OPTIMIZING DIRECT SALES WITH DATABASE MANAGEMENT

FINAL GROUP PROJECT REPORT

PREPARED BY GROUP 14

MEMBERS:

WEIBO ZHENG
SAMET TEMURCIN

PROJECT OVERVIEW

The goal of this project is to apply the knowledge learned from the IE6700 course to a real-world business case. In this project, we simulate the operations of a direct sales company, focusing on how data systems can support sales and recruitment. The main purpose is to help the company owner, managers, and employees get a clear and organized view of how the business is running—especially in terms of tracking agent activities, order statuses, and onboarding progress. A well-structured database system can reduce confusion, improve accuracy, and support better decision-making. Instead of relying on Excel sheets or messaging apps, which can easily cause errors or data loss, we aim to build a system that is more reliable and efficient for daily use.

There are four major steps in this project. First, we design the Enhanced Entity-Relationship (EER) diagram and UML model based on how the company operates in real life. These diagrams help us understand the relationships between departments, employees, orders, and customers. Second, we build the relational model by identifying primary keys, foreign keys, and other important constraints that maintain data consistency. In the third step, we write Python scripts to generate sample data that mimics real company data. We then use this data to build the database schema in MySQL. At the same time, we also store the data in MongoDB to show how it can be represented in a NoSQL format. Finally, we connect both databases to Python, run queries, and create visualizations like plots and charts to show useful insights, such as agent performance or order progress.

The final database system allows managers to easily monitor the daily activities of sales agents. They can use the information to give agents specific feedback, suggest better sales locations, or improve training. For the company owner and other employees, the system provides an easy way to keep track of agent onboarding and customer orders. This helps reduce chargebacks caused by early cancellations and improves overall company performance. With better data organization and reporting, the company can make smarter decisions and run more smoothly.

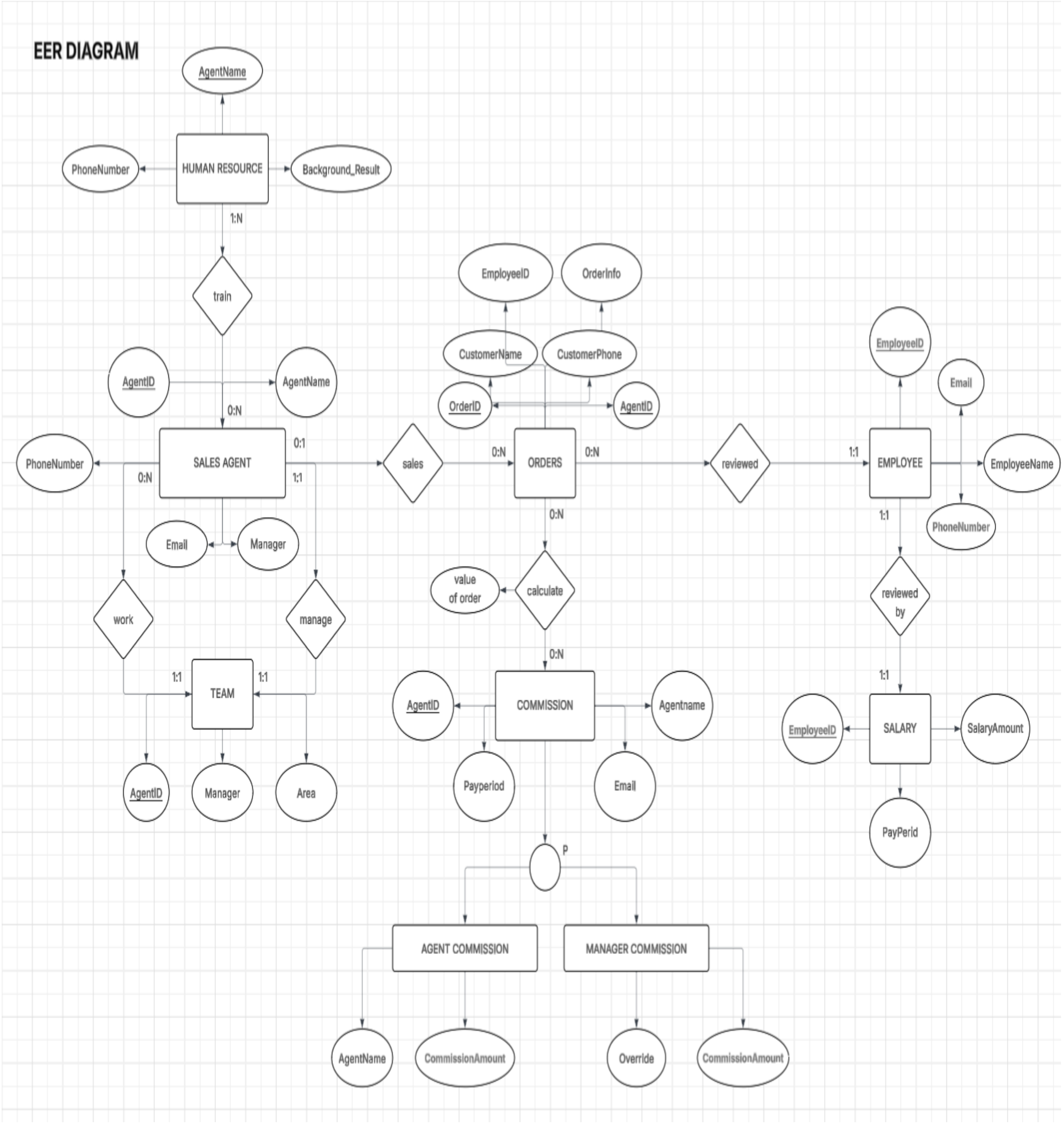
INTRODUCTION

Direct sales have become a dominant method for offering services like cable, home energy, and home security in the United States. In this model, sales representatives typically go door-to-door, engaging with potential customers and persuading them to purchase services. If a customer agrees to the offer, the sales representative gathers the necessary information and helps facilitate the signing of a contract with the service provider. However, managing this process manually, without a proper system in place, can lead to inefficiencies and errors. As a result, a robust database system is essential for streamlining operations and minimizing the risk of mistakes. Yet, many companies continue to rely on outdated tools like Excel spreadsheets and communication apps to track sales data. This approach often results in issues such as data duplication, inaccuracies in records, and incorrect commission calculations. A well-structured database system, on the other hand, can significantly enhance operational efficiency, reduce the likelihood of errors, and ultimately improve customer satisfaction.

