Neo4j Work: Advanced-Level Cypher Queries in Neo4j

Prepared by: Mert OLÇAMAN

Week: 5-6

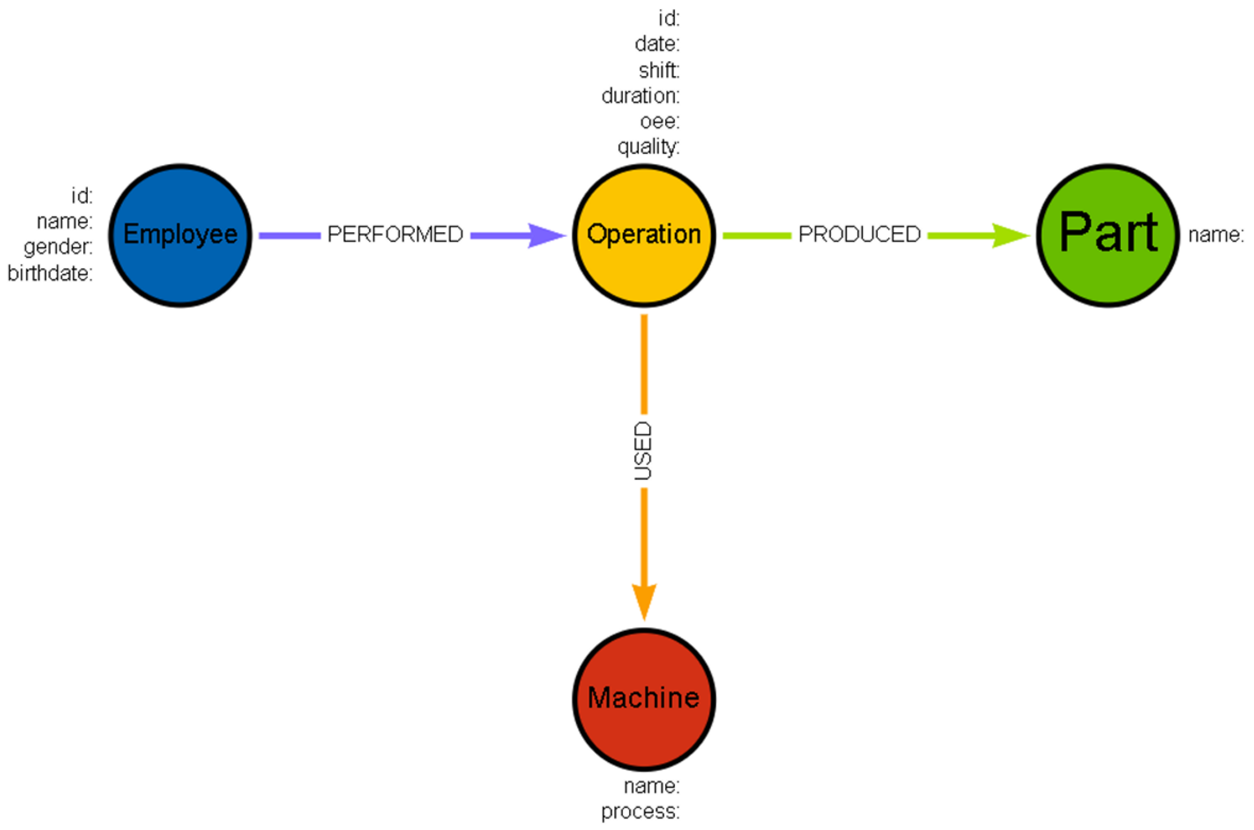
patika.dev

NewMind AI Bootcamp

INTRODUCTION

In this work, advanced-level cypher queries were performed to get detailed insights from the dataset. The dataset and the graph database design can be seen below respectively.

EmployeeID	Name	Gender	DateOfBirth	OperationID	Part	Process	Shift	Machine	Date	Duration	OEE	Quality
Employee1	Danielle Johnson	M	17/01/1983	Operation1	Piston	Milling	Day	M2	26/04/2025	64	0.68	Medium
Employee1	Danielle Johnson	M	17/01/1983	Operation19	Piston	Assembly	Morning	M3	30/04/2025	38	0.73	Medium
Employee1	Danielle Johnson	M	17/01/1983	Operation25	Gearbox	Drilling	Morning	M1	25/04/2025	31	0.61	Medium
Employee1	Danielle Johnson	M	17/01/1983	Operation32	Shaft	Drilling	Morning	M1	29/04/2025	85	0.77	Medium
Employee1	Danielle Johnson	M	17/01/1983	Operation41	Bearing	Painting	Night	M4	25/04/2025	179	0.63	Low
Employee1	Danielle Johnson	M	17/01/1983	Operation60	Valve	Milling	Day	M2	25/04/2025	31	0.83	Medium
Employee1	Danielle Johnson	M	17/01/1983	Operation78	Piston	Painting	Morning	M4	29/04/2025	171	0.7	Medium
Employee1	Danielle Johnson	M	17/01/1983	Operation81	Bearing	Painting	Night	M4	28/04/2025	126	0.63	Medium
Employee1	Danielle Johnson	M	17/01/1983	Operation106	Valve	Milling	Morning	M2	29/04/2025	176	0.79	High
Employee1	Danielle Johnson	M	17/01/1983	Operation169	Shaft	Assembly	Morning	M3	30/04/2025	106	0.88	Low
Employee1	Danielle Johnson	M	17/01/1983	Operation170	Gearbox	Milling	Day	M2	24/04/2025	146	0.68	Medium
Employee1	Danielle Johnson	M	17/01/1983	Operation171	Valve	Milling	Morning	M2	25/04/2025	40	0.75	Low
Employee1	Danielle Johnson	M	17/01/1983	Operation234	Shaft	Milling	Day	M2	30/04/2025	89	0.82	Low
Employee2	John Taylor	M	22/04/1971	Operation3	Bearing	Painting	Morning	M4	26/04/2025	176	0.73	High
Employee2	John Taylor	M	22/04/1971	Operation33	Piston	Milling	Morning	M2	28/04/2025	178	0.69	Medium
Employee2	John Taylor	M	22/04/1971	Operation36	Bearing	Painting	Night	M4	25/04/2025	45	0.94	Low
Employee2	John Taylor	M	22/04/1971	Operation77	Piston	Milling	Day	M2	27/04/2025	165	0.8	High
Employee2	John Taylor	M	22/04/1971	Operation115	Valve	Milling	Night	M2	30/04/2025	65	0.9	Low
Employee2	John Taylor	M	22/04/1971	Operation125	Gearbox	Painting	Day	M4	24/04/2025	60	0.64	Medium



