SN Computer Science (2020) 1:335
https://doi.org/10.1007/s42979-020-00338-1

ORIGINAL RESEARCH

An SQL Domain Ontology Learning for Analyzing Hierarchies of Structures in Pre-Learning Assessment Agents

Kennedy E. Ehimwenma[1,2] · Paul Crowther[3] · Martin Beer[3] · Safiya Al-Sharji[4]

Received: 27 March 2020 / Accepted: 17 September 2020

Abstract
This paper presents the use of description logics (DL) in the definition and development of a Structured Query Language (SQL) domain ontology for a multi-agent based pre-assessment system. Description logics is a knowledge representation language for defining terms or classes, the relationships between classes, their instances, including individuals and literals. In a formal school curriculum, modules of learning are inter-dependent. So, teaching and learning follows an ordered sequence of learning from lower-level module(s) to higher-level ones. This process enables students to gain mastery of lower-level materials before moving up the ladder to higher-level learning. To describe an SQL ontology and its representation for a multi-agent based system application, this paper uses a description logic language to present the organization of learning modules into *DesiredConcept* < ***D*** >, *PrerequisiteConcept* < ***C*** > and *LeafNodes* < ***N*** > as well as their associated relationships, namely, *hasPrerequisite* and *hasKB* between the learning modules. The paper thus presents a TBox and an Abox of a DL ontology and further transformation into a first-order predicate for a multi-agent based system that was implemented in Jason.

Keywords Knowledge representation · Description logics · First-order logic · SQL ontology · OWL · Multi-agents · Teaching and learning · Semantic web

Introduction

In teaching and learning, activities are arranged in an ordered sequence from known-to-unknown. Also, in a knowledge domain, concepts do not exist in isolation. Like nodes in a semantic net, unit of lessons are linked to one another with common knowledge boundaries or properties that exists in-between them. On one hand, one learning unit can be established to have a relationship with an immediate higher-level unit; and on the other hand, could have a relationship with an immediate lower-level unit. This is because the successful learning of a concept and a unit of lesson is dependent on some prerequisites knowledge. Like vertices in a graph network, units of lessons can be viewed as a connection of nodes representing learning structures. As stated in [17] vertices represent objects, such as; people, houses, cities, courses, concepts, etc.; that are modelled as nodes in a knowledge graph. Ontologies are models of information in a domain of interest. To build ontologies, a description logic (DL) language is required for the formal specification of the chosen terms, class names, class instances or individuals, literals and the relationships that exists in-between them in the ontology.

A DL as a representational language for defining ontologies from scratch is hereby used for: i) the analysis of the DL defined SQL ontology, ii) organization of learning modules as instances of the chosen

✉ Kennedy E. Ehimwenma
kehimwen@kean.edu

Paul Crowther
crowtherpaultas@gmail.com

Martin Beer
mdb.shu@gmail.com

Safiya Al-Sharji
safiya.alsharji@hct.edu.om

[1] Department of Computer Science, Wenzhou-Kean University, Wenzhou, China

[2] Department of Computer Science, International College, Hunan University of Arts and Science, Hunan, China

[3] Department of Computing, Sheffield Hallam University, Sheffield, UK

[4] Department of Information Technology, University of Technology and Applied Sciences, Muscat, Oman

Published online: 13 October 2020

terms—*DesiredConcept* $< \boldsymbol{D} >$, *PrerequisiteConcept* $< \boldsymbol{C} >$ and *LeafNodes* $< \boldsymbol{N} >$—as well as the associated properties—*hasPrerequisite* and *hasKB*—that exists between them. While the *hasPrerequisite* property presents the relationships between $< D >$ and $< C >$; *hasKB* on the other hand, depicts the relationships between the $< C >$ and $< N >$ as well as between $< D >$ and $< N >$. This conceptualization shows the hierarchy of a structured learning content which support students to study all the *LeafNode* $< \boldsymbol{N} >$ instances that are related to a chosen *DesiredConcept* $< \boldsymbol{D} >$ if all pre-assessments are passed or to study any *LeafNode* $< \boldsymbol{N} >$ instance(s) that are related to the *PrerequisiteConcept* $< \boldsymbol{C} >$ if a given pre-assessment node is failed. In this research study, the domain of SQL/databases of learning was chosen for the purpose of improving students' learning. This is because SQL as reported in [15] is a subject that has posed serious learning difficulties to students.

Description logics is a system of both a TBox (Terminology Box) and an ABox (Assertion Box). In [39] a TBox is a set of classes [terms] and property [relationship] descriptions, and axioms [rules] establishing equivalence and subsumption relationships between classes and properties; and an ABox which describes the state of an application domain by asserting that certain individuals are instances of certain classes and that certain individuals are related by a property. In this paper, we define the terms which gives class assertions to SQL modules, their properties, and constraints which renders the DL ontology to be extendible through the specification of a minimum cardinality on the properties between class names. The ABox counterpart asserts the instances of the defined classes (e.g. C(a) where a is the individual that belongs to the class C) and the relationships between instances as e.g. r(a, b) such that $r \in R$. To ensure compliance with the web ontology language (OWL) DL ontology, we validated the formalized ontology by: (i) its visualization on the Protégé [28] ontology editor, and (ii) through consistency checks by running the *FACT++* reasoner. In furtherance, was the rendering of the built ontology in Jena Turtle format [1] and output of the individual members of the ontology and their properties as defined in OWL2 DL.

Contribution of this Paper

The contribution of this paper are therefore as follows:

i) To demonstrate the use of description logics in the definition of a structured learning content.

ii) To define the relationships between learning modules in a sequential order such that teaching and learning can be supported using multiagent systems.

iii) To show the significance of reasoning with DL language in the organization and selection of materials for improvement of student learning.

iv) To show a representation of classes, individual members, literals and their relationships in first order predicate beliefs for multiagents.

v) To analyze the ontology and visualize the linked nodes in their ordered hierarchy.

This section continues with ontology languages and formalized methods for ontology representation. Section II presents research works on multi-agent systems, OWL DL and ontologies. Section III describes the ontological methodology as applied in the development of the SQL ontology. This includes classes and their relationships for the ontology. In section IV, a SQL learning content is presented in a structured hierarchy of learning and their representation in DL. This covers the arrangement of learning modules in a manner that can support effective management of students' learning and progress. Section V discusses the construction of the ontology with the use of Protégé and its rendering in Turtle syntax as well as the transformation of the rendered ontology into predicate logic form for multiagents in Jason [13] agent language. Section VI conclusions and further work.

Ontology

As mentioned earlier, ontologies describe things, terms or classes, class instances or individuals, and the relationships amongst them. A relationship can be one-to-one or one-to-many. Given a domain of interest, ontology is an explicit specification of a conceptualization: objects *e.g. robin*, concepts *e.g. FlyingBird*, and other entities in the domain *e.g. Birds* and the relationships that exist amongst them, including constraints over those objects [26, 27] as well as the information about an object itself [29] *e.g FlyingBird(robin)*. In [35], ontology is a 5-tuple definition $O =< C;R;F;A;I >$ where:

`C: finite set of named concepts.`

R: finite set of binary relations among concepts.

F: functions that relates concept and relations.

A: set of axioms that are valid in the conceptualisation.

I: set of individuals belonging to a domain.

From the foregoing, let a named concept **C** be *desired_Concept* $< D >$ [23] for our SQL ontology; and the set of binary relations or properties **R** be *hasPrerequisite*, *hasKB*, *hasContent*, and *isPrerequisiteOf* which exists between individuals **I**. With the *hasPrerequisite* relation; we state an example of a binary relation as *insert hasPrerequisite select*, and the inverse as *select isPrerequisteOf insert*. Both statements are valid relations in a structured SQL learning curriculum (details in subsequent sections).

Ontology Languages

Ontology languages such as *RDF* (Resource Description Framework), *RDFS* (Resource Description Framework

Schema), and *OWL* (Web Ontology Language) has undergone several stages of developments. In this section, we look at the standardized languages i.e. *RDF*, *RDFS*, *OWL* and *OWL2* (Web Ontology Language 2); and the use of OWL2 DL in the establishment of relationships between objects of learning in an SQL ontology for a Multi-agent system (MAS).

RDF: Resource Description Framework

RDF is a standard model for data interchange on the Web [5]. RDF extends the linking structure of the Web by using URIs (uniform resource identifiers) to name the relationships between things as well as the two ends of the link to form "triples". This linking structure forms a directed labelled graph where edges represents named links between two resources or graph nodes [5]. RDF as triples (a, P, b) or set of triples which are expressed as logical formulas of the form $P(a, b)$ are therefore binary statements in which the binary predicate P relates the subject a to object b. The relationships or graphical connection between a node subject a and a node object b via a predicate P is a semantic net. While RDF is very scalable, it is not very expressive and as such does not provide support for much semantics [7]. Furthermore, RDF is not data format, but a data model with a choice of syntaxes for storing data [19]. RDF has been given the syntax of XML [5].

RDFS: Resource Description Framework Schema

RDFS is object oriented in its nature. This implies that it is fundamentally about describing classes of objects. RDFS supports semantics of data by class and property descriptions, class hierarchies and inheritance, and property hierarchy. It gives flexibility to the definition of data in that a data of a particular class may be expressed to have various type declaration i.e. *RDFS: type*; or different property declaration which is *RDFS: property*.

OWL: Web Ontology Language

Unlike RDF (see Fig. 1), OWL1 which is also known as *OWL* (web ontology language) is an increasingly expressive language. OWL is based on *SHOIN(D)* and it is NExpTime-complete. One of such expressiveness is its power to specify property values (or constraints) and validate relationships while maintaining upward compatibility with RDF and RDFS. OWL is a formalism that is based on DL and are used in representing ontologies for semantically based applications [18]. OWL has three sublanguages, namely, OWL Lite, OWL DL and OWL Full (*for details see* [8]). Of these three sublanguages, OWL DL is the language of the semantic web. OWL DL constructs have restrictions such as: (i) a class cannot be both an individual and a property (ii) a property cannot be an individual as well as a class [34].

