Competency Questions and Query Language Comparison: SPARQL vs. Cypher

1. Introduction

Our Knowledge graph (KG) has emerged as a powerful paradigm for representing complex, interconnected data, enabling sophisticated querying and inferencing capabilities. The choice of query language significantly impacts how information is extracted and analyzed from the graph. This document is an extended detailed section delves into the practical application of two prominent graph query languages, SPARQL for RDF-based knowledge graphs (e.g., managed by GraphDB) and Cypher for property graph databases (e.g., Neo4j). We demonstrate their utility through a set of competency questions derived from a real-world industrial maintenance knowledge graph, highlighting their respective strengths and differences in expressing complex queries and presenting results, while maintaining data confidentiality through ambiguous result representation.

2. Competency Questions for Knowledge Graph Analysis

Competency questions (CQs) serve as a crucial bridge between informal user requirements and formal queries, ensuring that the knowledge graph can answer the questions it was designed to address. For our industrial maintenance knowledge graph, which integrates information about personnel, equipment, locations, tasks, and safety protocols, the following samples of competency questions are important for operational efficiency and safety management:

CQ1: Personnel Expertise and Availability

- "Which available personnel possess expertise in 'Vacuum, cryogenics' and have an accumulated radiation dose below a certain threshold?"

CQ2: Task Assignment and Location Safety

- "Identify all tasks assigned to personnel with a specific skill set that are to be performed in locations requiring Personal Protective Equipment (PPE) and having an 'Orange' radiation classification."

CQ3: Equipment Utilization and Maintenance History

- "Retrieve the maintenance history for a given piece of equipment, including the types of maintenance performed, the personnel involved, and the duration of each activity."

CQ4: Critical Path Analysis for Maintenance Activities

- "Determine the sequence of maintenance activities that constitute the critical path for a major system overhaul, considering task dependencies and personnel availability."

CQ5: Radiation Exposure and Personnel Reassignment

- "Given a personnel member whose accumulated radiation dose exceeds a predefined limit, identify alternative qualified personnel for their currently assigned tasks in high-radiation areas."

CQ6: Infrastructure Utilization and Capacity

- "Which maintenance workshops are currently underutilized, and what types of equipment are typically maintained there, indicating potential for increased throughput or specialized service expansion?"

CQ7: Safety Protocol Compliance

- "Verify that all tasks performed in 'Red' radiation classification areas are assigned to personnel with the appropriate safety certifications and that all required PPE is specified for the location."

CQ8: Cross-Disciplinary Task Identification

- "Find tasks that require expertise from at least two different areas of knowledge (e.g., 'Mechanics' and 'Electricity, Electronics, including RF') and identify the personnel capable of performing such tasks."

CQ9: Anomaly Detection in Task Durations

- "Identify maintenance tasks whose actual duration significantly deviates from their estimated duration, considering the handling method (e.g., 'Hands-off' vs. 'Hands-on') and location radiation levels."

CQ10: Resource Allocation Optimization

- "For a set of high-priority maintenance activities, suggest an optimal allocation of available personnel and equipment to minimize overall completion time, taking into account their respective skills, certifications, and current assignments."

These competency questions guide the formulation of queries in both SPARQL and Cypher, allowing for a direct comparison of their expressive power and performance characteristics in a practical context.

3. Query Language Implementation and Comparison

This section presents the implementation of queries for each competency question using both SPARQL and Cypher, followed by a comparative analysis of their syntax, expressiveness, and the nature of their results. To maintain data confidentiality, all query results are presented in an ambiguous, generalized format.

CQ1: Personnel Expertise and Availability

Competency Question: "Which available personnel possess expertise in 'Vacuum, cryogenics' and have an accumulated radiation dose below a certain threshold?"

This question requires filtering personnel based on their area of knowledge, availability status, and a numerical threshold on their accumulated radiation dose. It tests the ability of both query languages to handle property filtering and numerical comparisons.

SPARQL Query (GraphDB)

```
 PREFIX : <http://www.semanticweb.org/maintenance-ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?personnelUID ?areaOfKnowledge ?availability ?accumulatedDose
WHERE {
    ?personnel a :Personnel ;
               :hasPersonnelUID ?personnelUID ;
               :hasAreaOfKnowledge "Vacuum, cryogenics" ;
               :hasPersonnelAvailability "true"^^xsd:boolean ;
               :hasAccumulatedDoze ?accumulatedDose .
    FILTER (?accumulatedDose < 10) .
}
ORDER BY ?personnelUID
```

Ambiguous SPARQL Result (Example):

personnelUID	areaOfKnowledge	availability	accumulatedDose
PersXXX	Vacuum, cryogenics	true	5
PersYYY	Vacuum, cryogenics	true	8
PersZZZ	Vacuum, cryogenics	true	3
...	...	...	...

Explanation of SPARQL Query:

The SPARQL query leverages prefixes for readability and directly queries for individuals of type `:Personnel`. It uses property paths to retrieve the `personnelUID`, `hasAreaOfKnowledge`, `hasPersonnelAvailability`, and `hasAccumulatedDoze`. A `FILTER` clause is applied to restrict the results to personnel with

an accumulated dose less than 10. The `ORDER BY` clause ensures consistent result ordering.

Cypher Query (Neo4j)

```
 MATCH (p:Personnel)
WHERE p.hasAreaOfKnowledge = 'Vacuum, cryogenics'
  AND p.hasPersonnelAvailability = true
  AND p.hasAccumulatedDoze < 10
RETURN p.hasPersonnelUID AS personnelUID, p.hasAreaOfKnowledge AS areaOfKnowledge, p.hasPersonnelAvailability AS
availability, p.hasAccumulatedDoze AS accumulatedDose
ORDER BY personnelUID
```

Ambiguous Cypher Result (Example):

personnelUID	areaOfKnowledge	availability	accumulatedDose
PersXXX	Vacuum, cryogenics	true	5
PersYYY	Vacuum, cryogenics	true	8
PersZZZ	Vacuum, cryogenics	true	3
...	...	...	...

