
- Execution Plan:** A small window on the right shows the execution plan for the query.

10. Subquery in FROM:

```

SELECT AVG(PaymentCount) as AvgPaymentsPerCustomer
FROM (
    SELECT CustomerID, COUNT(*) as PaymentCount
    FROM Makes
    GROUP BY CustomerID
) as CustomerPayments;
    
```

The screenshot shows a database query interface with the following details:

- Project - Warning - not supported** is displayed at the top.
- Administration Schemas** are listed on the left.
- Table: RSOrder** is selected in the center.
- Columns:** OrderID int PK, Quantity int are listed.
- Query 1:**

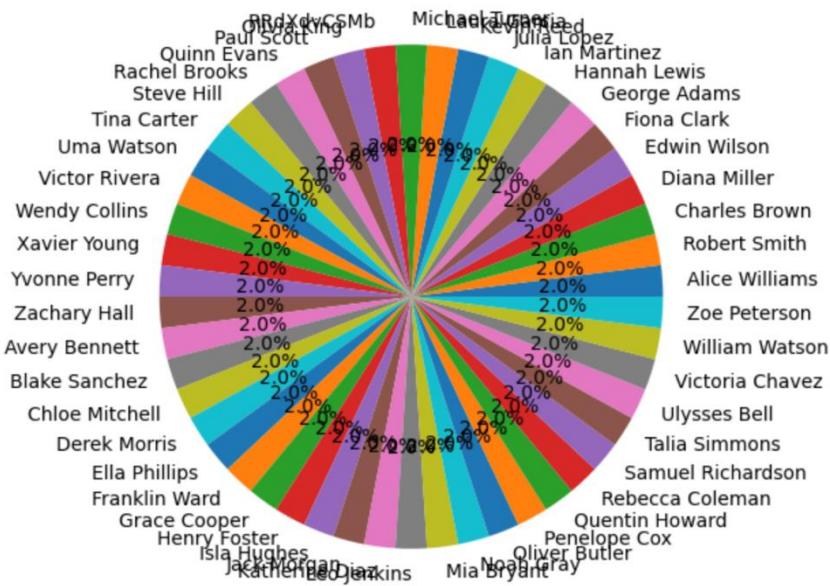
```

1   SELECT AVG(PaymentCount) as AvgPaymentsPerCustomer
2   FROM (
3       SELECT CustomerID, COUNT(*) as PaymentCount
4       FROM Makes
5       GROUP BY CustomerID
6   ) as CustomerPayments;
    
```
- Result Grid:** Shows a table with one column: AvgPaymentsPerCustomer. The value is 1.0000.
- Execution Plan:** A small window on the right shows the execution plan for the query.

