USE CASE STUDY REPORT

Group No: Group 2
Student Names: Aravind Anbazhagan & Aniruthan SA

Enhancing Customer Service Excellence: A Comprehensive Incident Management Solution

I)Executive Summary:

The emerging technology firm's strategic initiative focuses on establishing a specialized service team to address customer concerns for its cloud-based product offerings. This initiative includes a robust cloud-based monitoring tool that triggers incidents for immediate resolution. The service structure is tiered: Level 1 (L1) engineers handle initial incident assessments and monitoring, Level 2 (L2) engineers conduct in-depth troubleshooting, and Level 3 (L3) engineers tackle recurrent problem analysis and solution recommendation. The incident management workflow is comprehensive, involving incidents, cases, problems, and change requests, each linked in a structured manner and subject to strict oversight and approval. Central to this initiative is the development of an advanced database system, designed to centralize, streamline, and analyze incident-related data. This system will automate incident handling, enhance knowledge management, and feature KPI dashboards for monitoring SLA compliance, identifying automation opportunities, and optimizing resource use. The ultimate goal is to transform customer service from reactive to proactive, ensuring a data-driven, process-improvement approach that elevates customer satisfaction.

The technology firm aims to develop a comprehensive database system to enhance its customer service for cloud-based products. The system will centralize incident data, automate incident handling, and manage knowledge effectively. It's designed to address challenges in tracking and resolving incidents, cases, problems, and changes across different support levels (L1, L2, L3). The goal is to improve SLA compliance, identify automation opportunities, and optimize resource utilization through a dashboard that supports data-driven decision-making and process improvement in incident management workflows.