a) Importing data into Neo4j

At first, the generated csv file was uploaded to github directory. Secondly, it was directly imported by using function.

Step1 - Adding Constraints: Before starting everything, constraints should be specified. Otherwise, there might be some improper nodes, which causes wrong results.

```
// --- ADDING CONSTRAINTS --- \\

// Employee constraint
CREATE CONSTRAINT employee_id IF NOT EXISTS
FOR (e:Employee)
REQUIRE e.employee_id IS UNIQUE;

// Operation constraint
CREATE CONSTRAINT operation_id IF NOT EXISTS
FOR (o:Operation)
REQUIRE o.operation_id IS UNIQUE;

// Part constraint
CREATE CONSTRAINT part_name IF NOT EXISTS
FOR (p:Part)
REQUIRE p.part_name IS UNIQUE;

// Machine constraint
CREATE CONSTRAINT machine_name IF NOT EXISTS
FOR (m:Machine)
REQUIRE m.machine_name IS UNIQUE;
```

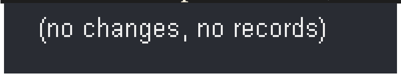
Step2 – Loading CSV Files & Creating Nodes with Properties & Adding Relationships: While loading data from the link, an alias should be specified to use in the other parts of the cypher query. It was specifed as **data** in this work.
For each node created, the unique property has to be specified in the merge line as a property. As for Employee node, only employee_id was set as unique property, so it was the only property which was provided in the merge, as it can be seen in the code cell below. If there had been multiple unique properties, they would have been specifed as well. Additionally, data alias was used to call the relevant data from the imported dataset by using the same column name as the imported dataset had. What it is meant here that EmployeeID was the name of the column in the dataset. It is valid for the other variables which start with the alias of data such as data.Name, data.Gender, and data.Shift.
On the other hand, the numerical values like duration must be converted into float by toFloat() function while calling the data from the csv file. The column of date was converted to date, as the format was only data (time not included).

```
// --- LOADING CSV FILE --- \\
LOAD CSV WITH HEADERS FROM
'https://media.githubusercontent.com/media/mertolcaman/patika_newmind_ai_bootcamp/refs/heads/main/4.week/mydata.csv' as data

// --- CREATING NODES WITH PROPERTIES--- \\

// Creating Employee Node
MERGE (e:Employee {employee_id: data.EmployeeID})
SET e.name = data.Name,
e.gender = data.Gender,
e.birthdate = data.DateOfBirth

// Creating Operation Node
MERGE (o:Operation {operation_id: data.OperationID})
SET o.date = date(data.Date),
o.shift = data.Shift,
o.duration = toFloat(data.Duration),
o.oee = toFloat(data.OEE),
o.quality = data.Quality

// Creating Machine Node
MERGE (m:Machine {machine_name: data.Machine})
SET m.process = data.Process

// Creating Part Node
MERGE (p:Part {part_name: data.Part})

// --- CREATING RELATIONSHIPS --- \\

// Creating PERFORMED Relationship: Employee -> Operation
MERGE (e)-[:PERFORMED]->(o)

// Creating USED Relationship: Operation -> Machine
MERGE (o)-[:USED]->(m)

// Creating PRODUCED Relationship: Operation -> Person
MERGE (o)-[:PRODUCED]->(p);
```

TEST

After loading a dataset, it should be checked if it has any errors. According to the query, there is no record found, which is related to not linked nodes.

```
MATCH (o:Operation)
WHERE NOT (o)<-[:PERFORMED]-(:Employee)
   OR NOT (o)-[:USED]->(:Machine)
   OR NOT (o)-[:PRODUCED]->(:Part)
RETURN o.operation_id;
```



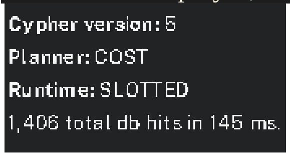
```
(no changes, no records)
```

The test query was run to see if there was any deficiency in the graph database model. However, no record was found, which is a good sign.

