User Access Control in Policy-Protected Virtual Knowledge Graphs

Divya Baura[1][0000−0002−5237−9927] and Diego Calvanese[2][1111−2222−3333−4444]

[1] Umeå Universitet, Umeå, Sweden
[2] Free University of Bozen-Bolzano, Bolzano, Italy

Abstract. *Virtual Knowledge Graph* (VKG) is a well-established framework in which users can access a relational data source through an ontology and declarative mappings. VKG systems traditionally assume uniform access rights for all users, an assumption that does not always hold in real-world scenarios involving diverse user roles and sensitive information requiring protection. *Controlled Query Evaluation* (CQE) provides a privacy-preserving framework by enforcing policies that define confidential information and implementing censors to prevent policy violations. However, it does not account for differences in user privileges during query answering. To address this gap, we extend the *Policy-Protected VKG* (PPVKG) framework, which embeds CQE policies into VKG mappings, by enabling role-sensitive query answering. Specifically, we incorporate *Role-Based Access Control* (RBAC) into PPVKG, by associating to each user role a specific set of policies, and ensuring that during query evaluation, only the policies relevant to the user's role are applied. We validate our RBAC enhanced PPVKG approach using the MIMIC-III critical-care database, mapped to the *Fast Healthcare Interoperability Resources* (FHIR) ontology. Our experiments, conducted with the open-source VKG system Ontop, demonstrate effective policy enforcement with RBAC.

Keywords: Virtual Knowledge Graph · Controlled Query Evaluation · Policy-protected VKG · Role-Based Access Control

1 Introduction

The Virtual Knowledge Graph (VKG) paradigm [16,20] enables seamless integration and querying of heterogeneous relational databases through an ontology layer, typically expressed in OWL 2 QL [15], using declarative mappings (e.g., R2RML [7]) that connect database schema elements to ontology classes and properties. This allows users to pose high-level semantic queries over the ontology without directly engaging with the underlying data sources.

Despite its strengths, VKG inherits critical privacy and inference risks, since TBox axioms may implicitly reveal sensitive information through logical entailments. *Controlled Query Evaluation (CQE)* addresses this by modeling confidentiality requirements as policies and applying censoring functions that evaluate

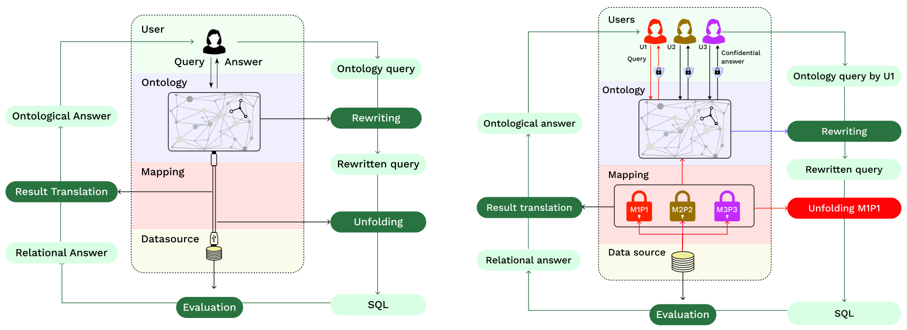

Fig. 1. Query processing in VKG (left) vs. RBAC-PPVKG (right)

and restrict query answers [3,11,14]. Extending this concept, *Policy-Protected VKG* (PPVKG) integrates CQE in VKG [6,2]. PPVKG specifies policies that are first-order denial constraints. These policies are then embedded into VKG mapping assertions, resulting in new mapping assertion that enforce policy compliance during the rewriting process. However, both VKG and PPVKG assume a uniform access model, granting identical data visibility to all authenticated users regardless of their operational context or responsibilities. This one-size-fits-all paradigm proves inadequate in sensitive domains like healthcare, where granular access differentiation based on professional roles is essential not only to prevent data leakage but also to comply with stringent regulations like the *General Data Protection Regulation* (GDPR) [19].

Role-Based Access Control (RBAC) [18,9] provides the necessary conceptual framework to address these limitations by enforcing the privacy principle of least privilege, restricting users to the minimum level of access required for their organizational roles or responsibilities. In this work, we show that mappings can be enriched with RBAC, thereby ensuring that query results are dynamically filtered based on user roles and data exposure is minimized. We have implemented constrained RBAC, enforcing session-specific separation-of-duty rules and limiting role activations per session, in compliance with the NIST RBAC standard's more secure models. This integration enables enterprise-grade features such as role-differentiated views and dynamic session-aware policy enforcement. The resulting RBAC-enhanced PPVKG framework combines static privacy policies with runtime, context-aware access control, making it suitable for deployment in complex domains like healthcare and finance, where both confidentiality and compliance are paramount. As illustrated in Figure 1, our approach activates role-specific mapping assertions during query processing. For example, when User 1 (specified with red color) queries the ontology, only policy augmented mappings applicable to their role with the specific policy (e.g., M1P1) are activated.

In the rest of the paper, after introducing technical preliminaries on VKG and PPVKG (Section 2), we define the notion of RBAC (Section 3) and introduce

PPVKG with RBAC functionalities (Section 4). We evaluate our implementation of role-base policy embedding into mappings on the MIMIC-III clinical dataset [13], demonstrating that role-aware query answering delivers fine-grained access control with minimal performance overhead (Section 5).

Our implementation and experimental evaluation is available on GitHub[3].