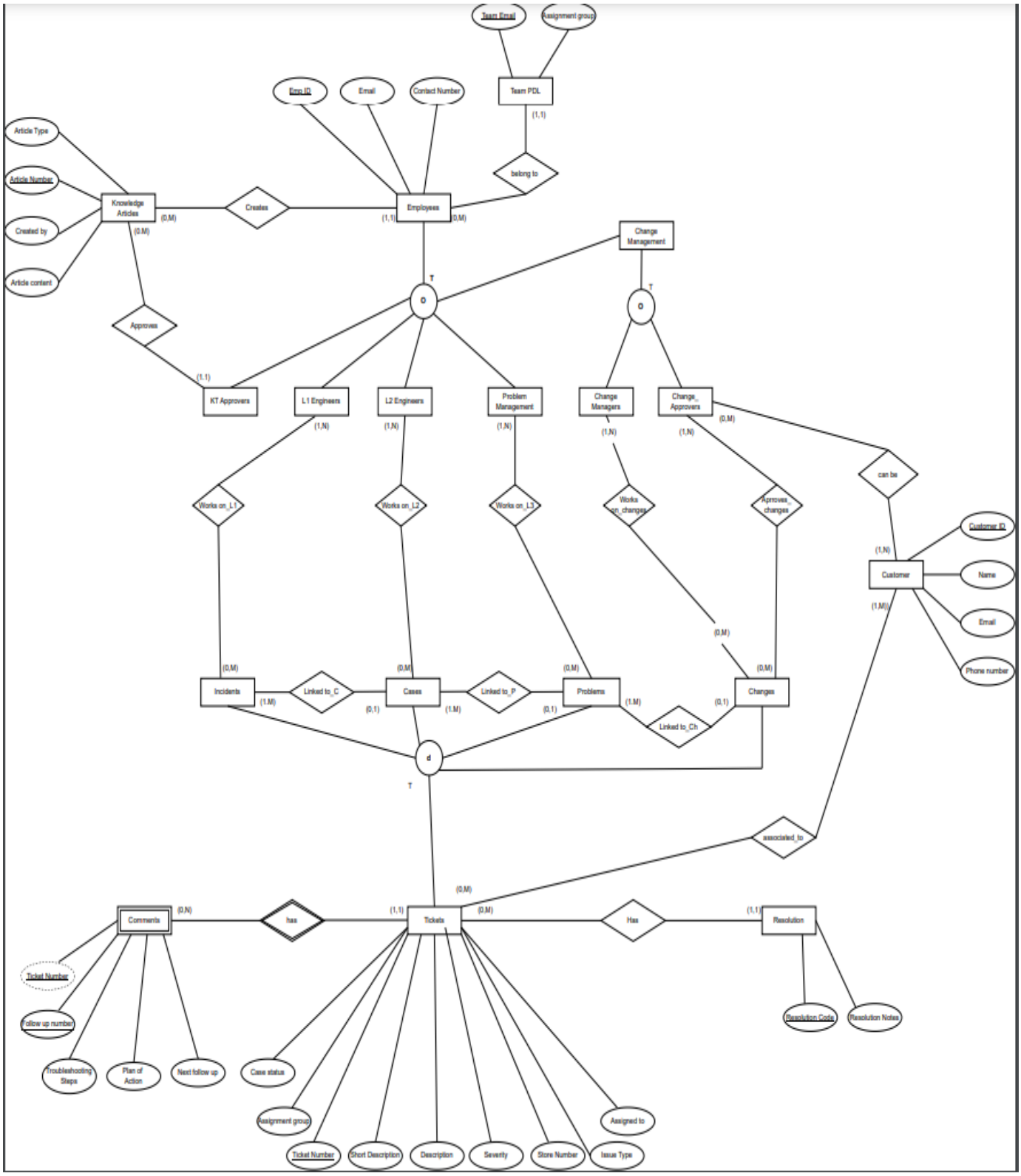
Introduction

We unveil our strategic initiative to revolutionize customer service for our cloud-based products. We propose a comprehensive database system designed to centralize and streamline incident-related data. This innovative system aims to automate incident handling, enhance knowledge management, and provide insightful KPI dashboards for SLA compliance, automation opportunities, and resource optimization. Our goal is to transform customer service from reactive to proactive, ensuring data-driven decision-making and elevating customer satisfaction.

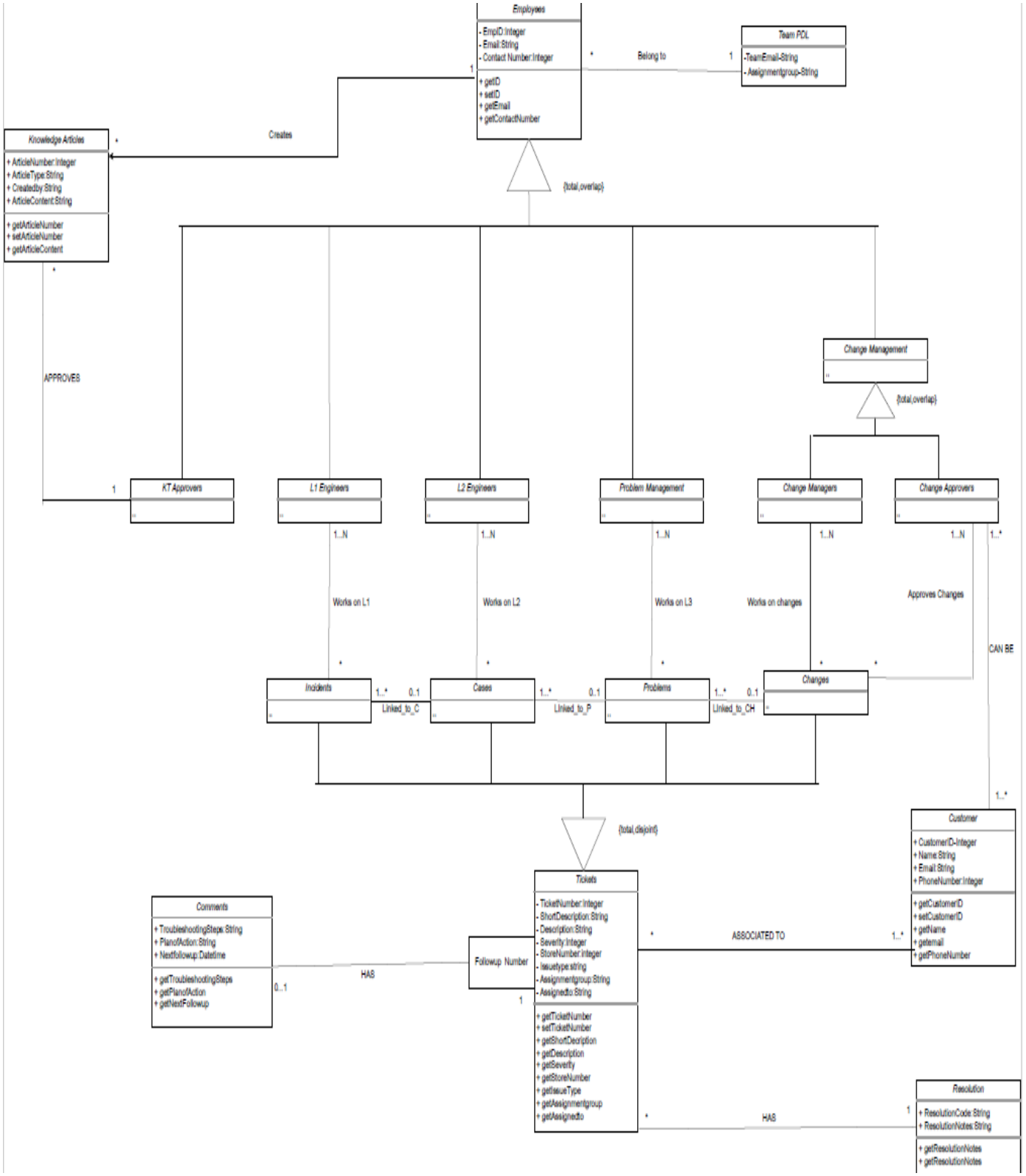
\

II) CONCEPTUAL MODELLING:

EER MODEL:



UML DIAGRAM:


Employees
- EmpID:Integer
- Email:String
- Contact Number:Integer
+ getID
+ setID
+ getEmail
+ getContactNumber
Belong to
Team PDL
-TeamEmail-String
- Assignmentgroup-String
Knowledge Articles
+ ArticleNumber:Integer
+ ArticleType:String
+ Createdby:String
+ ArticleContent:String
+ getArticleNumber
+ setArticleNumber
+ getArticleContent
Creates
{total,overlap}
APPROVES
Change Management
{total,overlap}
KT Approvers
L1 Engineers
L2 Engineers
Problem Management
Change Managers
Change Approvers
Works on L1
Works on L2
Works on L3
Works on changes
Approves Changes
CAN BE
Incidents
Cases
Problems
Changes
Linked_to_C
Linked_to_P
Linked_to_CH
{total,disjoint}
Tickets
- TicketNumber:Integer
- ShortDescription:String
- Description:String
- Severity:Integer
- StoreNumber:Integer
- issuetype:string
- Assignmentgroup:String
- Assignedto:String
+ getTicketNumber
+ setTicketNumber
+ getShortDecription
+ getDescription
+ getSeverity
+ getStoreNumber
+ getIssueType
+ getAssignmentgroup
+ getAssignedto
Customer
+ CustomerID-Integer
+ Name:String
+ Email:String
+ PhoneNumber:Integer
+ getCustomerID
+ setCustomerID
+ getName
+ getemail
+ getPhoneNumber
ASSOCIATED TO
Comments
+ TroubleshootingSteps:String
+ PlanofAction:String
+ Nextfollowup:Datetime
+ getTroubleshootingSteps
+ getPlanofAction
+ getNextFollowup
HAS
Followup Number
Resolution
+ ResolutionCode:String
+ ResolutionNotes:String
+ getResolutionNotes
+ getResolutionNotes
HAS

III) RELATIONAL MODEL:

•Employees (Emp name,Emp ID, Email, Contact number, Team Email) Primary Key: Emp ID

Foreign Key: Team email refers to Team PDL and it's NOT NULL

•Team PDL (Team Email, Assignment group) Primary Key: Team Email

•Knowledge Articles (Article Type, Article Number, created by, Article content, Emp ID)

Primary Key: Article Number

Foreign Key: EMP ID refers to Team Employees and it's NOT NULL

•L1 Engineers (Emp ID), L2 Engineers (Emp ID), Problem Management (Emp ID), Change Management (Emp ID), Change Managers (Emp ID), Change Approvers (Emp ID)

Emp ID is the primary key for the above entities and foreign key referring to employees.

•Incidents (Incident_Ticket_Number, Case_Ticket_number) Primary Key: Incident_Ticket_Number

Foreign Key: Incident_Ticket_Number refers to tickets and Case_Ticket_Number refers to Cases. NULL values are allowed

•Cases (Case_Ticket_Number, PRB_Ticket_Number) Primary Key: Case_Ticket_Number

Foreign Key: Case_Ticket_Number refers to tickets and PRB_Ticket_Number refers to Problems. NULL values are allowed

•Problems (PRB_Ticket_Number, CHG_Ticket_Number) Primary Key: PRB_Ticket_Number

Foreign Key: PRB_Ticket_Number refers to problems and CHG_Ticket_Number refers to Changes. NULL values are allowed

•Changes (CHG_Ticket_Number) Primary Key: CHG_Ticket_Number

Foreign Key: CHG_Ticket_Number refers to tickets.

•Tickets (Ticket Number, Case status, Short Description, Description, Severity, Store Number, Issue Type, Assigned to, Resolution Code)

Primary Key: Ticket Number

Foreign Key: Resolution code refers to Resolution. NULL values not allowed.

•Resolution (Resolution Code, Resolution Notes) Primary Key: Resolution Code

•Comments (Ticket Number, Follow up Number, Troubleshooting steps, Plan of Action, Next Follow up)

Primary Key: Ticket Number, Follow up Number

Foreign Key: Ticket Number refers to Comments. NULL values not allowed.

•Works_on_L1 (Emp ID, Incident_Ticket_Number) Primary Key: Emp ID, Incident_Ticket_Number

Foreign Key: Emp Id refers to L1 Engineers and Incident ticket number refers to Incidents

•Works_on_L2 (Emp ID, Case_Ticket_Number) Primary Key: Emp ID, Case_Ticket_Number

Foreign Key: Emp Id refers to L2 Engineers and Case ticket number refers to Cases

•Works_on_L3 (Emp ID, PRB_Ticket_Number)
Primary Key: Emp ID, PRB_Ticket_Number

Foreign Key: Emp Id refers to Problem management and EMP ID number refers to Problems

•Works_on_changes (Emp ID, CHG_Ticket_Number)
Primary Key: Emp ID, CHG_Ticket_Number

Foreign Key: Emp Id refers to changes and EMP ID number refers to change managers

•Approve changes (Emp ID, CHG_Ticket_Number)
Primary Key: Emp ID, CHG_Ticket_Number

Foreign Key: Emp Id refers to changes and EMP ID number refers to change approvers

•Customer (Customer ID, Name, Email, Phone Number) Primary Key: Customer ID

•Can be (Emp ID, Customer ID) Primary Key: Emp ID, Customer ID

Foreign Key: Emp Id refers to change approvers and Customer ID refers to customer

•Associated _to (Ticket number, Customer ID) Primary Key: Ticket Number and Customer ID

Foreign Key: Ticket Number refers to tickets and Customer ID refers to customer

IV) Implementation in MySQL:

1.) Simple query: Retrieves all columns for employees with team_email=L1_engineers@dma.com.