a) Query performance comparison

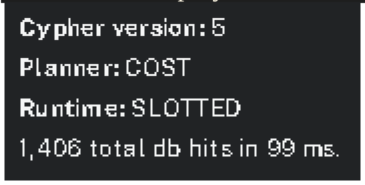
Query comparison-1:

```
//creating a detailed table for the best employees

PROFILE CALL {
  MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
  WITH e.name AS employee, COUNT(CASE WHEN o.quality = "High" THEN 1 END) AS high
  ORDER BY high DESC
  LIMIT 5
  RETURN collect(employee) AS top_names
}
UNWIND top_names AS employee_name
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
WHERE e.name=employee_name
RETURN
  employee_name AS employee,
  o.shift AS shift,
  COUNT(*) AS op_count,
  COUNT(CASE WHEN o.quality = "High" THEN 1 END) AS high_quality,
  COUNT(CASE WHEN o.quality = "Low" THEN 1 END) AS low_quality,
  ROUND(AVG(o.oee), 2) AS avg_oee
ORDER BY employee, shift
```



```
Cypher version: 5
Planner: COST
Runtime: SLOTTED
1,406 total db hits in 145 ms.
```

```
//creating a detailed table for the best employees
PROFILE  CALL {
  MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
  WITH e.name AS employee, COUNT(CASE WHEN o.quality = "High" THEN 1 END) AS high
  ORDER BY high DESC
  LIMIT 5
  RETURN collect(employee) AS top_names
}
UNWIND top_names AS employee_name
MATCH (e:Employee {name: employee_name})-[:PERFORMED]->(o:Operation)
RETURN
  employee_name AS employee,
  o.shift AS shift,
  COUNT(*) AS op_count,
  COUNT(CASE WHEN o.quality = "High" THEN 1 END) AS high_quality,
  COUNT(CASE WHEN o.quality = "Low" THEN 1 END) AS low_quality,
  ROUND(AVG(o.oee), 2) AS avg_oee
ORDER BY employee, shift
```

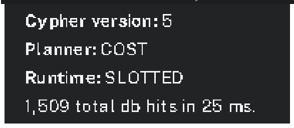


```
Cypher version: 5
Planner: COST
Runtime: SLOTTED
1,406 total db hits in 99 ms.
```

In the first query, employee name is filtered in WHERE part, which makes the query slower.

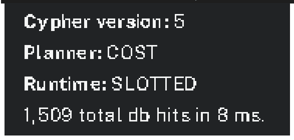
Query comparison-2:

```
PROFILE
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
MATCH (o)-[:USED]->(m:Machine)
RETURN e.name, COUNT(*) AS op_count
```



```
Cypher version: 5
Planner: COST
Runtime: SLOTTED
1,509 total db hits in 25 ms.
```

```
PROFILE
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)-[:USED]->(m:Machine)
RETURN e.name, COUNT(*) AS op_count
```

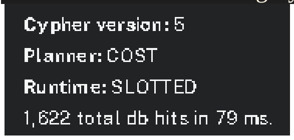


```
Cypher version: 5
Planner: COST
Runtime: SLOTTED
1,509 total db hits in 8 ms.
```

In the first query 2 different MATCH was used. Indicating all query in 1 MATCH makes the query faster.

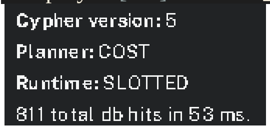
Query comparison-3:

```
PROFILE MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
WITH e.name AS employee, AVG(o.oee) AS avg_oee
ORDER BY avg_oee DESC
LIMIT 3
RETURN 'Best' AS category, employee, ROUND(avg_oee, 2) AS avg_oee
UNION
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
WITH e.name AS employee, AVG(o.oee) AS avg_oee
ORDER BY avg_oee ASC
LIMIT 3
RETURN 'Worst' AS category, employee, ROUND(avg_oee, 2) AS avg_oee
```



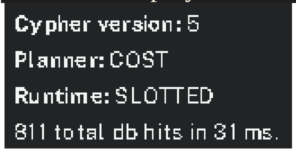
```
Cypher version: 5
Planner: COST
Runtime: SLOTTED
1,622 total db hits in 79 ms.
```

```
PROFILE
CALL {
  MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
  WITH e.name AS employee, AVG(o.oee) AS avg_oee
  ORDER BY avg_oee DESC
  RETURN collect({
    employee: employee,
    avg_oee: ROUND(avg_oee, 2)
  }) AS employees
}
WITH employees
RETURN
  employees[..3] AS best_employees,   // Top 3
  employees[-3..] AS worst_employees  // Bottom 3
```



```
Cypher version: 5
Planner: COST
Runtime: SLOTTED
811 total db hits in 53 ms.
```

```
:param limit=>3;
PROFILE call{
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
WITH e.name AS employee, AVG(o.oee) AS avg_oee
ORDER BY avg_oee DESC
RETURN collect({
  employee: employee,
  avg_oee: ROUND(avg_oee, 2)
}) AS employees
}
RETURN employees[..$limit] as best_employees, employees[-$limit..] as worst_employees
```