2 Preliminaries

2.1 Virtual Knowledge Graph

In VKG, users can query a data source through an ontology TBox linked to the data source via declarative mappings [16,20]. We formalize VKG using the notion of *VKG specification*, which is a triple $\mathcal{V} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{T}$ is a TBox, $\mathcal{S}$ a relational database (DB) schema, and $\mathcal{M}$ a mapping between $\mathcal{T}$ and $\mathcal{S}$. The TBox captures knowledge at the intensional level about the classes (denoting sets of individuals) and properties (denoting binary relations between individuals or values) of the domain of interest. It is expressed in the lightweight description logic (DL) *DL-Lite*$_\mathcal{R}$ [5], which is the formal counterpart of the ontology language OWL 2 QL, standardized by the W3C [15]. For the formal details about *DL-Lite*$_\mathcal{R}$ and OWL 2 QL we refer to [5,15]. The *mapping* $\mathcal{M}$ is a finite set of *mapping assertions* from $\mathcal{S}$ to $\mathcal{T}$, each of the form $\varphi(\boldsymbol{x}) \rightsquigarrow \psi(\mathsf{IRI}(\boldsymbol{x}))$. Here, $\varphi(\boldsymbol{x})$ denotes a FOL (or SQL) source query over $\mathcal{S}$ with answer variables $\boldsymbol{x}$, while $\psi(\mathsf{IRI}(\boldsymbol{x}))$, referred to as the *target* of the mapping, is an atom whose predicate is a class or property of $\mathcal{T}$ over the variables in $\boldsymbol{x}$ and so-called *IRI-templates* $\mathsf{IRI}(\boldsymbol{x})$. Each IRI-template $\mathsf{iri}(\boldsymbol{x})$ in $\mathsf{IRI}(\boldsymbol{x})$ is a term that concatenates string values and the answer variables in $\boldsymbol{x}$, and is used to construct values (i.e., literals) and object identifiers (i.e., IRIs) from the DB values returned by $\varphi(\boldsymbol{x})$. A concrete mapping language that provides such mappings is R2RML, standardized by the W3C [7]. In our examples we provide mappings in the mapping language of the Ontop system (see Section 2.3), where the source query is expressed in SQL, the target atom is specified in RDF syntax as a *triple template* (in which the SQL answer variables of the IRI-templates are enclosed in `{...}`), and multiple mapping assertions with the same source query might be grouped together, resulting in a target consisting of multiple RDF triple templates.

Given a VKG specification $\mathcal{V} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ and a DB instance $\mathcal{D}$ for $\mathcal{S}$, the pair $\langle \mathcal{V}, \mathcal{D} \rangle$ is called a *VKG instance*. The *retrieved ABox* for $\langle \mathcal{V}, \mathcal{D} \rangle$, denoted $\mathsf{ret}(\mathcal{V}, \mathcal{D})$, consists of all facts $\psi(\mathsf{IRI}(\boldsymbol{t}))$, where $\varphi(\boldsymbol{x}) \rightsquigarrow \psi(\mathsf{IRI}(\boldsymbol{x}))$ is a mapping assertion in $\mathcal{M}$, and $\boldsymbol{t} \in \mathsf{eval}(\varphi(\boldsymbol{x}), \mathcal{D})$ is a tuple of constants in the evaluation of $\varphi(\boldsymbol{x})$ over $\mathcal{D}$. Notice that the fact $\psi(\mathsf{IRI}(\boldsymbol{t}))$ contains IRIs of the form $\mathsf{iri}(\boldsymbol{t})$ constructed from the answer tuple $\boldsymbol{t}$ using the IRI template $\mathsf{iri}(\boldsymbol{x})$ in the mapping target. Hence, $\mathsf{ret}(\mathcal{V}, \mathcal{D})$ is an ABox over constants in the set $\Delta_{\mathcal{O},\mathcal{D}}$, consisting of: *(i)* all DB values in $\mathcal{D}$ and *(ii)* all possible IRIs $\mathsf{iri}(\boldsymbol{t})$ constructed from some tuple $\boldsymbol{t}$ of values in $\mathcal{D}$ and some IRI template $\mathsf{iri}(\boldsymbol{x})$ in a mapping assertion in $\mathcal{M}$.

[3] https://github.com/divyabaura/PPVKG-role-based-policies

A *model* of the VKG instance $\langle \mathcal{V}, \mathcal{D} \rangle$ is defined as a model of the DL knowledge base $\langle \mathcal{T}, \mathsf{ret}(\mathcal{V}, \mathcal{D}) \rangle$. We denote the set of models of $\langle \mathcal{V}, \mathcal{D} \rangle$ by $\mathsf{Mod}(\mathcal{V}, \mathcal{D})$, and we say that $\langle \mathcal{V}, \mathcal{D} \rangle$ is *inconsistent* if $\mathsf{Mod}(\mathcal{V}, \mathcal{D}) = \emptyset$. Moreover, $\langle \mathcal{V}, \mathcal{D} \rangle \models \alpha$, indicating that $\langle \mathcal{V}, \mathcal{D} \rangle$ entails a sentence α, holds if α is true in every model in $\mathsf{Mod}(\mathcal{V}, \mathcal{D})$. Here, we adopt the *standard name assumption*, i.e., given a VKG instance $\langle \mathcal{V}, \mathcal{D} \rangle$, we consider interpretations over a fixed domain Δ containing $\Delta_{\mathcal{O},\mathcal{D}}$ such that all values in $\Delta_{\mathcal{O},\mathcal{D}}$ are interpreted as themselves.

We provide an example of VKG mappings in a medical domain, on which we will build in the rest of the paper.

Example 1. We assume to have in the ontology a class `:Patient` with data property `:Patient.gender`, and a class `:Prescription` with object property `:Prescription.subject` relating a prescription to the patient who is the subject of the prescription. In the data source we have a table `Person`, which is mapped to the class `:Patient` and the `:Patient.gender` data property, and a table `DrugExposure`, which is mapped to the class `:Prescription` and the `:Prescription.subject` object property, as follows:

```
mappingId m1
target :Patient/{person_id} a :Patient ; :Patient.gender {gender}^^xsd:string .
source SELECT person_id, gender FROM Person P

mappingId m2
target :Prescription/{drug_exposure_id} a :Prescription ;
         :Prescription.subject :Patient/{person_id} .
source SELECT drug_exposure_id, person_id FROM DrugExposure
```

$\triangleleft$

2.2 Policy-Protected VKG

Policy-Protected VKG (PPVKG) extends the traditional VKG paradigm by embedding data protection policies directly into VKG mapping definitions [6,2]. While standard VKG enables querying heterogeneous sources through an ontological semantic layer, PPVKG makes use of policies to specify information that should be protected. Specifically, policies in PPVKG are expressed as denial constraints and the disclosure of protected information, i.e., information that causes a violation of the denial constraint, is prevented by dynamically rewriting queries in such a way that protected information is suppressed from query results. This is achieved by compiling the policy rules into the declarative VKG mappings. In this way, a policy-aware VKG is transformed into a standard VKG that transparently enforces the policies.