SELECT * FROM Employees where team_email='L1_engineers@dma.com';

emp_name	emp_id	personal_email	contact_number	team_email
Employee1	1	Employee1@example.com	1234560001	L1_Engineers@dma.com
Employee5	5	Employee5@example.com	1234560005	L1_Engineers@dma.com
Employee9	9	Employee9@example.com	1234560009	L1_Engineers@dma.com
Employee13	13	Employee13@example.com	1234560013	L1_Engineers@dma.com
Employee17	17	Employee17@example.com	1234560017	L1_Engineers@dma.com
Employee21	21	Employee21@example.com	1234560021	L1_Engineers@dma.com
Employee25	25	Employee25@example.com	1234560025	L1_Engineers@dma.com
Employee29	29	Employee29@example.com	1234560029	L1_Engineers@dma.com
Employee33	33	Employee33@example.com	1234560033	L1_Engineers@dma.com

2.) Aggregate query: Counts the number of tickets for each distinct resolution code in the Tickets table.

SELECT resolution_code, count(*) as resolution_count from tickets group by resolution_code order by resolution_count desc;

resolution_code	resolution_count
reboot	17
replacement	17
updated configuration	17
vendor resolved	17
others	16
remote fix	16

3.) Joins: Retrieves the names of employees and their corresponding assignment group by joining the Employees and Team_PDL tables on the team_email.:

SELECT Employees.emp_name, Team_PDL.assignment_group
FROM Employees INNER JOIN Team_PDL
ON Employees.team_email = Team_PDL.team_email;

emp_name	assignment_group
Employee4	Change_Management
Employee8	Change_Management
Employee12	Change_Management
Employee16	Change_Management
Employee20	Change_Management
Employee24	Change_Management
Employee28	Change_Management
Employee32	Change_Management
Employee36	Change_Management
Employee40	Change_Management

Outer join: Retrieves all employees' names and their associated knowledge article numbers, including those without an article, by performing a left join on the Employees and Knowledge_Articles tables.

```
SELECT Employees.emp_name, Knowledge_Articles.article_number
FROM Employees LEFT OUTER JOIN Knowledge_Articles
ON Employees.emp_id = Knowledge_Articles.emp_id;
```

emp_name	article_number
Employee1	KA0001
Employee2	KA0002
Employee3	KA0003
Employee4	KA0004
Employee5	KA0005
Employee6	KA0006
Employee7	KA0007
Employee8	KA0008
Employee9	KA0009
Employee10	KA0010
Employee11	KA0011

4.) Nested query: Selects the names of employees who are listed as L1 engineers by using a subquery within the WHERE clause.

```
SELECT emp_name
FROM Employees
WHERE emp_id IN (SELECT emp_id FROM L1_engineers);
```

emp_name
Employee1
Employee5
Employee9
Employee13
Employee17
Employee21
Employee25
Employee29
Employee33
Employee37
Employee41

5.) Correlated query: For each employee, counts how many tickets are assigned to them using a correlated subquery.

```
SELECT emp_name, (SELECT COUNT(*)
FROM Tickets WHERE Tickets.assigned_to = Employees.emp_name)
AS num_tickets FROM Employees;
```

emp_name	num_tickets
Employee1	1
Employee2	1
Employee3	1
Employee4	1
Employee5	1
Employee6	1
Employee7	1
Employee8	1
Employee9	1
Employee10	1
Employee11	1

6.) Query using ALL: Selects the names of employees whose emp_id is greater than or equal to all emp_ids in the L1_engineers table.

SELECT emp_name FROM Employees WHERE emp_id >= ALL (SELECT emp_id FROM L1_engineers);

emp_name
Employee97
Employee98
Employee99
Employee100

7)Query using ANY: Selects the names of employees whose emp_id is greater than the emp_id of any engineer in the L1_engineers table.
SELECT emp_name FROM Employees WHERE emp_id > ANY (SELECT emp_id FROM L1_engineers);

emp_name
Employee2
Employee3
Employee4
Employee5
Employee6
Employee7
Employee8
Employee9
Employee10
Employee11
Employee12

8.) Query using EXISTS: Selects the names of employees who have at least one ticket assigned to them..

SELECT emp_name FROM Employees WHERE EXISTS (SELECT * FROM Tickets WHERE Tickets.assigned_to = Employees.emp_name);

emp_name
Employee1
Employee2
Employee3
Employee4
Employee5
Employee6
Employee7
Employee8
Employee9
Employee10
Employee11

9)Query using not exists: Selects the names of employees who do not have any tickets assigned to them.
SELECT emp_name FROM Employees WHERE NOT EXISTS (SELECT * FROM Tickets WHERE Tickets.assigned_to = Employees.emp_name);

emp_name

10.) Set operations: Selects all unique employee IDs from both L1 and Knowledge_articles tables.