Comparison for CQ1:

For CQ1, both SPARQL and Cypher demonstrate straightforward filtering capabilities. SPARQL, being RDF-centric, uses triple patterns (`?subject ?predicate ?object`) and `FILTER` clauses for conditions. Its verbose nature with full URIs (though mitigated by `PREFIX` declarations) makes it explicit about the semantic relationships. Cypher, on the other hand, utilizes a more intuitive pattern-matching syntax (`MATCH (node)-[relationship]->(node)`) and property access (`node.property`). The `WHERE` clause in Cypher serves a similar purpose to SPARQL's `FILTER`. The results from both queries, when executed against their respective graph databases, would yield similar sets of personnel, albeit with potentially different internal representations of the data. Cypher's syntax is arguably more concise and readable for this type of direct property filtering, resembling natural language more closely.

CQ2: Task Assignment and Location Safety

Competency Question: "Identify all tasks assigned to personnel with a specific skill set that are to be performed in locations requiring Personal Protective Equipment (PPE) and having an 'Orange' radiation classification."

This competency question requires traversing relationships between tasks, personnel, and locations, and applying multiple filtering conditions on properties of both personnel and locations. This will highlight the graph traversal capabilities of both languages.

SPARQL Query (GraphDB)

```
 PREFIX : <http://www.semanticweb.org/maintenance-ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX ns: <http://www.semanticweb.org/maintenance-ontology#>

SELECT DISTINCT ?taskUID ?personnelUID ?locationName ?radiationClassification
WHERE {
    ?task a :MaintenanceTask ;
          :hasTaskUID ?taskUID ;
          :isPerformedBy ?personnel .
    ?personnel a :Personnel ;
               :hasPersonnelUID ?personnelUID ;
               :hasAreaOfKnowledge "Metrology, Control, Instrumentation" ;
               :worksAt ?location.
    ?task :worksAt ?location .
    ?location a :Location ;
              :requiresPPE "true"^^xsd:boolean ;
              :hasRadiationClassification ns:Orange ;
              :hasLocationName ?locationName ;
              :hasRadiationClassification ?radiationClassification .

    BIND("Orange" AS ?radiationClassification)
}
ORDER BY ?taskUID
```

Ambiguous SPARQL Result (Example):

taskUID	personnelUID	locationName	radiationClassification
Task0XX	PersYYY	LocationA	Orange
Task0ZZ	PersAAA	LocationB	Orange
...	...	...	...

Explanation of SPARQL Query:

This SPARQL query identifies `MaintenanceTask` instances (`?task`) that are performed by `Personnel` (`?personnel`) with a specific `hasAreaOfKnowledge` (e.g., "Metrology, Control, Instrumentation"). It then links these tasks to `Location` instances (`?location`) where `requiresPPE` is true and `hasRadiationClassification` is 'Orange'. The `DISTINCT` keyword ensures unique task entries. The query demonstrates property path traversal and filtering across multiple entities.

Cypher Query (Neo4j)

```
 MATCH (t:MaintenanceTask)-[:isPerformedBy]->(p:Personnel),
      (t)-[:worksAt]->(l:Location)-[:hasRadiationClassification]->(rc:RadiationClassification)
WHERE p.hasAreaOfKnowledge = 'Metrology, Control, Instrumentation'
  AND l.requiresPPE = true
  AND (rc.uri ENDS WITH '#Orange' OR rc.id = 'Orange')
RETURN t.hasTaskUID AS taskUID, p.hasPersonnelUID AS personnelUID, l.hasLocationName AS locationName,
l.hasRadiationClassification AS radiationClassification
ORDER BY taskUID
```

Ambiguous Cypher Result (Example):

taskUID	personnelUID	locationName	radiationClassification
Task0XX	PersYYY	LocationA	Orange
Task0ZZ	PersAAA	LocationB	Orange
...	...	...	...

Comparison for CQ2:

For CQ2, the differences in graph traversal become more apparent. SPARQL uses multiple triple patterns to connect `?task`, `?personnel`, and `?location`, relying on explicit predicate URIs. The relationships are implicitly defined by the shared variables. Cypher, conversely, uses a more visual, ASCII-art-like syntax to represent graph patterns (`(node)-[relationship]->(node)`). This makes the relationships between entities explicit and often more readable for graph-native structures. Both languages effectively

handle multi-hop queries and filtering on properties of connected nodes. Cypher's pattern matching can feel more natural for complex graph traversals, while SPARQL's explicit triple patterns offer a precise, RDF-standardized way to express relationships.

CQ3: Equipment Utilization and Maintenance History

Competency Question: "Retrieve the maintenance history for a given piece of equipment, including the types of maintenance performed, the personnel involved, and the duration of each activity."

This question requires retrieving a list of related activities and their details, which involves traversing multiple relationships and potentially aggregating information. This will demonstrate how each language handles retrieving connected data and presenting it as a cohesive history.

SPARQL Query (GraphDB)

```
PREFIX : <http://www.semanticweb.org/maintenance-ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?activityName ?activityUID ?maintenanceType ?personnelUID ?taskDuration
WHERE {
    ?MaintenanceMeans a :MaintenanceMeans ;
                :hasMeansName "EquipmentX" .
    ?activity :hasActivityMaintainableItem ?equipment ;
              :hasMaintenanceActivityName ?activityName ;
              :hasMaintenanceActivityUID ?activityUID ;
              :hasMaintenanceType ?maintenanceType ;
              :requiresPersonnel ?personnel .
    ?personnel :hasPersonnelUID ?personnelUID .
    OPTIONAL { ?activity :hasTask ?task .
               ?task :hasTaskDuration ?taskDuration . }
}
ORDER BY ?activityUID
```

Ambiguous SPARQL Result (Example):

activityName	activityUID	maintenanceType	personnelUID	taskDuration
ActivityA	Act001	Preventive	Pers001	120
ActivityB	Act002	Corrective	Pers002	60
ActivityC	Act003	Preventive	Pers001	90
...	...	...	...	...