PPVKG builds upon the principles of *controlled query evaluation* (CQE), which is a privacy-preserving framework for query answering in the presence of ontologies [3,11,14], and it extends the VKG framework to incorporate CQE. To formally introduce the PPVKG setting, we first define a *denial (assertion)* as a first-order logic (FOL) sentence of the form $\forall \boldsymbol{x}.\varphi(\boldsymbol{x}) \rightarrow \bot$, where $\exists \boldsymbol{x}.\varphi(\boldsymbol{x})$ is a Boolean conjunctive query (BCQ). Given a set $\mathcal{V}$ of FOL sentences (e.g., a TBox) and a denial δ, we say that $\mathcal{V} \cup \{\delta\}$ is *consistent* if $\mathcal{V} \not\models \exists \boldsymbol{x}.\varphi(\boldsymbol{x})$.

Following [6], we define a *PPVKG specification* as a 4-tuple $\mathcal{E} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M}, \mathcal{P} \rangle$, where $\langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ is a VKG specification and $\mathcal{P}$ is a *policy*, i.e., a finite set of denials over the signature of $\mathcal{T}$, such that $\mathcal{T} \cup \mathcal{P}$ is consistent. The semantics of a PPVKG specification is the same as that of the underlying VKG specification, and all other notions (source DB $\mathcal{D}$, instance $\langle \mathcal{E}, \mathcal{D} \rangle$, retrieved ABox $\mathsf{ret}(\mathcal{E}, \mathcal{D})$, and set $\mathsf{Mod}(\mathcal{E}, \mathcal{D})$ of models) naturally extend to PPVKG.

For a query language $\mathcal{L}$ (e.g., the language **CQ** of conjunctive queries or **GA** of ground atoms), let $\mathcal{L}(\mathcal{T})$ denote the restriction of $\mathcal{L}$ to the predicates in $\mathcal{T}$, and $\mathcal{L}_{\mathcal{D}}$ the formulas in $\mathcal{L}$ mentioning only constants in $\mathcal{D}$. An *optimal censor* for $\mathcal{E}$ in $\mathcal{L}$ is a function $\mathsf{cens}(\cdot)$ that, for each source DB $\mathcal{D}$ for $\mathcal{E}$, returns a set $\mathsf{cens}(\mathcal{D}) \subseteq \mathcal{L}_{\mathcal{D}}$ such that *(i)* $\langle \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle, \mathcal{D} \rangle \models \varphi$, for each $\varphi \in \mathsf{cens}(\mathcal{D})$, and *(ii)* $\mathcal{T} \cup \mathcal{P} \cup \mathsf{cens}(\mathcal{D})$ is consistent. $\mathcal{L}$ is called the *censor language*. We are interested in *optimal* censors, i.e., those that return as many formulas as possible. The set of all optimal censors in $\mathcal{L}$ for a PPVKG specification $\mathcal{E}$ is denoted $\mathcal{L}\text{-}\mathsf{OptCens}_{\mathcal{E}}$.

To obtain a notion of censor that allows for embedding a policy into the mapping, [6] define censors that approximate censors for $\mathcal{E}$ in **GA** [14].

Definition 2 (IGA censor [6]). Given a PPVKG specification $\mathcal{E} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M}, \mathcal{P} \rangle$, the *intersection GA (IGA)* censor for $\mathcal{E}$ is the function $\mathsf{cens}_{\mathrm{IGA}}(\cdot)$ such that, for every DB instance $\mathcal{D}$ for $\mathcal{S}$, $\mathsf{cens}_{\mathrm{IGA}}(\mathcal{D}) = \bigcap_{\mathsf{cens} \in \mathbf{GA}\text{-}\mathsf{OptCens}_{\mathcal{E}}} \mathsf{cens}(\mathcal{D})$. ◁

Thus, an IGA censor, when applied to a DB instance $\mathcal{D}$ for the source schema $\mathcal{S}$ of $\mathcal{E}$, returns the intersection of the sets of ground atoms computed by all optimal censors. In [2], an algorithm called EncodeMapping is presented, which generates a new VKG mapping $\mathcal{M}'$ that embeds a policy $\mathcal{P}$ into a given mapping $\mathcal{M}$. When the VKG engine processes queries posed over the ontology using $\mathcal{M}'$, the obtained answers comply with $\mathcal{P}$ according to an IGA censor.

Example 3 (Example 1 cont'd). Let us now consider two policies p_1 : $\forall x.\forall y.\mathit{Patient.gender}(x, y) \wedge \forall z.\mathit{Patient.address}(x, z) \rightarrow \bot$, which ensures the confidentiality of the combination of a patient's gender and address, and p_2 : $\forall x.\mathit{Prescription}(x) \wedge \forall y.\mathit{Prescription.subject}(x, y) \rightarrow \bot$, which ensures the confidentiality of patient ids for which a prescription is made. Let us assume that the `Person` table contains also a `location_id` attribute that is mapped to the *Patient.address* data property. By compiling these policies into m_1 and m_2, we obtain the following policy-protected mapping assertions:

```
mappingId mp1
target :Patient/{person_id} a :Patient ;  :Patient.gender {gender}^^xsd:string .
source SELECT person_id, gender FROM Person WHERE location_id IS NULL

mappingId mp2
target :Prescription/{drug_exposure_id} a :Prescription ;
         :Prescription.subject :Patient/{person_id} .
source SELECT drug_exposure_id, person_id FROM DrugExposure WHERE person_id IS NULL
```

Intuitively, the condition `location_id IS NULL` in *mp₁* ensures that the gender location can be returned only for patients whose `location_id` is not known. Similarly, mp_2 ensures that prescriptions can only be disclosed when the `person_id` is not known. $\triangleleft$

2.3 The VKG System Ontop

We utilize the state-of-the-art open-source VKG system Ontop [4,21], which supports query answering over a VKG instance through query transformation and sophisticated optimization techniques. An Ontop installation operates on a VKG specification $\mathcal{V} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{T}$ is an OWL 2 QL TBox, $\mathcal{S}$ a relational schema with constraints, and $\mathcal{M}$ an R2RML [7] mapping. Ontop supports also a proprietary mapping language that is more user-friendly and fully interoperable with R2RML, and in our examples we have used such language.

Ontop efficiently answers SPARQL queries [12] over a VKG instance $\langle \mathcal{V}, \mathcal{D} \rangle$, where $\mathcal{D}$ is a DB instance for $\mathcal{S}$, using the OWL 2 QL entailment regime [10]. It does so by implementing *query answering by rewriting*. This process involves pre-processing $\mathcal{T}$, $\mathcal{S}$, and $\mathcal{M}$ for optimization purposes. At runtime, Ontop transforms a given SPARQL query into an equivalent SQL query and optimizes it using mapping information and database constraints in $\mathcal{S}$. The resulting SQL query gets then executed by the underlying DB engine.