SELECT emp_id FROM L1_engineers UNION SELECT emp_id FROM knowledge_articles;

emp_id
1
5
9
13
17
21
25
29
33
37
41

9.) Subqueries in select:
For each employee, selects the name and uses a subquery to retrieve the corresponding assignment group from the Team_PDL table.
SELECT emp_name, (SELECT assignment_group FROM Team_PDL WHERE Team_PDL.team_email = Employees.team_email) as assignment_group FROM Employees;

emp_name	assignment_group
Employee1	L1_Engineers
Employee2	L2_Engineers
Employee3	Problem_Management
Employee4	Change_Management
Employee5	L1_Engineers
Employee6	L2_Engineers
Employee7	Problem_Management
Employee8	Change_Management
Employee9	L1_Engineers
Employee10	L2_Engineers
Employee11	Problem_Management

10) Subqueries using from Selects all unique employee IDs from both L1 and Knowledge_articles tables.
SELECT emp_id FROM L1_engineers UNION SELECT emp_id FROM knowledge_articles;

emp_id
1
5
9
13
17
21
25
29
33
37
41

V) Implementation in NoSQL:

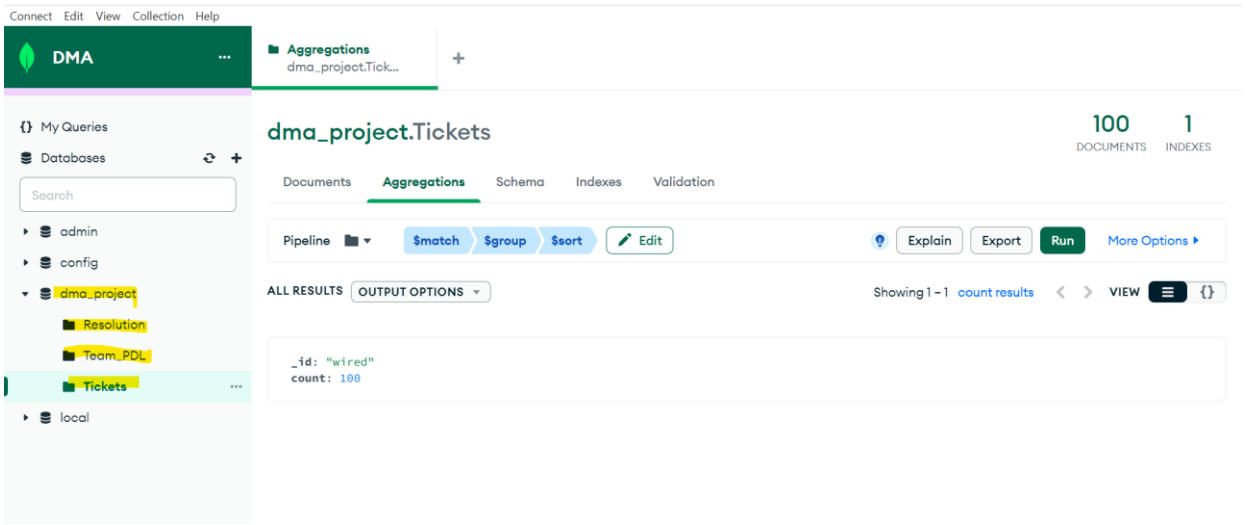
The “dma_project” database tables were converted into csv files using the below python script.

```
import pymysql
import pandas as pd

# Database connection details
host = 'localhost'  # e.g., 'localhost'
user = 'root'
password = 'admin123'
database = 'dma_project'

# Establishing a database connection
connection = pymysql.connect(host=host, user=user, password=password, database=database)

try:
    # Fetching the list of tables
    with connection.cursor() as cursor:
        cursor.execute("SHOW TABLES")
        tables = cursor.fetchall()
```

```
    # Export each table to a CSV file
    for table_name in tables:
        query = f"SELECT * FROM {table_name[0]}"
        df = pd.read_sql(query, connection)
        df.to_csv(f"{table_name[0]}.csv", index=False)

finally:
    # Closing the database connection
    connection.close()
```

NoSQL implementation:

- Downloaded and Installed MongoDB compass
- Created a new database dma_project with three collections namely Resolution, Team_PDL and Tickets.



- Imported data for the three collections using the csv files generated from SQL database tables using the python script.
- After importing the data, three queries were implemented using aggregation.

When working with the aggregation framework in MongoDB Compass, you can build your pipeline using the graphical user interface. Below are examples of what each of these queries would look like as stages in an aggregation pipeline.