Fig. 1 Comparison of RDF, RDFS and OWL languages

RDF	RDFS	OWL	OWL2
*Domain independent. *States fact in triples and establishing the relation between two ends.	*Provide mechanism for defining specific domain. *States class and property relation. *Declares class and subclasses in subsumption, supports property and subproperty, domain and range restriction. *Logical combination beyond its use.	*Compatible with several existing ontology languages e.g. OIL, DAML + OIL. *Extends RDF fact stating ability, and RDFS class and property structure ability. *Declares class and subclasses in subsumption hierarchy. *Classes can be logical combinations (intersection, union, negation) of other classes, or as enumeration of other specific object. *Extends RDFS by declaring properties as transitive, symmetric, functional or inverse. *Expresses disjoint, equivalence, individuality of object, quantification and value restriction. *Relies on XML schema (xsd) for listing datatypes. *Based on SHOIN(D) and NExpTime-complete.	*Compatible with OWL. *Extends and Improves datatype handling. *Additional property and addresses qualified cardinality restriction. *Simplified meta-modelling and extended annotation. *Primary exchange syntax is RDF/XML. Like OWL, OWL2 has other syntaxes e.g. Turtle, XML, RDF graph, Manchester syntax. *Has three profiles OWL2 EL, OWL2 QL, OWL2 RL with trade-offs for expressivity. *Based on STROIQ(D) and 2NExpTime-complete.

OWL2 and OWL2 Profiles

Like OWL, *OWL2 DL* which is informally referred to as OWL2 is an ontology language for the Semantic Web; and an extension and revision of OWL expressions. OWL2 DL is based on *SROIQ(D)* and it is 2NExpTime-complete. Designed to facilitate upward ontology development, OWL2 supports a variety of serialization formats e.g. Manchester syntax, RDF/XML, TURTLE, OWL/XML, etc., and exchange of ontologies. For the efficiency of reasoning, OWL2 has traded off some expressive power of OWL DL [3]. The trade-off is the result of three sub-languages known as the OWL2 Profiles, namely OWL2 EL, OWL2 QL, and OWL2 RL. Each is more restrictive than OWL DL, and each trades off different aspects of OWL's expressive power in return for different computational and implementational benefits. Like its OWL DL counterpart, OWL2 DL also has a restriction—which is that a name is not used for more than one property type i.e. object, datatype or annotation property [6].

1. OWL2 EL: This is based on EL++ [9], which is geared towards scalable reasoning in a TBox (i.e., polynomial-time reasoning (PTime-complete) for most inference tasks such as classification). Some of the expressive features not supported in OWL2 EL with respect to OWL DL are, namely class negation, inverse object property, universal quantification to a class expression or data range, self restriction, cardinality restriction, and disjoint property [6, 31] .
2. OWL2-QL: Based on DL-Lite (NLogSpace-complete) [14], OWL2-QL is geared towards scalable query answering in an ABox (when dealing with lots of instance data and a relatively simple TBox). Expressively, features of Entity-Relationship and UML diagrams can be represented [6]. As against OWL2 DL, a list of axiomatic features not supported are universal and existential quantification, self restriction, cardinality restriction, disjoint property, transitive properties, individual equality, negative property assertions, and property inclusion or property chaining [31].
3. OWL2-RL: This is based on Description Logic Programs (DLP) PTime-complete [24], OWL2-RL has an expressivity that subsets OWL2 DL. OWL2-RL supports scalable reasoning in their applications without loosing its power of expressiveness. It also supports RDF(S) applications that requires added expressivity from OWL2 [6]. However, OWL2-RL does not support disjoint union of classes, reflexive object property, universal quantification and negation on left side of $\sqsubseteq$ symbol; and no $\exists$ and $\sqcup$ symbols on the right side of the $\sqsubseteq$ symbol [31].

From the foregoing, OWL2 DL covers a wide range of axiomatic features than its profiles. On this basis, this paper presents the use of OWL2 DL for the definition of classes and property relations for an SQL ontology domain before showing the instances of class names and their subsequent visualization.

TBox and ABox

The TBox and an ABox which are used to describe two-different but-related kinds of statements for ontologies together make up a knowledge base. Reference [2] states that knowledge representation (KR) systems that are based on DLs are made up of two parts, namely, TBox and ABox; such that the ABox is the complement of the terms defined in the TBox. A DL language allows the formal specification of the concepts and their relationships in a domain. While classes may comprise a set of individuals, literals are themselves set of individuals denoting data values such as strings, and integers [3].

With the formalized logic symbols (Fig. 2), related terms in a domain are defined, and relevant properties are established in-between terms as well as possible quantification and constraints specification on the terms with respect to their properties. The following example,

$$\mathsf{Human} \sqcap \neg \mathsf{Female}\,(\exists \mathsf{married}.\mathsf{Doctor}) \sqcap (\forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Professor}))$$

defines the concept of *A man that is married to a doctor, and all of whose children are either doctors or professors* [10]. This axiom which is from the human factor domain, is a TBox representation with classes as Human, Female, Doctor, and Professor. The axiom has two properties, namely, *married* and *hasChild* with *existential quantification* and

DL Symbols	Read
$\sqcap$	*and*
$\sqcup$	*or*
$\neg$	*not*
$\forall$	*for all*
$\exists$	*there exists*
$\equiv$	*equivalence*
$\leq$	*max. cardinality*
$\geq$	*min. cardinality*
:	*concept assertion*

Fig. 2 Description logic ALC symbols

universal quantification, respectively. Reference [40] states that TBox contains *intensional* knowledge in the form of terminology or taxonomy and it is built through declarations that describes general properties of concepts. The "terminology" in a TBox denotes a hierarchy of structure that is built to provide an *intensional* representation of the domain of interest [40]. Thus, a TBox describes the vocabulary that make up a knowledge base in an application domain. Basically, this vocabulary are the terms (that comprises the set of individuals) plus the roles (relationship between terms).

An ABox which complements a TBox are assertions about the terms described in the TBox. That is, the ABox which is a description of the world contains assertional knowledge called ground facts [41] . It asserts and introduces individuals or instances of the world and their properties or relations. Properties can be unary and binary relations. Given that C is an atomic concept, R as role concept, and a, b, and c as individuals, it follows that [11, 41]:

1. C(a) concept assertion implies that a belongs to (the interpretation of) C, where C is a class name.
2. R(b, c) role assertions implies $c \in C$ is a filler of the role R for b such that there exists the property $r \in R$.

The domain and purpose of an ontology determines the terms that are defined in a TBox. In this study, an SQL domain ontology has been defined based on the need to design a MAS based eLearning system that can support adaptive prediction of learning materials to students after some pre-assessments. The SQL DL ontology has the term *DesiredConcept* which represents any module or topic that a student may enter at the beginning of an episode of learning. With a unary property in first-order predicate logic, the formula *desired_Concept(a)* specifies what class the individual a belongs—which is a statement of class assertion. At any level in the ontology tree, a *desired_Concept* is a parent node that has a link to i) some immediate leafnode instances through the *hasKB* relation, and ii) to some prerequisite subclass nodes by the *hasPrerequisite* relation. While the leafnodes are the terminal nodes that represents the individual learning units, the prerequisite subclass nodes are recursively parent nodes that connects other parent nodes and leaf nodes in a top-down design approach respectively. Thus, as an example of a binary relation, the statement *hasPrerequisite(a, b)* relates two classes a and b; and the statement *hasKB(b, c)* relates a class b to some leaf node instance c.

Related Works

Works on software agents, description logics, and ontologies have been reported in [32, 34, 36, 37, 39]. In AgentSpeak_DL, [39] used DL for describing domain ontologies for agents in such a way that the AgentSpeak language was also extended for belief base updates and query. In JASDL, [32] also applied a technique that combined semantic web and agent programming. In the project, OWL API (application programming interface) was developed to provide features such as plan triggers for agents on the premise of ontological knowledge that are present in a belief base. For the CooL-AgentSpeak [36, 37] system, a set of 5 tuples were derived and were used to define parameters for ontology matching. In a series of statements, first-order logic syntax were bounded to variables and ground facts to represent belief semantics for pattern matching in agent plans. *ALC* (Attributive Language with Complements) semantics were used to define classes and properties of and for the different ontologies in the study, and Mascardi *et al.* [36] then implemented the CooL-AgentSpeak in Jason agent language. The CooL-AgentSpeak system which extends AgentSpeak_DL performed: ontology alignment, concept matching and plan substitution between two-different but-related ontologies. For example, the CooL-AgentSpeak system would query ontologies for similarity in names. For instance, whether named identifiers given in the form $< \text{oid1} >$ and $< \text{oid2} >$ for two different food ontologies is the same knowledge base. Or whether an atomic formula that was formalized as $C(t) \in \text{oid}$ was present in any given ontology id. Other functions of the system included comparisons of predicate symbols. For instance, whether the predicate symbols C and D of the formulas C(t)[o(oid1)] and D(t́)[o(oid2)] were same and equal. One other example was whether the literal that was symbolized as t unified with another literal t́ in a given logic formula during inter-agent communication. In furtherance, the work of [36] in conjunction with the OntArt artifact which is supported by the JWNL ontology matching algorithm, searched collective agents' plan library for beliefs (from individual agentś library) for a match of a given triggering_event; and when such a match was found, the relevant plan(s) were retrieved. So, depending on the architecture of a system, agents perform task that they are delegated to. In the foregoing analysis, ontologies and agents have been used to realise different services and purpose. The common practice is the use of OWL DL language for defining the semantics of ontologies for agent based systems.

For agents in a MAS to communicate, there has to be a belief unification that comes with a triggering_event. In our work, belief matching and information retrieval is one major aspect of the pre-assessment multiagent system. Much so that, the end-point is the recommendation of learning materials that are relative to a *DesiredConcept* $< D >$ (i.e. higher-level materials) through the *hasKB* property if all pre-assessments are passed; or the recommendation of lower-level materials that are linked to *PrerequisiteConcepts* $< C >$ through the *hasKB* relation. The list of recommended materials is dependent on student performance at the end of a pre-assessment episode.

