JURNAL ELTEK
Vol. 24, No. 2, October 2026, pp. 111~120
p-ISSN: 1693 – 4024 | e-ISSN: 2355-0740
DOI: 10.33795/eltek.v24i2.10896

A comparative study of property-to-label and property-to-node refactoring in ontology-based graph data modeling

Ade Hodijah[1], Sri Ratna Wulan[2], Jonner Hutahaean[3], Rahil Jumiyani[4]
e-mail: adehodijah@jtk.polban.ac.id, sri.ratna@polban.ac.id, jonnerh@jtk.polban.ac.id, rahil@jtk.polban.ac.id
[1,2,3,4]Department of Informatics and Computer Engineering, Politeknik Negeri Bandung, Indonesia

Article Information

Article History

Received August 9, 2026
Revised August 20, 2026
Published October 30, 2026

***Keywords*:**

Protégé
Neo4j
Neosemantics
Owlready2

ABSTRACT

Integrating Web Ontology Language with Neo4j through the neosemantics (n10s) framework enables semantic validation and automated reasoning that are not natively supported by Labeled Property Graphs. However, the choice of graph refactoring pattern influences query performance. This study compares two ontology-based graph refactoring approaches, namely Property-to-Label (P2L) and Property-to-Node (P2N), implemented in Protégé and integrated into Neo4j using n10s and Owlready2. Both models were evaluated for semantic validation and ontology reasoning using the PET ontology, while their query execution performance was benchmarked using 78 Cypher query variants derived from the IMDB dataset. The evaluation examined the relationship between execution time and reductions in `WHERE` clauses following schema refactoring. The results show that both approaches successfully enforce ontology constraints and infer missing classifications from incomplete data. Performance analysis indicates that both P2L and P2N improved query execution time, with the highest reduction percentage was 89% for queries with four total number of the `WHERE`. P2L exhibited a more consistent relationship between `WHERE` clause reduction and execution time improvement due to Neo4j's native label indexing.

Corresponding Author:
Ade Hodijah,
Department of Informatics and Computer Engineering,
Bandung State of Polytechnic,
Jl. Gegerkalong Hilir, Bandung, West Java, Indonesia.
Email: ade.hodijah@polban.ac.id

1. INTRODUCTION

Graph databases, particularly Neo4j, have emerged as the industry standard for managing highly interconnected and complex datasets [1], [2]. However, a major structural limitation of Labeled Property Graphs (LPGs) is their lack of native semantic validation mechanisms at the database level, where some researches has conducted to optimize the query [3], [4], and others doing the transformation between RDF to property graph [5], [6] with combining the ontology layer [7], [8], but without implementing the refactoring pattern of graph-based schmea in the semantic layer of ontology. In a standard Neo4j schema, relations are defined structurally rather than semantically, meaning any arbitrary relation such as `:Owner -[:OWNS]-> :Animal` can be created without semantic restrictions [3]. A typical structural modeling layout under a standard LPG environment is illustrated in Figure 1, demonstrating an unconstrained relationship instance.

This architectural limitation poses a significant challenge when business rules require strict data compliance, such as restricting the `:OWNS` relationship exclusively to animals categorized as "`Pet`" while prohibiting it for "`Wild`" animals [9]. Implementing such dynamic, property-based constraints natively in Neo4j usually requires complex application-level logic or custom Cypher transaction triggers, which are difficult to maintain [1]. Furthermore, graph databases cannot perform automated reasoning to infer missing classifications when faced with incomplete open-world data [3], [10]. To bridge this gap, integrating Web Ontology Language (OWL) schemas through the neosemantics (n10s) framework allows Neo4j to leverage semantic validation and

description logic (DL) reasoning [10], [11]. Within this integration framework, two primary ontology modeling and graph refactoring patterns can be applied: Property-to-Label (P2L) and Property-to-Node (P2N) [12]. The P2L approach maps properties directly to ontological classes/labels, whereas the P2N approach reifies properties into distinct structural nodes within the graph [13].

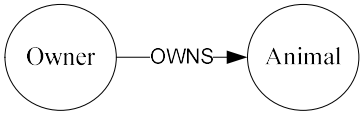


Figure 1. Sample of data modeling using Neo4j as the baseline data model

The simple Pet ontology [9] is used to elaborate the idea of P2L and P2N. A person may have a pet (`hasPet`). The subject of the `hasPet` relationship is of type `PetOwner`, while the object is of type `PetAnimal`. The `hasPet` relationship can be considered a subproperty of the `owns` relationship. In addition to `PetAnimal`, there is also the class `WildAnimal`. A `PetAnimal` cannot be a `WildAnimal`. Cats and dogs are both subclasses of `PetAnimal`. Both `PetAnimal` and `WildAnimal` are subclasses of `Animal`. Furthermore, any person who `owns` something of type `PetAnimal` can also be classified as a `PetOwner`. This study compares both modeling approaches to evaluate their ability to infer that an unclassified individual "Tom" is a Pet based solely on the incoming assertion `:OWNS` Tom from an owner [14], [9]. Finally, this paper evaluates the performance trade-offs of both models, analyzing how these ontological structures impact Cypher query execution times when imported back into Neo4j [12].