Explanation of SPARQL Query:

This SPARQL query starts by identifying a specific `Equipment` instance (e.g., "EquipmentX"). It then finds all `MaintenanceActivity` instances (`?activity`) associated with this equipment via the `:hasActivityMaintainableItem` property. For each activity, it retrieves its name, UID, and type. It also links to the `Personnel` involved and their UIDs. The `OPTIONAL` clause is used to retrieve `taskDuration` if available, as not all activities might have directly associated tasks with durations. This query demonstrates how SPARQL can traverse relationships to gather comprehensive information about an entity's history.

Cypher Query (Neo4j)

```
MATCH (e:MaintenanceMeans {hasMeansName: 'EquipmentX'})<-[:hasActivityMaintainableItem]-(a:MaintenanceActivity)
OPTIONAL MATCH (a)-[:requiresPersonnel]->(p:Personnel)
OPTIONAL MATCH (a)-[:HAS_TASK]->(t:MaintenanceTask)
RETURN a.hasMaintenanceActivityName AS activityName, a.hasMaintenanceActivityUID AS activityUID, a.hasMaintenanceType AS
maintenanceType, p.hasPersonnelUID AS personnelUID, t.hasTaskDuration AS taskDuration
ORDER BY activityUID
```

Ambiguous Cypher Result (Example):

activityName	activityUID	maintenanceType	personnelUID	taskDuration
ActivityA	Act001	Preventive	Pers001	120
ActivityB	Act002	Corrective	Pers002	60
ActivityC	Act003	Preventive	Pers001	90
...	...	...	...	...

Comparison for CQ3:

For CQ3, both languages effectively retrieve connected data to form a historical record. Cypher's pattern matching `(e:Equipment {hasItemname: 'EquipmentX'})<-[:hasActivityMaintainableItem]-(a:MaintenanceActivity)` clearly shows the direction of the relationship and the filtering on the equipment name. The use of `OPTIONAL MATCH` in Cypher is analogous to SPARQL's `OPTIONAL` clause, allowing for the inclusion of data that might not be present for all activities (e.g., `taskDuration`). While SPARQL relies on explicit triple patterns and variable binding, Cypher's graph patterns provide a more visual and often more intuitive way to express these relationships, especially when dealing with complex interconnected data. The readability of Cypher for graph traversal and data retrieval is a notable advantage in this scenario.

CQ4: Critical Path Analysis for Maintenance Activities

Competency Question: "Determine the sequence of maintenance activities that constitute the critical path for a major system overhaul, considering task dependencies and personnel availability."

This is a complex question that goes beyond simple data retrieval and requires graph algorithms or sophisticated pathfinding capabilities. While neither SPARQL nor Cypher inherently provide full-fledged critical path algorithms, they can be used to extract the necessary data for such analysis or to model dependencies. For the purpose of this comparison, we will focus on identifying chains of dependent activities.

SPARQL Query (GraphDB)

```
 PREFIX : <http://www.semanticweb.org/maintenance-ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?activity1UID ?activity2UID ?activity1Name ?activity2Name
WHERE {
    ?activity1 a :MaintenanceActivity ;
               :hasMaintenanceActivityUID ?activity1UID ;
               :hasMaintenanceActivityName ?activity1Name .
    ?activity2 a :MaintenanceActivity ;
               :hasMaintenanceActivityUID ?activity2UID ;
               :hasMaintenanceActivityName ?activity2Name .
    ?activity1 :hasActivitySucessor ?activity2 .
}
ORDER BY ?activity1UID ?activity2UID
```

Ambiguous SPARQL Result (Example):

activity1UID	activity2UID	activity1Name	activity2Name
Act001	Act002	PrepPhase	MainRepair
Act002	Act003	MainRepair	TestPhase
...	...	...	...

Explanation of SPARQL Query:

This SPARQL query identifies direct successor relationships between maintenance activities. It finds pairs of activities where one activity (`?activity1`) has another (`?activity2`) as a successor. While this query doesn't compute the critical path directly, it provides the foundational data (activity dependencies) needed for external critical path analysis. More complex SPARQL queries involving property paths (`+` or `*`) could be used to find all indirect dependencies, but true critical path analysis often requires iterative algorithms or specialized graph processing tools beyond standard SPARQL capabilities. This query focuses on extracting the immediate dependency structure.

Cypher Query (Neo4j)

```
MATCH (a1:MaintenanceActivity)-[:hasActivitySuccessor]->(a2:MaintenanceActivity)
RETURN a1.hasMaintenanceActivityUID AS activity1UID, a2.hasMaintenanceActivityUID AS activity2UID,
a1.hasMaintenanceActivityName AS activity1Name, a2.hasMaintenanceActivityName AS activity2Name
ORDER BY activity1UID, activity2UID
```

Ambiguous Cypher Result (Example):

activity1UID	activity2UID	activity1Name	activity2Name
Act001	Act002	PrepPhase	MainRepair
Act002	Act003	MainRepair	TestPhase
...	...	...	...

Comparison for CQ4:

For critical path analysis, both SPARQL and Cypher, in their basic forms, are primarily designed for querying existing graph structures rather than executing complex graph algorithms like critical path determination. Both queries for CQ4 focus on identifying direct successor relationships, which forms the basis for such analysis. Cypher's pattern matching `(a1:MaintenanceActivity)-[:hasActivitySuccessor]->(a2:MaintenanceActivity)` is again more visually intuitive for representing these directed relationships. While both languages can be extended or integrated with external tools for full critical path computation (e.g., using property paths in SPARQL or APOC procedures in Neo4j for Cypher), their core syntax for this type of query is about extracting the raw dependency data. The choice between them for this specific task often comes down to the underlying graph database technology and the ecosystem of tools available for each.

CQ5: Radiation Exposure and Personnel Reassignment

Competency Question: "Given a personnel member whose accumulated radiation dose exceeds a predefined limit, identify alternative qualified personnel for their currently assigned tasks in high-radiation areas."

This question requires identifying personnel based on a threshold, then finding their assigned tasks, and finally searching for other personnel who are qualified for those tasks and are not exceeding the radiation limit. This involves multi-step reasoning and filtering.