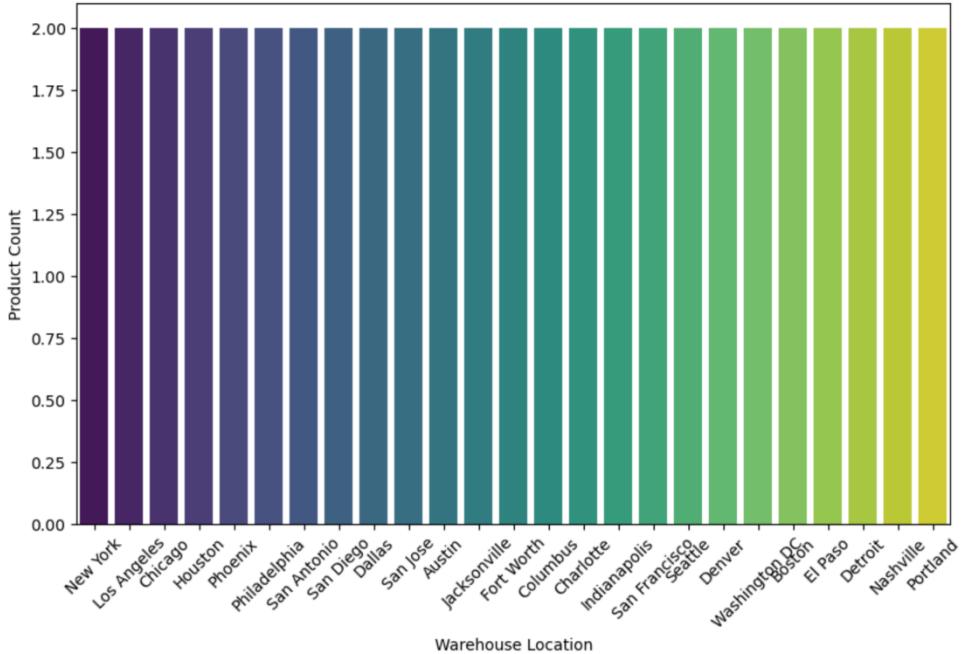
Database Access via Python

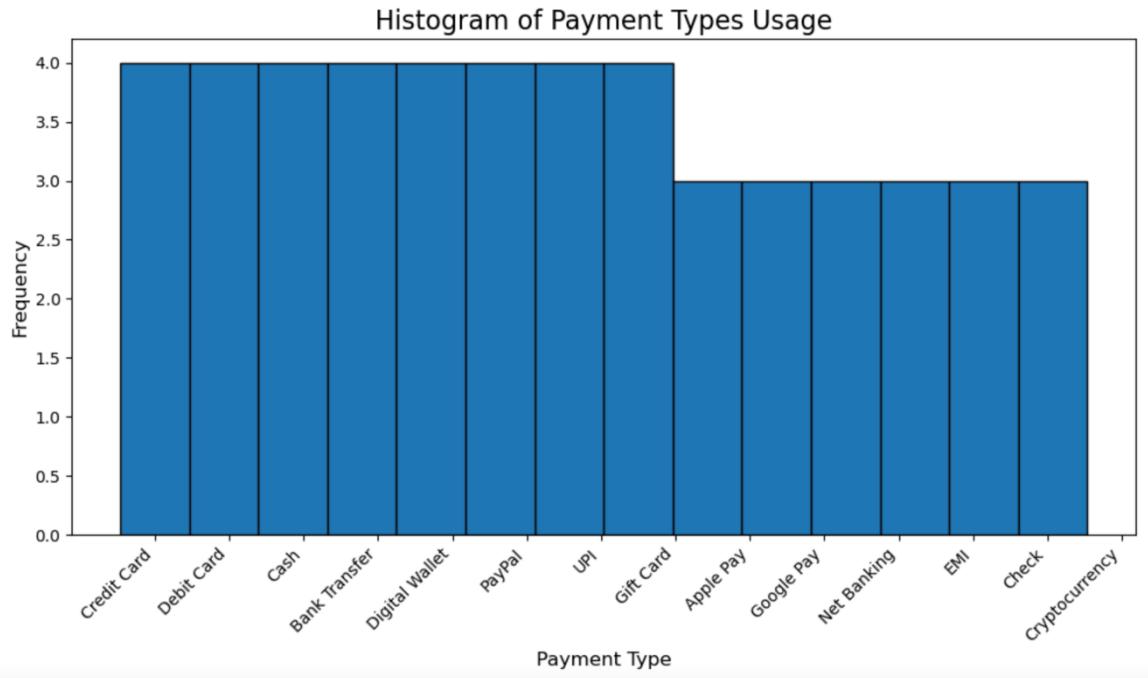
The database is accessed using Python and visualization of analyzed data is shown below. The connection of MySQL to Python is done using mysql.connector, followed by converting the list into a dataframe using pandas library and using matplotlib to plot the graphs for the analytics.

Orders Distribution by Customer



Product Distribution in Warehouses





NoSQL Implementation

FOR CREATING AND INSERTING TABLE CODE USED:

```
// Create Product collection
db.createCollection("Product")
db.Product.insertMany([
  {_id: 1, name: "Laptop", price: 999.99, category: "Electronics", stock: 50 },
  {_id: 2, name: "Smartphone", price: 599.99, category: "Electronics", stock: 100 },
  {_id: 3, name: "Headphones", price: 149.99, category: "Electronics", stock: 200 },
  {_id: 4, name: "T-shirt", price: 19.99, category: "Clothing", stock: 500 },
  {_id: 5, name: "Jeans", price: 59.99, category: "Clothing", stock: 300 }
])

// Create Customer collection
db.createCollection("Customer")
db.Customer.insertMany([
  {_id: 1, name: "John Doe", email: "john@example.com", loyaltyPoints: 100 },
  {_id: 2, name: "Jane Smith", email: "jane@example.com", loyaltyPoints: 150 },
  {_id: 3, name: "Bob Johnson", email: "bob@example.com", loyaltyPoints: 75 }
])

// Create Order collection
db.createCollection("Order")
db.Order.insertMany([
  {_id: 1, customerId: 1, products: [{ productId: 1, quantity: 1 }, { productId: 3, quantity: 2 }],
    totalAmount: 1299.97, status: "Completed" },
  {_id: 2, customerId: 2, products: [{ productId: 2, quantity: 1 }, { productId: 4, quantity: 3 }],
    totalAmount: 659.96, status: "Processing" },
  {_id: 3, customerId: 3, products: [{ productId: 5, quantity: 2 }], totalAmount: 119.98, status: "Placed" }
])
```

```
"Shipped" }  
])
```