3 Role Based Access Control

Role-Based Access Control (RBAC) is a well-established authorization model that assigns permissions based on organizational roles rather than individual identities. This abstraction simplifies policy management by aligning access rights with job responsibilities, reducing administrative overhead and mitigating risks from over-privileged users [18]. In the RBAC paradigm, three core entities interact: *users* (human or system agents), *roles* (functional positions within an organization), and *permissions* (authorizations to perform operations on protected resources).

The foundational variant of RBAC, known as *Flat RBAC* or *Core RBAC*, mandates many-to-many relations between users and roles and between roles and permissions, it also requires that users can activate multiple roles concurrently. Building upon this, *Hierarchical RBAC* introduces role inheritance, where senior roles automatically inherit permissions from their juniors, supporting partial-order structures or restricted hierarchies. A further enhancement, *Constrained RBAC*, enforces separation of duty constraints, static or dynamic, to prevent incompatible roles from being assigned or activated simultaneously. The *Symmetric RBAC* (or RBAC3), retains all the functionality of Constrained RBAC and adds the crucial capability of permission-role review. In effect, the four RBAC levels—Flat, Hierarchical, Constrained, and Symmetric—form a cumulative hierarchy in which each level builds upon the previous one by introducing a single new dimension of control: role hierarchy, separation of duties, and role–permission auditability, respectively. [17].

In this work, we present a technique to incorporate the principles of constrained role-based access Control within the PPVKG framework. This is achieved by enforcing access control through the concept of sessions, where each session represents a specific user operating under a defined access role.

We formalize the Flat RBAC model through the following notion.

Definition 4 (RBAC specification). A *role-based access control (RBAC) specification* is a tuple $\langle \mathcal{O}, \mathcal{A}, \mathcal{U}, \mathcal{R}, \mathsf{UR}, \mathsf{P}, \mathsf{perms} \rangle$, where:

- $\mathcal{O}$ is a finite set of *objects*, representing protected resources;
- $\mathcal{A}$ is a finite set of *actions*, representing operations that can be performed on the objects;
- $\mathcal{U}$ is a finite set of *users*;
- $\mathcal{R}$ is a finite set of *roles* (e.g., *doctor*, *nurse*);
- $\mathsf{UR} \subseteq \mathcal{U} \times \mathcal{R}$ is a *user-role assignment* relation;
- $\mathsf{P} \subseteq \mathcal{O} \times \mathcal{A}$ represents *permissions*, where each pair $\langle o, a \rangle \in \mathsf{P}$ represents a permission to perform action a on object o; and
- $\mathsf{perms} : \mathcal{R} \to 2^{\mathsf{P}}$ is a function mapping roles to sets of their permissions. $\triangleleft$

Modern RBAC implementations increasingly incorporate dynamic policy-driven mechanisms where role assignments become context-sensitive rather than static. Such user to role mappings may dynamically activate based on temporal constraints (e.g., emergency room physicians gaining after-hours access), environmental conditions (e.g., location-based restrictions for mobile clinicians) or runtime attributes (e.g., current patient treatment relationships). This dynamic capability transforms RBAC from a rigid structure into an adaptive policy enforcement engine, where roles serve as policy anchors that respond to real-world operational contexts. We will take into account the dynamicity of actual role assignments by allowing users to log into a PPOBDA system with a specific role, and it is such role that determines the access rights of the user, and hence the PPVKG policies that should be applied.

The GDPR compliance [8] advantages of RBAC are particularly pronounced in policy-driven implementations. *Article 5(1)(c)*'s data minimization principle is reinforced through just-in-time permission activation, ensuring users access only currently necessary data. Similarly, *Article 32(1)(b)*'s confidentiality requirements benefit from dynamic permission revocation when contexts change (e.g., automatically downgrading access when clinicians rotate between departments). By enabling fine-grained, context-aware authorization that restricts data visibility to the minimal necessary set, policy-driven RBAC provides a robust technical foundation for fulfilling legal obligations while enforcing least-privilege access across evolving operational scenarios.

4 PPVKG with RBAC Functionalities

In this section, we extend the PPVKG framework by integrating RBAC to overcome the limitations of traditional policy-protected VKGs that expose the same data to all users regardless of their roles, thus enforcing static policies that lack adaptability and failing to align with real-world organizational structures. We do so by *(i)* suitably duplicating mappings according to user roles and embedding policies associated to roles where appropriate, *(ii)* enabling for each user role only the relevant mapping copies during query unfolding, and *(iii)* ensuring that the proper mappings are taken into account when, for query answering, a user logs into the system with a specific roles.

4.1 Formalizing PPVKG with RBAC functionalities

We start by providing the formalization of the proposed extension of PPVKG with RBAC functionalities.

Definition 5 (RBAC-PPVKG specification). An *RBAC policy-protected VKG* (RBAC-PPVKG) is a tuple $\mathcal{H} = \langle \mathcal{E}, \mathcal{R}, \mathsf{RP}, \mathcal{U}, \mathsf{UR} \rangle$, where:

– $\mathcal{E} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M}, \mathcal{P} \rangle$ is a PPVKG,
– $\mathcal{R}$ is a finite set of roles,
– $\mathsf{RP} : \mathcal{R} \to 2^{\mathcal{P}}$ is a function assigning to each role a set of policies,
– $\mathcal{U}$ is a finite set of users, and
– $\mathsf{UR} \subseteq \mathcal{U} \times \mathcal{R}$ is a user-role assignment. $\triangleleft$

To relate an RBAC-PPVKG specification to an RBAC specification according to Definition 4, we observe that we are interested in answering queries posed by a user over a PPVKG, and the information the user is allowed to access, according to their role, depends on the policy-protected mappings that they can access while unfolding the query. Hence, $\mathcal{A}$ consists of the single action of accessing a mapping during query unfolding. Instead, the set $\mathcal{O}$ of objects (to be accessed) consists of the union of suitable copies of mapping assertions, with one copy $\mathcal{M}_r$ for each role $r \in \mathcal{R}$, obtained from $\mathcal{M}$ by applying to all its mapping assertions the policies $\mathsf{RP}(r)$ assigned to r.