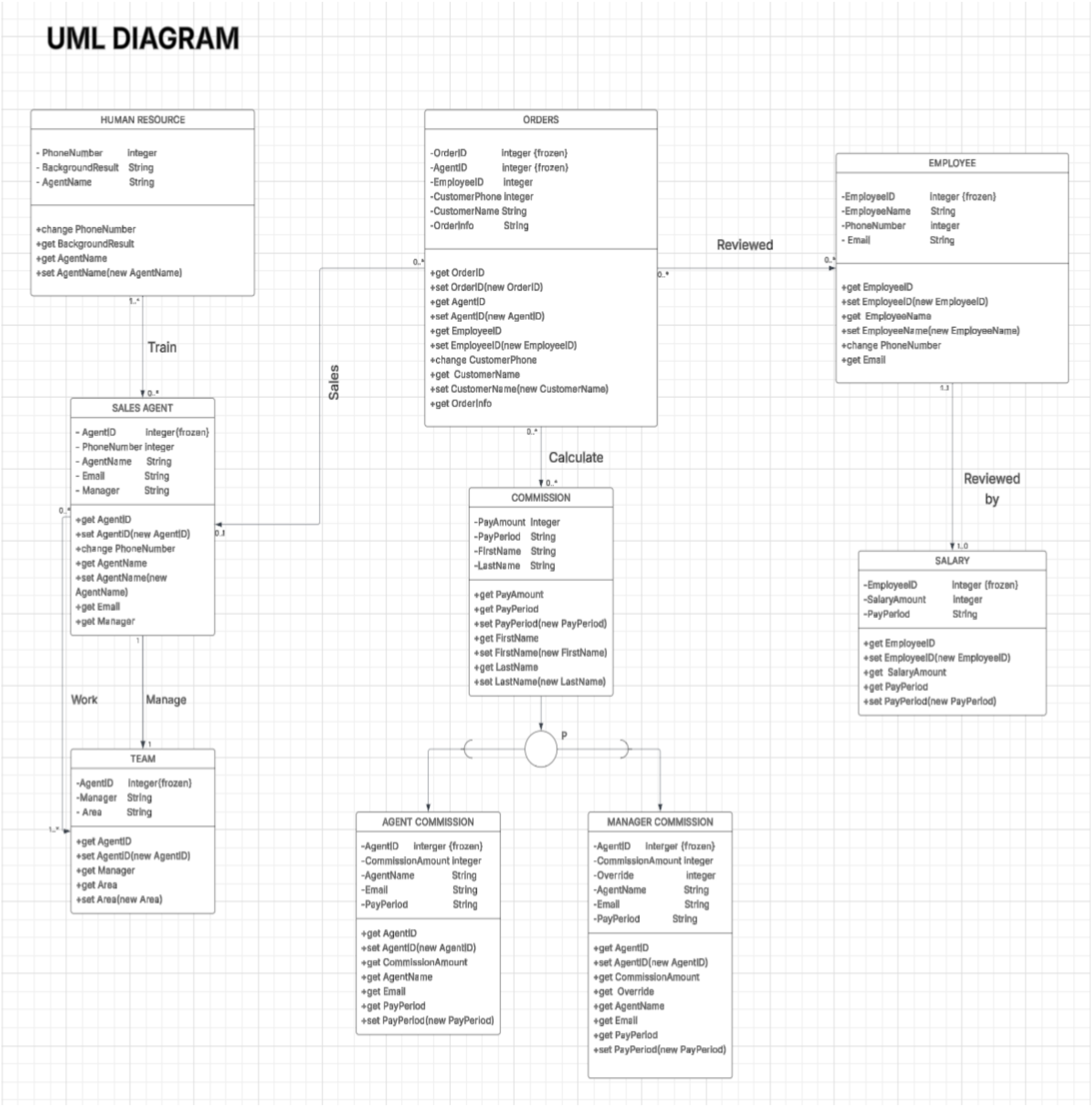
The simulated direct sales company for this project consists of three primary departments: Human Resources (HR), Sales, and Customer Service. The HR department is responsible for recruiting, hiring, and training sales representatives. Once sales representatives pass background checks, they are assigned a unique ID, which is used to track their performance throughout the sales process. The Sales department plays a critical role in monitoring sales activities, calculating commissions, and managing any overrides made by managers. When a customer places an order, the relevant details are captured in the system, including the sales representative's ID, the date of the sale, the customer's information, and the type of service ordered. The Sales department uses this data to compute commissions for both individual sales representatives and their managers. Meanwhile, the Customer Service department focuses on managing customer interactions, addressing concerns, and actively working to prevent service cancellations within the first 180 days. Cancellations within this window result in commission chargebacks, which adds an additional layer of complexity to the tracking and management process. A comprehensive database ensures smooth coordination between these departments, leading to more accurate record-keeping and improved service quality.