Ontology Representation And Agents

In [45] an agent was defined as a computer system that is situated in some environment in which they exhibit some properties, e.g. reactivity, deliberation, etc., in order to meet their design objectives. Agents can observe and perceive the state of their environment, and can perform actions intended [42]. A group of agents that interacts in order to achieve a common goal is referred to as multi-agents.

Agent Interaction in Jason

Communication in MAS is typically based on the speech act performative [13]. For a group of agents to communication, there must be a sender, a receiver, the performative and the content as shown in the following construct: $<$ sender, illoc_force, propositional_content $>$; where the sender is an AgentSpeak atom (i.e. a simple term), meaning the name of the agent that sends the message; illoc_force is the performative, the intention of the sender; and propositional_content, is the act to be accomplished [13]. The above constructs are only executable as part of a plan i.e. a series of executable instructions.

Beliefs

Beliefs in Jason are logic based representation that holds the knowledge i.e. propositional_content that an agent has about the world. One agent can perceive the world and another can update the world. Every agent has a belief base (BB) that contains the beliefs or mental status of the agent at a given point in time. In other words, a BB is a knowledge base that comprises a set of statements that an agent can act upon [42] or information—semantic literals that agents can understand and communicate. Thus, beliefs are assertion of the agent's knowledge about its world or environment. They are similar to ABox assertional knowledge, and generally presented in predicate logic form as either predicate(object) or predicate(subject, object).

Goals

Goals can be considered as events which are needed to be achieved. They are the part of a plan that makes the entire plan to be fulfilled or completed. As such, goals are the post-conditions of a plan [13]. In Jason there are two types of goals:

1. *Achievement Goals*: Achievement goals are those prefixed by the **!** operator and they are *goals to do*. In an agent plan, the syntax is given as *!p(a)*, e.g. !write(book). Which is assigning the goal to write a book.
2. *Test Goals*: Test goals are those prefixed by the **?** operator and they are *goals to test* the *truthness* of a belief in order to retrieve the information from the agent's BB. The syntax formation is *?p(a)*, e.g. ?publisher(P). This tests whether the named individual for the identifier *P* is a *publisher*. This returns *true* if the predicate formula publisher(P) is part of the agent's beliefs.

Mental Notes

At runtime or execution of a MAS, agents are capable of creating beliefs and adding them to their BB. These kinds of dynamically-created beliefs are referred to as *mental notes*. These could be belief updates which is as a result of the changes that has occurred in the environment that the agent is part of, or as a result of the arithmetic operations performed or the messages (or percepts) passed by other agents. For instance, +percept(NewPercept) is a formula that adds a new percept to an agent beliefs; where percept is the predicate name and NewPercept is the variable name that maps to incoming arguments.

Ontology Development Methodology

Developing ontologies from scratch entails the use of ontology development life cycle methodologies which are closely knitted to the phases of object-oriented software development methods [25, 44]. Like object oriented development life cycles, ontology construction also requires systematic developmental procedure. As obtainable with SDLC (software development life cycle), methontology [25] prescribes the stages of planning, analysis, design,

implementation and maintenance for building domain ontologies; and the activities to undertake in the development process as: ascertaining the purpose, stating the scope, specifying the requirements, building the ontology, evaluation and documentation.

Purpose and Scope

SQL queries and database programming poses learning challenges to students [15, 30, 43]. As a result, several research has evolved strategies on how to reduce the amount of difficulty and challenges faced by SQL/database students. One strategy in this work was to employ the use of prior-learning evaluation and recommendation of learning. Ontology organizes knowledge systematically. As such, the content of the ontology has to be in levels of hierarchies for agents' traversal from node to node for students' pre-assessments and recommendation. To ensure a scalable development of the ontology, we defined a TBox terminology and an ABox assertion; and considered the following topics in databases as its scope: *union*, *join*, *update*, *delete*, *insert* and *select* statements.

Requirements Specification

This is the phase where we decided on the terms (or classes), and the relationships that exists in-between them. Firstly, the term *DesiredConcept* was created. This term is a class assertion C(a) of the scope of our ontology domain which represents a student's intended topic of learning. Next was the term *PrerequisiteConcept* which denotes subsequent (or subclass) nodes underneath a *DesiredConcept*. At the system′s interface, the term *DesiredConcept* also prompts a student to enter a topic (i.e. the module intended for learning) in order to start a pre-assessment episode.

Secondly, the topics e.g. *update*, *delete*, *insert*, etc., which are the instances of the class *DesiredConcept* were connected by the property *hasPrerequisite* to each other in the form *P(a, b)*; in a class-to-class relation. In like manner, the *hasKB* property was also used to connect the topics to their respective *LeafNodes* $< N >$ instances (i.e. the subtopic); in a class-to-leafnode relation. In first-order logic, the chosen property names are predicates i.e. unary property or binary relation. In object oriented class design, these predicate names has doubled as functions for the purpose of the SQL ontology.

In order to build an effective ontology for our agent-based system, opinions and information were sourced for. The combined sources of information best informed the procedure we took to organize the learning content that would support students to solidify their previous learning before higher queries. These sources included database books and materials. For example, Transact-SQL [38],

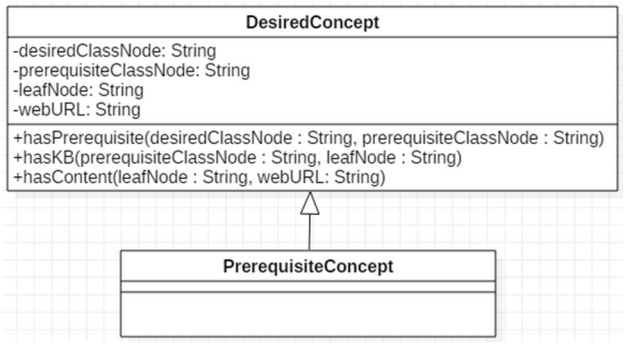


Fig. 3 Class Diagram

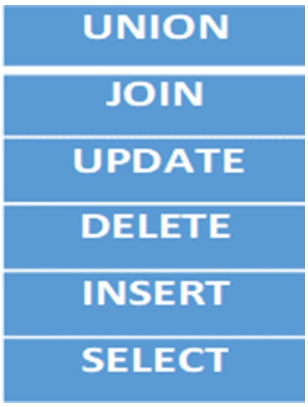


Fig. 4 Hierarchy of six SQL Modules Learning Structure. Source: [20, 23]

Database Systems [16], and higher education database curriculum. Other sources were face-to-face interaction with lecturers in databases/SQL programming and due consideration from members of the research team who are experts in databases. The consultation ensured **validation** (focus on the user's need) and **verification** (focus on the ontology and the agent based system requirements). Using a UML (unified modelling language) class diagram, we show the class DesiredConcept, the subclass PrerequisiteConcept that is dependent-on the immediate top class, and their attributes and functions (Fig. 3).

In Fig. 3, from object oriented design paradigm, the class *PrerequisiteConcept* is a derived class which inherits from the base class *DesiredConcept*. The base class has four attributes namely, *desiredClassNode*, *prerequisiteClassNode*, *leafNode* and *webURL*; and three parameterized functions which are *hasPrerequisite()*, *hasKB()* and *hasContent()* in object oriented principles. Whereas in logic based orientation, as we have seen so far, these functions are predicates. At the invocation of these functions,

two related arguments will generate binary predicate statements.

Building the DL Ontology

The next stage of this work was the formal construction of the SQL ontology using the DL language; which covered the definition of terms, existential quantification and constraint specification on the properties of the terms or class names. The SQL content of learning upon which the TBox and ABox were defined is shown in a top-down hierarchy as per level of difficulty (as was decided for this work, Fig. 4).

Given the listed modules in Fig. 4, each module in a teaching-learning process have their respective sub-units. For example, the *select* module (or topic) has several sub-units such as *selectWhere*; *selecOrderBy*; *selectDistinct*; *selectAll*, etc., for sorting and matching data queries. As the first query statement, the *select* topic and its associated sub-learning units therefore comes as the lowest statement (Fig. 4). The various sub-units of the individual topic thus constitutes respective leafNodes $< N >$ in this ontology specification.

A Formal Specification of an SQL Ontology

Visualization of a domain ontology can be preceded by a formal ontology model using a DL language. Description logic as a technique of knowledge representation (KR), is the set of background structure of knowledge that is declared for intelligent systems to function: to reason, to unify, to query, to make judgement or prediction. For our teaching-learning DL ontology, we have adopted the technique of [11] formalism. This sort of KR in artificial intelligence (AI) as ascertained in [11] is usually on methods for providing high-level description of a domain of interest

SqlNode ≡ SqlClassNode ⊓ SqlSubClassNode

LeafNode ≡ SqlSubClassNode ⊓ (∃hasQuiz.Quiz
⊓ ∃hasAnswer.Answer
⊓ ∃hasContent.WebUrl)
⊓ ¬ SqlClassNode

PrerequisiteConcept ≡ SqlClassNode ⊓ ≥ 2 hasKB.LeafNode
⊓ ((∃hasPrerequisite.SqlSubClassNode
⊓ ∃isPrerequisiteOf.SqlClassNode)
⊔ (∃hasPrerequisite.SqlSubClassNode))

DesiredConcept ≡ SqlNode ⊓ ∃hasPrerequisite.PrerequisiteConcept

isPrerequisiteOf ≡ hasPrerequisite⁻

Fig. 5 TBox Description of an SQL Domain

(or world). This is relevant in first-order predicates for building intelligent agent application.

TBox Description for an SQL Ontology

In Fig. 5 is a TBox (*hierarchical*) description [40] of concept names for a SQL domain ontology. Concept names are *named symbols* on the left-hand-side of the *equivalence* ≡ symbol in an axiom and defined on the right-hand-side are the *base symbols* [11].