2. RESEARCH METHOD

The methodology of this study employs a rigorous pipeline encompassing ontology design, semantic mapping, and empirical database benchmarking. The process begins with ontology modeling in Protégé, followed by semantic validation and reasoning using the PET ontology, and concludes with separate performance benchmarking using Cypher queries derived from the IMDB dataset.

There are three competency questions [15], [9]: (a) Which individuals are cats?; (b) Which individuals are pets?; (c) Who are the pet owners?. The individual (ABox) data as follows: (a) MammyTwoShoes has a pet named Tom; (b) Garfield is a cat (pet animal); (c) Simba is a lion (wild animal); (d) Tom without any type information. These competency questions were used for semantic validation and reasoning, rather than for performance benchmarking. The conceptual breakdown and sequential stages of this framework are mapped in the architectural pipeline of semantic integration shown in Figure 2.

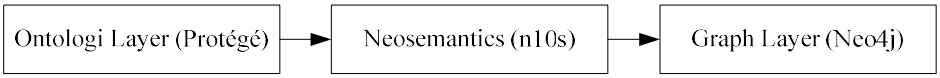


Figure 2. Architectural pipeline of semantic integration (ontology) using Neo4j and neosemantics (n10s) [16] and owlReady2 [17]

In the ontology layer, only four datasets (individuals or ABox) used in order to consistency testing of the TBox and RBox using HermiT reasoner in Protégé. Based on the rule of importing RDF data into Neo4j using n10s pluging [18], `MammyTwoShoes`, `Garfield`, `Simba`, and `Tom` saved as the node in Neo4j, `hasPet` (object property) as the relationship in Neo4j, while the `PetAnimal` categorization was determined by data property (P2L) or sub-class (P2N). Table 1 shows P2L and P2N comparison in ontology layer and graph layer using n10s. The same idea implemented for the question of "Which individuals are pets?". However, for the question of "Who are the pet owners?", the reasoner executed by the definition in the TBox for both P2L and P2N.

Table 1. Neosemantics perspective for question "Which individuals are cats?"

Aspect	P2L	P2N
Species assertion	Uses datatype property `<species>`	Uses `rdf:type` referencing Classes (`Cat`/`Lion`)
Representation in Neo4j	Node Property (`n.species = 'cat'`)	Node Label (`:Cat`, `:Lion`)
Cypher Query Execution	Requires property filtering/scanning: `MATCH (n) WHERE n.species = 'cat'`	Uses native Neo4j Label matching `MATCH (n:Cat)`
`hasPet` Relationship	Mapped as Relationship `-[:hasPet]->`	Mapped as Relationship `-[:hasPet]->`

Based on the neosemantics mechanism, the dataset of IMDB in [4] used for performance testing with focusing on the query having `WHERE` operations with only using an `AND` as the logical operator. The initial 300 queries were classified into 10 groups, resulting in 190 queries after removing redundant patterns. Of these, 110 queries containing `OR` operators and Disjunctive Normal Form (DNF) patterns were excluded, while 2 additional queries were excluded because they exceeded the 10-minute timeout. Consequently, 78 queries were successfully executed across the baseline, P2L, and P2N schemas. The total of the combination of the `AND` operator is four: (1) `A`; (2) `A AND B`; (3) `A AND B AND C`; (4) `A AND B AND C AND D`, where `'A', 'B', 'C',` and `'D'` is the property or attributes used in the query of `WHERE`. Table 2. shows some queries used for experimentation. The experimental tested in a single server environment using a PC with Intel (R) Core (TM) specifications i5-8250U CPU @ 1.60GHz 1.80 GHz uses 12.0 GB RAM and uses the operating system Microsoft Windows 10.

Table 2. Sample of baseline queries for two, three, and four `AND` operator combination using IMDB dataset [4] for performance testing

`WHERE` Number	Logical Operator	Sample Baseline Query (IMDB) [4]	Cypher for Experimentation (Pet) for P2N model (Table 10)
1	`A`	`MATCH (a: Adventure) WHERE a.category = 'movie'`	`MATCH(n1:Animal)-[:HAS_SPECIES]->(n2:Cat)`
2	`(A AND B)`	`MATCH (n1:actress)-[:PRODUCED]->(n3:romance), (n1:soundtrack)-[:ACTED_IN]->(n4:thriller), (n1:soundtrack)[:ACTED_IN]->(n2:drama) WHERE (n4.category="tvMovie" AND n2.category="movie")`	`MATCH (n1:Animal)-[:HAS_SPECIES]->(n2:Cat), (n1:Animal)-[:HAS_OWNER]->(n3:John)`
3	`(A AND B AND C)`	`MATCH (n3:music_department)-[:COMPOSED]->(n2:documentary), (n3:music_department)-[:COMPOSED]->(n1:documentary) WHERE (n3.name="Philip Glass" AND n3.birth=1937 AND n1.category="video")`	`MATCH (n1:Animal)-[:HAS_SPECIES]->(n2:Cat), (n1:Animal)-[:HAS_OWNER]->(n3:John), (n1:Animal)-[:HAS_FOOD]->(n4:Fish)`
4	`(A AND B AND C AND D)`	`MATCH (n1:actor)-[:ACTED_IN]->(n2:romance), (n1:actor)-[:ACTED_IN]->(n4:war), (n1:actor)-[:ACTED_IN]->(n3:comedy) WHERE (n3.category="movie" AND n1.birth=1895 AND n3.year=1927 AND n4.year=1929)`	`MATCH (n1:Animal)-[:HAS_SPECIES]->(n2:Cat), (n1:Animal)-[:HAS_OWNER]->(n3:John), (n1:Animal)-[:HAS_FOOD]->(n4:Fish), (n1:Animal)-[:HAS_BREED]->(n4:Persian)`