SPARQL Query (GraphDB)

```
PREFIX : <http://www.semanticweb.org/maintenance-ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?overExposedPersonnelUID ?taskUID ?alternativePersonnelUID ?alternativePersonnelAreaOfKnowledge
WHERE {
    ?overExposedPersonnel a :Personnel ;
                          :hasPersonnelUID ?overExposedPersonnelUID ;
                          :hasAccumulatedDoze ?accumulatedDoze .
    FILTER (?accumulatedDoze > 10) .

    ?task :isPerformedBy ?overExposedPersonnel ;
          :hasTaskUID ?taskUID.
    ?location :hasRadiationClassification :Orange .

    ?alternativePersonnel a :Personnel ;
                          :hasPersonnelUID ?alternativePersonnelUID ;
                          :hasAreaOfKnowledge ?alternativePersonnelAreaOfKnowledge ;
                          :hasPersonnelAvailability "true"^^xsd:boolean ;
                          :hasAccumulatedDoze ?alternativeAccumulatedDoze;
                          :worksAt ?location .
    FILTER (?alternativeAccumulatedDoze <= 10) .

    ?task :isPerformedBy ?alternativePersonnel .
    FILTER (?overExposedPersonnel != ?alternativePersonnel) .
}
ORDER BY ?overExposedPersonnelUID ?taskUID
```

Ambiguous SPARQL Result (Example):

overExposedPersonnelUID	taskUID	alternativePersonnelUID	alternativePersonnelAreaOfKnowledge
Pers001	Task005	Pers007	Mechanics
Pers001	Task012	Pers010	Vacuum, cryogenics
Pers003	Task020	Pers009	RH
...	...	...	...

Explanation of SPARQL Query:

This SPARQL query first identifies personnel (`?overExposedPersonnel`) whose `hasAccumulatedDoze` exceeds a threshold (e.g., 10). It then finds the `MaintenanceTask` instances (`?task`) performed by these over-exposed personnel, specifically focusing on tasks performed in 'Orange' radiation classification locations. Finally, it searches for `alternativePersonnel` who are available, have an accumulated dose below the threshold, and are also assigned to the same tasks. A `FILTER` clause ensures that the alternative personnel are not the same as the over-exposed personnel. This query demonstrates SPARQL's ability to combine multiple conditions and traverse complex paths to find suitable alternatives based on specific criteria. The query structure reflects a logical flow of identifying a problem (over-exposed personnel), finding associated tasks, and then seeking qualified replacements.

Cypher Query (Neo4j)

```
 MATCH (t:MaintenanceTask)-[:isPerformedBy]->(op:Personnel)-[worksAt]->(l:Location)-[:hasRadiationClassification]->
(rc:RadiationClassification)
WHERE op.hasAccumulatedDoze > 10
  AND rc.uri ENDS WITH '#Orange'
MATCH (t)-[:isPerformedBy]->(ap:Personnel)
WHERE ap.hasAccumulatedDoze <= 10
  AND ap.hasPersonnelAvailability = true
  AND ap <> op
RETURN op.hasPersonnelUID AS overExposedPersonnelUID, t.hasTaskUID AS taskUID, ap.hasPersonnelUID AS
alternativePersonnelUID, ap.hasAreaOfKnowledge AS alternativePersonnelAreaOfKnowledge
ORDER BY overExposedPersonnelUID, taskUID
```

Ambiguous Cypher Result (Example):

overExposedPersonnelUID	taskUID	alternativePersonnelUID	alternativePersonnelAreaOfKnowledge
Pers001	Task005	Pers007	Mechanics
Pers001	Task012	Pers010	Vacuum, cryogenics
Pers003	Task020	Pers009	RH
...	...	...	...

Comparison for CQ5:

For CQ5, both SPARQL and Cypher handle the multi-step reasoning and filtering effectively. Cypher's pattern matching, with its clear representation of relationships and properties, makes the query flow quite readable: first, identify over-exposed personnel and their tasks in high-radiation areas, then find alternative personnel for those same tasks. The `MATCH` clauses in Cypher are powerful for expressing these complex relationships. SPARQL, while achieving the same result, requires more explicit triple patterns and `FILTER` clauses to define the relationships and conditions. The readability of Cypher for this type of complex relational query, especially with multiple `MATCH` clauses, can be seen as an advantage for those familiar with graph database concepts. Both languages demonstrate their capability to perform sophisticated queries that involve multiple entities and conditions, crucial for decision-making in a knowledge graph.

CQ6: Infrastructure Utilization and Capacity

Competency Question: "Which maintenance workshops are currently underutilized, and what types of equipment are typically maintained there, indicating potential for increased throughput or specialized service expansion?"

This question requires identifying locations of a specific type (maintenance workshops), assessing their utilization (which would typically involve counting assigned activities or equipment, and comparing against a capacity), and then linking them to the types

of equipment they handle. For the purpose of this example, we will focus on identifying workshops and the equipment associated with them, and assume 'underutilized' can be inferred by a low number of associated activities/MaintenanceMeans (though a true measure would require more data).

SPARQL Query (GraphDB)

```
 PREFIX : <http://www.semanticweb.org/maintenance-ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?workshopName
       (COUNT(DISTINCT ?equipment) AS ?equipmentCount)
       (GROUP_CONCAT(DISTINCT ?equipmentTypeLabel; SEPARATOR=", ") AS ?equipmentTypes)
WHERE {
  ?workshop a :MaintenanceWorkshop ;
            :hasLocationName ?workshopName .

  OPTIONAL {
    ?equipment :storedAt ?workshop ;
               a ?equipmentTypeClass .

    VALUES ?equipmentTypeClass {
      :Crane
      :Tool
      :SparePart
      :Consumable
      :TransportMean
      :Material
      :ConditionMonitoring
      :Control
      :EmergencyResponse
    }
    BIND(STRAFTER(STR(?equipmentTypeClass), "#") AS ?equipmentTypeLabel)
  }
}
GROUP BY ?workshopName
ORDER BY ?equipmentCount
LIMIT 5
```

Ambiguous SPARQL Result (Example):

workshopName	equipmentCount	equipmentTypes
WorkshopA	2	Tool, TransportMean
WorkshopB	3	Crane, Tool
WorkshopC	5	TransportMean, Consumable
...	...	...