CONCEPTUAL DATA MODELING

EER DIAGRAM



UML DIAGRAM


UML DIAGRAM
HUMAN RESOURCE
- PhoneNumber integer
- BackgroundResult String
- AgentName String
+change PhoneNumber
+get BackgroundResult
+get AgentName
+set AgentName(new AgentName)
1..*
Train
0..*
SALES AGENT
- AgentID integer{frozen}
- PhoneNumber integer
- AgentName String
- Email String
- Manager String
+get AgentID
+set AgentID(new AgentID)
+change PhoneNumber
+get AgentName
+set AgentName(new AgentName)
+get Email
+get Manager
0..*
0..1
1
Work
Manage
1
1..*
TEAM
-AgentID integer(frozen)
-Manager String
- Area String
+get AgentID
+set AgentID(new AgentID)
+get Manager
+get Area
+set Area(new Area)
Sales
0..*
ORDERS
-OrderID integer {frozen}
-AgentID integer {frozen}
-EmployeeID integer
-CustomerPhone integer
-CustomerName String
-OrderInfo String
+get OrderID
+set OrderID(new OrderID)
+get AgentID
+set AgentID(new AgentID)
+get EmployeeID
+set EmployeeID(new EmployeeID)
+change CustomerPhone
+get CustomerName
+set CustomerName(new CustomerName)
+get OrderInfo
0..*
Calculate
0..*
COMMISSION
-PayAmount Integer
-PayPeriod String
-FirstName String
-LastName String
+get PayAmount
+get PayPeriod
+set PayPeriod(new PayPeriod)
+get FirstName
+set FirstName(new FirstName)
+get LastName
+set LastName(new LastName)
P
AGENT COMMISSION
-AgentID interger {frozen}
-CommissionAmount integer
-AgentName String
-Email String
-PayPeriod String
+get AgentID
+set AgentID(new AgentID)
+get CommissionAmount
+get AgentName
+get Email
+get PayPeriod
+set PayPeriod(new PayPeriod)
MANAGER COMMISSION
-AgentID interger {frozen}
-CommissionAmount integer
-Override integer
-AgentName String
-Email String
-PayPeriod String
+get AgentID
+set AgentID(new AgentID)
+get CommissionAmount
+get Override
+get AgentName
+get Email
+get PayPeriod
+set PayPeriod(new PayPeriod)
0..*
Reviewed
0..*
EMPLOYEE
-EmployeeID Integer {frozen}
-EmployeeName String
-PhoneNumber integer
- Email String
+get EmployeeID
+set EmployeeID(new EmployeeID)
+get EmployeeName
+set EmployeeName(new EmployeeName)
+change PhoneNumber
+get Email
1..1
Reviewed by
1..0
SALARY
-EmployeeID Integer {frozen}
-SalaryAmount Integer
-PayPeriod String
+get EmployeeID
+set EmployeeID(new EmployeeID)
+get SalaryAmount
+get PayPeriod
+set PayPeriod(new PayPeriod)

MAPPING CONCEPTUAL MODEL TO RELATIONAL MODEL

PK: Primary Key, FK: Foreign Key

1. **HR (*AgentName*, background_result, phone) - AgentName PK**
2. **SalesAgent (*AgentID,* AgentName, PhoneNumber, Email ,Manager) - AgentID PK ,AgentName FK null not allowed**
3. **Team (*AgentID*, Area, Manager) - AgentID PK and FK from SalesAgent null not allowed**
4. **Orders (*OrderID, AgentID*, CustomerName, OrderInfo, CustomerPhone, EmployeeID) - OrderID and AgentID are PK, AgentID from salesagent null not allowed**
5. **Commission (*AgentID*, Payperiod, AgentName, Email) - AgentID PK and FK from salesagent null not allowed**
6. **Calculate (*OrderID, AgentID*, Value of Order) - OrderID, AgentID both PK and OrderID FK from orders, AgentID FK from salesagent, both null not allowed**
7. **AgentCommission (*AgentID*, PayPeriod, AgentName, Email, CommissionAmount) - AgentID PK and FK from sales agent null not allowed**
8. **MamangerCommission (*AgentID*, PayPeriod, AgentName, Email, CommissionAmount, Override) - AgentID PK and FK from sales agent null not allowed**
9. **Employee (*EmployeeID*, EmployeeName, Email, PhoneNumber) EmployeeID PK**
10. **Salary (EmployeeID, PayPeriod, SalaryAmount) EmployeeID PK and FK from employee, null not allowed**