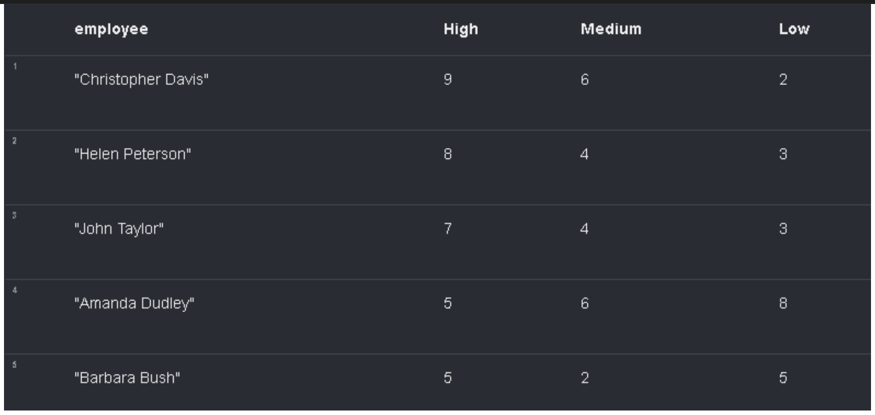


```
Cypher version: 5
Planner: COST
Runtime: SLOTTED
811 total db hits in 31 ms.
```

In the first query, two separate MATCH and aggregation operations are performed for best and worst results, which leads to redundant graph traversals and slower execution. The second and third queries improve performance by using a single MATCH with collect() and slicing, avoiding duplicate computations. The third query also adds flexibility by parameterizing the limit without affecting speed.

b) Advanced-level cypher queries

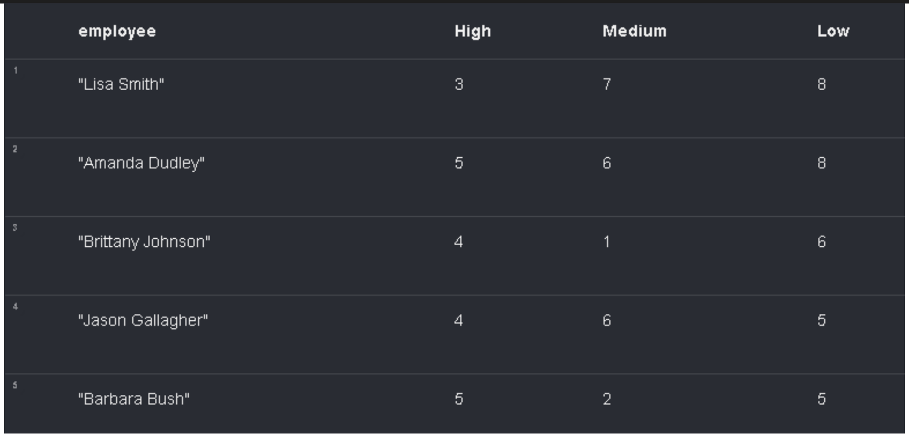
```
//Top 5 employees perform with high quality
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
RETURN e.name AS employee,
       COUNT(CASE WHEN o.quality = "High" THEN 1 END) AS High,
       COUNT(CASE WHEN o.quality = "Medium" THEN 1 END) AS Medium,
       COUNT(CASE WHEN o.quality = "Low" THEN 1 END) AS Low
ORDER BY High DESC
LIMIT 5
```



employee	High	Medium	Low
"Christopher Davis"	9	6	2
"Helen Peterson"	8	4	3
"John Taylor"	7	4	3
"Amanda Dudley"	5	6	8
"Barbara Bush"	5	2	5

Christopher Davis shows the best performance with 9 high- and only 2 low-quality operations. In contrast, Amanda Dudley has 8 low-quality outputs, indicating inconsistency, while Barbara Bush shows a balanced but less remarkable result.

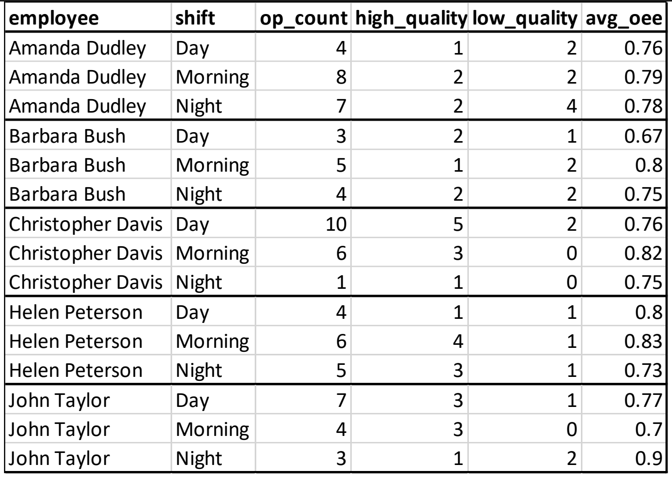
```
//Top 5 employees perform with low quality
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
RETURN e.name AS employee,
       COUNT(CASE WHEN o.quality = "High" THEN 1 END) AS High,
       COUNT(CASE WHEN o.quality = "Medium" THEN 1 END) AS Medium,
       COUNT(CASE WHEN o.quality = "Low" THEN 1 END) AS Low
ORDER BY Low DESC
LIMIT 5
```



employee	High	Medium	Low
"Lisa Smith"	3	7	8
"Amanda Dudley"	5	6	8
"Brittany Johnson"	4	1	6
"Jason Gallagher"	4	6	5
"Barbara Bush"	5	2	5

Lisa Smith and Amanda Dudley have the weakest performance, each with 8 low-quality operations, highlighting a consistent quality issue. Among the group, Barbara Bush stands out as the best, with 5 high- and only 5 low-quality results, showing a relatively balanced output.