Explanation of SPARQL Query:

This SPARQL query identifies `MaintenanceWorkshop` instances and their names. It then uses an `OPTIONAL` block to find equipment associated with activities performed at these workshops. The `FILTER` and `BIND` clauses are used to extract the simple class name of the equipment (e.g., 'Tool', 'Crane') from its full URI. The `GROUP BY` and `COUNT(DISTINCT ?equipment)` are used to count the number of unique equipment items associated with each workshop, providing a proxy for utilization. `GROUP_CONCAT` aggregates the types of equipment. The `ORDER BY ?equipmentCount ASC` and `LIMIT 5` are used to identify potentially

underutilized workshops (those with fewer associated equipment items). This query demonstrates SPARQL's aggregation capabilities and string manipulation for presenting more user-friendly results.

Cypher Query (Neo4j)

```
 MATCH (w:MaintenanceWorkshop)
OPTIONAL MATCH (e)-[:storedAt]->(w)
WHERE ANY(label IN labels(e) WHERE label IN [
  'Crane', 'Tool', 'SparePart', 'Consumable', 'TransportMean',
  'Material', 'ConditionMonitoring', 'Control', 'EmergencyResponse'
])
WITH w.hasLocationName AS workshopName,
     COLLECT(DISTINCT e) AS equipmentList,
     REDUCE(s = [], l IN labels(e) |
         CASE
             WHEN l IN ['Crane', 'Tool', 'SparePart', 'Consumable', 'TransportMean',
                        'Material', 'ConditionMonitoring', 'Control', 'EmergencyResponse']
             THEN s + l
             ELSE s
         END) AS filteredLabels
RETURN workshopName,
       SIZE(equipmentList) AS equipmentCount,
       REDUCE(output = "", label IN filteredLabels |
          output + CASE WHEN output = "" THEN label ELSE ", " + label END
       ) AS equipmentTypes
ORDER BY equipmentCount
LIMIT 5
```

Ambiguous Cypher Result (Example):

workshopName	equipmentCount	equipmentTypes
WorkshopA	2	["Tool", "Equipment"]
WorkshopB	3	["Crane", "Equipment"]
WorkshopC	5	["TransportMean", "Equipment"]
...	...	...

Comparison for CQ6:

For CQ6, both languages provide mechanisms for aggregation and grouping to assess utilization. Cypher's approach to identifying equipment types using `labels(e)` is more direct and idiomatic for property graphs, as types are often represented as labels. SPARQL requires more explicit filtering and string manipulation to extract class names from URIs. Both queries use `OPTIONAL MATCH` (Cypher) or `OPTIONAL` (SPARQL) to ensure that workshops with no associated equipment are still included in the results. The `COUNT(DISTINCT ...)` and `ORDER BY ... ASC` clauses are functionally equivalent in both languages for identifying underutilized workshops. Cypher's syntax for collecting distinct labels is arguably more concise than SPARQL's `GROUP_CONCAT` with string manipulation for this specific task. This comparison highlights how both languages can be used for analytical queries, with Cypher often offering a more streamlined syntax for property graph structures.

CQ7: Safety Protocol Compliance

Competency Question: "Verify that all tasks performed in 'Red' radiation classification areas are assigned to personnel with the appropriate safety certifications and that all required PPE is specified for the location."

This question requires checking multiple conditions across different entities and relationships to ensure compliance with safety protocols. It involves filtering, joining, and potentially identifying cases where compliance is *not* met.

SPARQL Query (GraphDB)

```
 PREFIX : <http://www.semanticweb.org/maintenance-ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?taskUID ?locationName ?personnelUID ?personnelCertification ?locationRequiresPPE
WHERE {
    ?task a :MaintenanceTask ;
          :hasTaskUID ?taskUID ;
          :isPerformedBy ?personnel .
    ?location a :Location ;
              :hasRadiationClassification :Red ;
              :hasLocationName ?locationName ;
              :requiresPPE ?locationRequiresPPE .
    ?personnel a :Personnel ;
               :worksAt ?location ;
               :hasPersonnelUID ?personnelUID .
    OPTIONAL { ?personnel :hasCertification ?personnelCertification . }

    FILTER (
        ?locationRequiresPPE = "true"^^xsd:boolean &&
        (BOUND(?personnelCertification) && ?personnelCertification = "Liquid metal")
    )
}
ORDER BY ?taskUID
```

Ambiguous SPARQL Result (Example):

taskUID	locationName	personnelUID	personnelCertification	locationRequiresPPE
Task001	RedZoneA	Pers001	Liquid metal	true
Task005	RedZoneB	Pers003	Liquid metal	true
...	...	...	...	...

Explanation of SPARQL Query:

This SPARQL query identifies tasks performed in locations with a `:Red` radiation classification. It then retrieves the associated personnel and checks if the location requires PPE and if the personnel have the necessary

certification (simulated as "Liquid metal" for demonstration, as the provided KG.ttl doesn't have explicit safety certifications for personnel beyond general areas of knowledge). The `FILTER` clause combines these conditions. This query demonstrates SPARQL's ability to enforce complex logical conditions across multiple entities to verify compliance.

Cypher Query (Neo4j)

```
 MATCH (t:MaintenanceTask)-[:isPerformedBy]->(p:Personnel)-[:worsAt]->(l:Location)-[:hasRadiationClassification]->
(rc:RadiationClassification)
WHERE rc.id = 'Red'
  AND l.requiresPPE = true
  AND p.hasCertification = 'Liquid metal'
RETURN t.hasTaskUID AS taskUID, l.hasLocationName AS locationName, p.hasPersonnelUID AS personnelUID, p.hasCertification
AS personnelCertification, l.requiresPPE AS locationRequiresPPE
ORDER BY taskUID
```

Ambiguous Cypher Result (Example):

taskUID	locationName	personnelUID	personnelCertification	locationRequiresPPE
Task001	RedZoneA	Pers001	Liquid metal	true
Task005	RedZoneB	Pers003	Liquid metal	true
...	...	...	...	...