With this correspondence in mind, we want now to transform an RBAC-PPVKG specification into an ordinary VKG specification, which can be processed by a VKG engine like Ontop, and that automatically enforces the appropriate policies for the role with which a user logs into the systems for query answering. We perform this transformation in two steps:

1. For each role $r \in \mathcal{R}$, we create from $\mathcal{M}$ the set $\mathcal{M}_r$ of mapping assertions that embed in $\mathcal{M}$ the policies relevant for role r.
2. We ensure that the mapping assertions in $\mathcal{M}_r$ are activated only for users that log into the system with role r.

Step 2 of the transformation is achieved by making use of a special boolean SQL function `ontop_contains_role`(r), which takes a role r as argument, and evaluates to true only if it is executed while the user is logged in the system with role r. Such function is treated by Ontop in a special way during query processing. Specifically, Ontop evaluates the function at query unfolding time (rather than leaving it in the generated SQL query), and substitutes the function call with the boolean constant `true` or `false` resulting from this evaluation. Then, the query optimization algorithm of Ontop can take into account the value of such constant and simplify the generated SQL query accordingly. We exploit `ontop_contains_role` by embedding in the source query of a mapping assertion m the condition `ontop_contains_role`(r), in conjunction (at the top level of the `WHERE` clause) with the existing condition. Hence: *(i)* If the user is logged with role r, at query unfolding time the condition evaluates to `true` and the mapping assertion m is actually used for the unfolding. *(ii)* Instead, if the user is logged with a role different from r, the condition evaluates to `false` and the overall source query of m is considered by Ontop as equivalent to a query that returns the empty answer. As a result, m is effectively ignored in query unfolding.

To concretely realize the described mapping transformation approach, we make use of two functions that manipulate mapping assertions:

- The function $\textsc{EncodeP}$ takes as arguments a set $\mathcal{M}$ of mapping assertions and a set $\mathcal{P}$ of policies, and returns the set $\textsc{EncodeP}(\mathcal{M},\mathcal{P})$ of mapping assertions in which all policies in $\mathcal{P}$ have been encoded into the mapping assertions of $\mathcal{M}$, according to the technique described in [6,2].[4]
- The function $\textsc{EncodeR}$ takes as arguments a mapping assertion m and a role $r \in \mathcal{R}$ and returns a new mapping assertion $\textsc{EncodeR}(m, r)$ in which the source query of m is modified by conjoining the condition it its `WHERE` clause with an additional filtering condition of the form `ontop_contains_role`(r). We then extend $\textsc{EncodeR}$ to a set $\mathcal{M}$ of mapping assertions by defining $\textsc{EncodeR}(\mathcal{M}, r) = \bigcup_{m\in\mathcal{M}}\{\textsc{EncodeR}(m, r)\}$.

We are now ready to describe how to construct, given an RBAC-PPVKG specification $\mathcal{H} = \langle\mathcal{E}, \mathcal{R}, \mathsf{RP}, \mathcal{U}, \mathsf{UR}\rangle$, where $\mathcal{E} = \langle\mathcal{T}, \mathcal{S}, \mathcal{M}, \mathcal{P}\rangle$, a VKG specification $\mathcal{V}_\mathcal{H} = \langle\mathcal{T}, \mathcal{S}, \mathcal{M}_\mathcal{H}\rangle$ with an enriched set $\mathcal{M}_\mathcal{H}$ of mapping assertions that encode both the appropriate role-specific policies according to $\mathcal{P}$ and RP, and the role-based activation conditions for each role in $\mathcal{R}$. Specifically, for a role $r \in \mathcal{R}$, we define $\textsc{EncodeRP}(\mathcal{M}, r) = \textsc{EncodeR}(\textsc{EncodeP}(\mathcal{M}, \mathsf{RP}(r)), r)$. Then, $\mathcal{M}_\mathcal{H} = \bigcup_{r\in\mathcal{R}} \textsc{EncodeRP}(\mathcal{M}, r)$. Algorithm 1 ($\textsc{EncodeRolePolicies}$) computes the function $\mathcal{M}_\mathcal{H}$ according to this definition.

Example 6 (Example 3 cont'd). Consider a healthcare domain with users $\mathcal{U} = \{\mathit{alice}, \mathit{bob}\}$ and roles $\mathcal{R} = \{\mathit{pharmacist}, \mathit{receptionist}\}$, and assume that $\mathsf{RP}(\mathit{pharmacist}) = \{p_1\}$ and $\mathsf{RP}(\mathit{receptionist}) = \{p_2\}$. Then, $\textsc{EncodeP}(\{m_1, m_2\}, \mathsf{RP}(\mathit{pharmacist})) = \{mp_1, m_2\}$ and $\textsc{EncodeP}(\{m_1, m_2\}, \mathsf{RP}(\mathit{receptionist})) = \{m_1, mp_2\}$. Moreover, by embedding also the role-based activation conditions, we obtain the following mapping assertions:

```
mappingId mrp1
target :Patient/{person_id} a :Patient ;
        :Patient.gender {gender}^^xsd:string .
source SELECT person_id, gender FROM Person
```

[4] The function $\textsc{EncodeP}$ was called $\textsc{EncodeMapping}$ in [2].

Algorithm 1: EncodeRolePolicies

```
Input: A PPVKG specification $\langle \mathcal{T}, \mathcal{S}, \mathcal{M}, \mathcal{P} \rangle$, a finite set $\mathcal{R}$ of roles, a
       role-policy assignment $\mathsf{RP}$
Output: A role-based permission assigned mapping $\mathcal{M}_{\mathcal{H}}$
1 $\mathcal{M}_{\mathcal{H}} \leftarrow \emptyset$;
2 for each role $r \in \mathcal{R}$ do
3     $\mathcal{M}_0 \leftarrow \textsc{EncodeP}(\mathcal{M}, \mathsf{RP}(r))$;
4     $\mathcal{M}_r \leftarrow \emptyset$;
5     for each mapping assertion $m \in \mathcal{M}_0$ do
6         $\mathcal{M}_r \leftarrow \mathcal{M}_r \cup \{\textsc{EncodeR}(m, r)\}$;
7     $\mathcal{M}_{\mathcal{H}} \leftarrow \mathcal{M}_{\mathcal{H}} \cup \mathcal{M}_r$;
8 return $\mathcal{M}_{\mathcal{H}}$;
```

```
       WHERE location_id IS NULL AND
             ontop_contains_role ('pharmacist')

mappingId mr2
target :Prescription/{drug_exposure_id} a :Prescription ;
         :Prescription.subject :Patient/{person_id} .
source SELECT drug_exposure_id, person_id FROM DrugExposure
       WHERE ontop_contains_role ('pharmacist')

mappingId mr1
target :Patient/{person_id} a :Patient ;
         :Patient.gender {gender}^^xsd:string .
source SELECT person_id, gender FROM Person P
       WHERE ontop_contains_role ('receptionist')

mappingId mrp2
target :Prescription/{drug_exposure_id} a :Prescription ;
         :Prescription.subject :Patient/{person_id} .
source SELECT drug_exposure_id, person_id FROM DrugExposure
       WHERE person_id IS NULL AND
             ontop_contains_role ('receptionist')
```