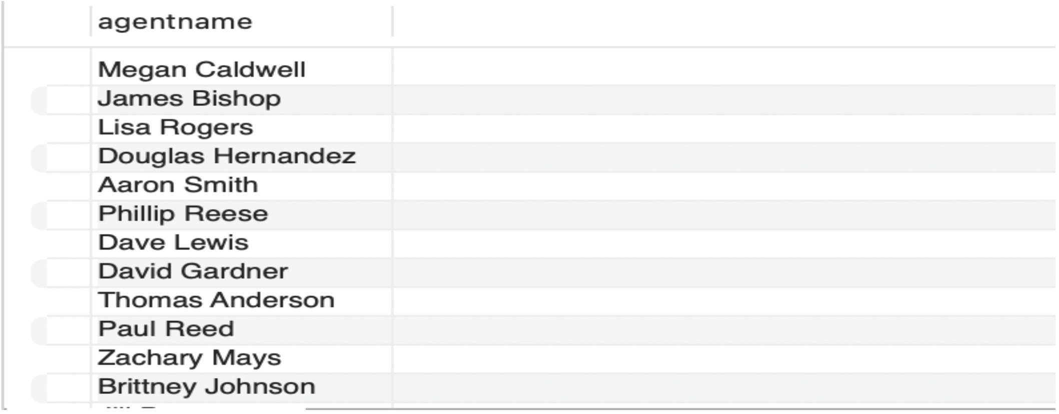
IMPLEMENTATION OF RELATION MODEL VIA MYSQL AND NOSQL

MYSQL IMPLEMENTATION

The database is created in MySQL database and following queries were performed

1) **Query 1: Retrieve all the agent name from salesagent table**

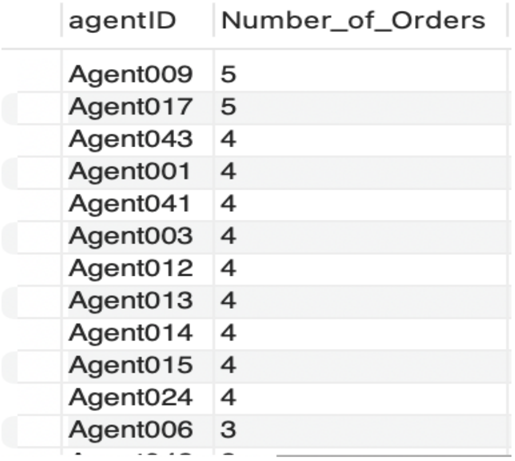
```
Select agentname from salesagent
```



agentname
Megan Caldwell
James Bishop
Lisa Rogers
Douglas Hernandez
Aaron Smith
Phillip Reese
Dave Lewis
David Gardner
Thomas Anderson
Paul Reed
Zachary Mays
Brittney Johnson

2) **Query 2: Retrieve number of orders for each agent from orders table**

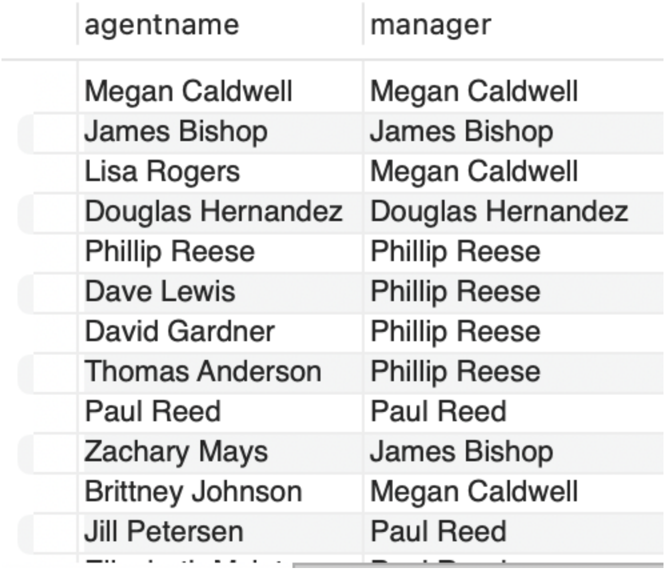
```
Select distinct agentID, count(orderInfo) as Number_of_Orders
From orders
Group by AgentID
Order by Number_of_Orders desc
```



agentID	Number_of_Orders
Agent009	5
Agent017	5
Agent043	4
Agent001	4
Agent041	4
Agent003	4
Agent012	4
Agent013	4
Agent014	4
Agent015	4
Agent024	4
Agent006	3

3) **Query 3: Retrieve all the agent name with at least one order and their manager name**

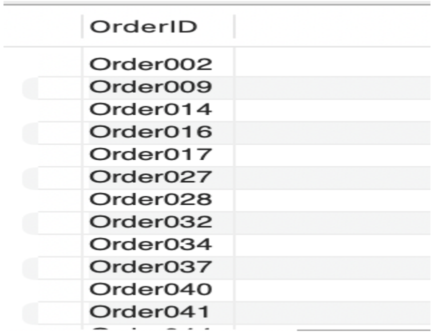
Select distinct(ag.agentname), ag.manager
From salesagent ag,orders od
Where ag.agentID=od.AgentID



agentname	manager
Megan Caldwell	Megan Caldwell
James Bishop	James Bishop
Lisa Rogers	Megan Caldwell
Douglas Hernandez	Douglas Hernandez
Phillip Reese	Phillip Reese
Dave Lewis	Phillip Reese
David Gardner	Phillip Reese
Thomas Anderson	Phillip Reese
Paul Reed	Paul Reed
Zachary Mays	James Bishop
Brittney Johnson	Megan Caldwell
Jill Petersen	Paul Reed

4) **Query 4: Retrieve orders which is currently reviewed by employee Andrew Kelly**

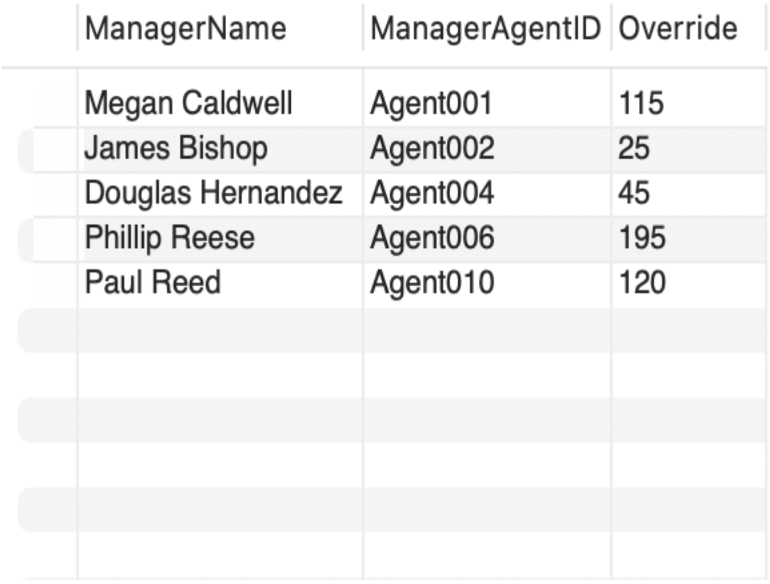
Select OrderID
From orders
Where EmployeeID =
(select employeeID
From employee
Where EmployeeName =’Andrew Kelly’)