Table 3 shows how Node Labels used to boost the cypher performance [12] from the Relationship Types. For example, `MATCH (n:Animal) WHERE n.species = 'cat'` for the first of P2L model (Table 8) and `MATCH (n:Cat)` for the second of P2L model (Table 9) have one the `AND` operator. In [4], sample query which has one the `AND` operator is `MATCH (a: Adventure) WHERE a.category = 'movie'` (Table 2). The relationship types of `HAS_PET` was not used in the performance bencmarking testing, but it was used in the logical validation and inference testing.

Table 3. Cypher for experimentation of P2L model by using the cypher of the P2N model (Table 10)

WHERE Number	Logical Operator	Cypher for the First P2L Model	Cypher for the Second of P2L Model
1	A	`MATCH (n:Animal) WHERE n.species = 'cat'`	`MATCH (n:Animal:Cat:Pet)`
2	(A AND B)	`MATCH (n:Animal) WHERE n.species = 'cat' AND n.owner = 'John'`	`MATCH (n:Animal:Cat:Pet:John)`
3	(A AND B AND C)	`MATCH (n:Animal) WHERE n.species = 'cat' AND n.owner = 'John' AND n.food = 'Fish'`	`MATCH (n:Animal:Cat:Pet:John:Fish)`
4	(A AND B AND C AND D)	`MATCH (n:Animal) WHERE n.species = 'cat' AND n.owner = 'John' AND n.food = 'Fish' AND n.breed = 'Persian'`	`MATCH (n:Animal:Cat:Pet:John:Fish: Persian)`

2.1. Ontological Design in Protégé

Both transformation models were designed in Protégé to explicitly mirror Neo4j's classic graph refactoring patterns [12], [13]. In production graph database engineering, structural data refactoring is heavily driven by the performance requirements of frequent workloads and query paths, which similar to competency question in ontology [15]. According to the Neo4j refactoring paradigm, optimization becomes highly critical because the Cypher queries for specific use cases often do not perform optimally, especially when the graph scales upward. Therefore, embedding these performance-oriented refactoring patterns into the ontological layer at the design phase ensures that the resulting graph transformation satisfies both formal description logic constraints and high-throughput production execution benchmarks.

For Property-to-Label refactoring or (the first P2L model), the property value "`Pet`" or "`Wild`" is transformed directly into an OWL class hierarchy, as seen in Table 5. Meanwhile, for the second P2L model, property values are reified into standalone entity nodes to decouple classification from the core hierarchy. The ontology defines a helper class `Animal` containing the individuals `Pet` and `Wild`, as seen in Table 6. An equivalent class `Animal` classified as `Pet` or `Wild` is then constructed using the existential restriction for both model (P2L and P2N), which are the definition of `Owner` and disjointness. However, for Property-to-Node Refactoring or P2N (Table 4), two additional object properties (`hasCatSpecies` and `hasLionSpecies`) and two additional individuals (`CatIndividuals` and `LionIndividuals`) are introduced as part of the "property becomes node" refactoring for Neo4j.

Table 4. TBox, RBox, and ABox for P2N model with sample of Pet was a cat and Wild was a lion

Description Logic	Type	Definition
Pet ⊑ Animal	TBox	`Pet` is a subclass of the class `Animal`.
Wild ⊑ Animal	TBox	`Wild` is a subclass of `Animal`.
Pet ⊓ Wild ⊑⊥	TBox	No animal can be both a `Pet` and a `Wild` individual.
Owner ≡ ∃hasPet. Pet	TBox	A person is classified as an Owner if and only if they have at least one `hasPet` relationship with an instance of the class `Pet`. No animal can be both a `Pet` and a `Wild` individual.
Wild ≡ ∃ hasLionSpecies.Lion	TBox	A `Wild` is exactly a thing that has at least one `Lion` species
CatIndividuals	ABox	`CatIndividuals` is declared as an instance of the `Cat` class.
LionIndividuals	ABox	`LionIndividuals` is declared as an instance of the `Lion` class.
Garfield: Cat	ABox	`Garfield` is declared as an instance of the `Cat` class.
Simba: Lion	ABox	`Simba` is declared to be an instance of the `Lion` class.
(MammyTwoShoes, Tom): hasP	ABox	`MammyTwoShoes` has the `hasPet` relationship with `Tom`.
Tom	ABox	`Tom` is an individual classified by the `hasPet` relationship.

