**Neo4j Work:** Advanced-Level Cypher Queries in Neo4j

**Prepared by:** Mert OLÇAMAN

**Week:** 5-6

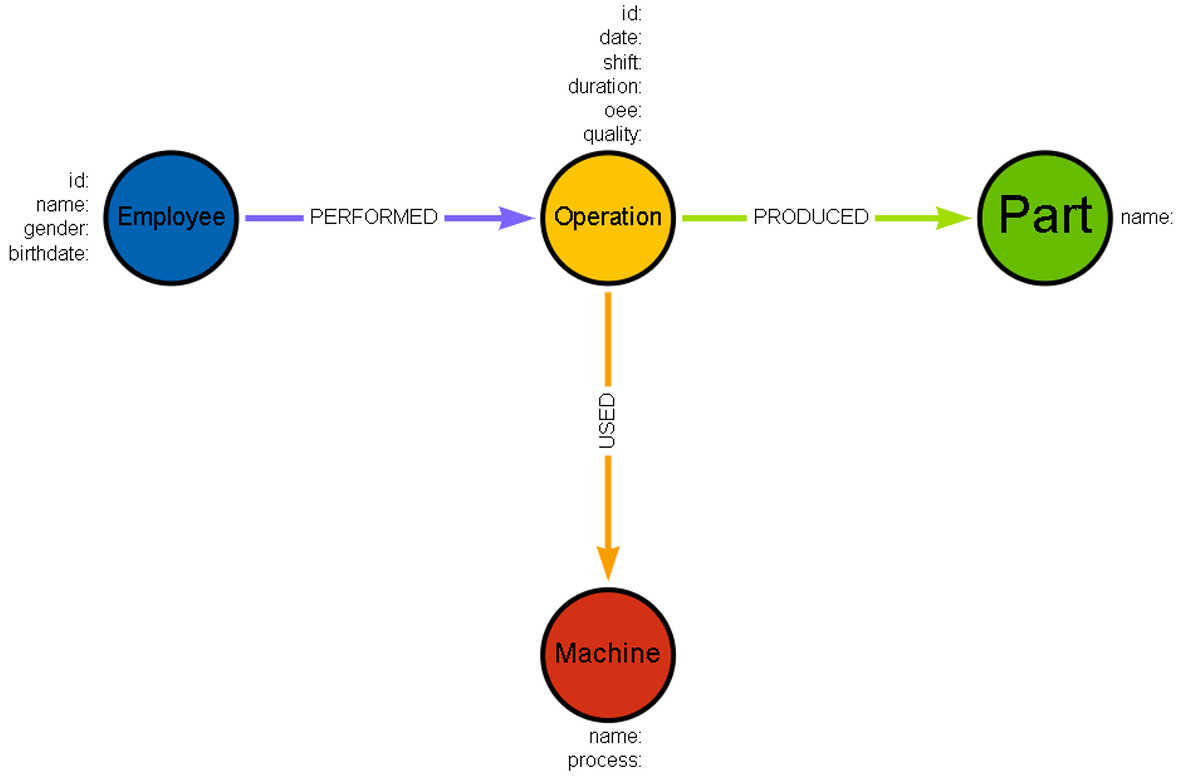
**patika.dev**

**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.





1. **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);

**GRAPH DESIGN**

**TEST**

**QUESTIONS**

***Before starting, each question related to the query was specified on top of the query.***

**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;

****

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.

1. **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,

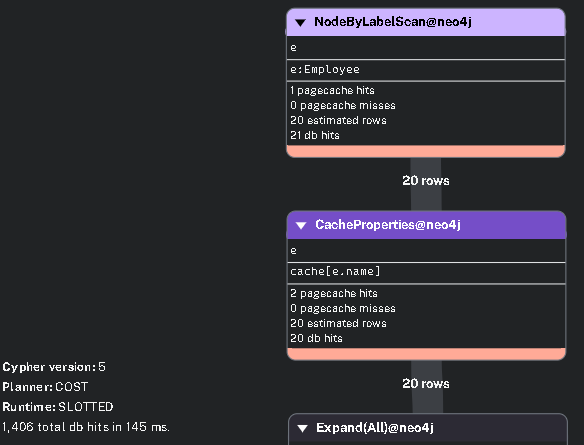
  COUNT(\*) AS op\_count,

  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

****

//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,

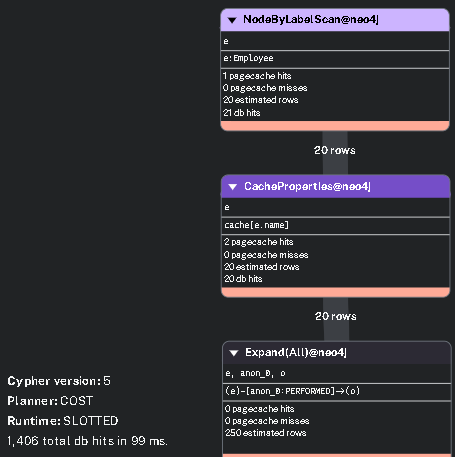
  COUNT(\*) AS op\_count,

  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



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

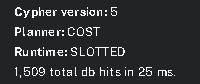
**Query comparison-2:**

PROFILE

MATCH (e:Employee)-[:PERFORMED]->(o:Operation)