- A Simple Query (as an Aggregation)
  To find all documents where `case_status` is "new":

  Stage 1: `$match`
  ```json
  { "case_status": "new" }
```

- A More Complex Query (as an Aggregation)
  To find all documents where `case_status` is "new" and either `issue_type` is "wired" or `resolution_code` is "reboot":

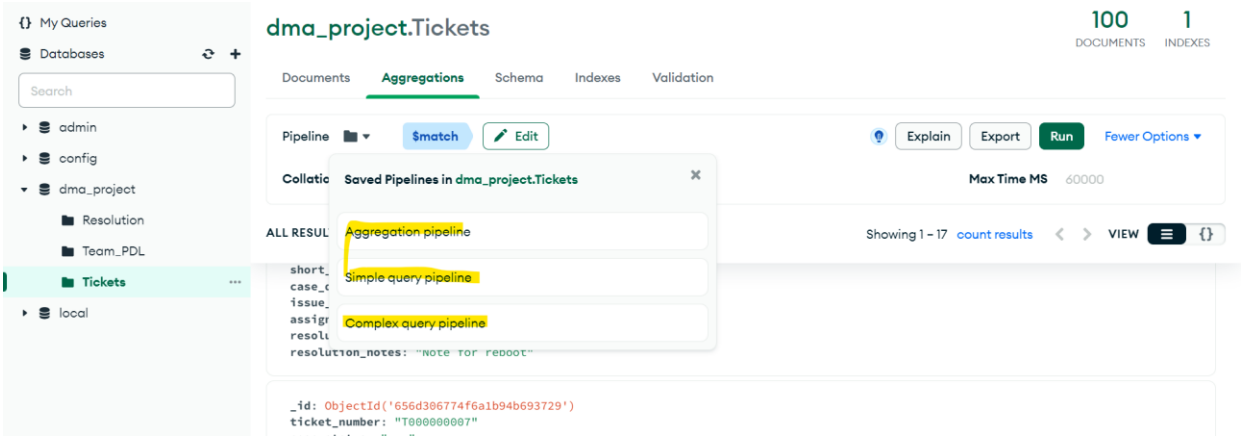
Stage 1: `$match`
```json
{
  "$and": [
    { "case_status": "new" },
    {
      "$and": [
        { "issue_type": "wired" },
        { "resolution_code": "reboot" }
      ]
    }
  ]
}
```

- An Aggregate Query
  To group documents by `issue_type` and count them, only including documents where `case_status` is "new":

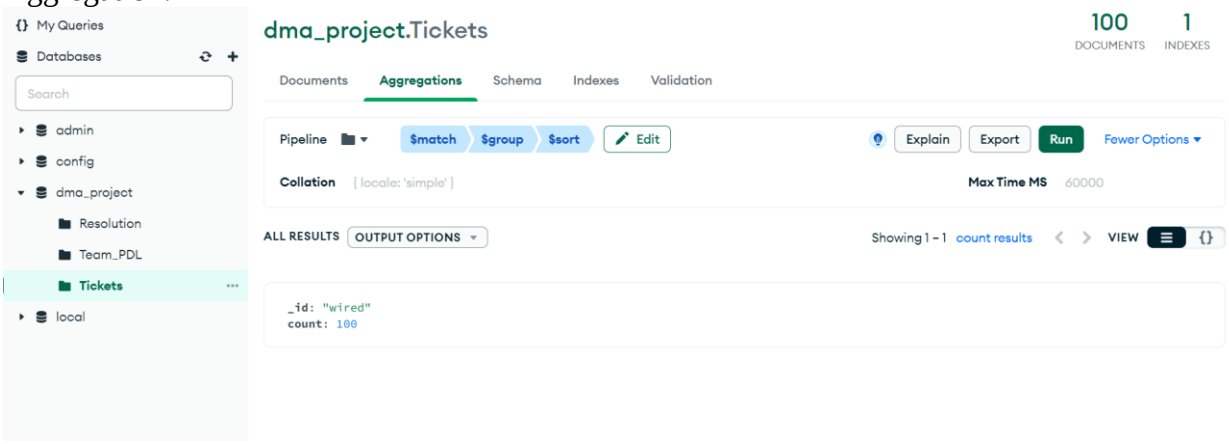
  Stage 1: `$match`
  ```json
  { "case_status": "new" }
  ```

  Stage 2: `$group`
  ```json
  {
    "_id": "$issue_type",
    "count": { "$sum": 1 }
  }
  ```

  Stage 3: `$sort`
  ```json
  { "count": -1
  ```
- Saved the three pipelines as shown in the below image.



- Outputs of the queries:
  1. Aggregation:



  2. Simple:


```
{
 "_id": {
  "$oid": "656d306774f6a1b94b693723"
 },
 "ticket_number": "T000000001",
 "case_status": "new",
 "short_desc": "Short Desc 1",
 "case_desc": "Case Desc 1",
 "issue_type": "wired",
 "assigned_to": "Assignee1",
 "resolution_code": "reboot",
 "resolution_notes": "Note for reboot"
}

{
 "_id": {
  "$oid": "656d306774f6a1b94b693724"
 },
 "ticket_number": "T000000002",
 "case_status": "new",
 "short_desc": "Short Desc 2",
 "case_desc": "Case Desc 2",
 "issue_type": "wired",
 "assigned_to": "Assignee2",
 "resolution_code": "replacement",
 "resolution_notes": "Note for replacement"
}
```


