Comprehensive Database Management System for Mining and Stone Crushing Operations

Milestone: Project Report

Group 20
Nithin Arumugam Raja
Harrish Ebi Francis Peter Joshua

857-370-7645
857-268-9141

arumugamraja.n@northeastern.edu
peterjoshua.h@northeastern.edu

Percentage of Effort Contributed by Student1: 50

Percentage of Effort Contributed by Student2: 50

Signature of Student 1:

Signature of Student 2:

Submission Date: 12/07/2024

Comprehensive Database Management System for Mining and Stone Crushing Operations

The Mining and stone crushing industry is a big operation that houses multiple resources. It has machinery to be monitored, vehicles for transportation, inventory management, skilled and unskilled employees, purchase and sales. All these aspects are intertwined and heavily dependent on one another for smooth operation. To overlook all of these through traditional bookkeeping methods is near to impossible as it takes a lot of human effort and something important can be easily overlooked. This is where database management system comes in. By using a database, you can centralize all data / information together making it easier to manage all these resources in a organized and efficient manner. This project would showcase how the implementation of database would greatly reduce the production costs and improve efficiency, this can create a more efficient workflow and help the business thrive in a competitive environment.

Theory of Database usage in Mining and Stone Crushing Operations:

The primary objective of this project is to develop a robust database management system that integrates multiple assets of mining operations machinery management, vehicle tracking, employee records, inventory control, customer management, sales, and purchases. The Machinery Management module will maintain detailed records of different machinery, capturing essential information such as type, condition, erection date, total operating hours, and the employee responsible for operation. This system will facilitate regular maintenance scheduling and tracking to ensure optimal performance.

In Vehicle Management, we will track vital vehicle details, including vehicle ID, type, registration numbers, working conditions, total hours used, operator ID, purchase dates, and maintenance schedules. The system will also provide alerts for upcoming maintenance and inspections, ensuring that vehicles remain in peak condition.

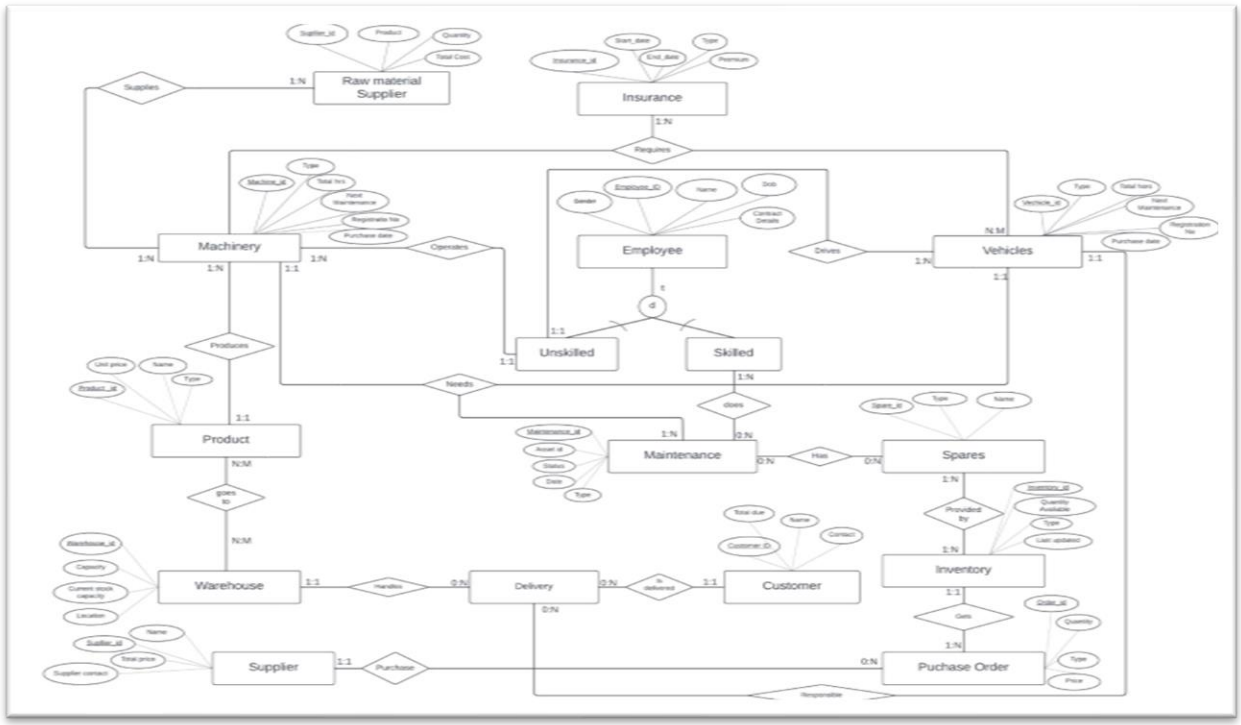
The Employee Management module will maintain comprehensive records for all employees, including their employee ID, salary, name, gender, date of birth, phone number, and address. This will provide easy access to personnel and payroll records, streamlining HR operations.