Comparison for CQ7:

For CQ7, both SPARQL and Cypher are capable of expressing the complex conditions required for safety protocol compliance. Cypher's pattern matching again provides a clear and concise way to link tasks, locations, and personnel. The `WHERE` clause in Cypher directly combines the conditions on location radiation, PPE requirement, and personnel certification. SPARQL achieves this through a series of triple patterns and a combined `FILTER` clause. The main difference lies in the syntax and how

relationships are expressed. Cypher's graph-centric syntax often leads to more readable queries for complex graph patterns, while SPARQL's explicit triple patterns offer a standardized and precise way to define relationships in an RDF context. Both languages are equally powerful in performing these compliance checks, and the choice often depends on the underlying graph database and the user's familiarity with the syntax.

CQ8: Cross-Disciplinary Task Identification

Competency Question: "Find tasks that require expertise from at least two different areas of knowledge (e.g., 'Mechanics' and 'Electricity, Electronics, including RF') and identify the personnel capable of performing such tasks."

This question requires identifying tasks that are associated with personnel having diverse skill sets. This will test the ability to query for multiple property values for a single entity and then link those entities to tasks.

SPARQL Query (GraphDB)

```
 PREFIX : <http://www.semanticweb.org/maintenance-ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?taskUID ?personnelUID ?areaOfKnowledge1 ?areaOfKnowledge2
WHERE {
    ?task a :MaintenanceTask ;
          :hasTaskUID ?taskUID ;
          :isPerformedBy ?personnel .
    ?personnel a :Personnel ;
               :hasPersonnelUID ?personnelUID ;
               :hasAreaOfKnowledge ?areaOfKnowledge1 ;
               :hasAreaOfKnowledge ?areaOfKnowledge2 .
    FILTER (?areaOfKnowledge1 != ?areaOfKnowledge2) .
    FILTER (
        (?areaOfKnowledge1 = "Mechanics" && ?areaOfKnowledge2 = "Electricity, Electronics, including RF") ||
        (?areaOfKnowledge1 = "Electricity, Electronics, including RF" && ?areaOfKnowledge2 = "Mechanics")
    )
}
ORDER BY ?taskUID ?personnelUID
```

Ambiguous SPARQL Result (Example):

taskUID	personnelUID	areaOfKnowledge1	areaOfKnowledge2
Task010	Pers005	Mechanics	Electricity, Electronics, including RF
Task025	Pers011	Electricity, Electronics, including RF	Mechanics
...	...	...	...

Explanation of SPARQL Query:

This SPARQL query identifies `MaintenanceTask` instances and the `Personnel` performing them. It then looks for personnel who possess at least two different `hasAreaOfKnowledge` values. The `FILTER` clauses ensure that the two identified areas of knowledge are distinct and match the specified cross-disciplinary requirements (e.g., 'Mechanics' and 'Electricity, Electronics, including RF'). This query demonstrates SPARQL's ability to handle multiple values for a single property and apply complex logical conditions to identify specific combinations of expertise. The use of `DISTINCT` ensures that each unique task-personnel combination is listed only once.

Cypher Query (Neo4j)

```
 MATCH (t:MaintenanceTask)-[:isPerformedBy]->(p:Personnel)
WHERE (
    ("Mechanics" IN p.hasAreaOfKnowledge AND "Electricity, Electronics, including RF" IN p.hasAreaOfKnowledge)
    OR
    ("Electricity, Electronics, including RF" IN p.hasAreaOfKnowledge AND "Mechanics" IN p.hasAreaOfKnowledge)
)
RETURN t.hasTaskUID AS taskUID, p.hasPersonnelUID AS personnelUID, p.hasAreaOfKnowledge AS areasOfKnowledge
ORDER BY taskUID, personnelUID
```

Ambiguous Cypher Result (Example):

taskUID	personnelUID	areasOfKnowledge
Task010	Pers005	["Mechanics", "Electricity, Electronics, including RF"]
Task025	Pers011	["Electricity, Electronics, including RF", "Mechanics"]
...	...	...

Comparison for CQ8:

For CQ8, both SPARQL and Cypher effectively identify tasks requiring cross-disciplinary expertise. Cypher, with its ability to directly query properties that hold multiple values (like `p.hasAreaOfKnowledge` being a list), offers a more concise way to express the condition of having multiple areas of knowledge using the `IN` operator. SPARQL achieves this by binding the same property to different variables and then filtering for distinct values. While both are capable, Cypher's list-based property handling can be more elegant for this type of multi-value attribute querying. The readability of Cypher's `IN` operator for checking membership in a collection of values is a clear advantage here.

CQ9: Anomaly Detection in Task Durations

Competency Question: "Identify maintenance tasks whose actual duration significantly deviates from their estimated duration compared to their designated Maintenance Activity, considering the handling method (e.g., 'Hands-off' vs. 'Hands-on' vs. 'RH') and location radiation levels."

This question requires comparing two numerical values (actual vs. estimated overall activity duration) and applying conditional logic based on categorical properties (handling method, radiation level). This highlights the numerical and conditional processing capabilities of the query languages. However this case will never happen because we have already assured that the task duration should never exceed the duration of the activity in which is part of by the previously defined rules. We have enabled it in this example just to show the capabilities of our model's Knowledge extraction

SPARQL Query (GraphDB)

```
 PREFIX : <http://www.semanticweb.org/maintenance-ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ns: <http://www.semanticweb.org/maintenance-ontology#>

SELECT ?taskUID ?taskDescription ?actualDuration ?activityDuration ?handlingMethod ?radiationClassification
WHERE {
  ?task a :MaintenanceTask ;
        :hasTaskUID ?taskUID ;
        :hasTaskDescription ?taskDescription ;
        :hasTaskDuration ?actualDuration ;
        :hasHandlingMethod ?handlingMethod ;
        :isPartOf ?activity ;
        :performedAt ?location .

  ?activity a :MaintenanceActivity ;
            :hasMaintenanceDuration ?activityDuration .

  ?location :hasRadiationClassification ?radiationClassification.

  FILTER (xsd:float(?actualDuration) > xsd:float(?activityDuration)) .
  FILTER (?handlingMethod = ns:HandsOff || ?handlingMethod = ns:HandsOn || ?handlingMethod = ns:RH) .
}
ORDER BY ?taskUID
```

Ambiguous SPARQL Result (Example):

taskUID	taskDescription	actualDuration	MaintenanceDuration	handlingMethod	radiationClassification
Task007	Inspect Valve	17.28	16.3	Hands-off	Green
Task015	Replace Component	19.12	12.7	Hands-on	Orange
...	...	...	...	...	...