MATCH (o)-[:USED]->(m:Machine)

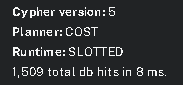
RETURN e.name, COUNT(\*) AS op\_count



PROFILE

MATCH (e:Employee)-[:PERFORMED]->(o:Operation)-[:USED]->(m:Machine)

RETURN e.name, COUNT(\*) AS op\_count

****

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

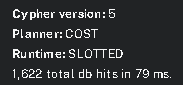
MATCH (e:Employee)-[:PERFORMED]->(o:Operation)

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

****

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

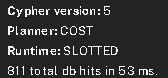
}

WITH employees

RETURN

  employees[..3] AS best\_employees,   // Top 3

  employees[-3..] AS worst\_employees  // Bottom 3



: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({

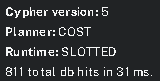
  employee: employee,

  avg\_oee: ROUND(avg\_oee, 2)

}) AS employees

}

RETURN employees[..$limit] as best\_employees, employees[-$limit..] as worst\_employees

****

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.

1. **Advanced-level cypher queries**

//Top 5 employees perform with high quality

MATCH (e:Employee)-[:PERFORMED]->(o:Operation)

RETURN e.name AS employee,

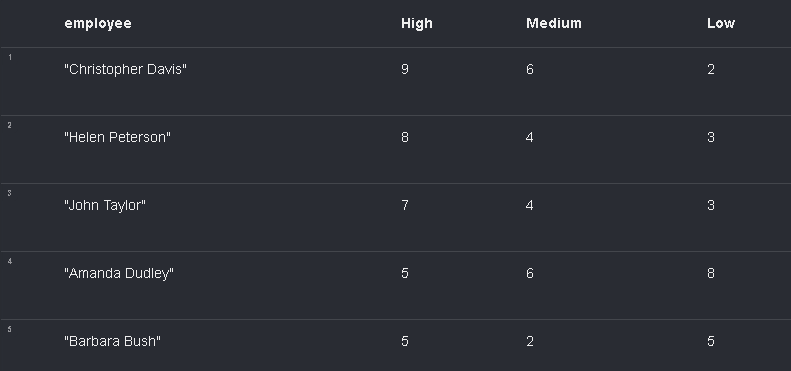
       COUNT(CASE WHEN o.quality = "High" THEN 1 END) AS High,

       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



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,

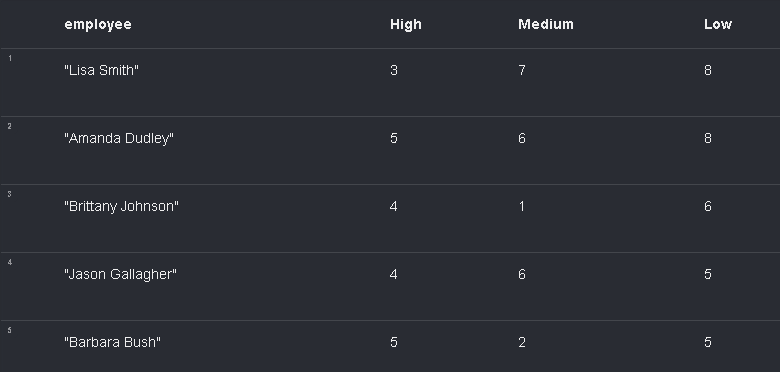
       COUNT(CASE WHEN o.quality = "High" THEN 1 END) AS High,

       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



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



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

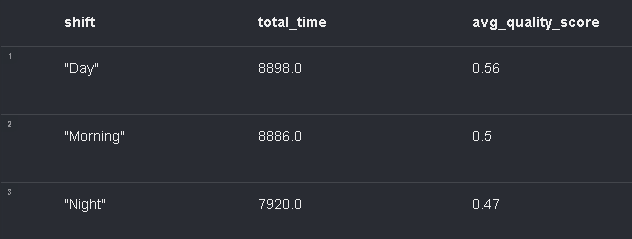
            WHEN o.quality = "Medium" THEN 0.5