Our Inventory Management system will meticulously track product inventory, product ID, detailing product types, quantities, IDs, conditions, and the latest purchase dates. This module will also include stock level alerts to prevent shortages, ensuring that products are always available when needed. The Customer Management module will maintain a robust database that includes customer id, customer types, names, IDs, and total dues, enhancing customer relationship management and billing processes.

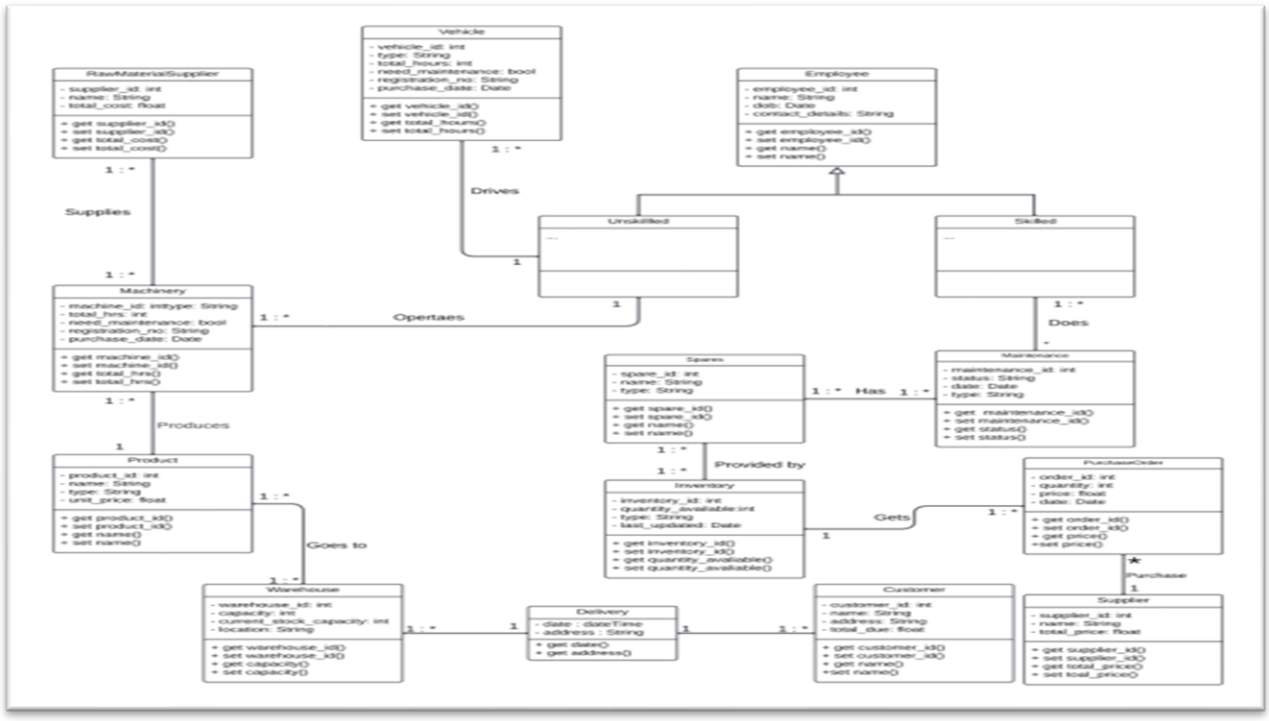
In Purchase Management, we will maintain detailed purchase records that include purchase ID, product types, prices, product ID, and quantities. This module will integrate seamlessly with inventory management for real-time stock updates.

Lastly, the Production Details module will maintain records of daily production, capturing operation hours, raw material details, daily income, and daily expenses. This comprehensive data will support strategic planning and operational efficiency.

ENHANCED ENTITY RELATIONSHIP:



UNIFIED MODELING LANGUAGE:



RELATIONAL MODEL:

RawMaterialSupplier (SupplierID, Product, Quantity, TotalCost)
Here, SupplierID is the primary key.

Insurance (InsuranceID, StartDate, EndDate, Type, Premium)
Here, InsuranceID is the primary key.

Employee (EmployeeID, Gender, Name, DateOfBirth, ContactDetails, SkillType)
Here, EmployeeID is the primary key.

UnskilledEmployee (EmployeeID)
Here, EmployeeID is a foreign key to Employee, and NULL is NOT allowed.

SkilledEmployee (EmployeeID, Skill)
Here, EmployeeID is a foreign key to Employee, and NULL is NOT allowed.

Machinery (MachineryID, Type, LastMaintenance, RegistrationNo, PurchaseDate)
Here, MachineryID is the primary key.Each machinery is operated by one or more employees.

Vehicles (VehicleID, Type, LastMaintenance, RegistrationNo, PurchaseDate)
Here, VehicleID is the primary key.Each vehicle is driven by one or more employees.

Product (ProductID, UnitPrice, Name, Type)
Here, ProductID is the primary key.

Warehouse (WarehouseID, Capacity, CurrentStock, Location)
Here, WarehouseID is the primary key.

Supplier (SupplierID, Name, Location, TotalCost)
Here, SupplierID is the primary key. Stores general supplier details.