Given the DL syntax ∃r.C that a thing has a role or relation r with the concept C as in ∃hasChild.Lawyer, and ∃r.{x} that a thing has some relationship with a some instances as in ∃citizenOf.{USA} [10]; then from Fig. 5, we deduce the following axiom,

$$\text{SqlNode} \equiv \text{SqlClassNode} \sqcap \text{SqlSubClassNode}$$

to define the term SqlNode to be a class node as well as a subclass node. This is because an **SqlNode** is a class node, and the same **SqlNode** can also be a subclass node. For instance, let us consider the class instance delete in the ontology tree to be a *DesiredConcept D = delete*. In one episode of pre-assessment, the delete node could be the highest class node target for learning by a student. Whereas, in another episode when the *DesiredConcept D = update* node, the delete node then becomes a subclass node *PrerequisiteConcept C* to the higher class node i.e. update. This is the case of intersection which qualifies the node that is symbolized as D to exist in dual state: as in D ≡ C. This makes the individual delete ∈ D in one instance, and in another delete ∈ C depending on a student's chosen node D of interest. Detail of this is covered in [23]. A class node represents a student's *DesiredConcept* intended to be studied upon which some pre-assessments are carried out on the student. An episode begins at a given class node, then through the class's subclass (i.e. prerequisite) node(s) and down to their terminal (leaf) nodes. The following axiom

$$\begin{aligned}\text{LeafNode} \equiv\ & \text{SqlSubClassNode} \sqcap (\exists \text{hasQuiz.Quiz} \\ & \sqcap \exists \text{hasAnswer.Answer} \\ & \sqcap \exists \text{hasContent.WebUrl}) \\ & \sqcap \neg \text{SqlClassNode}\end{aligned}$$

uses *existential quantifier* ∃ to define the term *LeafNode* as subclass nodes that have some quizzes, answers and web URLs (universal resource locator) via their respective *hasQuiz*, *hasAnswer* and *hasContent* properties, also with the classical *negation* ¬ symbol that the leaf nodes are not parent class nodes per se. The identifiers *Quiz*, *Answer* and *WebUrl* as the names depicts indicate the quizzes, answers to the quizzes, and web URL references to learning materials. The *Quiz*, *Answer* and *WebUrl* are identifiers that

holds the corresponding string arguments of the defined term *LeafNode*. As a pre-assessment system, quizzes are presented by the system to students, and students answer(s) are compared with the predefined answers in the system. Based on the outcome of this comparison, web URL are recommended at the end of the episode. The next axiom from Fig. 5 uses a *minimum cardinality* restriction of 2.

$$\begin{aligned}\text{PrerequisiteConcept} \equiv\ & \text{SqlClassNode} \sqcap \geqslant 2\text{hasKB.LeafNode} \\ & \sqcap ((\exists \text{hasPrerequisite.SqlSubClassNode} \\ & \sqcap \exists \text{isPrerequisiteOf.SqlClassNode}) \\ & \sqcup (\exists \text{hasPrerequisite.SqlSubClassNode}))\end{aligned}$$

In the axiom, the term *PrerequisiteConcept* is defined as a class node that has at least two leaf nodes connection by the *hasKB* property as well as the *hasPrerequisite* property to a subclass node and may or may not have the *isPrerequisiteOf* inverse property from the subclass node. Then, the axiom

$$\text{DesiredConcept} \equiv \text{SqlNode} \sqcap \exists \text{hasPrerequisite.PrerequisiteConcept}$$

defines the term *DesiredConcept* as class nodes that have some prerequisite class nodes (i.e. *PrerequisiteConcept)* via the *hasPrerequisite* property; and finally,

$$\text{isPrerequisiteOf} \equiv \text{hasPrerequisite}^{-}$$

which states that the *isPrerequisiteOf* relation is the inverse of the *hasPrerequisite* property.

From the DL syntax, the *DesiredConcept* is unary predicate for a student's desired concept in a first-order logic (FOL) ground facts. The *hasPrerequisite*, *isPrerequisiteOf*, and *hasKB* are binary predicates that connects two individuals. Suitable model of these ground fact statements in predicate logic constitutes belief representation for predicate logic-based agent and inter-agent communication.

Instance Declaration in Description Logics

Individuals as ascertained in [11] are not only meant to be asserted in an ABox. Also, they can be instantiated in a TBox. By implication, the SQL DL ontology defined above thus have class instances listed within the DL statements. For example, the term *DesiredConcept* is instantiated as

$$\begin{aligned}\text{DesiredConcept} \equiv\ & \{\text{insert}\} \sqcap \text{hasPrerequisite.}\{\text{select}\} \\ & \sqcap (\text{hasKB.}\{\text{selectWhere}\} \sqcap \text{hasContent.}\{\text{http} : //...\})\end{aligned}$$

which states that the individual *insert* is a desired concept that has a *hasPrerequisite* property relation with *select* which has a knowledge base (KB) through the *hasKB* property to *selectWhere* that further has a URL link *http://...* via the *hasContent* property.

ABox Assertion for an SQL Ontology

An ABox contains assertional knowledge called ground fact which are individuals and their properties [41]. Based on the SQL learning structure (Fig. 4), a list of individuals of the class *DesiredConcepts* is given as

$$\text{DesiredConcept} = \{\text{union, join, update, delete, insert, select}\}$$

and the set of query statements (i.e. the *leafNode* instances) which are the module's sub-units covered in this work as

$$\begin{aligned}\text{Leaf Node} = \{&\text{unionAll, unionDistinct, self Join, fullJoin,} \\ &\text{innerJoin, updateSelect, updateWhere,} \\ &\text{deleteSelect, deleteWhere, insertSelect,} \\ &\text{insertWhere, selectWhere, selectAll,} \\ &\text{selectOrderBy, selectDistinct}\}\end{aligned}$$

Given the *predicate(object)* and *predicate(subject, object)* data structure, and the logic formula C(a) that a belongs to the interpretation of C e.g. *father(peter)*; and R(b, c) that c is a filler for the role R for b [11], then, the following ABox assertions therefore holds to show valid representations of some individuals and their properties in their unary and binary predicates forms:

desiredConcept(update)

hasPrerequisite(update, delete)

isPrerequisiteOf(delete, update)

hasKB(update, updateSelect)

hasContent(updateSelect, WebURL)

The asserted ground (first-order) atomic formula are valid Jason agent language belief representation. As a set of beliefs, they are the agent's *knowhow* of its world [12].

Digraph Analysis of Defined SQL Ontology Model

From the SQL TBox definition, we have created different ontology model visualization so as to illustrate the knowledge modules (topics) in the domain of SQL and to validate the DL ontology. Graphical analysis of the constructed models are shown (below in the following section) as *regular* ontology and *non-regular* ontologies, respectively. The ontology models are directed graphs, and the directed edges indicate the flow of navigation between nodes.

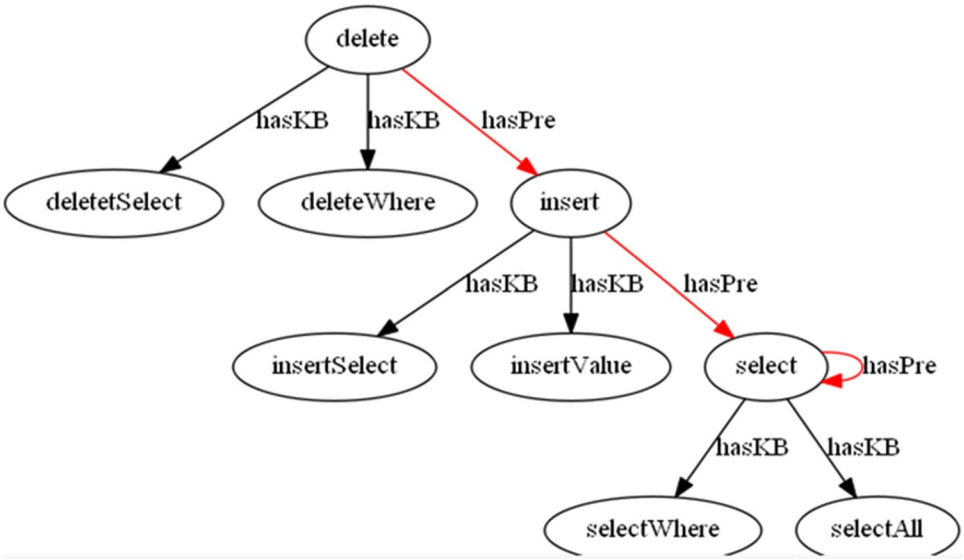


Fig. 6 A snapshot of regular ontology of two leaf nodes per parent class node. *select* node is reflexive

A Regular SQL Ontology

A regular ontology is an ontology with an equal number of leaf-nodes across all parent class nodes in a tree [21, 22]. The Fig. 6 is a *regular* ontology with a linear configuration from top to bottom showing two leaf nodes each per parent class node. An immediate-lower class node is a prerequisite to its top class node. For instance, *hasPrerequisite(delete, insert)* or *isPrerequisiteOf(insert, delete)* relation.

Non-Regular SQL Ontology Model

Recall that in the DL syntax a *minimum cardinality* constraint of at least two leaf nodes per parent class node was defined. This condition enables a class node to have two or more leaf nodes. In a *non-regular* ontology such as in Fig. 7, this condition accounted for the differences in the number of leafnodes between the *select* parent node and other parent nodes in the tree.

While the *select* parent node has four leafnodes, other parent nodes have two leafnodes each. This is also a valid representation as specified by the description in the TBox given the *minimum cardinality* for LeafNodes $< N >\geq 2$.