Description Logic	Type	Definition
Domain(hasPet) = Owner	RBox	Every individual that has the `hasPet` relationship is inferred to be a member of the `Owner` class.
Range(hasPet) = Pet	RBox	Every individual that is the object of a `hasPet` relationship is inferred to be a member of the `Pet` class.
Domain(hasCatSpecies) = Animal	RBox	Every individual that has the `hasCatSpecies` relationship is inferred to be a member of the `Animal` class.
Range(hasCatSpecies) = Cat	RBox	Every individual that is the object of a `hasCatSpecies` relationship is inferred to be a member of the `Cat` class.
Domain(hasLionSpecies) = Animal	RBox	Every individual that has the `hasLionSpecies` relationship is inferred to be a member of the `Animal` class.
Range(hasLionSpecies) = Lion	RBox	Every individual that is the object of a `hasLionSpecies` relationship is inferred to be a member of the `Lion` class.

Table 5. TBox, RBox, and ABox for the first P2L model with sample of Pet was a cat and Wild was a lion

Description Logic	Type	Definition
Pet ⊑ Animal	TBox	`Pet` is a subclass of the class `Animal`.
Wild ⊑ Animal	TBox	`Wild` is a subclass of `Animal`.
Pet ⊓ Wild ⊑⊥	TBox	No animal can be both a `Pet` and a `Wild` individual.
Owner ≡ ∃hasPet. Pet	TBox	A person is classified as an Owner if and only if they have at least one `hasPet` relationship with an instance of the class `Pet`. No animal can be both a `Pet` and a `Wild` individual.
Pet ≡ Animal ⊓ ∃species. {"cat"}	TBox	The `Pet` class is defined as the class of animals whose `species` property has the value "`cat`".
Wild ≡ Animal ⊓ ∃species. {"lion"}	TBox	The `Wild` class is defined as the class of animals whose `species` property has the value "`lion`".
Garfield: Cat	ABox	`Garfield` is declared as an instance of the `Cat` class.
Simba: Lion	ABox	`Simba` is declared to be an instance of the `Lion` class.
(MammyTwoShoes, Tom): hasP	ABox	`MammyTwoShoes` has the `hasPet` relationship with `Tom`.
Tom	ABox	`Tom` is an individual classified by the `hasPet` relationship.
Domain(hasPet) = Owner	RBox	Every individual that has the `hasPet` relationship is inferred to be a member of the `Owner` class.
Range(hasPet) = Pet	RBox	Every individual that is the object of a `hasPet` relationship is inferred to be a member of the `Pet` class.
Domain(species) = Animal	RBox	Every individual that has the `species` data property is inferred to be a member of the `Animal` class.
Range(species) = xsd:string	RBox	The value of a `species` must be a string.

Table 6. TBox, RBox, and ABox for the second P2L model with sample of Pet was a cat and Wild was a lion

Description Logic	Type	Definition
Cat ⊑ Pet	TBox	`Cat` is a subclass of the `Pet` class.
Lion ⊑ Wild	TBox	`Lion` is a subclass of the `Wild` class.
Pet ⊑ Animal	TBox	`Pet` is a subclass of the class `Animal`.
Wild ⊑ Animal	TBox	`Wild` is a subclass of `Animal`.
Pet ⊓ Wild ⊑⊥	TBox	No animal can be both a `Pet` and a `Wild` individual.
Owner ≡ ∃hasPet. Pet	TBox	A person is classified as an Owner if and only if they have at least one `hasPet` relationship with an instance of the class `Pet`. No animal can be both a `Pet` and a `Wild` individual.
Garfield: Cat	ABox	`Garfield` is declared as an instance of the `Cat` class.
Simba: Lion	ABox	`Simba` is declared to be an instance of the `Lion` class.
(MammyTwoShoes, Tom): hasP	ABox	`MammyTwoShoes` has the `hasPet` relationship with `Tom`.
Tom	ABox	`Tom` is an individual classified by the `hasPet` relationship.
Domain(hasPet) = Owner	RBox	Every individual that has the `hasPet` relationship is inferred to be a member of the `Owner` class.

Range(hasPet) = Pet	RBox	Every individual that is the object of a `hasPet` relationship is inferred to be a member of the `Pet` class.

The TBox defines concepts (classes) and the hierarchical relationships among classes, while the RBox defines the characteristics of properties (roles), and the ABox contains assertions about specific individuals in the ontology [14].

2.2. Integration via Neosemantics (n10s)

The ontologies designed in Protégé were exported as RDF Turtle (.ttl) files and imported into Neo4j using the neosemantics (n10s) plugin [16]. During the import phase, the n10s validation engine was configured to listen to transaction commits and block any write operations that violate the OWL constraints [17]. For P2L model, RDF types are converted directly into native Neo4j node labels, mapping the ontology class `Pet` to the node label `:Pet`, for P2N model, the import engine generates a multi-hop path topology where each animal node is linked via a `:hasPet` relationship and `species` data property to a dedicated classification node.