PurchaseOrder (OrderID, SupplierID, Date, Quantity, UnitCost, TotalCost)
Here, OrderID is the primary key.

SupplierID is a foreign key to Supplier, and NULL is NOT allowed.
Customer (CustomerID, Name, Address, ContactNo)Here, CustomerID is the primary key.

Inventory (InventoryID, QuantityAvailable, LastUpdated, WarehouseID, ProductID)
Here, InventoryID is the primary key.
WarehouseID is a foreign key to Warehouse, and NULL is NOT allowed.
ProductID is a foreign key to Product, and NULL is NOT allowed.

Maintenance (MaintenanceID, AssetID, Date, Status, Type)
Here, MaintenanceID is the primary key. Each maintenance record is associated with an asset, such as machinery or vehicles.

Spares (SpareID, Type, Name, Stock, Cost)
Here, SpareID is the primary key.

Delivery (DeliveryID, OrderID, CustomerID, Date, Address)
Here, DeliveryID is the primary key. OrderID is a foreign key to PurchaseOrder, and NULL is NOT allowed. CustomerID is a foreign key to Customer, and NULL is NOT allowed.

MachineryID is a foreign key to Machinery, and NULL is NOT allowed.
Primary Key: (EmployeeID, MachineryID) to represent the many-to-many relationship between Employee and Machinery.

VehicleID is a foreign key to Vehicles, and NULL is NOT allowed.
Primary Key: (EmployeeID, VehicleID) to represent the many-to-many relationship between Employee and Vehicles.

Needs (ProductID, WarehouseID)
Here, ProductID is a foreign key to Product, and NULL is NOT allowed.

WarehouseID is a foreign key to Warehouse, and NULL is NOT allowed.
Provides (SpareID, InventoryID)
Here, SpareID is a foreign key to Spares, and NULL is NOT allowed.
InventoryID is a foreign key to Inventory, and NULL is NOT allowed.

QUERY IN SQL:

1.Simple query: Using select to get productId and unit price from product

Select ProductID, UnitPrice from product;

ProductID	UnitPrice
101-3mm	250.00
201-120mm	380.00
301-40mm	570.00
302-40mm	550.00
401-70mm	120.00
501-100mm	180.00
503-100mm	200.00
601-Powder	510.00
701-Msand	880.00
708-Msand	790.00

2.Aggregate query: Average cost of each product from all suppliers

Select Product, round(avg(Cost)) as Average_Cost from RawMaterialSupplier group by Product;

Product	Average_Cost
Sand	43
Boulders	42
Aggreate	46
Rock blocks	54

3.Joins: To find sum of type dynamic, fixed and moving in leased and rental inventories where count is greater than 50

```
Select  s.Type as PartType, i.Type as InventoryType, sum(s.Stock) as Stock
from Spares as s, Inventory as i
where s.inventoryid = i.inventoryid
and s.Stock >50 and i.Type in ('Rental', 'Leased')
group by s.Type, i.type
order by s.Type;
```

PartType	InventoryType	Stock
Dynamic	Leased	130
Dynamic	Rental	94
Fixed	Leased	379
Fixed	Rental	425
Moving	Leased	150
Moving	Rental	456

4.Nested query: This query finds the OrderID, OrddeDate, and Quantity for fixed spare part, which stock > 50 and in owned inventory'.

```
select po.OrderID, po.OrddeDate, po.Quantity from PurchaseOrder as po
where po.SpareID in (
   select s.SpareID from Spares as s
   where s.Type = 'Fixed'
    and s.Stock > 50
    and s.InventoryId in (select i.InventoryId from Inventory as i where i.Type = 'Owned')) order by
po.OrderID;
```

OrderID	OrddeDate	Quantity
315	2024-11-06	29
520	2024-08-02	47
597	2024-02-24	19
913	2024-05-30	28
NULL	NULL	NULL

5.Correlated query: To find each purchase order where quantity is greater than avg quantity of all purchase in the same inventory

```
Select * from PurchaseOrder as po
where po.Quantity >
                (select avg(po_inner.Quantity) from PurchaseOrder as po_inner where
po_inner.OrderInventoryId = po.OrderInventoryId);
```

OrderID	OrddeDate	OrderInventoryId	SpareID	Quantity
108	2024-03-29	37 mv	6268	40
112	2023-12-11	37 mv	2864	41
130	2024-03-24	15 io	8842	45
140	2024-01-26	18 dl	8861	31
164	2024-09-14	37 cm	2766	29
183	2024-05-01	39 wi	3066	47
223	2023-12-28	90 gi	2766	23
315	2024-11-06	99 oh	8803	29
395	2023-12-14	72 vh	2440	34