Unlike Figs. 6 and 7 that has a single *hasPrerequisite* property between a parent class node and its prerequisite class node, Fig. 8 presents a different model with multiple *hasPrerequisite* directed properties (or edges) from one parent class to other parent classes. This model places two parent classes at the same horizontal level, for example, the union and join nodes. But in teaching and learning, one topic or unit of lesson must be taught before the other. In that case, the Fig. 8 model invalidates the ordered sequence of the learning structure presented in Fig. 4. However, the model satisfies the TBox definition in Fig. 5. The point here is that whilst a model may validate a DL definition, it may

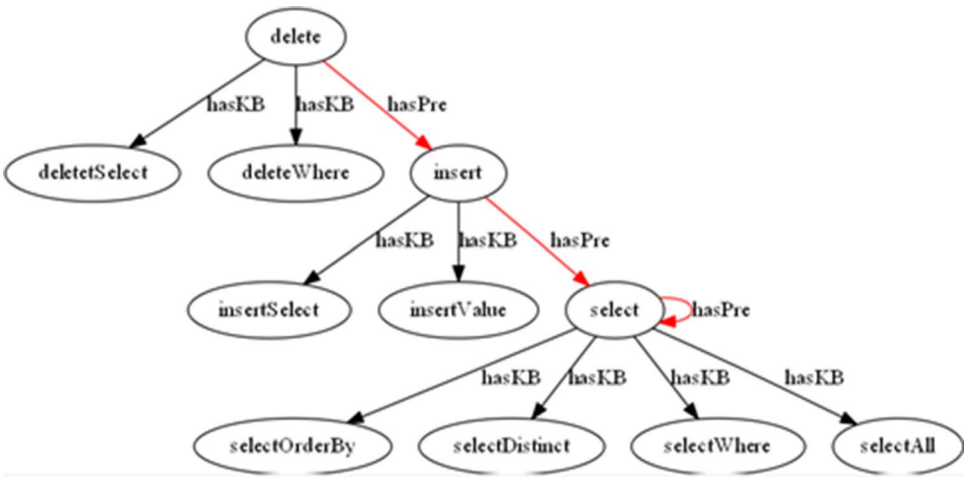
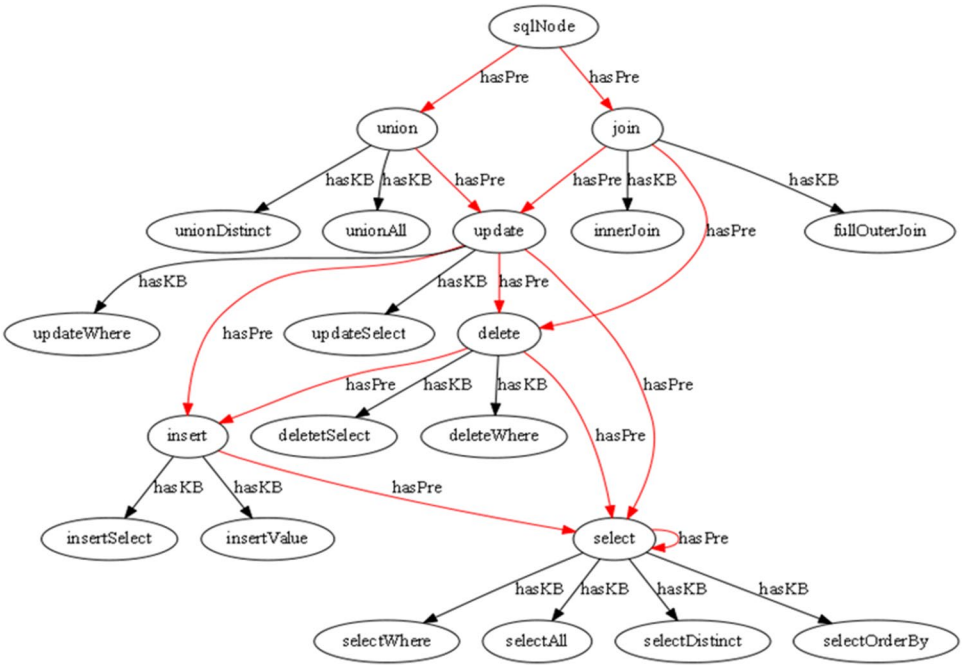


Fig. 7 A snapshot of a non-regular ontological model from the TBox



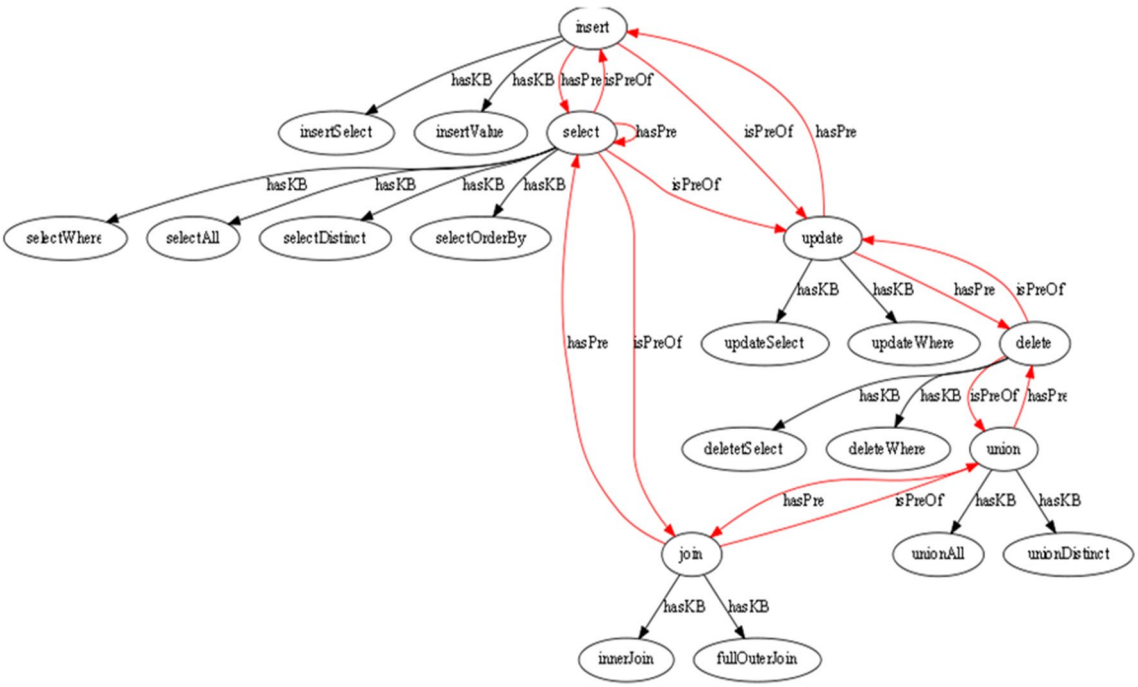


Fig. 8 A non-linear hierarchy of the SQL learning structure with multiple property relationships between parent nodes

Fig. 9 A variant ontology model of the TBox description and its navigation. But some class relationships not in the structured sequence in Fig. 4

however not satisfy the sequential process of teaching and learning as shown in the foregoing graphical models. Therefore, we must strike a synergy between a DL definition and its graphical models for teaching and learning.

Thus, face-to-face or computer assisted learning should follow an ordered sequence of learning where a learned module can support the learning of a new and unknown module seamlessly. This would help to check any gaps that may inhibit new learning. We use another ontology structure (Fig. 9) to illustrate this point. From the TBox definition that a PrerequisiteConcept is

$\equiv$ SqlClassNode $\sqcap \geqslant$ 2hasKB.LeafNode
$\sqcap$ (($\exists$hasPrerequisite.SqlSubClassNode $\sqcap \exists$isPrerequisiteOf.SqlSubClassNode)
$\sqcup$ ($\exists$hasPrerequisite.SqlSubClassNode))

then it holds that Fig. 9 is a valid visualization with the *hasPrerequisite* and its inverse object property *isPrerequisiteOf*. As with the preceding sessions, the *hasPrerequisite* shows the navigation from a top-level class node to a lower-level node; and the *isPrerequisiteOf* the link from a lower-level class node to a top-level class node. Though, this visualization satisfies the defined specification in the TBox, the question here is: does it follow the order of sequence of the structured content of learning in Fig. 4? The answer is, no; because there is a violation of the order. For example, consider the connection between the join and select which implies a random selection of any of the node as a start node for learning (Fig. 9); or that a start node can be join and thereafter a select. Ontology structures such as this can create gaps in learning. Therefore, we conclude that ontologies for learning should be devoid of random arrangements of learning modules.

Another drawback of the model of Figure 9 is the infinite loop traversal across parent class nodes and the inconsistency that may arose for not following the structured content of learning. Such inconsistency meant every knowledge engineer would subjectively pick any node (topic) as the start-node, and/or ascribe the highest level of difficulty to any arbitrary node in the design of a knowledge base for the system. Which is not to be. But suppose it does, from our observation, this would lead to the creation of inconsistent predicate ground facts assertions in the respective agent beliefs from inter-agent communication; and in-turn leads to inaccurate and inconsistent recommendation of learning material. To prevent this type of multi-agent behaviour, Figures 6 and 7 are therefore the optimal model-approach for the pre-assessment system [20].

Navigation of Ontology Nodes for Agent Goals

As stated earlier, in a standard structured-curriculum, teaching and learning is sequential, ordered, and from *known* to *unknown*. Based on the TBox, the different graphical ontology models has shown how we used a DL language in describing a body of knowledge and the relationships between sub-units using the binary properties. In a directed graph, these properties provide a sense of navigation from node to node. For instance, the binary relations (e.g. Fig. 7) showed possible navigation paths through which nodes are linked for pre-assessments. This can be established either on the strategy of: *Pre-Assessment By Immediate Prerequisite Class* or *Pre-Assessment By Multiple Prerequisite Classes* [20, 23]. The directed edges in the visualized models are the navigational paths from one node to another namely, class-to-class, class-to-subclass, class-to-leafnode, and subclass-to-leafnode. In the Pre-assessment System, the predicates i.e. unary and binary representation form the basis for Jason agents' beliefs (which is first-order logic form) and goals execution for the pre-assessment of students' SQL knowledge.