OrderID
Order002
Order009
Order014
Order016
Order017
Order027
Order028
Order032
Order034
Order037
Order040
Order041

5) **Query 5: Retrieve Override Amount for each Manager Based on the number of orders from their team sales record (each order worth $5 Override)**

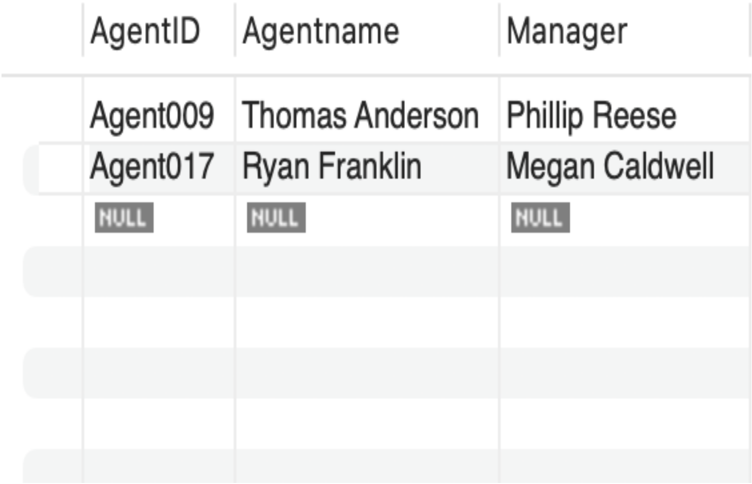
```
SELECT m.AgentName AS ManagerName, m.AgentID AS
ManagerAgentID,
   (
      SELECT COUNT(*)*5
      FROM Orders o
      WHERE o.AgentID IN (
         SELECT a.AgentID
         FROM SalesAgent a
         WHERE a.Manager = m.Manager
      )
   ) AS Override
FROM SalesAgent m
WHERE m.AgentName IN (
SELECT DISTINCT Manager FROM SalesAgent)
```



ManagerName	ManagerAgentID	Override
Megan Caldwell	Agent001	115
James Bishop	Agent002	25
Douglas Hernandez	Agent004	45
Phillip Reese	Agent006	195
Paul Reed	Agent010	120

6) **Query 6: Retrieve The AgentID, AgentName,Manager who has most sales record**

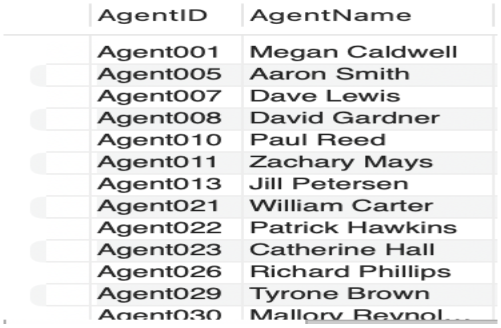
```
select AgentID,Agentname,Manager
from salesagent
where AgentID in
(select AgentID
from orders
group by AgentID
Having count(orderID)>=ALL(
select count(od1.orderID)
from orders as od1
group by od1.AgentID))
```



AgentID	Agentname	Manager
Agent009	Thomas Anderson	Phillip Reese
Agent017	Ryan Franklin	Megan Caldwell
NULL	NULL	NULL

7) **Query 7: Retrieve AgentID,AgentName Who do not have TV Order**

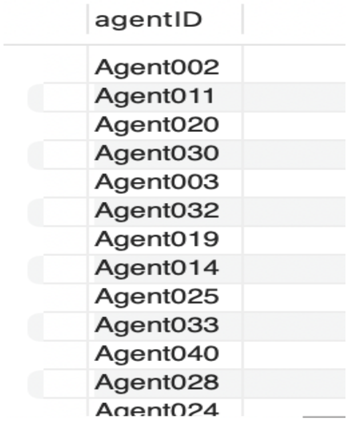
```
 select sa.AgentID,sa.AgentName
from SalesAgent sa
where not exists(
select 1
from orders od
where sa.AgentID = od.AgentID
And od.OrderInfo ='TV')
```



AgentID	AgentName
Agent001	Megan Caldwell
Agent005	Aaron Smith
Agent007	Dave Lewis
Agent008	David Gardner
Agent010	Paul Reed
Agent011	Zachary Mays
Agent013	Jill Petersen
Agent021	William Carter
Agent022	Patrick Hawkins
Agent023	Catherine Hall
Agent026	Richard Phillips
Agent029	Tyrone Brown
Agent030	Mallory Reynol...

8) **Query 8: Retrieve AgentID of agent who under manager James Bishop or he has TV order**

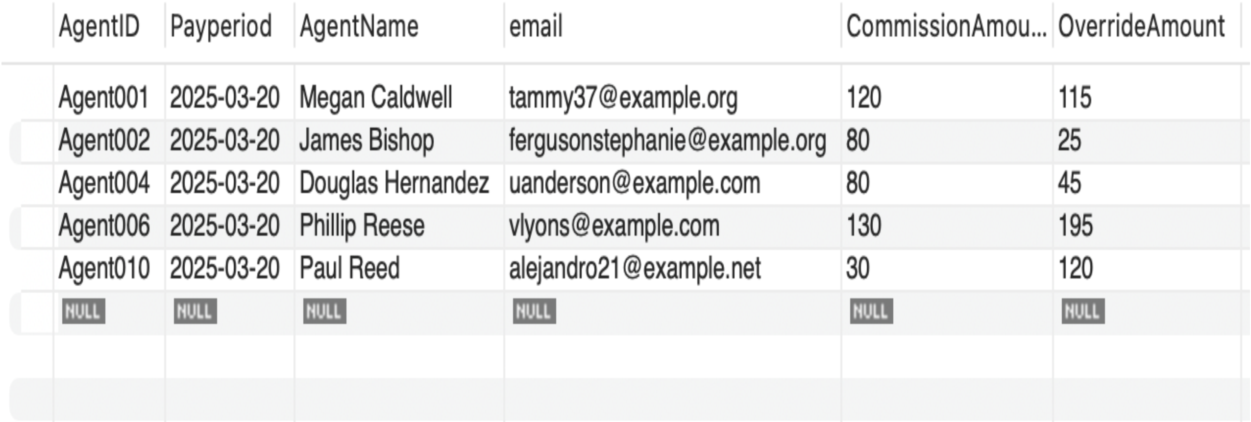
```
select agentID
from salesagent
where manager ='James Bishop'
union
select agentID
from orders
where orderinfo ='TV'
```



agentID
Agent002
Agent011
Agent020
Agent030
Agent003
Agent032
Agent019
Agent014
Agent025
Agent033
Agent040
Agent028
Agent024