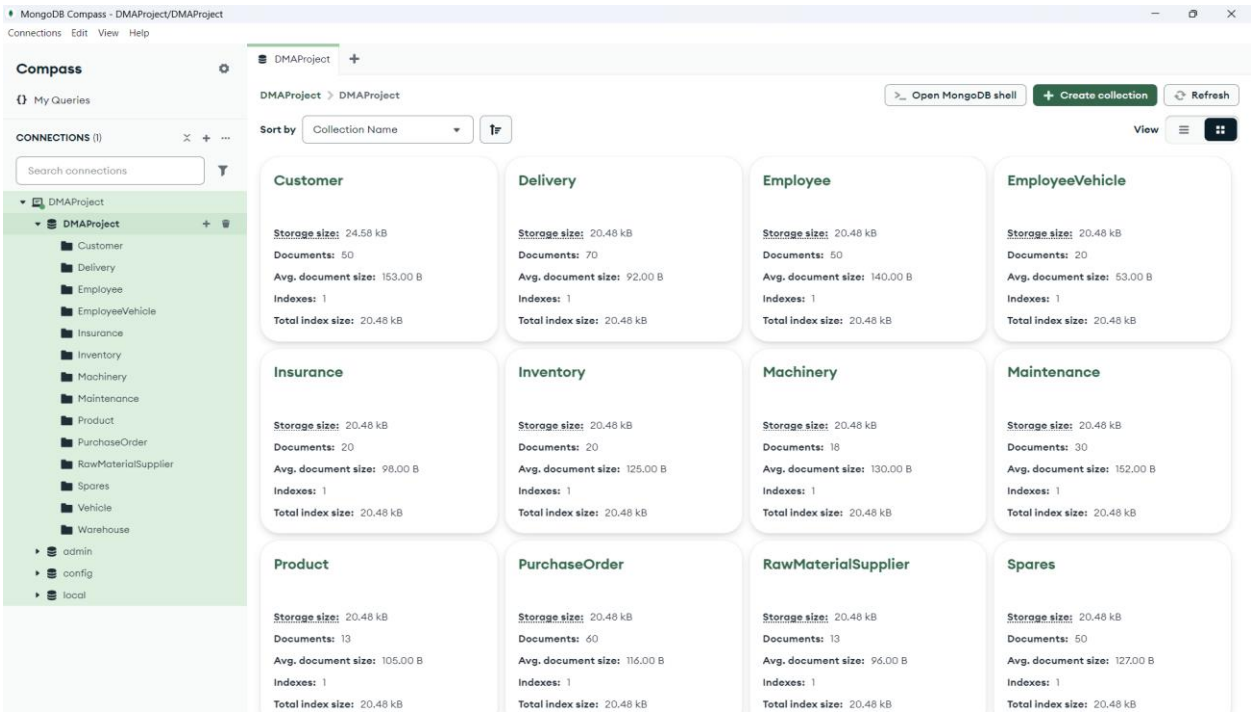
6.Subqueries in select: This query will return each employee's EmployeeID, Name, and the count of vehicles assigned to them.

```
Select e.EmployeeID,  e.Name, e.SkillType,
   (select count(*) from EmployeeVehicle ev where ev.EmployeeId = e.EmployeeID) as VehicleCount
From Employee e;
```

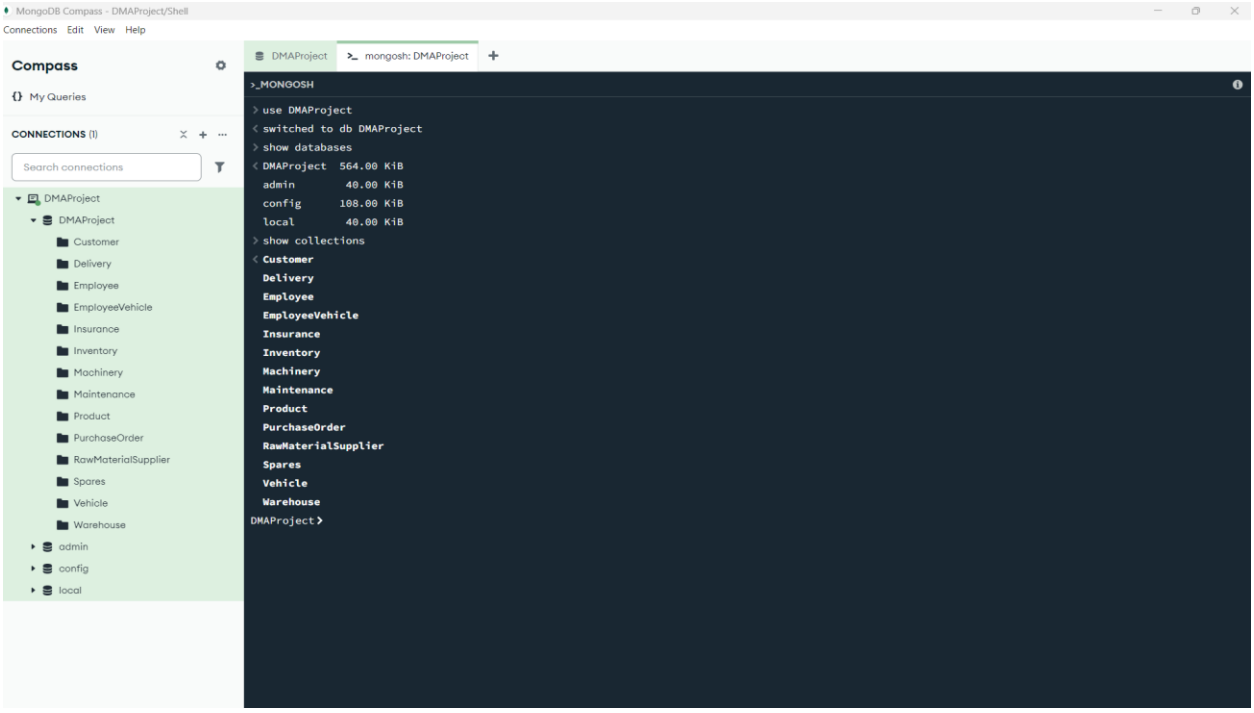
EmployeeID	Name	SkillType	VehicleCount
1019	Monserrate	Unskilled	1
1208	Orin	Unskilled	1
1615	Tyra	Unskilled	1
1816	Helena	Unskilled	1
2228	Nash	Skilled	1
2350	Daisha	Skilled	1
2858	Tyler	Unskilled	1
3484	Retha	Skilled	1
3490	Anjali	Skilled	1
3579	Ashtyn	Unskilled	1
3895	Santino	Skilled	1
3926	Garry	Unskilled	1