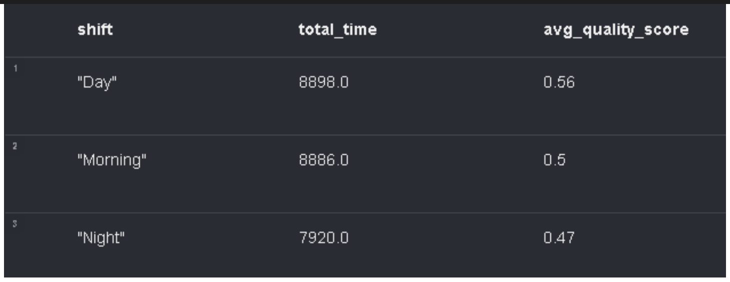
```
//creating a detailed table for the best employees
CALL {
  MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
  WITH e.name AS employee, COUNT(CASE WHEN o.quality = "High" THEN 1 END) AS high
  ORDER BY high DESC
  LIMIT 5
  RETURN collect(employee) AS top_names
}
UNWIND top_names AS employee_name
MATCH (e:Employee {name: employee_name})-[:PERFORMED]->(o:Operation)
RETURN
  employee_name AS employee,
  o.shift AS shift,
  COUNT(*) AS op_count,
  COUNT(CASE WHEN o.quality = "High" THEN 1 END) AS high_quality,
  COUNT(CASE WHEN o.quality = "Low" THEN 1 END) AS low_quality,
  ROUND(AVG(o.oee), 2) AS avg_oee
ORDER BY employee, shift
```



employee	shift	op_count	high_quality	low_quality	avg_oee
Amanda Dudley	Day	4	1	2	0.76
Amanda Dudley	Morning	8	2	2	0.79
Amanda Dudley	Night	7	2	4	0.78
Barbara Bush	Day	3	2	1	0.67
Barbara Bush	Morning	5	1	2	0.8
Barbara Bush	Night	4	2	2	0.75
Christopher Davis	Day	10	5	2	0.76
Christopher Davis	Morning	6	3	0	0.82
Christopher Davis	Night	1	1	0	0.75
Helen Peterson	Day	4	1	1	0.8
Helen Peterson	Morning	6	4	1	0.83
Helen Peterson	Night	5	3	1	0.73
John Taylor	Day	7	3	1	0.77
John Taylor	Morning	4	3	0	0.7
John Taylor	Night	3	1	2	0.9

According to the table above, which shows the information about the best 5 employees by the high quality, Christopher Davis worked hard during the day shift with the average oee rate of 0.76. Barbara Bush was the least hardworking among these, worked 12 times with the lowest oee ratio.

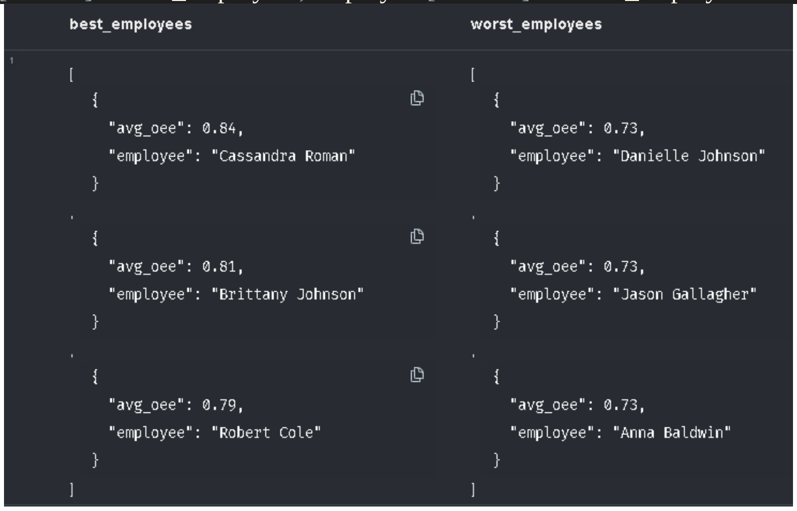
```
//Rank Shifts by Total Production Time and Average Quality
MATCH (:Employee)-[:PERFORMED]->(o:Operation)
WITH o.shift AS shift,
    SUM(o.duration) AS total_time,
    AVG(CASE
          WHEN o.quality = "High" THEN 1
          WHEN o.quality = "Medium" THEN 0.5
          ELSE 0
        END) AS quality_score
RETURN shift, total_time, ROUND(quality_score, 2) AS avg_quality_score
ORDER BY total_time DESC
```



shift	total_time	avg_quality_score
"Day"	8898.0	0.56
"Morning"	8886.0	0.5
"Night"	7920.0	0.47

In the table above, day and morning was really close to each other in terms of total minutes of working, but the average quality score was lower for the night inspite of the least working duration.