Etc…

  3. Complex:


```
{
 "_id": {
  "$oid": "656d306774f6a1b94b693723"
 },
 "ticket_number": "T000000001",
 "case_status": "new",
```

```
  "short_desc": "Short Desc 1",
  "case_desc": "Case Desc 1",
  "issue_type": "wired",
  "assigned_to": "Assignee1",
  "resolution_code": "reboot",
  "resolution_notes": "Note for reboot"
}

{
  "_id": {
    "$oid": "656d306774f6a1b94b693729"
  },
  "ticket_number": "T000000007",
  "case_status": "new",
  "short_desc": "Short Desc 7",
  "case_desc": "Case Desc 7",
  "issue_type": "wired",
  "assigned_to": "Assignee7",
  "resolution_code": "reboot",
  "resolution_notes": "Note for reboot"
}
Etc…
```


VI) Implementation in Python via Database Access

The database is accessed using Python and visualization of analyzed data is shown below. The connection of MySQL to Python is done using mysql alchemy, followed by pandas.read_sql fundtion to fetch the sql query and run it, storing it into a dataframe using the pandas library. Furthermore, used matplotlib to plot the graphs from the dataframes for analytics.

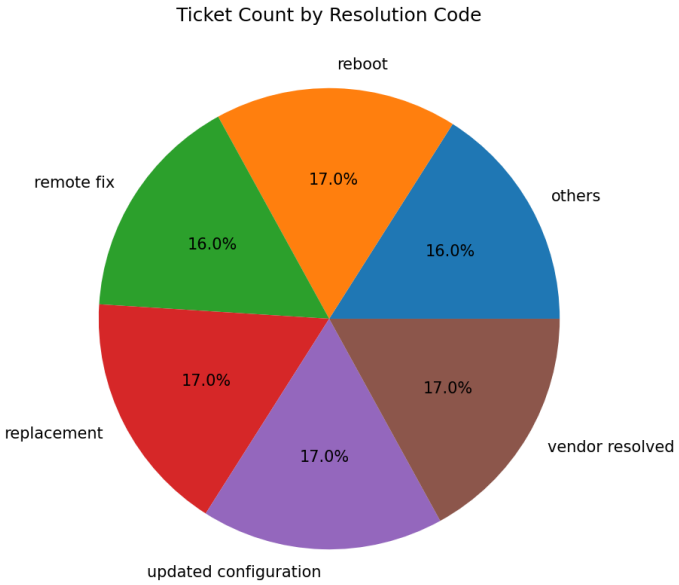
Code:

```
import pymysql
import matplotlib.pyplot as plt

# Define the database connection
connection = pymysql.connect(host='localhost',
                             user='ani',
                             password='ani123',
                             database='dma_project2')


# Define the queries
query1 = "SELECT resolution_code, COUNT(*) as resolution_count FROM tickets
GROUP BY resolution_code"
query2 = "SELECT emp_name, (SELECT COUNT(*) FROM tickets WHERE
tickets.assigned_to = employees.emp_name) as ticket_count FROM employees"
query3 = "SELECT case_status, COUNT(*) as status_count FROM tickets GROUP BY
case_status"
query4 = """
SELECT E.emp_id, COUNT(*) as ticket_count
FROM tickets T
JOIN employees E ON T.assigned_to = E.emp_name
GROUP BY E.emp_id
"""
```

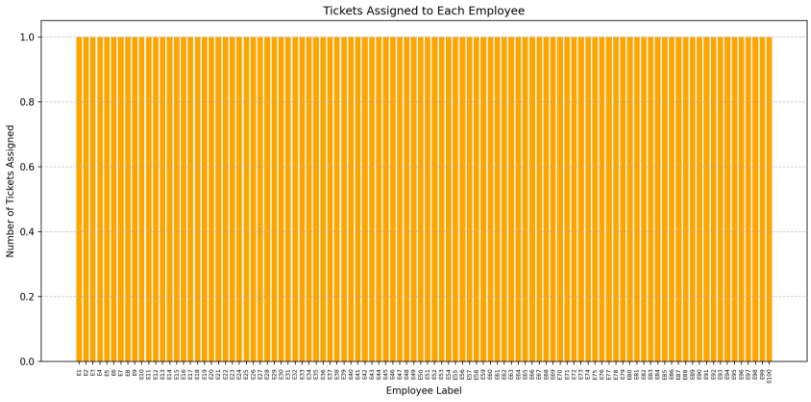
```
# Execute the queries and retrieve the results
try:
    with connection.cursor(pymysql.cursors.DictCursor) as cursor:
        cursor.execute(query1)
        result1 = cursor.fetchall()

        cursor.execute(query2)
        result2 = cursor.fetchall()

        cursor.execute(query3)
        result3 = cursor.fetchall()

        cursor.execute(query4)
        result4 = cursor.fetchall()

finally:
    connection.close()

# Process the data for each query
# Pie chart data
resolution_codes = [row['resolution_code'] for row in result1]
resolution_counts = [row['resolution_count'] for row in result1]

# Bar chart data
emp_names = [row['emp_name'] for row in result2]
ticket_counts = [row['ticket_count'] for row in result2]
emp_labels = ['E{}'.format(i+1) for i in range(len(emp_names))]

# Histogram data
case_statuses = [row['case_status'] for row in result3]
status_counts = [row['status_count'] for row in result3]

# Scatter plot data
emp_ids = [row['emp_id'] for row in result4]
resolved_ticket_counts = [row['ticket_count'] for row in result4]

# Pie chart
plt.figure(figsize=(8, 8))
plt.pie(resolution_counts, labels=resolution_codes, autopct='%1.1f%%')
plt.title('Ticket Count by Resolution Code')
plt.show()

# Bar chart with improved labeling
plt.figure(figsize=(12, 6))
plt.bar(emp_labels, ticket_counts, color='orange')
plt.title('Tickets Assigned to Each Employee')
plt.xlabel('Employee Label')
plt.ylabel('Number of Tickets Assigned')
plt.xticks(rotation=90, fontsize=6)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Histogram
plt.figure(figsize=(8, 6))
plt.hist(status_counts, bins=len(set(case_statuses)), color='green')
plt.title('Histogram of Tickets per Case Status')
plt.xlabel('Number of Tickets')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