Explanation of SPARQL Query:

This SPARQL query retrieves tasks along with their actual and estimated maximum durations, handling method, and the radiation classification of their location. The core of the anomaly detection is within the `FILTER` clause: `ABS(?actualDuration > ?`

`MaintenanceDuration)` identifies tasks where the aactual duration exceeds the maximum furation on the activity they are part of. Additional filters are applied to consider specific handling methods. This demonstrates SPARQL's ability to perform arithmetic operations and complex conditional filtering, which is essential for data analysis and anomaly detection within the knowledge graph. The query effectively combines numerical comparison with categorical filtering to pinpoint tasks that might warrant further investigation due to significant time discrepancies.

Cypher Query (Neo4j)

```
MATCH (task:MaintenanceTask)-[:isPartOf]->(activity:MaintenanceActivity),
      (task)-[:performedAt]->(location)-[:hasRadiationClassification]->(radiationClassification),
      (task)-[:hasHandlingMethod]->(rc:HandlingMethod)
WHERE toFloat(task.hasTaskDuration) > toFloat(activity.hasMaintenanceDuration)
  AND (rc.id IN ['HandsOff', 'HandsOn', 'RH'])
RETURN task.hasTaskUID AS taskUID,
       task.hasTaskDescription AS taskDescription,
       task.hasTaskDuration AS actualDuration,
       activity.hasMaintenanceDuration AS activityDuration,
       rc.id AS handlingMethod,
       radiationClassification.uri AS radiationClassification
ORDER BY taskUID
```

Ambiguous Cypher Result (Example):

taskUID	taskDescription	actualDuration	MaintenanceDuration	handlingMethod	radiationClassification
Task007	Inspect Valve	150	100	Hands-off	Green
Task015	Replace Component	200	250	Hands-on	Orange
...	...	...	...	...	...

Comparison for CQ9:

For CQ9, both SPARQL and Cypher demonstrate strong capabilities in numerical comparison and conditional filtering. Cypher's syntax for arithmetic operations and the `ABS` function is straightforward and directly integrated into the `WHERE` clause. Similarly, SPARQL uses standard arithmetic operators and `FILTER` for complex conditions. The `IN` operator in Cypher (`t.hasHandlingMethod IN ["Hands-off", "Hands-on", "RH"]`) provides a concise way to check for multiple categorical values, which in SPARQL would typically involve multiple `||` (OR) conditions. Both languages are well-suited for this type of analytical query, allowing for the identification of anomalies based on quantitative and qualitative data. The choice between them often comes down to the specific features and optimizations offered by the underlying graph database and the preference for a more declarative (SPARQL) versus pattern-matching (Cypher) syntax.

CQ10: Resource Allocation Optimization

Competency Question: "For a set of high-priority maintenance activities, suggest an optimal allocation of available personnel and equipment to minimize overall completion time, taking into account their respective skills, certifications, and current assignments."

This is the most complex competency question, as it involves optimization, which is typically beyond the scope of standard graph query languages. True optimization requires external algorithms and potentially iterative processing. However, both SPARQL and Cypher can be used to extract the necessary data to feed into an optimization algorithm. For this example, we will focus on identifying high-priority activities and listing available resources that could be allocated, demonstrating how the query languages can prepare data for such an optimization problem.

SPARQL Query (GraphDB)

```
 PREFIX : <http://www.semanticweb.org/maintenance-ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?activityUID ?activityName ?requirement ?availablePersonnelUID ?availablePersonnelSkill ?
availableEquipmentUID ?equipmentType
WHERE {
    ?activity a :MaintenanceActivity ;
              :hasMaintenanceActivityUID ?activityUID ;
              :hasMaintenanceActivityName ?activityName .

    OPTIONAL {
        ?activity :requiresSkill ?requiredSkill .
    }

    OPTIONAL {
        ?availablePersonnel a :Personnel ;
                            :hasPersonnelUID ?availablePersonnelUID ;
                            :hasPersonnelAvailability "true"^^xsd:boolean ;
                            :hasAreaOfKnowledge ?availablePersonnelSkill .
        FILTER NOT EXISTS {
            ?availablePersonnel :assignedToActivity ?otherActivity .
            FILTER (?otherActivity != ?activity) .
        }
    }

    OPTIONAL {
        ?availableEquipment a :MaintenanceMeans ;
                            :hasMeansname ?availableEquipmentUID ;
                            a ?equipmentTypeClass .
        FILTER (STRSTARTS(STR(?equipmentTypeClass), STR(:)) && ?equipmentTypeClass != :MaintenanceMeans)
        BIND (REPLACE(STR(?equipmentTypeClass), STR(:), "") AS ?equipmentType)
        FILTER NOT EXISTS {
            ?availableEquipment :assignedToActivity ?otherActivity .
            FILTER (?otherActivity != ?activity) .
        }
    }
}
ORDER BY ?activityUID
```

Ambiguous SPARQL Result (Example):

activityUID	activityName	requirement	availablePersonnelUID	availablePersonnelSkill	availableEquipmentUID	equipmentTy
Act001	CriticalFix	Mechanics	Pers005	Mechanics	Crane001	Crane
Act001	CriticalFix	Mechanics	Pers008	Electricity		
Act002	UrgentRepair	RH	Pers010	RH	ToolSetA	Tool
...	...	...	...	...	...	...