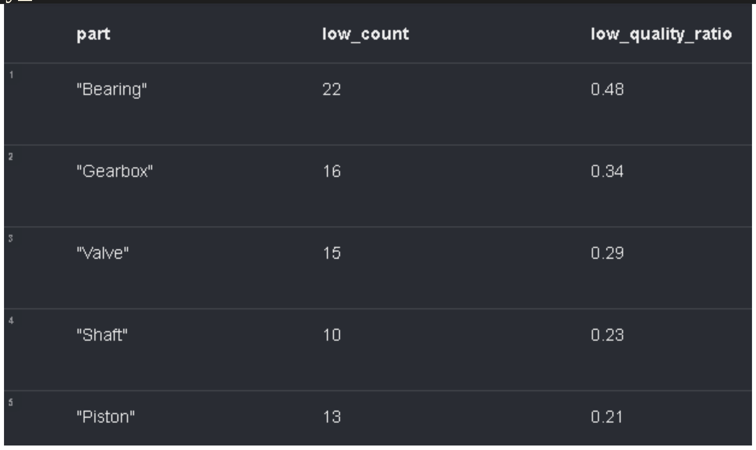
```
//3 best and 3 worst employees in terms of average oee
call{
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)
WITH e.name AS employee, AVG(o.oee) AS avg_oee
ORDER BY avg_oee DESC
RETURN collect({
  employee: employee,
  avg_oee: ROUND(avg_oee, 2)
}) AS employees
}
RETURN employees[..$limit] as best_employees, employees[-$limit..] as worst_employees
```



```
best_employees                              worst_employees

[                                           [
  {                                           {
    "avg_oee": 0.84,                            "avg_oee": 0.73,
    "employee": "Cassandra Roman"               "employee": "Danielle Johnson"
  }                                           }
,                                           ,
  {                                           {
    "avg_oee": 0.81,                            "avg_oee": 0.73,
    "employee": "Brittany Johnson"              "employee": "Jason Gallagher"
  }                                           }
,                                           ,
  {                                           {
    "avg_oee": 0.79,                            "avg_oee": 0.73,
    "employee": "Robert Cole"                   "employee": "Anna Baldwin"
  }                                           }
]                                           ]
```

3 the best and the worst (left and right respectivelty) can be seen with json formats. The lowest OEE score was 0.73, while that of the highest was found as 0.84 (Cassandra Roman) – who might be the most experienced operator.

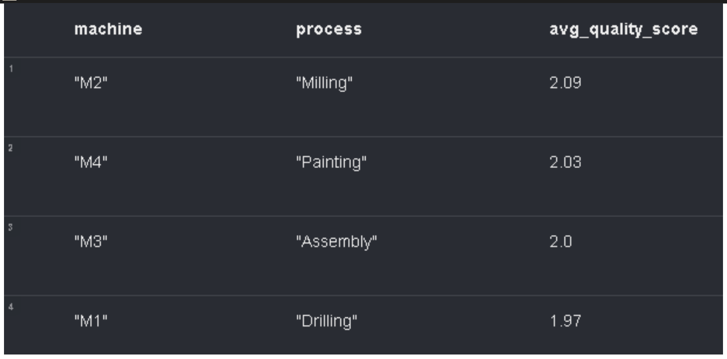
```
//Sort parts from low quality to high with the ratio
MATCH (o:Operation)-[:PRODUCED]->(p:Part)
WITH p.part_name AS part,
     COUNT(*) AS total,
     COUNT(CASE WHEN o.quality = 'Low' THEN 1 END) AS low_count
RETURN part,
       low_count,
       ROUND((1.0 * low_count) / total, 2) AS low_quality_ratio
ORDER BY low_quality_ratio DESC
```



part	low_count	low_quality_ratio
"Bearing"	22	0.48
"Gearbox"	16	0.34
"Valve"	15	0.29
"Shaft"	10	0.23
"Piston"	13	0.21

Bearing was found as the common part which was produced with the low quality among the all. Almost one in half of them was produced like that. On the other hand, piston held %0.79 with a good quality.

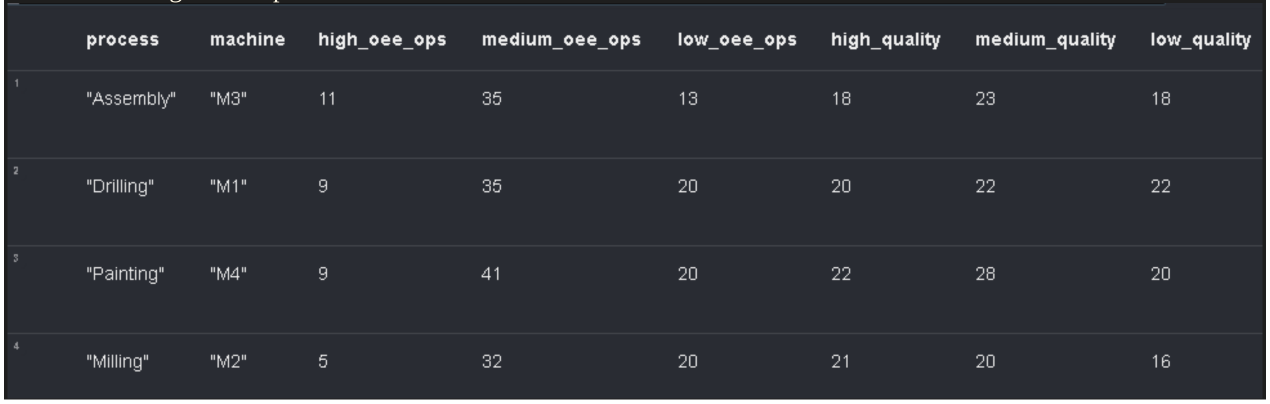
```
//Average quality score per process
MATCH (o:Operation)-[:USED]->(m:Machine)
WITH m.machine_name AS machine,
    m.process AS process,
     AVG(CASE o.quality
          WHEN 'High' THEN 3
          WHEN 'Medium' THEN 2
          ELSE 1 END) AS quality_score
RETURN
    machine,
    process,
    ROUND(quality_score, 2) AS avg_quality_score
ORDER BY avg_quality_score DESC
```



machine	process	avg_quality_score
"M2"	"Milling"	2.09
"M4"	"Painting"	2.03
"M3"	"Assembly"	2.0
"M1"	"Drilling"	1.97