2.3. Performance Testing Setup

To empirically evaluate the impact of the Property-to-Label (P2L) and Property-to-Node (P2N) graph refactoring patterns under 78 queries run in Neo4j, a comprehensive benchmarking framework was established. The evaluation matrix is structured into four primary performance dimensions, designed to measure query execution efficiency via structural traversal overhead [1]. The execution times of the 78 Cypher query variations derived from the IMDB dataset were recorded over 10 iterations to obtain average execution times [12], [19], [20]. Table 7 details the operational definitions, specific measurement methods using the `PROFILE MATCH` clause prefix on Cypher queries, and the research description for each variable.

Table 7. Performance metrics for model of P2L and P2N

Variable	Measurement	Description
Running times	Milliseconds (ms)	Evaluates the model's computing speed and resilience against degradation when the graph scales upward. It directly tests whether the Cypher query paths perform optimally under scalability stress.

3. RESULTS AND DISCUSSION

The empirical evaluation of the proposed (P2L and P2N) ontology-to-graph transformation models based on refactoring pattern yields critical insights into the interplay between formal semantic validation (ontology layer in Protégé) and graph database performance (Neo4j). The P2L model relied on checking string values (e.g., `"cat"` or `"lion"`), while the P2N model used taxonomic reasoning for classification. When the reasoner encounters the assertions `Garfield:Cat` and `Cat` $\sqsubseteq$ `Pet`, it automatically infers that Garfield is an instance of `Pet` through subclass transitivity. The P2N model was ontologically cleaner than P2L because it avoids using data properties for basic classification. As a result, the Tableau reasoner can process the class hierarchy more efficiently without evaluating string literals. The same idea with the refactoring in Neo4j that this approach shifts semantics from data (properties) to the graph/schema structure. Similarly, [12], [4] showed that representing semantics through the graph structure, rather than relying on property-based filtering, improves query efficiency. Their study demonstrated that pattern matching using `WHERE` clauses (filters) incurs performance degradation and increases computational overhead during condition evaluation.

3.1. Property-to-Node (P2N) Model

To have new relationship types in Neo4j, additional object properties (`hasCatSpecies` and `hasLionSpecies`) and individuals (`CatIndividuals` and `LionIndividuals`) are introduced. The additional individuals represent property values as explicit nodes, while the object properties represent relationships between the original entities and these nodes [21]. For example, hasCatSpecies(Garfield, CatIndividuals) can be transformed into a `HAS_CAT_SPECIES` relationship in Neo4j. A reasoner cannot arbitrarily create these object-property assertions because a class assertion such as Cat(Garfield) does not specify a particular target individual such as `CatIndividuals`. Therefore, the new relationships are explicitly represented in the ABox. The reasoner can subsequently apply domain, range, subclass, and other ontology axioms to derive additional facts. Although OWL reasoners can infer new object-property assertions when supported by explicit role axioms, such as inverse properties, property chains, transitivity, or symmetry, these mechanisms were not used in this research.

3.2. Property-to-Label (P2L) Model

Two P2L models were developed to transform ontology concepts into Neo4j node labels. The first uses `EquivalentTo`, defining `Pet` based on `species = "cat"` and `Wild` based on `species = "lion"`. Thus, a reasoner can automatically infer `Garfield` as a `Pet`. However, because species remains a data property, identifying `Cat` in Neo4j still requires property-based queries such as `MATCH (n) WHERE n.species = 'cat'`, rather than Node labels [12].

The second model uses explicit `SubClassOf` relationships (Relationship types), defining `Cat` ⊑ `Pet` and `Lion` ⊑ `Wild`. `Garfield` and `Simba` are explicitly asserted as `Cat` and `Lion`, respectively, allowing the reasoner to infer `Pet(Garfield)` and `Wild(Simba)`. Consequently, Neo4j can represent both specific and general concepts as node labels, such as `:Cat` and `:Pet`.

Thus, the two P2L models have different strengths. The first model, based on `EquivalentTo`, provides automatic classification from data property values and is useful for detecting general concepts such as `Pet`. The second model, based on `SubClassOf`, provides an explicit class hierarchy that is more directly aligned with the generation of Neo4j node labels for specific concepts such as `Cat` and `Lion`. Therefore, when the primary objective is to transform ontology concepts into explicit Neo4j node labels and support competency questions through graph labels, the second P2L model is more suitable. Conversely, if automatic classification from property values is the primary requirement, the first P2L model provides a useful reasoning mechanism.

3.3. Logical Validation and Inference Results

The logical evaluation demonstrated that both ontology models successfully enforced the required business rules (competency question) [15]. When a transaction attempted to link an `:Owner` to a "`Wild`" animal, the n10s validation listener successfully caught the conflict and aborted the transaction in both setups, as shown in Figure 8. Furthermore, the incomplete data scenario was resolved elegantly by both configurations based on the ABox specified. For P2L model, asserting the relationship `(:Owner)-[:HASPET]->(Tom)` allowed the HermiT reasoner to immediately infer the label `:Pet` for "`Tom`" due to the direct range constraint [1]. For P2N model, the reasoner correctly inferred that "`Tom`" belonged to the `Pet` class, though returning this inference to the graph database required generating an extra `hasPet` structural path [1]. For consistency testing purpose of the simple pet ontology, the ABox was added "MammyTwoShoes has a pet named Simba". Figure 3 shows the reasoner result.