Every parent class node have their leafnodes. For instance, the *insert* class node has its leafnodes which are *insertValue* and *insertSelect*. During student learning, these are the units of lessons in which the *insert* query skills would be tested on to ascertain whether there are gaps or no gaps in a student's skill sets. Where there is no lack of skill sets, the student progresses to a higher node, in this case, *delete* in that hierarchy. As defined in the TBox (Fig. 5), all leafnodes have their respective string literals identified as *quizzes*, *answers* and *url* data that are connected by the

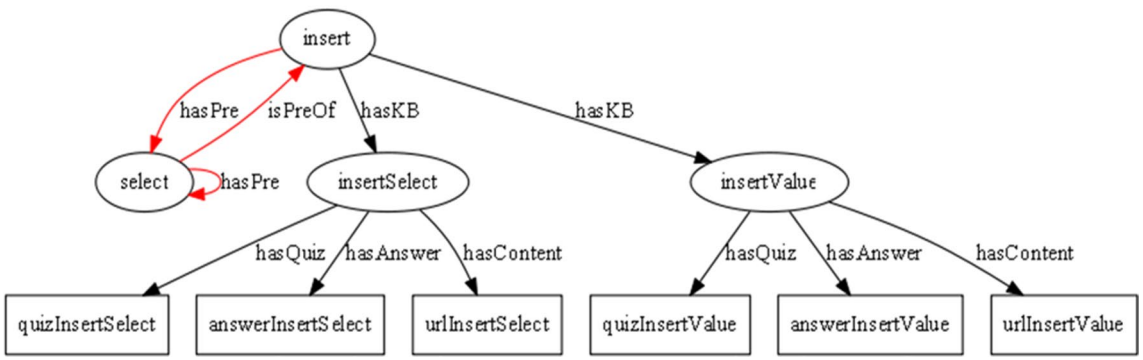


Fig. 10 The insert class example with its leaf node and String literal (or data) elements

hasQuiz, *hasAnwser* and *hasContent* properties. In Figure 10 is an explicit depiction of the axiom that connotes these Leafnodes and their literals. The literals e.g. (*quizInsertSelect*, *answerInsertSelect* and *urlInsertSelect*) shown in rectangles are variables that holds the *string* data.

The *quizInsertSelect* and *answerInsertSelect* literals are beliefs that were initialized in the BB of the agent *agSupport*: the agent that i) pre-assesses students using SQL query based quizzes, ii) take decisions on students' answer responses to quizzes, and iii) communicate the *passed(X)* or *failed(X)* predicate decision statement to the agent *agModelling* (the classifier agent) for SQL query skills classification. The classification process which is the categorization of student skill sets and recommendation for appropriate learning material(s) is a high-level reasoning procedure and has been presented using first-order logic syntax; see [20, 23].

Ontology Rendering

In the preceding sections, we presented a sequential learning structure, and analyzed and established valid relations to support effective learning. This section thus presents a snapshot of the SQL ontology constructed with Protégé [4] ontology editor where consistency checks were done with the *Fact* ++ reasoner. Furthermore, we parsed the SQL OWL ontology with Jena RDF API in Turtle format [1]. The parsed ontology showed upward compatibility of OWL DL with RDF ontology model; and finally, the

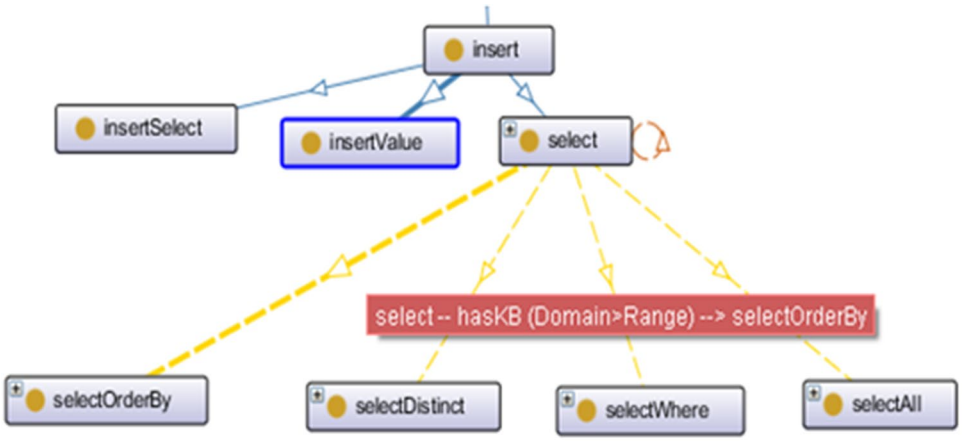


Fig. 11 A screen-shot of the hovered role assertion hasKB(select, selectOrderBy). The node Select is the Domain and the *selectOrderBy* its Range

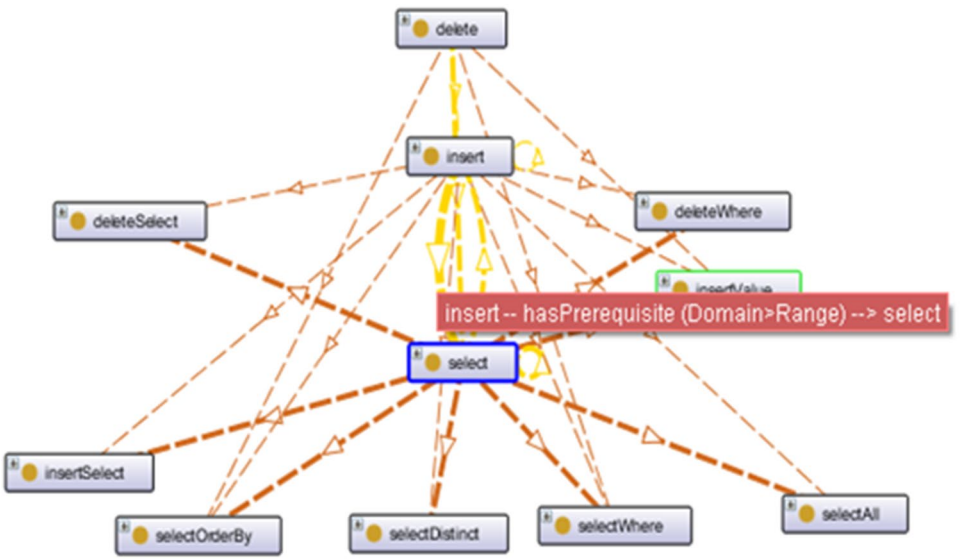


Fig. 12 The hovered axiom shows the role assertion in the statement hasPrerequisite(insert, select) and also depicts the inferred subsumption relationships on the object property hasPrerequisite which is the common property between the class names *DesiredConcept* and *PrerequisiteConcept*, and their recurrent class instances

```
<http://www.semanticweb.org/lette/ontologies/sql/delete>
        a       owl:Ontology .
:delete  a      owl:Class .
:deleteSelect  a         owl:Class ;
        rdfs:subClassOf  :delete .
:deleteWhere  a          owl:Class ;
        rdfs:subClassOf  :delete .
:insert  a               owl:Class ;
        rdfs:subClassOf  :delete .
:insertSelect  a         owl:Class ;
        rdfs:subClassOf  :insert .
:insertValue  a          owl:Class ;
        rdfs:subClassOf  :insert .
:select  a               owl:Class ;
        rdfs:subClassOf  :insert .
:selectAll  a            owl:Class ;
        rdfs:subClassOf  :select .
:selectWhere  a          owl:Class ;
        rdfs:subClassOf  :select .
:selectOrderBy  a        owl:Class ;
        rdfs:subClassOf  :select .
:selectDistinct  a       owl:Class ;
        rdfs:subClassOf  :select .
:hasKB  a            owl:ObjectProperty ;
        rdfs:domain  :delete , :select , :insert ;
        rdfs:range   :insertSelect , :deleteSelect , :deleteWhere ,
                     :selectOrderBy , :selectWhere , :selectAll ,
                      :insertValue , :selectDistinct .
:hasPrerequisite  a  owl:ObjectProperty ;
        rdfs:domain  :select , :insert , :delete ;
        rdfs:range   :select , :insert .
```

Fig. 13 Protégé OWL ontology using Turtle syntax from Jena API as given in the ABox

transformation of the *subject*, *predicate*, *object* format into first-order predicate representation.

Rendering in Jena

RDF define resources as connected graphs in their < *subject, predicate, object* > form. A class and the relationship it has with other classes are resources in RDF. Having established that Figure 6 or 7 is the optimal graphical model from our DL ontology; now, from Figure 7, let us consider a cross-section of the ontology with the following class nodes *delete*, *insert* and *select* and the relationships in-between them in order to illustrate the RDF ontology model. Though RDF data structures does not support unary predicate but are a set of triples < *a, p, b* > (Fig. 13) that can be expressed as logical formulas *p(a, b)*.

Axiom Visualization

Like the Jena API that has different output syntax; Protégé ontology editor constructs and renders ontology in different syntax e.g. RDF/XML syntax. Using the same cross-section of the ontology model in Figure 7 that comprise Delete, Insert and Select nodes; Figures 11 and 12 presents some logical axioms upon the visualization of connected edges between nodes in Protégé. While the hovered axiom in Figure 11 illustrates a parent class and its subclass relation with the *hasKB* object property, Figure 12 presents the relationship between two parent nodes with the *hasPrerequisite* object property in accordance to the DL ontology definition of Figure 5.

In Figure 11, with the named individuals, the structure depicts the hovered edge of OWL2 DL class inclusion $\{\text{select}\} \sqsubseteq \exists \text{hasKB}.\{\text{selectOrderBy}\}$. That,

$$\text{selectsubClassOf hasKBsomeselectOrderBy}$$

or from our DL definition that

$$\text{PrerequisiteConcept} \sqsubseteq \geq 2\ \text{hasKBsomeLeaf Node}.$$

The class name PrerequisiteConcept is therefore a subClassOf DesiredConcept which is in turn a subClassOf of OWL:Thing.

Similarly, as obtained in Figure 11, the OWL2 DL class inclusion for Figure 12 also state

$$\{\text{insert}\} \sqsubseteq \exists \text{hasPrerequisite}.\{\text{select}\}$$

and in general, from our DL definition as

$$\text{DesiredConcept} \sqsubseteq \exists \text{hasPrerequisite}.\text{PrerequisitConcept}.$$

To establish the backward compatibility of OWL syntax with RDF, we parsed the SQL OWL ontology rendering which was in RDF/XML to Jena API [1]:

```
import *.*;
public class TurtleOWL {
   public static void main(String[] args) {
      FileManager.get().addLocatorClassLoader(TurtleOWL.class.getClassLoader());
      Model m = FileManager.get().loadModel("C:/SqlOntology/delete.owl");
      m.write(System.out, "Turtle");
   }
}
```

Note that ontology rendering in formats such as RDF/XML and OWL/XML are in their fully qualified URIs (universal resource identifier). Jena takes these file formats and reproduce them into other formats based on the given syntax e.g. Turtle. So, in parsing the SQL OWL/XML ontology rendering in Jena with Turtle specified as the output syntax; Turtle output the ontology (i.e. the nodes and properties) in their given resource names in pairs with their (backward) compatibility languages i.e. owl and rdfs as well as the Class and subClassOf declaration of the nodes, respectively (see Fig. 13).