9) **Query 9: Insert Override amount into managercommission(Before this query, the overrideamount column is empty for each manager)**

```
SET SQL_SAFE_UPDATES = 0;
UPDATE managercommission mc
JOIN (
    SELECT m.AgentID AS ManagerAgentID,
        (
            SELECT COUNT(*) * 5
            FROM Orders o
            WHERE o.AgentID IN (
                SELECT a.AgentID
                FROM SalesAgent a
                WHERE a.Manager = m.Manager
            )
        ) AS Override
    FROM SalesAgent m
    WHERE m.AgentName IN (
        SELECT DISTINCT Manager FROM SalesAgent
    )
) AS calc ON mc.AgentID = calc.ManagerAgentID
SET mc.OverrideAmount = calc.Override
```

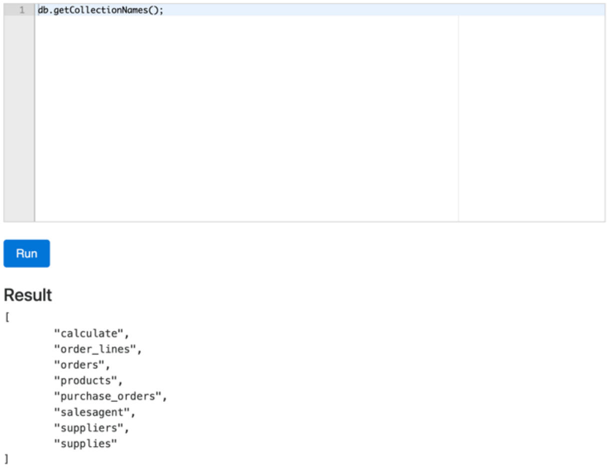


AgentID	Payperiod	AgentName	email	CommissionAmou...	OverrideAmount
Agent001	2025-03-20	Megan Caldwell	tammy37@example.org	120	115
Agent002	2025-03-20	James Bishop	fergusonstephanie@example.org	80	25
Agent004	2025-03-20	Douglas Hernandez	uanderson@example.com	80	45
Agent006	2025-03-20	Phillip Reese	vlyons@example.com	130	195
Agent010	2025-03-20	Paul Reed	alejandro21@example.net	30	120
NULL	NULL	NULL	NULL	NULL	NULL

NOSQL IMPLEMENTATION:

Three tables(Salesagent,Orders,Calculate) has been added into MongoDB playground. Here are the three following queries

1) Query 1: Retrieve all the collection we have

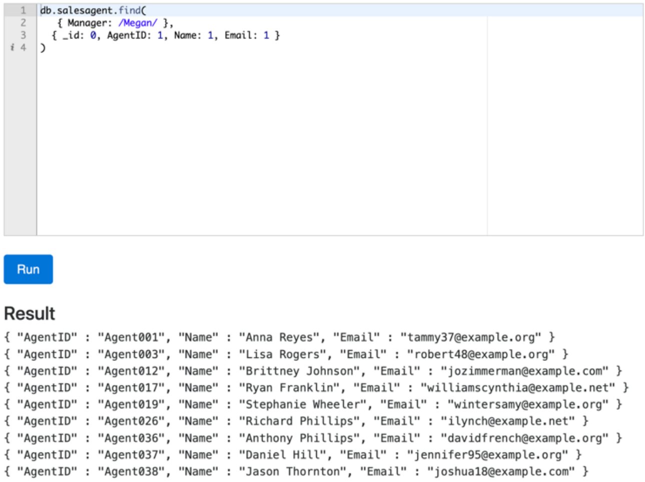


```
db.getCollectionNames();
```

Result

```
[
        "calculate",
        "order_lines",
        "orders",
        "products",
        "purchase_orders",
        "salesagent",
        "suppliers",
        "supplies"
]
```

2) Query 2: Retrieve all agents under Manager whose name contains Megan

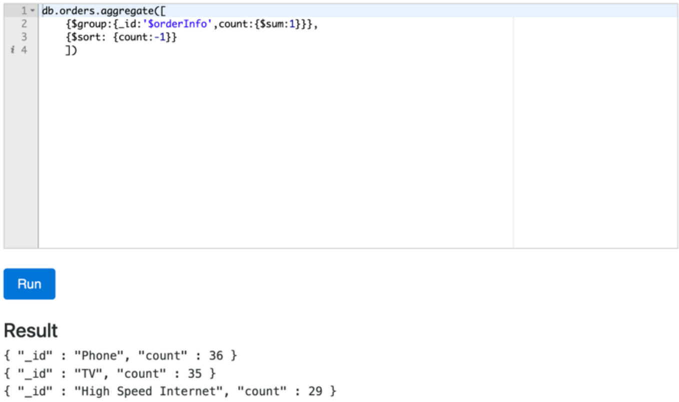


```
db.salesagent.find(
    { Manager: /Megan/ },
  { _id: 0, AgentID: 1, Name: 1, Email: 1 }
)
```