Figure 3. Reasoner result for consistency testing

3.4. Performance Benchmarking

The empirical benchmarking revealed that the two refactored schemas exhibited very similar distributions of queries across the execution time change categories measured by running time in milliseconds using 78 variance queries [4]. The running time of the 78 queries was further analyzed by examining its relationship with the percentage reduction in `WHERE` clauses. The results were categorized according to the number of properties contained in the `WHERE` clause of the baseline (see Figure 1) query and the percentage reduction in `WHERE` clauses after the queries were reconstructed according to the refactoring [12] schema. Table 8 shows data schema in Neo4j for P2L model.

Data property of `species` in P2L model using equivalentClass (Table 8) only have "`Pet`" node labels, while another node label of `Cat` in the same node was added in P2L model using subClassOf, as shown in Table 9, where it was used to support Cypher Query Execution performance [12] in Table 1 to answer competency question of "Which individuals are cats?" by using `MATCH (n:Cat)`. However, in P2N model, the new relationship and the new node label were created, as seen in Table 10.

In Neo4j, node labels are indexed natively using pointer-first lookup structures, allowing the execution engine to bypass irrelevant nodes entirely and scan only nodes labeled with `:Pet` [1]. Consequently, the Cypher query in P2L requires only a single-hop index traversal to retrieve the target dataset. In contrast, the P2N model stores the classification as a separate physical node, forcing the database engine to perform a two-hop graph

traversal across relationships [1]. In graph databases, executing multi-hop traversals introduces significant memory pointer chasing and CPU overhead, especially as the size of the graph scales up [12].

Table 8. Neo4j result for P2L model using equivalentClass

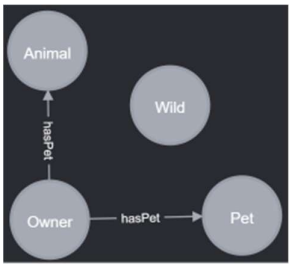
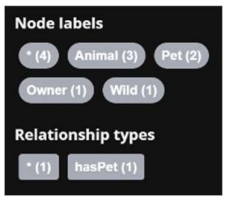
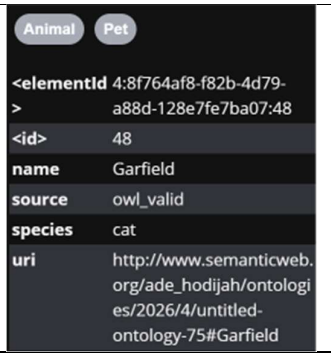
Schema	Node labels and Relationship types	Data properties
		

Table 9. Neo4j result for P2L model using subClassOf

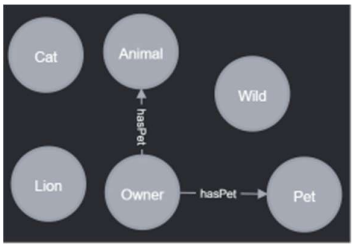
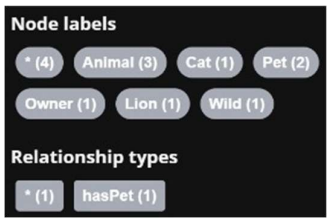
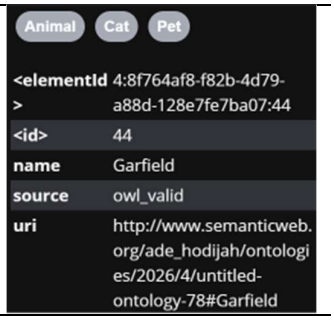
Schema	Node labels and Relationship types	Data properties
		

Table 10. Neo4j result for P2N model

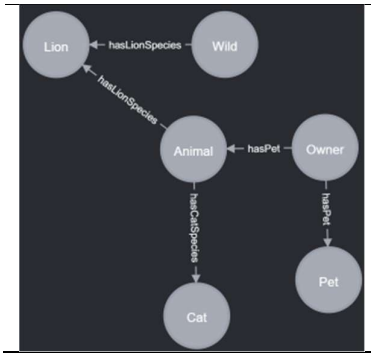
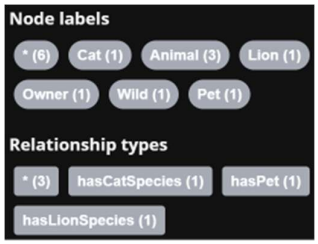
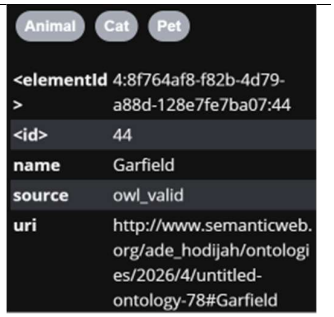
Schema	Node labels and Relationship types	Data properties
		

Table 11 shows that P2L consistently provides more query execution-time reductions than P2N. Across all 78 queries, P2L reduces execution time in 53 queries (67.95%), while execution time increases in 25 queries (32.05%). In comparison, P2N reduces execution time in 52 queries (66.67%) and increases execution time in 26 queries (33.33%).