            ELSE 0

         END) AS quality\_score

RETURN shift, total\_time, ROUND(quality\_score, 2) AS avg\_quality\_score

ORDER BY total\_time DESC



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({

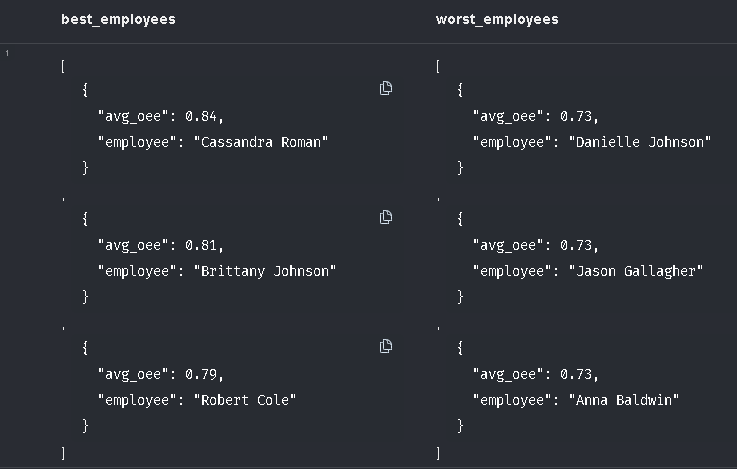
  employee: employee,

  avg\_oee: ROUND(avg\_oee, 2)

}) AS employees

}

RETURN employees[..$limit] as best\_employees, employees[-$limit..] as worst\_employees



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,

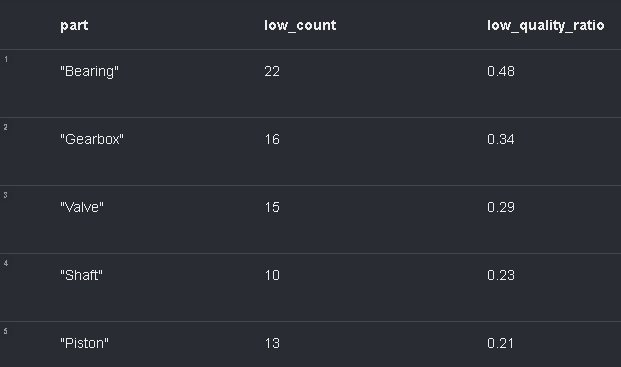
     COUNT(CASE WHEN o.quality = 'Low' THEN 1 END) AS low\_count

RETURN part,

       low\_count,

       ROUND((1.0 \* low\_count) / total, 2) AS low\_quality\_ratio

ORDER BY low\_quality\_ratio DESC



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

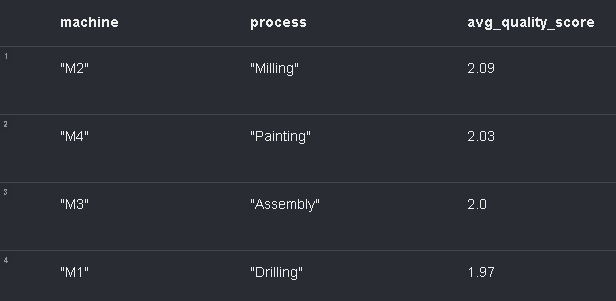
RETURN

    machine,

    process,

    ROUND(quality\_score, 2) AS avg\_quality\_score

ORDER BY avg\_quality\_score DESC



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,

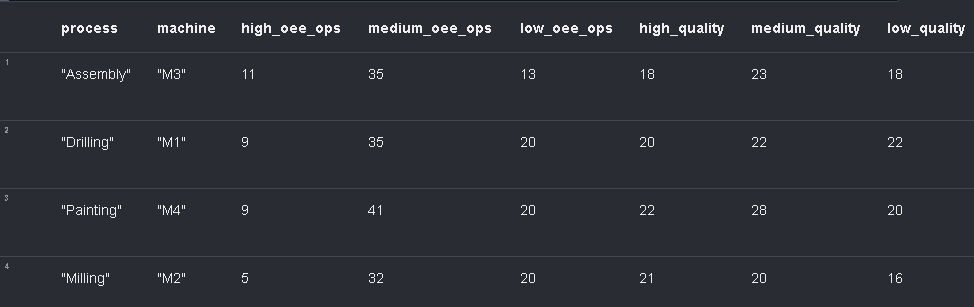
       COUNT(CASE WHEN o.oee >= 0.7 and o.oee < 0.9 THEN 1 END) AS medium\_oee\_ops,

       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  


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



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 performed 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



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.