NO SQL (MongoDB):

DataBase Implementation in MongoDB:



Collections in DB:



NoSQL Implementation

Connection to MongoDB and Jupyter Notebook

In [1]:
```
from pymongo import MongoClient
# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["DMAProject"]# Replace with your MongoDB URI
```

In [2]:
```
# Create or access a collection
collection = db['DMAProject']
# List all collections in the database
print(db.list_collection_names())
```

```
['Product', 'Customer', 'Vehicle', 'RawMaterialSupplier', 'Employee', 'Inventory', 'Maintenance', 'Warehouse', '
Machinery', 'Insurance', 'EmployeeVehicle', 'PurchaseOrder', 'Delivery', 'Spares']
```

To find Inventory where type is 'Rental'

In [3]:
```
result1 = db.Inventory.find({ "Type": "Owned" });
for doc in result1:
    print(doc)
```

```
{'_id': ObjectId('67464e549702a9cd8601307f'), 'InventoryId': '29 eq', 'Address': '7047 Nolan Village', 'Type': '
Owned', 'LastUpdated': '2020-04-16'}
{'_id': ObjectId('67464e549702a9cd86013082'), 'InventoryId': '38 wo', 'Address': '1491 Bobbie Island Apt. 724',
'Type': 'Owned', 'LastUpdated': '2023-10-19'}
{'_id': ObjectId('67464e549702a9cd86013084'), 'InventoryId': '41 ft', 'Address': '67396 Nettie Knolls Apt. 701',
'Type': 'Owned', 'LastUpdated': '2021-06-01'}
{'_id': ObjectId('67464e549702a9cd8601308b'), 'InventoryId': '92 ja', 'Address': '1695 Lebsack Island', 'Type':
'Owned', 'LastUpdated': '2024-03-29'}
{'_id': ObjectId('67464e549702a9cd8601308e'), 'InventoryId': '99 oh', 'Address': '4880 Magnolia Flat', 'Type': '
Owned', 'LastUpdated': '2022-06-05'}
```

To find Warehouse which has 'sand' in WarehouseId

In [4]:
```
result2 = db.Warehouse.find({"Warehouse_productID": {"$regex": "sand", "$options": "i"}})
for doc in result2:
    print(doc)
```

```
{'_id': ObjectId('67464e549702a9cd8601306f'), 'WarehouseID': '81 ik', 'Capacity': 45541, 'CurrentStock': 9309, '
Location': '8998 Destin Dale\nHailieville, IL 15791', 'Warehouse_productID': '701-Msand'}
{'_id': ObjectId('67464e549702a9cd86013070'), 'WarehouseID': '47 iz', 'Capacity': 33937, 'CurrentStock': 9892, '
Location': '57100 Ervin Pike\nPort Mollyville, MI 81246', 'Warehouse_productID': '708-Msand'}
{'_id': ObjectId('67464e549702a9cd86013071'), 'WarehouseID': '29 cg', 'Capacity': 71039, 'CurrentStock': 7523, '
Location': '489 Bernhard Village\nAdalinefurt, VT 59571', 'Warehouse_productID': '801-Psand'}
```

To find Machinery bought on or after Feb 2024

In [5]:
```
result3 = db.Machinery.find({"PurchaseDate" : {"$gt": "2024-03-01"} })
for doc in result3:
    print(doc)
```

```
{'_id': ObjectId('67464e549702a9cd86013194'), 'MachineryID': 41, 'Type': 'Impact crushers', 'PurchaseDate': '202
4-09-11', 'InsuranceID': 67, 'Employee_Incharge': 9223}
{'_id': ObjectId('67464e549702a9cd86013195'), 'MachineryID': 42, 'Type': 'Classifier', 'PurchaseDate': '2024-08-
09', 'InsuranceID': 74, 'Employee_Incharge': 6291}
```