Since each process belongs to a unique machine, the average quality score can be seen seperately. 1 means the lowest, while 3 means the highest quality. All of them were cumulated around moderate quality.

```
//Finding the relation between oee-machine-quality
MATCH (o:Operation)-[:USED]->(m:Machine)
RETURN m.process AS process,
       m.machine_name AS machine,
       COUNT(CASE WHEN o.oee >= 0.9 THEN 1 END) AS high_oee_ops,
       COUNT(CASE WHEN o.oee >= 0.7 and o.oee < 0.9 THEN 1 END) AS medium_oee_ops,
       COUNT(CASE WHEN o.oee < 0.7 THEN 1 END) AS low_oee_ops,
       COUNT(CASE WHEN o.quality='High' THEN 1 END) AS high_quality,
       COUNT(CASE WHEN o.quality='Medium' THEN 1 END) AS medium_quality,
       COUNT(CASE WHEN o.quality='Low' THEN 1 END) AS low_quality
ORDER BY high_oee_ops DESC
```



	process	machine	high_oee_ops	medium_oee_ops	low_oee_ops	high_quality	medium_quality	low_quality
1	"Assembly"	"M3"	11	35	13	18	23	18
2	"Drilling"	"M1"	9	35	20	20	22	22
3	"Painting"	"M4"	9	41	20	22	28	20
4	"Milling"	"M2"	5	32	20	21	20	16

According to the result above, assembly (machine 3) had the highest amount of high OEE score (greater than 0.9), and lowest amount of low OEE score (less than 0.7). Machine 3, in other words assembly, can be the best performed process among all.

```
//finding the shift-part-oee-quality relation
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)-[:PRODUCED]->(p:Part)
WITH o.shift as shift, p.part_name as part,  AVG(o.oee) as avg_oee,
AVG(CASE
   WHEN o.quality='High' THEN 1
   WHEN o.quality='Medium' THEN 0.5
   ELSE 0
END
) as quality_rate
WITH collect({
   shift: shift,
   part: part,
   avg_oee: ROUND(avg_oee,2),
   avg_quality: ROUND(quality_rate,2)

}) AS part_data
UNWIND part_data AS data
WITH data
RETURN data.shift as shift, data.part as part, data.avg_oee as avg_oee, data.avg_quality as avg_quality
ORDER BY part,shift
```

shift	part	avg_oee	avg_quality
Day	Bearing	0.76	0.38
Morning	Bearing	0.72	0.42
Night	Bearing	0.79	0.3
Day	Gearbox	0.74	0.4
Morning	Gearbox	0.77	0.5
Night	Gearbox	0.77	0.5
Day	Piston	0.78	0.72
Morning	Piston	0.79	0.48
Night	Piston	0.77	0.53
Day	Shaft	0.82	0.59
Morning	Shaft	0.74	0.46
Night	Shaft	0.75	0.68
Day	Valve	0.75	0.58
Morning	Valve	0.77	0.58
Night	Valve	0.77	0.43

According to the shift-part-oee-quality table above, the part of shaft had the highest average OEE rate in the day shift with 59% average quality. The greatest average quality was observed for the piston in the day shift with average OEE score of 0.78.

```
//Which operations took significantly longer than the average for the same part - and who perfor
med them, using which machine and in which shift?
:param threshold_factor => 1.5;
:param limit => 10;

MATCH (o:Operation)-[:PRODUCED]->(p:Part)
WITH p.part_name AS part, o
WITH part, AVG(o.duration) AS avg_duration
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)-[:PRODUCED]-
>(p:Part {part_name: part}),
      (o)-[:USED]->(m:Machine)
WHERE o.duration > avg_duration * $threshold_factor
RETURN
  part,
  o.shift AS shift,
  o.quality AS quality,
  o.duration AS actual_duration,
  ROUND(avg_duration, 1) AS avg_duration,
  e.name AS employee,
  m.machine_name AS machine,
  m.process AS process
ORDER BY actual_duration DESC
LIMIT $limit
```

part	shift	quality	actual_duration	avg_duration	employee	machine	process
Valve	Morning	Medium	180	99.3	Christian Carter	M4	Painting
Valve	Morning	Medium	180	99.3	Melissa Delacruz	M4	Painting
Bearing	Night	Low	179	104.6	Danielle Johnson	M4	Painting
Bearing	Day	Medium	178	104.6	Christopher Davis	M3	Assembly
Piston	Morning	Medium	178	92.2	John Taylor	M2	Milling
Valve	Night	Low	177	99.3	Cassandra Roman	M4	Painting
Valve	Morning	Medium	177	99.3	Amanda Dudley	M4	Painting
Gearbox	Morning	Low	176	106.9	Barbara Bush	M4	Painting
Piston	Night	High	176	92.2	Helen Peterson	M2	Milling
Bearing	Morning	High	176	104.6	John Taylor	M4	Painting