For instance, the statement

: insertaowl : Class;rdfs : subClassOf : delete.

is a classnode-to-classnode relationship that states *insert* is an owl class and by rdfs property, it is a subclass of *delete*. This specifies a module-to-module relationship; applicable, to all other modules in the learning structure of Figure 4.

Given the TBox and ABox assertion, and the analysis so far by the use of different graphical models; then, in Figure 14 we present the first-order ground facts representation that forms the knowledge base of the ontology agent in the multi-agent Pre-assessment System.

Inter-Agent Communication

From our SQL ontology (Fig. 14) where each statement has been annotated by the identifier [ont(sql)], the hasPrerequisite is the object property that states *hasPrerequisite(DesiredConcept, PrerequisiteConcept)*, and its counterpart hasKB is that which states *hasKB(PrerequisiteConcept, LeafNode)*. Since, every *LeafNode* < *N* > as defined in the

Fig. 14 Agents' ontological beliefs annotated with the id [ont(sql)] in Jason agentspeak syntax

```
hasPrerequisite(union, join)[ont(sql)].
        hasKB(join, selfJoin)[ont(sql)].
        hasKB(join, outerJoin)[ont(sql)].
        hasKB(join, innerJoin)[ont(sql)].
hasPrerequisite(join, update)[ont(sql)].
        hasKB(update, updateSelect)[ont(sql)].
        hasKB(update, updateWhere)[ont(sql)].
hasPrerequisite(update, delete)[ont(sql)].
        hasKB(delete, deleteSelect)[ont(sql)].
        hasKB(delete, deleteWhere)[ont(sql)].
hasPrerequisite(delete, insert)[ont(sql)].
        hasKB(insert, insertSelect)[ont(sql)].
        hasKB(insert, insertWhere)[ont(sql)].
hasPrerequisite(insert, select)[ont(sql)].
        hasKB(select, selectOrderBy)[ont(sql)].
        hasKB(select, selectDistinct)[ont(sql)].
        hasKB(select, selectWhere)[ont(sql)].
        hasKB(select, selectAll)[ont(sql)].
hasPrerequisite(select, select)[ont(sql)].
```

OWL DL ontology has some URL literals that contains learning material content; then by extension, the hasContent data property thus relates the logical expression *hasContent(LeafNode, WebUrl)*. From the foregoing, we then state as follows using symbolic logic that

$\forall X, \exists Y, \exists Z \text{hasPrerequisite}(X, Y) \implies \text{hasKB}(Y, Z) \wedge \text{hasContent}(Z, \text{WebUrl})$.

Otherwise, we have

$\forall X, \nexists Y, \exists Z \text{hasPrerequisite}(X, X) \implies \text{hasKB}(X, Z) \wedge \text{hasContent}(Z, \text{WebUrl})$;

where X is a reflexive node.

Recall that the general form of an agent plan in Jason AgentSpeak language is given as: **+triggering_event : context** <– **body** [13]. In our MAS, the atomic values in a formula such as *hasPrerequisite(X, Y)* implies messages that are communicated between agents. Simply put, when a message represented in the form *hasPrerequisite(DesiredConcept, PrerequisiteConcept)* is communicated, firstly, the agent's belief base is updated with this knowledge or belief template. For the relevant plan with respect to the updated belief to be triggered, this belief has to be part i.e. – context – of the agent's plan. The agent returns true when the belief matches or unifies with the knowledge in its belief base (BB). If the knowledge thus exists, the agent executes the relevant plan's body. Also, the receiving agent uses a test goal in the form **?***hasPrerequisite(DesiredConcept, PrerequisiteConcept)* or **?***hasKB(PrerequisiteConcept, LeafNode)* to verify whether a given formula matches any belief (updated or initialized) exists in its BB. Unlike the belief that is in a plan's context, a test goal about a belief is within a plan's body.

The ontology agent known as the *agMaterial* has two functions: i) to map and confirm the existence of ontological relations, and ii) to release relevant appropriate materials to students after pre-assessments. Now, let X be a commandline argument i.e. input from a user, and +value(X) a triggering_event for the interface agent *agInterface*. For the purpose of inter-agent communication and proper placement of arguments in their right predicate clauses such as *desiredConcept(X)* and *hasPrerequisite(X, Y)* or *answerResponseToX(X)* within the multi-agent system loop; we passed the arguments X through a single first-order predicate statement i.e. +value(X) which is the triggering_event. To remove ambiguity in every sense of it, one atomic value e.g. x ∈ X has to be passed and matched with existing predicate ground facts of each agent, respectively. This is because every agent have different duties, but the entire MAS depends on the percepts from the interface agent. So, each agent, based on their predicate name e.g. C as in C(a) or P as in P(a,b) interpret their incoming percepts according to their knowledge base representation. But the question is: during inter-agent message communication, how can the argument X in a unary predicate value(X) or *desired_Concept(X)* be passed as a ground fact into a binary predicate *hasPrerequisite(X, Y)*? Firstly, an agent must understand the knowledge it has and what it can do with it before passing the knowledge to other agents. Thus, within a plan, it is realized that an agent can pass and exchange atomic values. In the process, we tasked a given agent to adopt its perceived argument X from its triggering_event i.e. +value(X). Subsequently, in the same plan, the agent is given the task to pass the argument X into a variable location X of another receiving unary predicate as in *desiredConcept(X)* or of a binary predicate *hasPrerequisite(X, Y)* in the body of its plan. The argument X that is passed then maps to the X variable of the receiving clause (Fig. 15). Prior to this, the agent has confirmed that the value X is part of the knowledge it has its BB through a *test goal*.

```
// agent perceive X
+value(X)[source]: true
<-- .send(agReceiver, tell, desiredConcept(X));

// agent pass X into binary predicates and send
   .send(agReceiver, tell, hasPrerequisite(X, insert);

// agent asks whether the binary relation exists
   .send(agReceiver, askOne, hasPrerequisite(X, insert).
```

Fig. 15 Passing a value x from a unary predicate *p(x)* into binary relation *q(x, y)* and its unification

In the pre-assessment system, the ontology agent has the predicate logic-based ontology. It is the agent that enquires whether an incoming atomic value x ∈ X unifies with (any of) its modelled and initialized ground fact beliefs in [ont(sql)]. With the askOne performative (Fig. 15), a sending agent asks whether the ontological belief *hasPrerequisite(X, delete)* exist in the BB? Upon ascertaining that the stated belief or knowledge enquiry exists, the ontology agent automatically sends a reply back to the calling agent that .e.g. *hasPrerequisite(update, delete)* is in the [ont(sql)]. At least one ground fact in a binary predicate statement *P(a,b)* must be specified for the MAS optimal performance. For example, the value delete in the beliefs of the agent must be specified to avoid ambiguity (when it comes to context matching) from amongst the numerous beliefs that shares the same predicate or property name. Otherwise, correct plans would not be selected for classification and recommendation of learning to students.

At the end of a pre-assessment episode, and for the ontology agent to release appropriate learning materials to student users; firstly, the agent *agModelling* sends a message with the predicate statement e.g. hasKB(X, deleteSelect) to the ontology agent *agMaterial*. When this message is received,

it is added to the BB. Upon adopting the hasKB(X, deleteSelect), which is now a belief; the ontology agent uses a test goal *?hasContent(Z, WebUrl)* to enquire whether the argument WebUrl exists in its BB. Based on this confirmation, learning materials are released. From the system's interface, this process of knowledge perception, acquisition, and passing of atomic values from one predicate statement to another is given as follows: +value(X), *desiredConcept(X)*, *hasPrerequisite(X, Y)*, *hasKB(Y, Z)* and to *hasContent(Z, WebUrl)* where "+" in +value(X) is part of Jason syntax for percept acquisition.

Engaging the use of the **Pre** and **Post** conditions [33], we therefore present a step-wise belief conditions. The conditions are the facts that are true before and after an action was taken, the adoption and execution of perceived plans, and the release of materials by the ontology agent. Note that B as used in the following **Pre** and **Post** conditions depicts agent beliefs:

```
Pre: hasPrerequisite(X, Y)[source(self)] //B
```

Pre: hasKB(Y, Z)[source(self)] //B

Pre: hasContent(Z, WebUrl)[source(self)] //B

Post: +!hasPrerequisite(X, Y)[source(sender)]

Post: +!hasKB(Y, Z)[source(sender)]

Post: ?hasContent(Z, WebUrl)

Post: release material WebUrl

The step-wise process analysis depicts the relationships between individuals (or instances of classes) with the *hasPrerequisite* and *hasKB* ObjectProperty, repectively; and the *hasContent* DataProperty. The *hasContent* is the property that establishes the link between a given individual and its string type web url value. The combined prefix symbols **+!** to the predicates *hasPrerequisite* and *hasKB* indicates the adoption of a message perceived by an agent.

As reported in [21, 23], the multi-agent Pre-assessment System is made up of five agents, namely, *agInterface*, *agModel*, *agSupport*, *agModelling*, and *agMaterial* (the ontology agent). In the system, agents communicated their acquired knowledge in both unary predicate form *p(a)* and binary predicate form *p(a, b)*. For example, in the agent *agModelling*, every plan context has i) a logical formula of a desired concept as in *desired_Concept(X)* for receiving percepts from another agent, and ii) predefined predicate statement rules that are combinations of *passed(Y)* and *failed(Y)* decision statements for the classification of students' skill sets. The number of classification rules is dependent on 2^N number of LeafNodes < N > per episode of pre-assessment. In the form of C(a) class assertion, a desired input (or topic) e.g. *update* can be stated as *desired_Concept*("*update*"); and its prerequisite nodes as *hasPrerequisite*("*update*", "*delete*") and *hasPrerequisite*("*delete*", "*insert*"). Then, one plan that is showing the rules for the categorization in Jason AgentSpeak with respect to the prerequisites *delete* and *insert* is stated in Figure 16.