Specifically, $\textsc{EncodeRP}(\{m_1, m_2\}, \textit{pharmacist}) = \{mrp_1, mr_2\}$ and $\textsc{EncodeRP}(\{m_1, m_2\}, \textit{receptionist}) = \{mr_1, mrp_2\}$. $\triangleleft$

These role-conditioned mappings ensure that only users with the appropriate role can access the corresponding data, thereby enforcing both confidentiality and RBAC-based access control. The approach of extending PPVKG with RBAC enables for the policy embedded mappings activation of particular mapping assertion based on the roles of the user querying the data . Hence, the permission now represents the activation or inactivation of such mappings in accordance of user's roles.

4.2 RBAC-PPVKG Query Answering Sessions

In an RBAC specification, a user may be assigned multiple roles by the user-role assignment UR. However, when a user interested in query answering logs into an RBAC-PPVKG system, they will do so by assuming a specific role, and such role should be compatible with the roles assigned to that user. We capture this through the following definition.

Definition 7 (RBAC-PPVKG session). Let $\mathcal{H} = \langle \mathcal{E}, \mathcal{R}, \mathsf{RP}, \mathcal{U}, \mathsf{UR} \rangle$ be an RBAC-PPVKG specification, $r \in \mathcal{R}$ a role, and $u \in \mathcal{U}$ a user. An *RBAC-PPVKG session* is a triple $\langle \mathcal{H}, r, u \rangle$ such that $(u, r) \in \mathsf{UR}$. $\triangleleft$

Intuitively, an RBAC-PPVKG session $\langle \mathcal{H}, r, u \rangle$ represents a user u logged into an RBAC-PPVKG system formalized by $\mathcal{H}$ with a role r that is one of the roles assigned to u in $\mathcal{H}$.

In its setup phase, an RBAC-PPVKG system with specification $\mathcal{H} = \langle \mathcal{E}, \mathcal{R}, \mathsf{RP}, \mathcal{U}, \mathsf{UR} \rangle$, where $\mathcal{E} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M}, \mathcal{P} \rangle$, uses algorithm EncodeRoles to construct the mapping $\mathcal{M}_{\mathcal{H}}$, and exposes the VKG $\mathcal{V}_{\mathcal{H}} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M}_{\mathcal{H}} \rangle$ as a SPARQL endpoint. Consider now a session $\langle \mathcal{H}, r, u \rangle$, where a user u, who is logged into $\mathcal{H}$ with role r, issues queries to the SPARQL endpoint. Then, a given query is processed by the system according to $\mathcal{V}_{\mathcal{H}}$. However, of the overall set $\mathcal{M}_{\mathcal{H}}$ of mapping assertions, the ones actually used to unfold the query are only those in $\mathcal{M}_r \subseteq \mathcal{M}_{\mathcal{H}}$, i.e., those policy-protected according to the policies relevant for role r and activated for r. This ensures that while processing the query, the role-specific RBAC permissions are enforced on top of the role-specific PPVKG confidentiality policies.

5 Use Case in the Healthcare Domain

We now describe the main elements of a use case in the healthcare domain on which we carried out our experimental evaluation. We detail the VKG-related aspects of the use case and discuss a categorization of relevant roles.

VKG Specification and Dataset. For our experimentation, we make use of the *Medical Information Mart for Intensive Care* (MIMIC)-III clinical dataset [13], which aggregates de-identified healthcare records spanning 46,000 unique patients and over 60,000 intensive-care unit admissions across two ICU systems at Boston's Beth Israel Deaconess Medical Center (2001-2012). To standardize this heterogeneous data for semantic processing, we employed the open-source MIMIC-OMOP ETL Tool[5] for conversion into the *Observational Medical Outcomes Partnership Common Data Model* (OMOP-CDM)[6]. This community-developed standard normalizes observational health data structures to facilitate reproducible evidence generation through systematic analysis [1]. To represent the information of OMOP-CDM at the semantic level, me make use of the *FHIR Model Ontology* specified in the OWL ontology language[7].

For relating the FHIR Ontology to OMOP-CDM, we adopted the R2RML-compliant mappings developed by Xiao et al. [22], which establish precise correspondences between OMOP-CDM schemas and *Fast Healthcare Interoperability*

[5] https://github.com/MIT-LCP/mimic-omop
[6] https://ohdsi.github.io/CommonDataModel
[7] http://build.fhir.org/fhir.ttl

Table 1. Mapping of roles to accessible FHIR classes

RBAC roles	**Role name**	**Accessible FHIR class**
Physician/Nurse	r_1	all classes
Receptionist	r_2	*Encounter*, *Practitioner*
Researcher	r_3	all classes (since data are de-identified)
Data Administrator	r_4	*Encounter*
Pharmacist	r_5	*Prescription*, *Observation*
Patient	r_6	no class

Resources (FHIR) RDF representations. These mappings, executable within the Ontop platform, maintain a predominantly direct alignment where OMOP elements map to singular FHIR components (e.g., `person_id` maps to `Resource.id`). We observe, however, that certain temporal constructs necessitate intermediate blank node generation and therefore have a more complex structure, and that the mappings intentionally exclude redundant OMOP elements to optimize semantic representation, such as omitting duplicate temporal fields.

Role Definitions for the Hospital Domain (MIMIC-III Dataset). Using the MIMIC-III clinical care dataset, we define representative hospital roles and their access scopes for the RBAC framework (See Table 1):

– **Clinician (Physician/Nurse):** Full access to patient-level clinical data including demographics, lab results, medication orders, and clinical notes.
– **Receptionist:** Access limited to non-clinical patient demographics (e.g., contact information, appointment records) with exclusion of medical data.
– **Researcher:** Access to data of any type, provided they are de-identified.
– **Data Administrator:** Access to current medical procedures, and data for system maintenance, excluding patient-level clinical data.
– **Pharmacist:** Access to medication-focused data (e.g., medication order, dosage, and allergy) without direct patient identifiers.
– **Patient:** Access to their own records, but not to those of other patients.