Result

```
{ "AgentID" : "Agent001", "Name" : "Anna Reyes", "Email" : "tammy37@example.org" }
{ "AgentID" : "Agent003", "Name" : "Lisa Rogers", "Email" : "robert48@example.org" }
{ "AgentID" : "Agent012", "Name" : "Brittney Johnson", "Email" : "jozimmerman@example.com" }
{ "AgentID" : "Agent017", "Name" : "Ryan Franklin", "Email" : "williamscynthia@example.net" }
{ "AgentID" : "Agent019", "Name" : "Stephanie Wheeler", "Email" : "wintersamy@example.org" }
{ "AgentID" : "Agent026", "Name" : "Richard Phillips", "Email" : "ilynch@example.net" }
{ "AgentID" : "Agent036", "Name" : "Anthony Phillips", "Email" : "davidfrench@example.org" }
{ "AgentID" : "Agent037", "Name" : "Daniel Hill", "Email" : "jennifer95@example.org" }
{ "AgentID" : "Agent038", "Name" : "Jason Thornton", "Email" : "joshua18@example.com" }
```

3) Query 3: Retrieve order type in groups and count number of orders for each order type

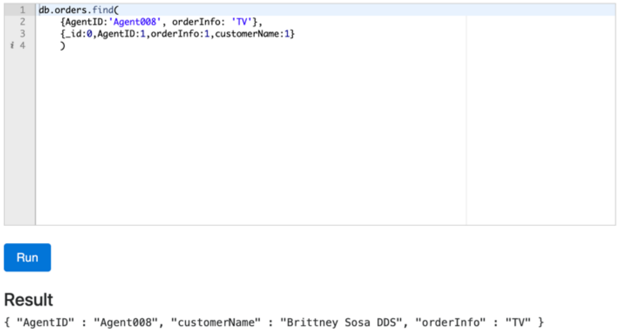


```
db.orders.aggregate([
    {$group:{_id:'$orderInfo',count:{$sum:1}}},
    {$sort: {count:-1}}
    ])
```

Result

```
{ "_id" : "Phone", "count" : 36 }
{ "_id" : "TV", "count" : 35 }
{ "_id" : "High Speed Internet", "count" : 29 }
```

4) Query 4: Retrieve TV order by Agent008



```
db.orders.find(
    {AgentID:'Agent008', orderInfo: 'TV'},
    {_id:0,AgentID:1,orderInfo:1,customerName:1}
    )
```

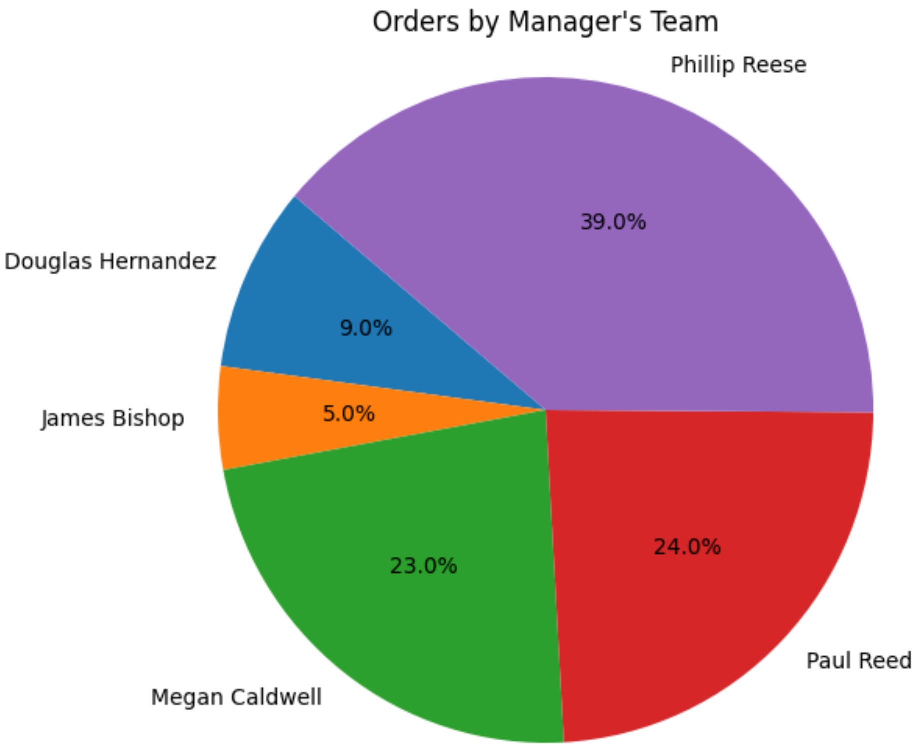
Result

```
{ "AgentID" : "Agent008", "customerName" : "Brittney Sosa DDS", "orderInfo" : "TV" }
```