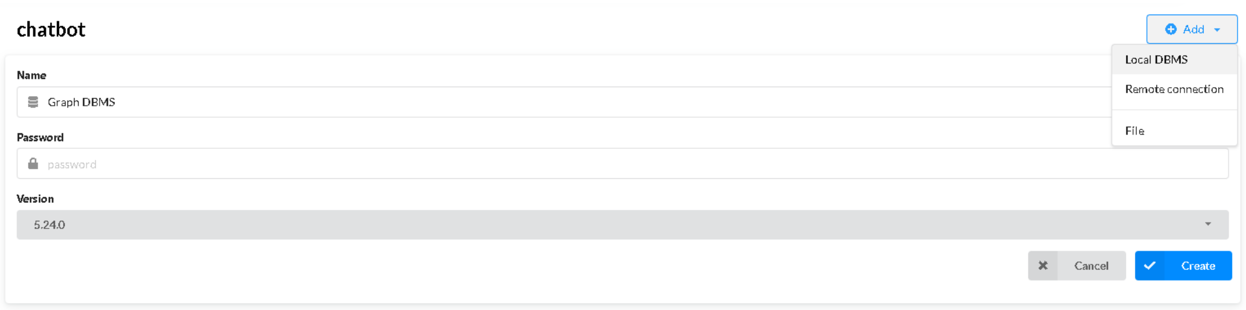
This table highlights operations that took significantly longer than the average duration for the same part. The longest delays occurred during Valve and Shaft productions, particularly in the Morning and Night shifts. Notably, Brittany Johnson and John Taylor are linked to multiple prolonged operations, often using Machine M2 (Milling), suggesting a potential performance or machine-related bottleneck. These patterns indicate that certain employee-machine combinations may require further investigation or support to reduce outliers and improve overall efficiency.

SUMMARY

In this assignment, advanced-level Cypher queries were used to extract detailed insights from a manufacturing dataset modeled in Neo4j. The analysis revealed standout performers like Christopher Davis, who consistently delivered high-quality outputs, and highlighted underperformers such as Amanda Dudley and Lisa Smith, suggesting the need for further evaluation or support. Among parts, bearings showed the highest defect rates, while pistons performed reliably with strong quality and OEE. Assembly (Machine M3) emerged as the most efficient process. Shift-wise, the Day shift proved optimal for producing parts like shafts and pistons, though overall productivity varied. Anomaly detection further identified operations that exceeded expected durations, particularly involving Valve and Shaft in Morning and Night shifts, often linked to Machine M2 and specific employees. Throughout the work, performance-optimized query structures-like single MATCH clauses and parameterized slicing-were effectively used to enhance both readability and execution speed.

EXTRA: Preparing Chatbot Requirements

Creating a DBMS in Neo4j desktop: In Neo4j Desktop application, a DBMS is created. Password is set here provided as an environment variable into .env file.


chatbot

Name: Graph DBMS

Password: password

Version: 5.24.0

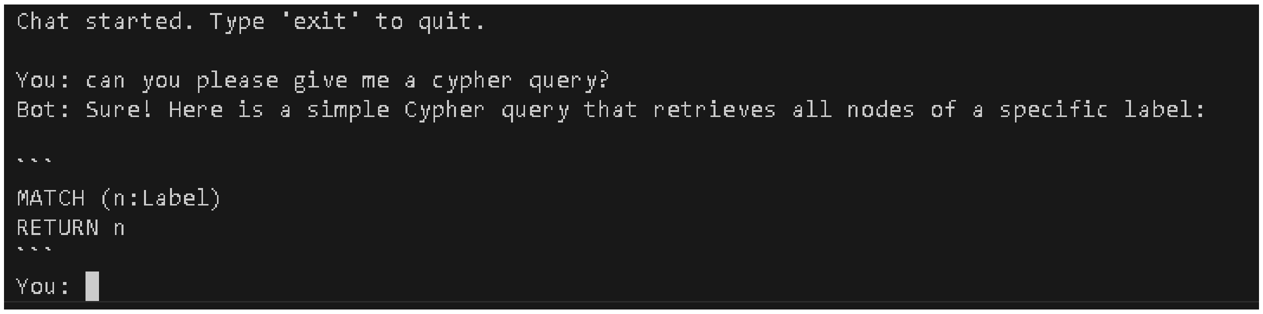
Installing required libraries: pip install openai langchain langchain-community neo4j python-dotenv

Providing .env file: It's required to access the relevant services. OpenAI key is necessary to send request to pre-trained LLM model. Other ones are related to the connection to Neo4j. To chat with LLM, only OpenAI API key will be enough to provide

```
OPENAI_API_KEY      =your-openai-api-key
NEO4J_URI           =bolt://your-neo4j-uri
NEO4J_USERNAME      =neo4j
NEO4J_PASSWORD      =your-password
```

Simple chat code: With the code below, simple the request to OpenAI LLM can be performed by API request. However, the model doesn't have a memory like this, so it doesn't remember any previous messages.

```
import os
from dotenv import load_dotenv
from langchain.chat_models import ChatOpenAI

# Load API key from .env file
load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")



# Initialize LLM
llm = ChatOpenAI(openai_api_key=api_key, temperature=0)



# Chat loop
print("Chat started. Type 'exit' to quit.\n")
while True:
    user_input = input("You: ")
    if user_input.lower() in ["exit", "quit"]:
        break
    response = llm.invoke(user_input)
    print("Bot:", response.content)
```



~~~
Chat started. Type 'exit' to quit.

You: can you please give me a cypher query?
Bot: Sure! Here is a simple Cypher query that retrieves all nodes of a specific label:

```
MATCH (n:Label)
RETURN n
```
You: 
~~~

As it can be seen above, the chatbot is asked if it could give a cypher query. It gave a simple query.