5.1 Implementation

To implement the RBAC-PPVKG framework, we utilize the open-source implementation of PPVKG described in [2], based on Ontop. Our system operates over an RBAC-PPVKG specification $\mathcal{H} = \langle \mathcal{E}, \mathcal{R}, \mathsf{RP}, \mathcal{U}, \mathsf{UR} \rangle$, where $\mathcal{E} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M}, \mathcal{P} \rangle$, and creates, for each role $r \in \mathcal{R}$, a separate copy $\textsc{EncodeP}(\mathcal{M}, \mathsf{RP}(r))$ of the mapping $\mathcal{M}$ in which all role-specific policies $\mathsf{RP}(r)$ are embedded.

To merge these into a unified, role-aware mapping $\mathcal{M}_{\mathcal{H}}$, we developed a Java utility class called `MappingCombinerRBAC`, which takes as input a file with multiple paths to a mapping file and an associated role. It then implements $\textsc{EncodeR}$ by parsing the mappings and injecting directly into the SQL `WHERE` clause of the source part of each mapping assertion conditions that make use of Ontop's built-in Boolean function `ontop_contains_role(`*`roleName`*`)`. When Ontop is configured with `ontop.authorization=true`, this function checks if the user role specified in the `x-roles` HTTP header equals *roleName*. As a result, such conditions encode RBAC that activates the mapping assertion according to the user role.

To execute queries against this RBAC-enforced mapping, we developed another Java class called `QueryExecutor`, which allows users to authenticate with their role and issue SPARQL queries. It issues an HTTP POST request to the Ontop endpoint using `HttpURLConnection`, setting headers such as `Content-Type`, `Accept`, and most importantly, `x-roles` to specify the user's role. One can also use a reverse proxy to let the user authenticate, inject these HTTP headers and send the modified HTTP request to the Ontop WebAPI.

The Ontop engine evaluates the query while applying the embedded access control logic from the mappings. Results are returned in SPARQL-JSON format. For our evaluation, we executed queries across different roles over multiple iterations to compute and report the average query execution time.

5.2 Experimental Evaluation

For our evaluation, we examine the impact of six roles and their associated policy sets on query execution, specifically analyzing how they affect the number of query results returned as well as the overall query evaluation time using the setup described above.

We employ the following seven denial policies (expressed over FHIR)[8]:

$p_1\colon \forall x.\forall y.\forall z.\mathit{Patient.gender}(x,y) \wedge \mathit{Patient.address}(x,z) \rightarrow \bot$
$p_2\colon \forall x.\forall y.\mathit{Prescription}(x) \wedge \mathit{Prescription.subject}(x,y) \rightarrow \bot$
$p_3\colon \forall x.\forall y.\forall z.\mathit{Patient.condition}(x,y) \wedge \mathit{Condition.code}(y,z) \rightarrow \bot$
$p_4\colon \forall x.\forall y.\forall z.\mathit{Practitioner.qualification}(x,y) \wedge \mathit{Practitioner.qualification.code}(y,z) \rightarrow \bot$
$p_5\colon \forall x.\forall y.\forall z.\mathit{Procedure.code}(x,y) \wedge \mathit{Procedure.performedDateTime}(y,z) \rightarrow \bot$
$p_6\colon \forall x.\forall y.\forall z.\mathit{Observation.valueQuantity}(x,y) \wedge \mathit{Quantity.value}(y,z) \rightarrow \bot$
$p_7\colon \forall x.\forall y.\forall z.\mathit{Encounter.location}(x,y) \wedge \mathit{Location.name}(y,z) \rightarrow \bot$

They are assigned to roles r_1–r_6 as specified below, coherently with Table 1:

r_1: All data are accessible. Hence, no policy applies.
r_2: p_2, p_3, p_5, and p_6.
r_3: All data are de-identified. Hence, no policy applies.
r_4: p_1, p_2, p_3, p_4, p_5 and p_6.
r_5: p_1, p_3, p_4, p_5 and p_7.
r_6: p_1, p_2, p_3, p_4, p_5, p_6, and p_7.

Notice that, since patients can access only information about themselves, and in RBAC-PPVKG policies are enforced based on roles (and not individual users), in order to avoid that a user accesses the data of other patients, we also have to forbid that they access their own data. Hence, for role Patient (r_6) we have to activate all policies. This represents a limitation of utility (in line with the tradeoff between utility and privacy).

[8] Additional policies are available in our Git repository.

Table 2. Impact of privacy policies with roles on execution times (in ms) and number of query results

	q_1		q_2		q_3		q_4		q_5		q_6	
Pol.	time	#res.	time	#res.	time	#res.	time	#res.	time	#res.	time	#res.
None	815	4730	1655	34	15282	4547	1042	944	18136	5155	1253	93
r_1	820	4730	1668	34	15376	4547	1186	944	18198	5155	1498	93
r_2	823	4730	490	0	1709	0	298	0	18193	5155	1499	93
r_3	828	4730	1664	34	15372	4547	1184	944	18194	5155	1496	93
r_4	212	0	491	0	1702	0	285	0	1191	0	1493	93
r_5	227	0	496	0	15371	4547	1181	944	1189	0	158	0
r_6	223	0	493	0	1692	0	286	0	1159	0	154	0

We have chosen six representative SPARQL queries, some of which are adapted from [22], covering a variety of clinical use cases. For each query, we have specified the roles for which it is answered, according to the policies:

q_1: Male patients with inpatient admissions lasting ≥ 5 days. – r_1, r_2, and r_3
q_2: Patients who delivered a baby. – r_1 and r_3
q_3: Patients taking Trazodone. – r_1, r_3 and r_5
q_4: Patients with an HbA1c $\geq 10\%$. – r_1, r_3 and r_5
q_5: Specialty of a Practitioner. – r_1, r_2 and r_3
q_6: Patient's treatment location name. – r_1, r_2, r_3 and r_4

Our experiments demonstrate that data can be effectively hidden based on user roles, validating the correctness of role-based access control within the PPVKG framework. Furthermore, the results show that enforcing role-based privacy policies introduces only a modest overhead in query execution time, thereby confirming the practical feasibility of such mechanisms for privacy-preserving query answering. As expected, execution times are notably reduced in cases where the applied policies yield empty query results, due to early pruning of the result set during query evaluation.