Explanation of SPARQL Query:

This SPARQL query focuses on extracting all relevant data for high-priority maintenance activities. It identifies activitiesvand then attempts to find available personnel and equipment. For personnel and equipment, it checks for `hasPersonnelAvailability` and ensures they are not currently `assignedToActivity` for other tasks (a simplified availability check). It also retrieves the `requirement` for the activity and the `hasAreaOfKnowledge` for personnel, and the `equipmentType`. The `OPTIONAL` clauses are crucial here, as not all activities will have specific skill requirements, and not all personnel/equipment will be available. This query demonstrates SPARQL's ability to gather a wide range of interconnected data, which can then be used as input for an external optimization engine. While SPARQL cannot perform the optimization itself, it is highly effective at preparing the dataset for such advanced analytical tasks.

Cypher Query (Neo4j)

```
 MATCH (activity:MaintenanceActivity)
OPTIONAL MATCH (activity)-[:hasMaintenanceActivityUID]->(activityUIDNode)
OPTIONAL MATCH (activity)-[:hasMaintenanceActivityName]->(activityNameNode)
WITH activity, activityUIDNode.value AS activityUID, activityNameNode.value AS activityName

// Optional required skill
OPTIONAL MATCH (activity)-[:requiresSkill]->(requiredSkill)
WITH activity, activityUID, activityName, requiredSkill

// Optional available personnel
OPTIONAL MATCH (availablePersonnel:Personnel)
WHERE availablePersonnel.hasPersonnelAvailability = true
  AND NOT EXISTS {
    MATCH (availablePersonnel)-[:assignedToActivity]->(otherActivity)
    WHERE otherActivity <> activity
  }
WITH activity, activityUID, activityName, requiredSkill,
     availablePersonnel.hasPersonnelUID AS availablePersonnelUID,
     availablePersonnel.hasAreaOfKnowledge AS availablePersonnelSkill

// Optional available equipment
OPTIONAL MATCH (availableEquipment:MaintenanceMeans)
WHERE NOT EXISTS {
  MATCH (availableEquipment)-[:assignedToActivity]->(otherActivity)
  WHERE otherActivity <> activity
}
WITH activity, activityUID, activityName, requiredSkill,
     availablePersonnelUID, availablePersonnelSkill,
     availableEquipment.hasMeansname AS availableEquipmentUID,
     labels(availableEquipment) AS equipmentLabels

// Filter out superclass "MaintenanceMeans" from labels
WITH activityUID, activityName, requiredSkill, availablePersonnelUID, availablePersonnelSkill,
     availableEquipmentUID,
     [label IN equipmentLabels WHERE label <> 'MaintenanceMeans'] AS specificLabels

// Assume only one specific type per equipment
WITH activityUID, activityName, requiredSkill, availablePersonnelUID, availablePersonnelSkill,
     availableEquipmentUID,
     CASE WHEN size(specificLabels) > 0 THEN head(specificLabels) ELSE NULL END AS equipmentType

RETURN DISTINCT activityUID, activityName, requiredSkill,
       availablePersonnelUID, availablePersonnelSkill,
       availableEquipmentUID, equipmentType
ORDER BY activityUID
```

Ambiguous Cypher Result (Example):

activityUID	activityName	requirement	availablePersonnelUID	availablePersonnelSkill	availableEquipmentUID	equipmentTy
Act001	CriticalFix	Mechanics	Pers005	["Mechanics"]	Crane001	["Crane", "Equipment"]
Act001	CriticalFix	Mechanics	Pers008	["Electricity"]		
Act002	UrgentRepair	RH	Pers010	["RH"]	ToolSetA	["Tool", "Equipment"]
...	...	...	...	...	...	...

Comparison for CQ10:

For CQ10, both SPARQL and Cypher serve as powerful data extraction tools to feed external optimization algorithms. Cypher's pattern matching, especially with `OPTIONAL MATCH` and `NOT (p)-[:assignedToActivity]->(:MaintenanceActivity)` for checking availability, provides a clear and concise way to express the complex conditions for resource allocation. SPARQL achieves similar results through `OPTIONAL` clauses and `FILTER NOT EXISTS`. The main difference lies in the expressiveness of their graph patterns. Cypher's visual syntax often makes it easier to conceptualize the relationships and constraints, particularly when dealing with the absence of certain relationships (e.g., not assigned to an activity). Both languages are equally capable of gathering the necessary data for optimization, but neither can perform the optimization itself. The choice between them for this type of complex data preparation often depends on the existing infrastructure and the developer's familiarity with the respective query paradigms.

4. Conclusion on Query Language Comparison

This comparative analysis of SPARQL and Cypher for querying a real-world industrial maintenance knowledge graph reveals that both languages are highly capable of extracting complex information, traversing relationships, and applying intricate filtering conditions. Their strengths, however, lie in different areas, largely influenced by their underlying data models:

- **SPARQL (RDF-centric):** Excels in its semantic richness and adherence to W3C standards. Its triple-based structure provides a highly flexible and extensible way to represent knowledge, making it ideal for scenarios requiring strong semantic interoperability and reasoning. While its syntax can be more verbose due to explicit URI usage, it offers powerful features like property paths and federation capabilities across distributed data sources. SPARQL's strength lies in its declarative nature, allowing users to describe what they want to retrieve rather than how to retrieve it.
- **Cypher (Property Graph-centric):** Shines in its intuitive, ASCII-art-like syntax for pattern matching, which often makes complex graph traversals more readable and easier to write for those accustomed to graph visualization. Its focus on nodes, relationships, and properties directly maps to the way many users conceptualize graph data. Cypher is particularly strong for local graph traversals and when the graph structure is relatively stable. Its procedural elements and support for graph algorithms (often through extensions like APOC in Neo4j) can provide powerful analytical capabilities.

In essence, the choice between SPARQL and Cypher often depends on the specific requirements of the project, the nature of the data, and the existing technology stack. For applications demanding strict semantic adherence, interoperability with other RDF data, and advanced reasoning capabilities, SPARQL and RDF databases like GraphDB are often preferred like in our case. For applications where the primary focus is on intuitive graph traversal, visual pattern matching, and native graph algorithm execution within a property graph model, Cypher and databases like Neo4j offer a compelling solution. Both are indispensable tools in the knowledge graph ecosystem, each offering unique advantages for different facets of data querying and analysis.