In Figure 16 there is a statement of inter-agent communication that is given as .send(agMaterial, achieve, has_KB(X, delete_select)). This is a message from agent *agModelling* to the ontology agent *agMaterial*. The statement uses the performative achieve [13] which informs the receiver agent to achieve what is in the given content. Upon

```
@updateRule5
+!recommendMaterial[source(agSupport)] :  desired_Concept("UPDATE")[source(agSupport)]
    & failed("The student has NOT passed the DELETE with SELECT question.")
    & passed("The student has passed the DELETE with WHERE question.")
    & passed("The student has passed the INSERT with SELECT question.")
    & passed("The student has passed the INSERT with VALUE question.")
    <- .send(agMaterial, achieve, has_KB(X, delete_select)).
```

Fig. 16 A classification and recommendation rule implementation in Jason agent oriented programming. The plan (a sequence of instruction) of the agent *agModelling* also contained the annotation [source(agSupport)] which is the agent that is responsible for the source of the decisions and another agent *agMaterial* (see Fig. 17) that is receiving the instruction of the classified learning materials that is recommended

```
@delete_selectURL
+!has_KB(X, delete_select)[source(agModelling)] // for failing DELETE...SELECT from desired_Concept("UPDATE")
    <- .println;
    .println(" You will learn the DELETE with SELECT clause. Please use the link below for learning materials:");
    ?hasContentText(deleteSelect, DS_textURL)[o(sql)]; //Test goal. Maps the variables  to the link in BB.
    .println("DELETE...SELECT query Text Link: ");
    .println(DS_textURL);
    .println.
```

Fig. 17 Agent agMaterial (the ontology agent) that releases learning material URL links based on the percept received. The web URL link is literal as shown in Figure 10

receiving the message has_KB(X, delete_Select), the receiver agent *agMaterial* adopts the content has_KB(X, delete_Select) and executes the relevant plan in its plan library with respect to this message.

Conclusions and Future Work

Ontologies have their knowledge organized in hierarchies of structures. In building ontologies for learning, teaching and assessments, we have outlined SQL learning modules in a manner that enables pre-learning of SQL queries to support future learning of SQL queries by database students. This is to avoid any gaps that may inhibit future learning. The paper has demonstrated the use of description logics in the definition of an SQL ontology for pre-learning assessment. With the chosen terms and properties and minimum cardinality constraint that has been specified, the SQL DL ontology is scalable, expandable and extensible. This paper has defined a TBox and its ABox counterpart . Then, showed the graphical design, analysis and serialization of the ontology in Turtle as well as in first-order predicate logic for Jason agents. In future work, we shall focus on first-order predicate API plug-in for an ontology editor. This is to enable the direct serialization of semantic web data in predicate logic beliefs for first-order predicate agent based systems.

Compliance with Ethical Standards

Conflict of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. The apache software foundation, apache jena. https://jena.apache.org/. Assessed 07 Aug 2019.
2. Obitko, M. (2007). ontologies of the semantic web. https://www.obitko.com/tutorials/ontologies-semantic-web/description-logics.html. Accessed 7 Aug 2019.
3. Owl 2 web ontology language. https://www.w3.org/tr/owl2-syntax/. Accessed 7 Aug 2019.
4. Protege (2016-2019). a free, open-source ontology editor and framework for building intelligent systems. https://protege.stanford.edu/. Accessed 11 Aug 2019.
5. W3c. 2004. resource description framework rdf. https://www.w3.org/rdf/. Assessed: August, 2nd,2019.
6. W3c owl2 web ontology language primer (second edition). w3c recommendation 11 december 2012. https://www.w3.org/tr/owl2-primer/#owl_2_dl_and_owl_2_full. Accessed 17 Aug 2019.
7. W3c recommendation (2004) rdf/xml syntax specification (revised). https://www.w3.org/tr/rec-rdf-syntax/#figure2. Assessed 2 Aug 2019.
8. W3c web ontology language (owl lite, owl dl, and owl full) feature synopsis version 1.0, w3c working draft january 2, 2002. http://www.ksl.stanford.edu/people/dlm/webont/owlfeaturesynopsis-jan22003.htm. Assessed 17 Aug 2019.
9. Baader F, Brandt S, Lutz C. Pushing the el envelope. In IJCAI. 2005;5:364–9.
10. Baader F, Horrocks I, Sattler U. Description logics. Found Artif Intell. 2008;3:135–79.
11. F. Baader and W. Nutt. Basic description logics. *Description Logic handbook*, pages 43–95, 2003.
12. R. H. Bordini, J. F. Hübner, and D. M. Tralamazza. Using jason to implement a team of gold miners. In *International Workshop on Computational Logic in Multi-Agent Systems*, pages 304–313. Springer, 2006.
13. Bordini RH, Hübner JF, Wooldridge M. Programming multi-agent systems in AgentSpeak using Jason, vol. 8. Hoboken: Wiley; 2007.
14. 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 Autom Reason. 2007;39(3):385–429.
15. J. Coleman Prior. Online assessment of sql query formulation skill. In *Proceedings of the fifth Australasian conference on Computing education*, volume 20, pages 247–256, 2003.
16. T. M. Connolly and C. E. Begg. *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.
17. N. B. Dale and J. Lewis. *Computer science illuminated*. Jones & Bartlett Learning, 2007.
18. Dong T, Le Duc C, Lamolle M. Tableau-based revision for expressive description logics with individuals. J Web Semant. 2017;45:63–79.
19. DuCharme, B. *Learning SPARQL: querying and updating with SPARQL 1.1*. "O'Reilly Media, Inc.", (2013).
20. Ehimwenma, K.E. *A multi-agent approach to adaptive learning using a structured ontology classification system*. PhD thesis, Sheffield Hallam University, (2017).
21. Ehimwenma, K.E., Beer, M., Crowther, P.. Student modelling and classification rules learning for educational resource prediction in a multiagent system. In *2015 7th Computer Science and Electronic Engineering Conference (CEEC)*, pages 59–64. IEEE, (2015).
22. Ehimwenma KE, Crowther P, Beer M. A system of serial computation for classified rules prediction in non-regular ontology trees. Int J Artif Intell Appl. 2016;7(2):23–35.
23. Ehimwenma KE, Crowther P, Beer M. Formalizing logic based rules for skills classification and recommendation of learning materials. Int. J. Inf. Technol. Comput. Sci. 2018;10(9):1–12.
24. Faruqui and, R.U., MacCaull, W. Owl ontdb: A scalable reasoning system for owl 2 rl ontologies with large aboxes. In *International Symposium on Foundations of Health Informatics Engineering and Systems*, pages 105–123. Springer, (2012).
25. Fernández-López M., Gómez-Pérez A., Juristo N. Methontology: from ontological art towards ontological engineering. (1997).
26. Gruber TR. A translation approach to portable ontology specifications. Knowl Acquis. 1993;5(2):199–220.
27. Gruber TR. Toward principles for the design of ontologies used for knowledge sharing? Int. J. Hum.-Comput. Stud. 1995;43(5–6):907–28.
28. Horridge, M., Jupp, S., Moulton, G., Rector, A., Stevens, R., Wroe, C. A practical guide to building owl ontologies using protégé 4 and co-ode tools edition1. 2. *The university of Manchester*, 107, (2009).
29. Horrocks I, Patel-Schneider PF, Van Harmelen F. From shiq and rdf to owl: The making of a web ontology language. Web semantics: science, services and agents on the World Wide Web. 2003;1(1):7–26.

30. Kawash, J. Formulating second-order logic conditions in sql. In *Proceedings of the 15th Annual Conference on Information technology education*, pages 115–120. ACM, (2014).
31. Keet CM. An introduction to ontology engineering. Cape Town: University of Cape Town; 2018.
32. Klapiscak, T., Bordini, R.H. Jasdl: A practical programming approach combining agent and semantic web technologies. In *International Workshop on Declarative Agent Languages and Technologies*, pages 91–110. Springer, (2008).
33. Labrou, Y., Finin, T. Semantics and conversations for an agent communication language. *Readings in agents*, pages 235–242, (1998).
34. Laclavik M, Balogh Z, Babik M, Hluchỳ L. Agentowl: Semantic knowledge model and agent architecture. Comput Inf. 2012;25(5):421–39.
35. Maedche A, Staab S. Ontology learning for the semantic web. IEEE Intell. Syst. 2001;16(2):72–9.
36. Mascardi V, Ancona D, Bordini RH, Ricci A. Cool-agentspeak: Enhancing agentspeak-dl agents with plan exchange and ontology services. Web Intell. Agent Syst. Int. J. 2007;5:1–23.
37. Mascardi, V., Ancona, D., Bordini, R.H., Ricci, A. Cool-agentspeak: Enhancing agentspeak-dl agents with plan exchange and ontology services. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02*, pages 109–116. IEEE Computer Society, (2011).
38. Mauer, L., Solomon, D., McEwan, B. *Sams teach yourself Transact-SQL in 21 days*. Sams, (2001).
39. Moreira, A.F., Vieira, R. Belief update in agentspeak-dl. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, (2008).
40. Nardi D, Brachman RJ. An introduction to description logics. Descr. Logic Handb. 2003;1:40.
41. Rudolph, S. Foundations of description logics. In *Reasoning Web International Summer School*, pages 76–136. Springer, (2011).
42. Russel S, Norvig P. Artificial intelligence: A modern approach. thrid ed. Boston Munich: Person Education; 2010.
43. Sadiq, S., Orlowska, M., Sadiq, W., Lin, J. Sqlator: an online sql learning workbench. In *ACM SIGCSE Bulletin*, volume 36, pages 223–227. ACM, (2004).
44. Uschold M, Gruninger M. Ontologies: Principles, methods and applications. Knowl. Eng. Rev. 1996;11(2):93–136.
45. Wooldridge M. An introduction to multiagent systems. Hoboken: Wiley; 2009.