To find count of each type part in spares collection

In [6]:
```
result4 = db.Spares.aggregate([{"$group": {"_id": "$Type", "count": {"$sum": 1}}}])
for doc in result4:
    print(doc)
```

```
{'_id': 'Dynamic', 'count': 8}
{'_id': 'Fixed', 'count': 24}
{'_id': 'Moving', 'count': 18}
```

To find sum of all outstanding from customers

In [7]:
```
result5 = db.Customer.aggregate([{ "$group": { "_id": "null", "TotalDue": { "$sum": "$TotalDue" }}}])
for doc in result5:
    print(doc)
```

```
{'_id': 'null', 'TotalDue': 2484509}
```

PYTHON DB CONNECTION:

import pymysql
import pandas as pd
from sqlalchemy import create_engine
import matplotlib.pyplot as plt

# Database configuration
host = '127.0.0.1'
user = 'root'
password = 'Nithinraja01!'
database = 'DMAProject'

# Create SQLAlchemy engine
connection = create_engine(f"mysql+pymysql://{user}:{password}@{host}/{database}")

# Test connection
try:
  with connection.connect() as conn:
    print("Connection to the database was successful!")
except Exception as e:
  print("Failed to connect to the database.")
  print("Error:", str(e))
  exit()

```python
import pymysql
import pandas as pd
from sqlalchemy import create_engine
import matplotlib.pyplot as plt

# Database configuration
host = '127.0.0.1'
user = 'root'
password = 'Nithinraja01!'
database = 'DMAProject'

# Create SQLAlchemy engine
connection = create_engine(f"mysql+pymysql://{user}:{password}@{host}/{database}")

# Test connection
try:
    with connection.connect() as conn:
        print("Connection to the database was successful! \nSuccessfully Connected to DMAProject")
except Exception as e:
    print("Failed to connect to the database.")
    print("Error:", str(e))
    exit()
```

```
Connection to the database was successful!
Successfully Connected to DMAProject
```

Query1: Basic Query

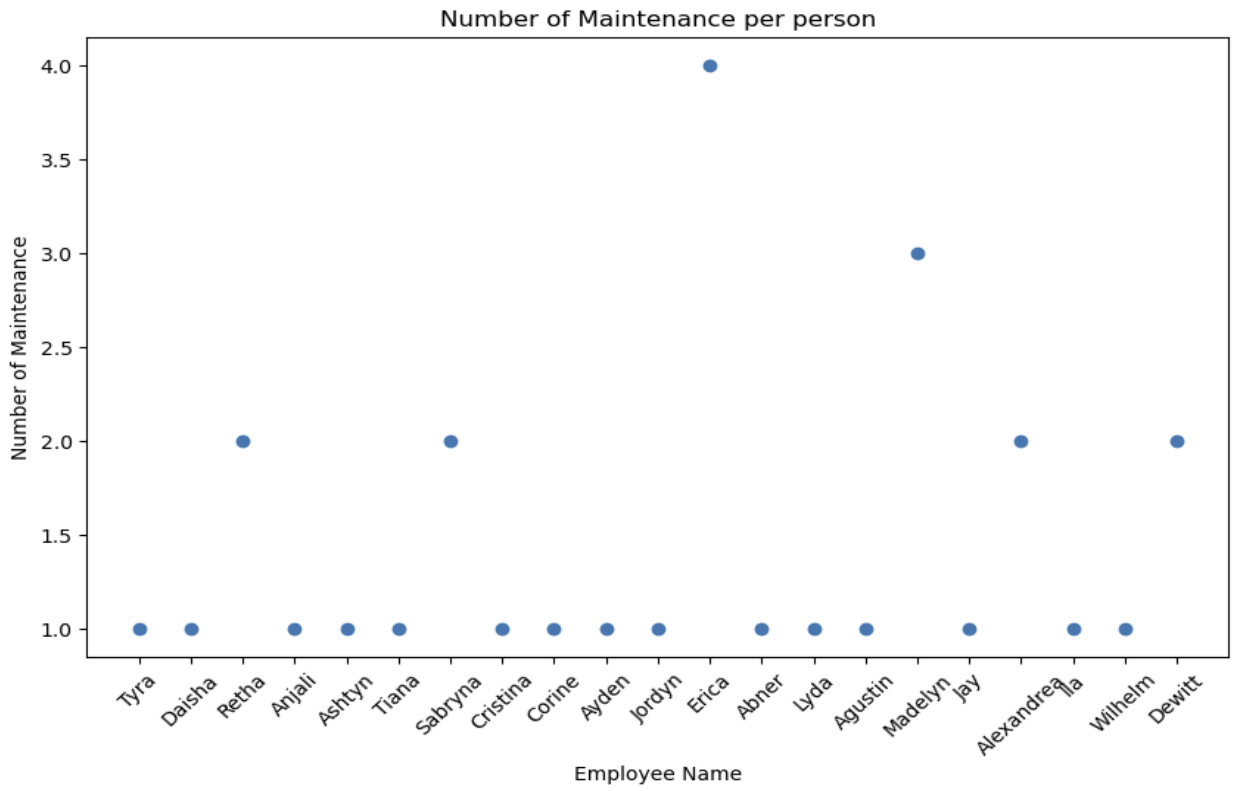
```
query1 = "select * from employee"
results_df1 = pd.read_sql(query1, con=connection)
print(results_df1.head())
```

```
#Query1

query1 = "select * from employee"
results_df1 = pd.read_sql(query1, con=connection)
print(results_df1.head())
```