Query 1: SIMPLE QUERY:

```
db.Product.find({ category: "Electronics" })
```

The screenshot shows a web-based MongoDB playground interface. The URL in the address bar is `pdbmbook.com/playground/mongo/wine/view/pgdb_1733081293_674cb8cdc56bc`. The page title is "Principles of Database Management". On the left, there is a text input field containing the MongoDB query `db.Product.find({ category: "Electronics" })`. Below the input field is a blue "Run" button. To the right of the input field, there is a "Result" section displaying the query results. The results show three documents from the "Product" collection:

```
{ "_id" : 1, "name" : "Laptop", "price" : 999.99, "category" : "Electronics", "stock" : 50 },  
{ "_id" : 2, "name" : "Smartphone", "price" : 599.99, "category" : "Electronics", "stock" : 100 },  
{ "_id" : 3, "name" : "Headphones", "price" : 149.99, "category" : "Electronics", "stock" : 200 }
```

Below the results, there is a note: "© Principles of Database Management 2024. Send us [your feedback](#)".

Query 2 : MORE COMPLEX QUERY:

```
db.Order.find({  
    totalAmount: { $gt: 500 },  
    status: { $in: ["Completed", "Shipped"] }  
})
```

The screenshot shows a MongoDB playground interface. On the left, a code editor contains a MongoDB query:

```

1 db.Order.find({
2   totalAmount: { $gt: 500 },
3   status: { $in: ["Completed", "Shipped"] }
4 })

```

A blue "Run" button is located below the code editor. To the right, a results panel displays the output of the query. At the top of the results panel, there are links for "Back to Playground overview" and "Reset". A "Tips" section provides instructions on how to use the playground.

Result

```

{
  "_id" : 1,
  "customerId" : 1,
  "products" : [
    {
      "productId" : 1,
      "quantity" : 1
    }
  ]
}

```

© Principles of Database Management 2024.
Send us [your feedback](#).

Query 3: AGGREGATE QUERY:

```

db.Order.aggregate([
  {
    $group: {
      _id: "$customerId",
      averageOrderTotal: { $avg: "$totalAmount" },
      totalOrders: { $sum: 1 }
    }
  },
  {
    $lookup: {
      from: "Customer",
      localField: "_id",
      foreignField: "_id",
      as: "customerInfo"
    }
  },
  {
    $project: {
      _id: 1,
      customerName: { $arrayElemAt: ["$customerInfo.name", 0] },
      averageOrderTotal: 1,
      totalOrders: 1
    }
  }
])

```

Principles of Database Management

Using: wine (mongo)

Write your statement below and press "Run" to see the result.

```
1+ db.Order.aggregate([
2+   {
3+     $group: {
4+       _id: "$customerId",
5+       averageOrderTotal: { $avg: "$totalAmount" },
6+       totalOrders: { $sum: 1 }
7+     },
8+   },
9+   {
10+     $lookup: {
11+       from: "Customer",
12+       localField: "_id",
13+       foreignField: "_id",
14+       as: "customerInfo"
15+     }
16+   },
17+   {
18+     $project: {
19+       _id: 1
20+     }
21+   }
22+ ])
```

Run

Result

```
{ "_id" : 3, "averageOrderTotal" : 119.98, "totalOrders" : 1, "customerName" : "Bob Johnson" }
{ "_id" : 2, "averageOrderTotal" : 659.96, "totalOrders" : 1, "customerName" : "Jane Smith" }
{ "_id" : 1, "averageOrderTotal" : 1299.97, "totalOrders" : 1, "customerName" : "John Doe" }
```

[← Back to Playground overview](#)

Reset

[Click here to reset the database to its initial state](#) (all your changes will be lost).

Tips

Enter MongoDB Javascript commands in the text area. Pressing "Run" will present the result of the MongoDB shell output. Try
`db.getCollectionNames();` to see defined collections and
`db.COLLECTIONNAME.find();` to retrieve a list of documents inside the given collection. See the [MongoDB reference](#) for useful commands.

© Principles of Database Management 2024.
Send us [your feedback](#).