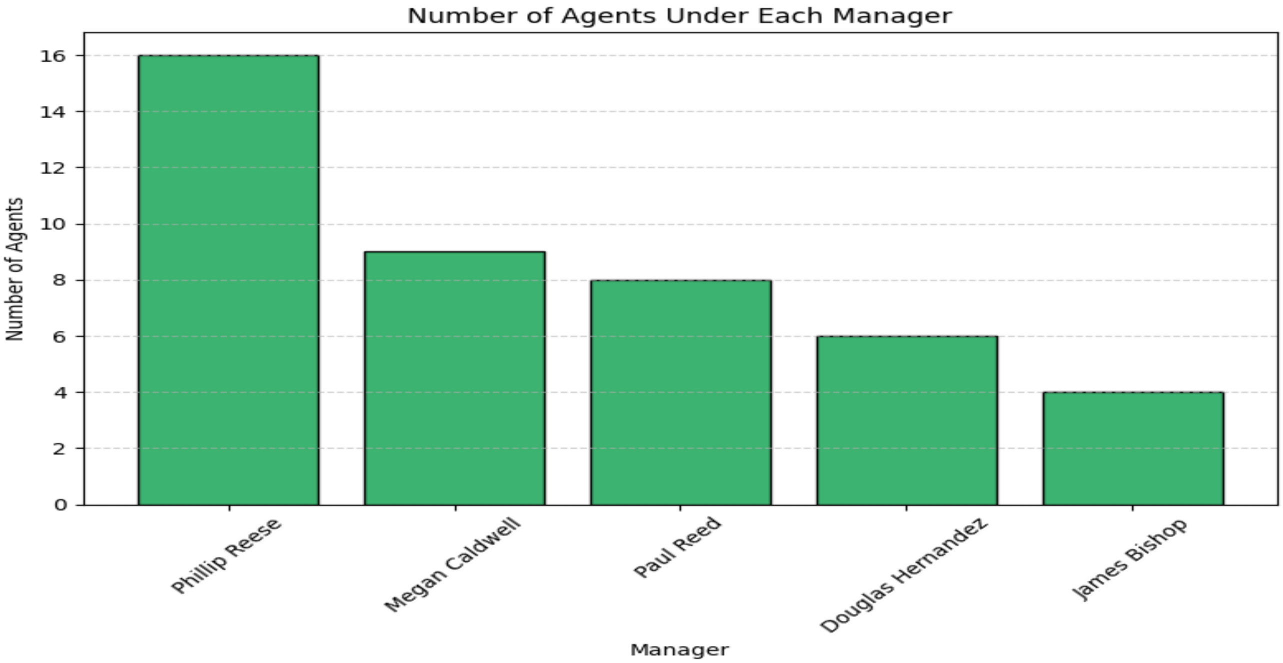
DATABASE ACCESS VIA PYTHON:

The database is accessed through python and using python code to visulize the analysis of data.

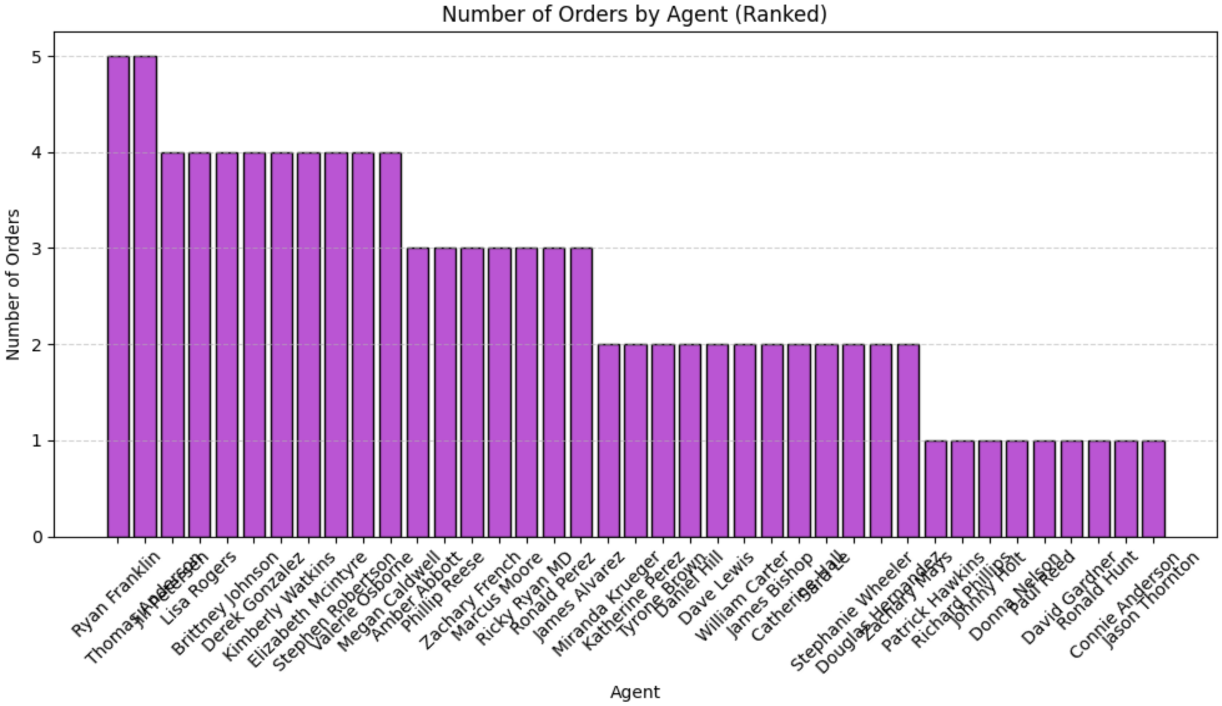
Graph 1: Using piechart to show the percentage of orders for each team



Graph 2: Using histrogram to show number of agents under each manager



Graph 3: Using Histrogram to show number of orders for each agent, follow by number of orders Descending



SUMMARY

The simulation of the direct sales company is grounded in real-world work experience, offering a realistic and practical representation of how such a business operates. It effectively visualizes the performance of each sales team, enabling business owners and decision-makers to gain valuable insights into sales dynamics and team effectiveness. With this data-driven approach, owners can make informed decisions on how to strategically improve operations, boost productivity, and optimize commission structures.

Although the simulation showcases key aspects of the business process, there are still several advanced functionalities that remain to be implemented. For example, one important enhancement would be the ability to assign role-based access controls, ensuring that agents at different levels within the hierarchy only see the information relevant to them. Another future development includes integrating the Python-based application into a cloud platform, which would allow seamless access to real-time sales and commission data for all members of the company—regardless of their location. These planned features will further increase the system's scalability, transparency, and overall usability, ultimately contributing to more efficient and informed business management.