	EmployeeID	Name	Gender	DOB	PhoneNo	SkillType
0	1019	Monserrate	Male	2005-01-07	1035051233	Unskilled
1	1208	Orin	Female	1998-05-14	3659173895	Unskilled
2	1615	Tyra	Other	2019-10-26	5781517476	Unskilled
3	1816	Helena	Male	2014-12-07	4757839668	Unskilled
4	2228	Nash	Male	1996-05-14	1775598246	Skilled

Query2: To find no of Maintenance assigned to each employee

```
query2 = """select Name, count(e.EmployeeID) as NumberOfMaintenance
from employee e , maintenance m
where e.EmployeeID = m.Employee_Incharge
group by e.EmployeeID ;"""
results_df2 = pd.read_sql(query2, con=connection)
print(results_df2)

# Scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(results_df2['Name'], results_df2['NumberOfMaintenance'])
plt.xlabel('Employee Name')
plt.ylabel('Number of Maintenance')
plt.title('Number of Maintenance per person')
plt.xticks(rotation=45)
plt.show()
```



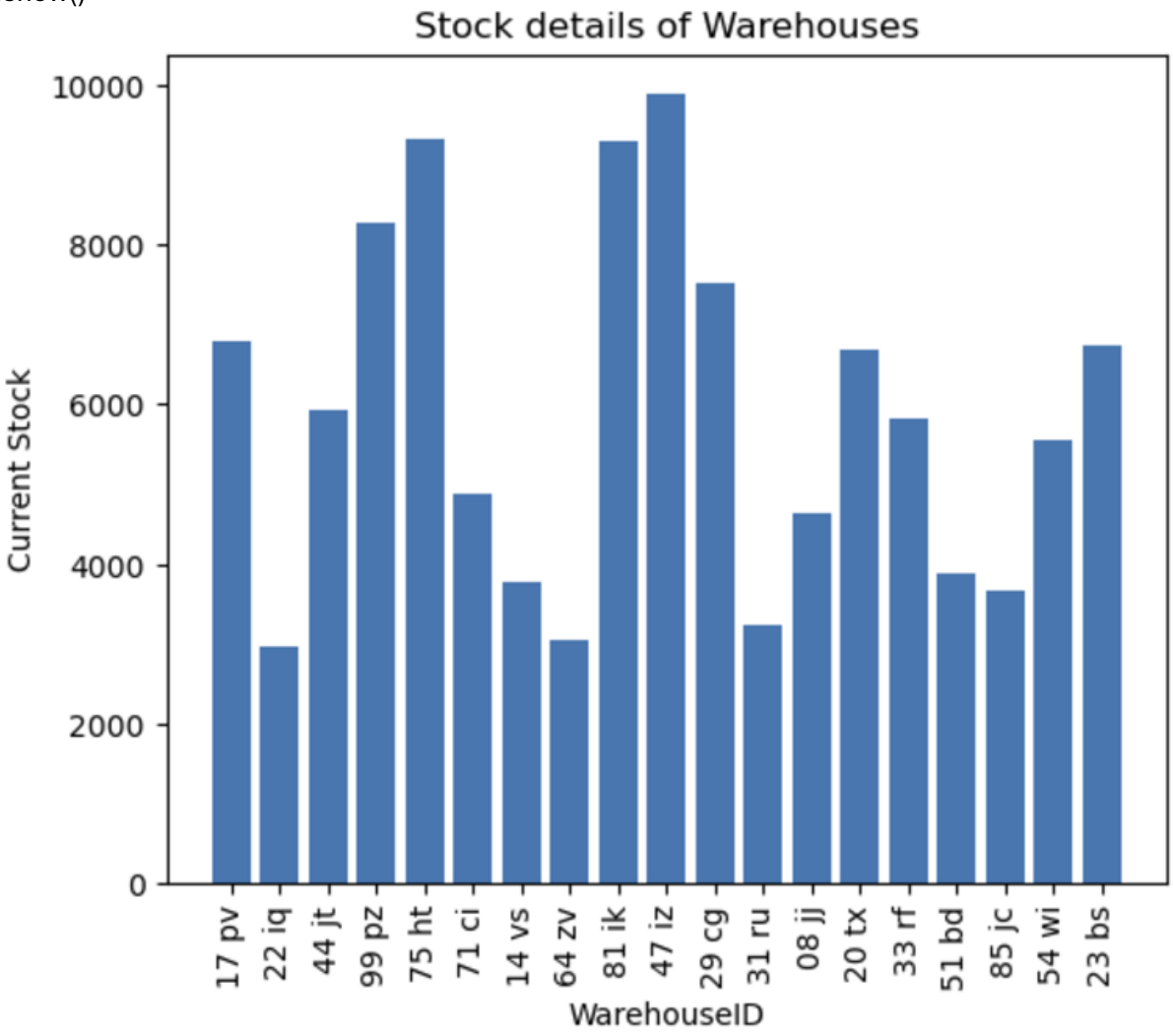
Query3: To find fixed part spares in inventory type owned where stock is greater than 50

```
query3 = """select s.SpareID, s.Stock from Spares as s
   where s.Type = 'Fixed'
     and s.Stock > 50
     and s.InventoryId in (select i.InventoryId from Inventory as i where i.Type = 'Owned');"""
results_df3 = pd.read_sql(query3, con=connection)
results_df3
```