6 Conclusions

In this paper, we presented an extension of the PPVKG framework to support Role-Based Access Control (RBAC), enabling privacy-preserving query answering that adapts to the access privileges of different user roles. Our approach enforces the principle of least privilege by injecting role-specific policy constraints directly into the VKG mappings, ensuring that users can only access ontology classes and properties permitted by their assigned roles. Our experiments confirm that this mechanism effectively restricts access according to roles, while introducing only minimal query performance overhead. These findings validate the practical applicability of role-based privacy enforcement in real-world knowledge graph settings. For future work, we aim to extend the RBAC-PPVKG framework by incorporating *Hierarchical Role-Based Access Control*, enabling structured role inheritance and more flexible policy management.

Acknowledgments. This research has been partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation, by the HEU project CyclOps (GA n. 101135513), by the Province of Bolzano and FWF through project OnTeGra (DOI 10.55776/PIN8884924). by the Province of Bolzano and EU through projects ERDF-FESR 1078 CRIMA, and ERDF-FESR 1047 AI-Lab, by MUR through the PRIN project 2022XERWK9 S-PIC4CHU, and by the EU and MUR through the PNRR project PE0000013-FAIR.

References

1. Baura, D., Calvanese, D.: Assessing privacy requirements for controlled query evaluation in OBDA. In: Proc. of the 22nd Int. Conf. on Modeling Decisions for Artificial Intelligence (MDAI). Lecture Notes in Computer Science, Springer (2025)
2. Baura, D., Calvanese, D., Marconi, L.: Implementing controlled query evaluation in OBDA. In: Proc. of the Joint Ontology Workshops Episode 10: The Tukker Zomer of Ontology (JOWO). CEUR-WS.org, vol. 3882. CEUR Workshop Proceedings (2024), https://ceur-ws.org/Vol-3882/st4dm-1.pdf
3. Bonatti, P.A., Sauro, L.: A confidentiality model for ontologies. In: Proc. of the 12th Int. Semantic Web Conf. (ISWC). Lecture Notes in Computer Science, vol. 8218, pp. 17–32. Springer (2013). https://doi.org/10.1007/978-3-642-41335-3_2
4. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL queries over relational databases. Semantic Web J. **8**(3), 471–487 (2017). https://doi.org/10.3233/SW-160217
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning **39**(3), 385–429 (2007). https://doi.org/10.1007/s10817-007-9078-x
6. Cima, G., Lembo, D., Marconi, L., Rosati, R., Savo, D.F.: Controlled query evaluation in ontology-based data access. In: Proc. of the 19th Int. Semantic Web Conf. (ISWC). Lecture Notes in Computer Science, vol. 12506, pp. 128–146. Springer (2020). https://doi.org/10.1007/978-3-030-62419-4_8
7. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. W3C Recommendation, World Wide Web Consortium (Sep 2012), http://www.w3.org/TR/r2rml/
8. European Parliament and Council of the European Union: General data protection regulation (GDPR). regulation (EU) 2016/679 of the European Parliament and of the Council. https://eur-lex.europa.eu/eli/reg/2016/679/oj (2016), articles 5(1)(c) and 32(1)(b)
9. Ferrari, E.: Role-based access control. In: Access Control in Data Management Systems, pp. 61–75. Springer (1992)
10. Glimm, B., Ogbuji, C.: SPARQL 1.1 entailment regimes. W3C Recommendation, World Wide Web Consortium (Mar 2013), http://www.w3.org/TR/sparql11-entailment/
11. Grau, B.C., Kharlamov, E., Kostylev, E.V., Zheleznyakov, D.: Controlled query evaluation for Datalog and OWL 2 Profile ontologies. In: Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI). pp. 2883–2889. AAAI Press (2015), https://ijcai.org/Abstract/15/408

12. Harris, S., Seaborne, A.: SPARQL 1.1 query language. W3C Recommendation, World Wide Web Consortium (Mar 2013), http://www.w3.org/TR/sparql11-query
13. Johnson, A.E., Pollard, T.J., Shen, L., Lehman, L.w.H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Anthony Celi, L., Mark, R.G.: MIMIC-III, a freely accessible critical care database. Scientific Data **3**(1), 1–9 (2016)
14. Lembo, D., Rosati, R., Savo, D.F.: Revisiting controlled query evaluation in description logics. In: Proc. of the 28th Int. Joint Conf. on Artificial Intelligence (IJCAI). pp. 1786–1792. ijcai.org (2019). https://doi.org/10.24963/IJCAI.2019/247
15. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language profiles (second edition). W3C Recommendation, World Wide Web Consortium (Dec 2012), http://www.w3.org/TR/owl2-profiles/
16. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics **10**, 133–173 (2008). https://doi.org/10.1007/978-3-540-77688-8_5
17. Sandhu, R., Ferraiolo, D., Kuhn, R.: The NIST model for role-based access control: Towards a unified standard. Tech. rep., NIST (2000), also known as ANSI/INCITS 359-2004
18. Sandhu, R.S.: Role-based access control. In: Advances in Computers, vol. 46, pp. 237–286. Elsevier (1998)
19. Voigt, P., von dem Bussche, A.: The EU General Data Protection Regulation (GDPR) – A Practical Guide. Springer (2017)
20. Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-based data access: A survey. In: Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI). pp. 5511–5519. IJCAI Org. (2018). https://doi.org/10.24963/ijcai.2018/777
21. Xiao, G., Lanti, D., Kontchakov, R., Komla-Ebri, S., Güzel-Kalayci, E., Ding, L., Corman, J., Cogrel, B., Calvanese, D., Botoeva, E.: The virtual knowledge graph system Ontop. In: Proc. of the 19th Int. Semantic Web Conf. (ISWC). Lecture Notes in Computer Science, vol. 12507, pp. 259–277. Springer (2020). https://doi.org/10.1007/978-3-030-62466-8_17
22. Xiao, G., Pfaff, E.R., Prud'hommeaux, E., Booth, D., Sharma, D.K., Huo, N., Yu, Y., Zong, N., Ruddy, K.J., Chute, C.G., Jiang, G.: FHIR-Ontop-OMOP: Building clinical knowledge graphs in FHIR RDF with the OMOP Common Data Model. J. of Biomedical Informatics **134**, 104201 (2022). https://doi.org/10.1016/J.JBI.2022.104201