```
# Scatter plot with improved labeling
plt.figure(figsize=(10, 6))
plt.scatter(emp_ids, resolved_ticket_counts, color='red', s=10)  # Adjusted
marker size
plt.title('Employee Performance')
plt.xlabel('Employee ID')
plt.ylabel('Number of Tickets Resolved')
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.tight_layout()
plt.show()
```

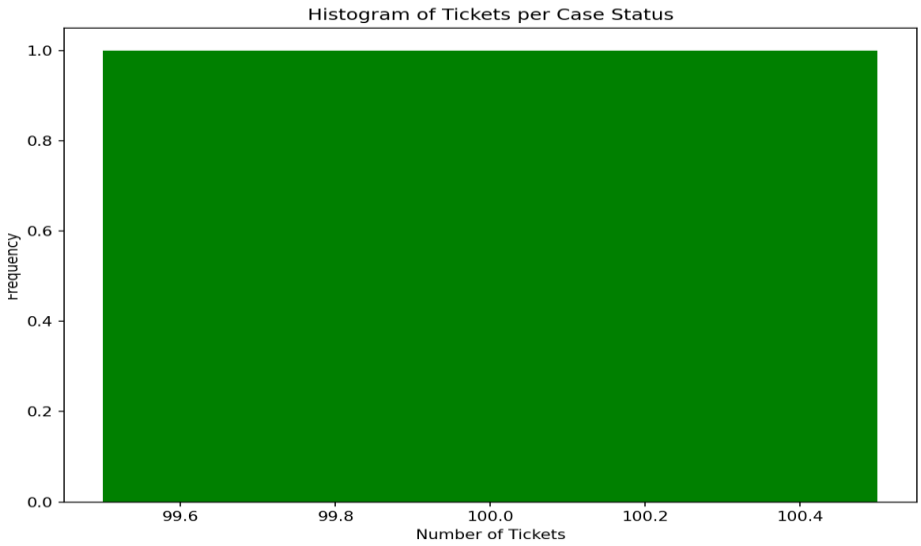
GRAPH 1:



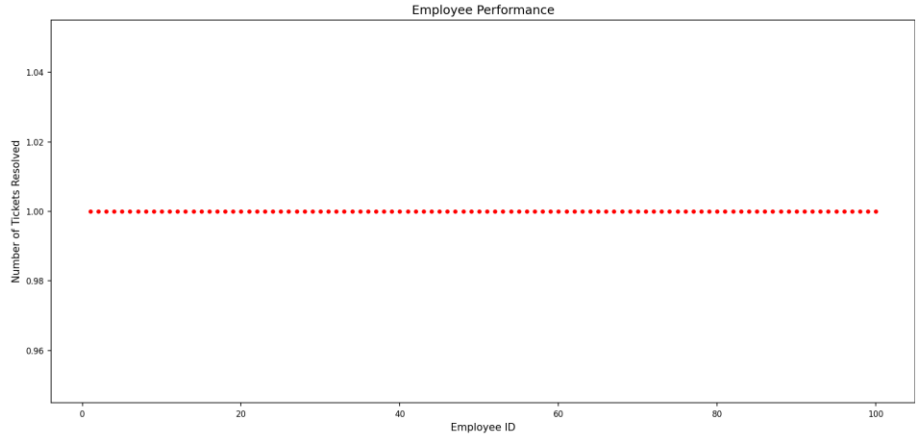
GRAPH 2:



GRAPH 3:



Graph 4:



VIII) Conclusion

In conclusion, the strategic initiative of the technology firm to establish a specialized service team for its cloud-based product offerings is poised to significantly enhance customer service efficiency. By implementing a comprehensive database system, the firm aims to centralize and streamline incident-related data management. This system will facilitate automated incident handling and efficient knowledge management. The introduction of an insightful dashboard, featuring key performance indicators (KPIs), will be instrumental in monitoring Service Level Agreement (SLA) compliance, identifying automation opportunities, and optimizing resource allocation. Overall, this initiative is expected to drive data-driven decision-making and continuous process improvement in customer service and incident management, thereby boosting the firm's operational effectiveness and customer satisfaction.