	SpareID	Stock
0	8803	55
1	9964	90
2	4209	88
3	7158	87

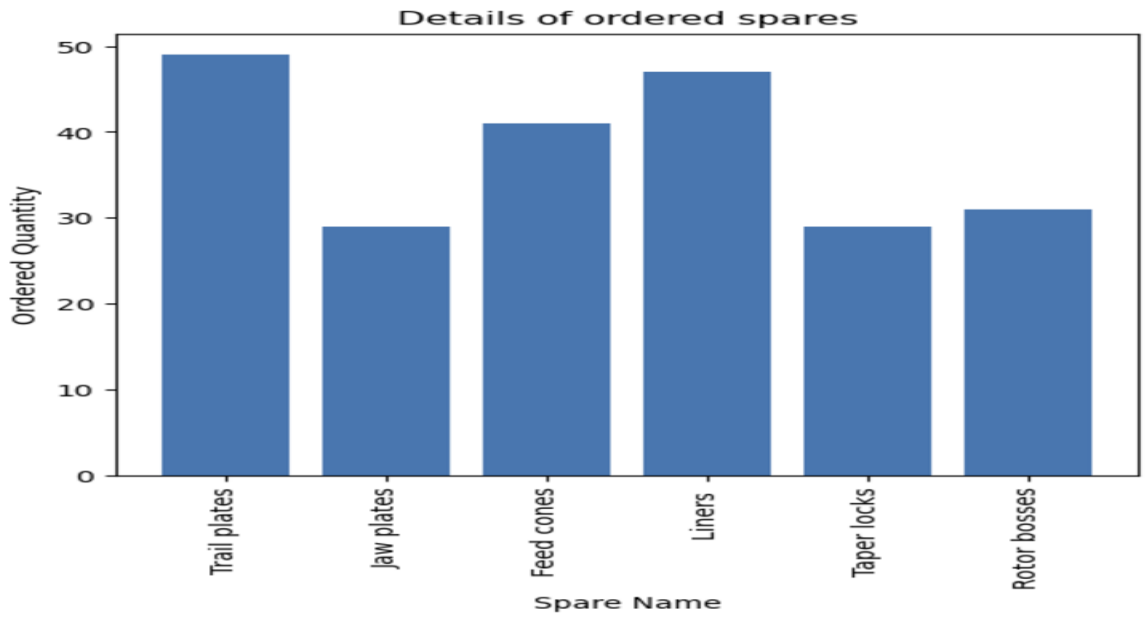
Query4: To find the Current stock level of warehouses

```
query4 = '''Select WarehouseID, Capacity, CurrentStock, Location, Warehouse_productID from
Warehouse
where CurrentStock > any (select CurrentStock from Warehouse where Capacity > 20000);'''
results_df4 = pd.read_sql(query4, con=connection)
print(results_df4)

plt.figure(figsize=(6, 5))
plt.bar(results_df4['WarehouseID'], results_df4['CurrentStock'])
plt.xlabel('WarehouseID')
plt.ylabel('Current Stock')
plt.title('Stock details of Warehouses')
plt.xticks(rotation=90)
plt.show()
```



Query5: To find the purchase orders of spares for the spares where their quantity is less the 20

```
query5 = '''select s.sparename, po.quantity from purchaseorder po, spares s
where po.spareid = s.spareid and s.stock < 20;'''
results_df5 = pd.read_sql(query5, con=connection)
print(results_df5)

plt.figure(figsize=(6, 5))
plt.bar(results_df5['sparename'], results_df5['quantity'])
plt.xlabel('WarehouseID')
plt.ylabel('Current Stock')
plt.title('Stock details of Warehouses')
plt.xticks(rotation=90)
plt.show()
```



CONCLUSION:

The future of the mining and stone crushing industry lies in the adoption of a robust database management system that integrates all aspects of the operation. By centralizing data related to machinery, vehicles, inventory, employees, and sales, businesses can significantly reduce production costs and enhance efficiency. The use of technologies like relational databases, data warehouses, and real-time processing tools allows for streamlined operations and better decision-making. Furthermore, leveraging advanced data analytics and predictive modeling can optimize resource management, improve workflow, and ultimately provide a competitive edge in the marketplace. This shift to a data-driven approach marks a critical step toward operational excellence and sustainable growth in the industry.