Table 11. Cypher query running times across varying number of queries and WHERE properties

Number of WHERE	Number of Queries	Reduction (P2N)	Increase (P2N)	Reduction (P2L)	Increase (P2L)
1	3	33% (1)	67% (2)	0% (0)	100% (3)
	15	87% (13)	13% (2)	60% (9)	40% (6)
2	4	50% (2)	50% (2)	50% (2)	50% (2)
	16	62.5% (10)	37.5% (6)	62.5% (10)	37.5% (6)
3	9	44% (4)	56% (5)	67% (6)	33% (3)
	11	73% (8)	27% (3)	82% (9)	18% (2)
4	11	55% (6)	45% (5)	82% (9)	18% (2)
	9	89% (8)	11% (1)	89% (8)	11% (1)

3.5. Discussion

This research aimed to know to what extent does Neo4j refactoring at the ontology data modeling level reduce the execution time of queries that use the WHERE clause. Based on the Table 11 results with adding the number of WHERE properties and queries scenario, reducing the number of WHERE clauses [4] has an impact on the following metrics:

- In the data schema resulting from the P2N refactoring technique, the effect of reducing the number of WHERE clauses on execution time reduction becomes consistently evident only when the reduction reaches 100%. At this level, 87%, 62.5%, 73%, and 89% of queries exhibited reduced execution times for query groups containing 1, 2, 3, and 4 WHERE properties, respectively. In contrast, the pattern is inconsistent for groups with partial reductions. These findings suggest that the performance improvement achieved by the property becoming a node technique becomes consistently observable only when the number of WHERE clauses is reduced by 100%.
- For the data schema produced using the P2L technique, the relationship between the reduction in WHERE clauses and the reduction in execution time is more consistent than that observed with property becoming a node, particularly for groups with partial reductions. A 67% reduction in WHERE clauses resulted in 67% of queries exhibiting reduced execution times, while a 75% reduction increased this proportion to 82%. When the reduction reached 100%, the proportion of queries with improved execution times increased further compared with the partial reduction groups. However, for query groups containing a single WHERE property, the label specification schema achieved a lower improvement than the property becoming a node schema under the 100% reduction scenario, with 60% and 87% of queries showing reduced execution times, respectively. This indicates that the advantage of the label specification approach over property becoming a node is not consistent across all query groups.

4. CONCLUSION

This study evaluated Property-to-Label (P2L) and Property-to-Node (P2N) graph refactoring approaches integrated with OWL ontologies using Protégé, Neo4j, and neosemantics (n10s). The PET domain served as a conceptual case study to demonstrate ontology design (TBox/RBox), semantic constraint enforcement, and reasoning capabilities on incomplete ABox data. Empirical performance benchmarking was conducted separately using 78 Cypher query variations with AND operators derived from the IMDB dataset. Both models effectively bridged semantic validation and graph database execution. However, performance analysis on the IMDB dataset demonstrated that P2L showed a slightly higher proportion of queries with reduced execution time than P2N and P2L showed a more consistent overall relationship between WHERE-clause reduction and execution-time improvement, although P2N performed better in some query groups.

A limitation of this research is that performance benchmarking was conducted using the IMDB dataset, while the PET case study was used exclusively for logical validation and reasoning rather than empirical execution-time experiments. Therefore, the performance characteristics of P2L and P2N have not yet been empirically evaluated on the PET ontology. Future work should conduct large-scale performance benchmarking directly on the PET domain. In addition, the P2N model has not yet fully explored reasoning over newly introduced relationships. In P2N, these relationships are explicitly represented in the ABox, allowing the reasoner to subsequently apply domain, range, subclass, and other ontology axioms to derive additional facts. Although OWL reasoners can infer new object-property assertions when supported by explicit role axioms, such as inverse properties, property chains, transitivity, or symmetry, these mechanisms were not used in this research. Future work should therefore evaluate P2N reasoning using explicit role axioms and investigate more complex query structures involving OR operators and Disjunctive Normal Form (DNF).

5. ACKNOWLEDGMENTS

This research was funded by the State Polytechnic of Bandung's DIPA under the Laboratory Capacity Building Research (Penelitian Peningkatan Kapasitas Laboratorium) scheme.

6. REFERENCES

[1] J. Barrasa and J. Webber, "Building Knowledge Graphs," *O'REILLY*, [Online]. Available: https://neo4j.com/knowledge-graphs-practitioners-guide/
[2] D. Bloor, N. Ugwuoke, D. Taylor, K. Lewis, L. Mur, and C. Lu, "Towards a Personal Health Knowledge Graph Framework for Patient Monitoring," Nov. 11, 2023, *arXiv*: arXiv:2311.06524. doi: 10.48550/arXiv.2311.06524.
[3] C. Sharma, P. Genevès, N. Gesbert, and N. Layaïda, "Schema-Based Query Optimisation for Graph Databases," *Proc. ACM Manag. Data*, vol. 3, no. 1, pp. 1–29, Feb. 2025, doi: 10.1145/3709722.
[4] G. Micale *et al.*, "MultiGraphMatch: A Subgraph Matching Algorithm for Multigraphs," *ACM Trans. Knowl. Discov. Data*, vol. 19, no. 5, pp. 1–36, Jun. 2025, doi: 10.1145/3728361.
[5] R. Angles and R. García, "Transforming RDF Data into Property Graphs," *IEEE Latin Am. Trans.*, vol. 18, no. 01, pp. 130–137, Jan. 2020, doi: 10.1109/TLA.2020.9049470.
[6] G. Abuoda, D. Dell'Aglio, A. Keen, and K. Hose, "Transforming RDF-star to Property Graphs: A Preliminary Analysis of Transformation Approaches".
[7] D. Tomaszuk, R. Angles, and H. Thakkar, "PGO: Describing Property Graphs in RDF," *IEEE Access*, vol. 8, pp. 118355–118369, 2020, doi: 10.1109/ACCESS.2020.3002018.
[8] S. Purohit, N. Van, and G. Chin, "Semantic Property Graph for Scalable Knowledge Graph Analytics," in *2021 IEEE International Conference on Big Data (Big Data)*, Orlando, FL, USA: IEEE, Dec. 2021, pp. 2672–2677. doi: 10.1109/BigData52589.2021.9671547.
[9] F. Darari, *Knowledge Graph Mini Handbook*, vol. v1.1.0. Faculty of Computer Science Universitas Indonesia.
[10] D. Bloor, "A Personal Health Knowledge Graph Framework for Management of Chronic Obstructive Pulmonary Disease," ABERYSTWYTH UNIVERSITY. [Online]. Available: https://pure.aber.ac.uk/ws/portalfiles/portal/105519514/Bloor_Daniel.pdf
[11] T. L. Andrews, "Graph schema validation at last? Work towards a StemmaREST ontology with Neo4J, RDF-STAR, and SHACL," *Digital Scholarship in the Humanities*, vol. 41, no. Supplement_1, pp. i3–i12, Apr. 2026, doi: 10.1093/llc/fqaf067.
[12] A. Hodijah and T. Gelar, "Analysis Graph-Based Data Model For Simple Search Application Using NER Data," in *Proceedings of the International Conference on Applied Science and Technology on Engineering Science 2023 (iCAST-ES 2023)*, vol. 230, M. U. H. Al Rasyid and M. R. Mufid, Eds., in Advances in Engineering Research, vol. 230. , Dordrecht: Atlantis Press International BV, 2024, pp. 943–957. doi: 10.2991/978-94-6463-364-1_86.
[13] A. M. Fonseca and L. Camolesi, "Refactoring Rules for Graph Databases," in *New Contributions in Information Systems and Technologies*, vol. 353, A. Rocha, A. M. Correia, S. Costanzo, and L. P. Reis, Eds., in Advances in Intelligent Systems and Computing, vol. 353. , Cham: Springer International Publishing, 2015, pp. 33–42. doi: 10.1007/978-3-319-16486-1_4.
[14] N. F. Noy and D. L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology".
[15] S. Aminah, A. Alfa Krisnadhi, and A. Nizar Hidayanto, "Ontological Framework for the Analysis of Outcome-Based Curriculum in Higher Education," *IEEE Access*, vol. 13, pp. 31497–31516, 2025, doi: 10.1109/ACCESS.2025.3542881.
[16] Neo4j Labs, *Neosemantics (n10s)*. [Online]. Available: https://github.com/neo4j-labs/neosemantics
[17] J.-B. Lamy, *Owlready2*. [Online]. Available: https://github.com/pwin/owlready2
[18] J.Barrasa, "Importing RDF data into Neo4j," Jesús Barrasa. Accessed: Aug. 03, 2026. [Online]. Available: https://jbarrasa.com/2016/06/07/importing-rdf-data-into-neo4j/
[19] A. Hodijah and U. T. Setijohatmo, "Analysis of frequent itemset generation based on trie data structure in Apriori algorithm," *TELKOMNIKA*, vol. 19, no. 5, p. 1553, Oct. 2021, doi: 10.12928/telkomnika.v19i5.19273.
[20] A. Hodijah, U. T. Setijohatmo, G. Munawar, and G. S. Ramadhanty, "Multi-threaded approach in generating frequent itemset of Apriori algorithm based on trie data structure," *TELKOMNIKA*, vol. 20, no. 5, p. 1034, Oct. 2022, doi: 10.12928/telkomnika.v20i5.24088.
[21] J.Barrasa, "Importing RDF data into Neo4j," Jesús Barrasa. Accessed: Aug. 20, 2026. [Online]. Available: https://jbarrasa.com/2016/06/07/importing-rdf-data-into-neo4j/