ONTOLOGY-BASED TOP-K GLOBAL SCHEMA GENERATION

A Thesis

by

YUZHE WEI

Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

Texas A&M University - Corpus Christi

Corpus Christi, Texas

May 2012

Major Subject: Computer Science

ONTOLOGY-BASED TOP-K GLOBAL SCHEMA GENERATION

A Thesis

by

YUZHE WEI

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Computer Science Graduate Program
School of Engineering and Computing Sciences
Texas A&M University - Corpus Christi

Approved by:

_____	_____
Longzhuang Li, Committee Chair	Ahmed M. Mahdy, Committee Member
_____	_____
Dulal C. Kar, Committee Member	John D. Fernandez, Department Chair

May 2012

Abstract

Global schema generation is the problem of generating a unified schema based on existing heterogeneous local schemas and a set of correspondences which are generated by a schema matching algorithm. The current schema integration approaches cannot satisfy users' requirements due to several reasons. The existing approaches cannot handle hierarchical schema structure, cannot solve conflicts problems, and only generates one merging result. To deal with these kinds of problems, top-k global schema generation approach is proposed in this research. The proposed approach consists of three steps: (1) Relational schemas are converted to ontologies; (2) Ontologies are merged and top-k results are generated; (3) Users can choose one of the top-k merged ontologies and convert it back to the relational schema. The approach utilizes ontology as a base merging model to create the global schema, because ontology can provide semantic and detail constraints. The objective of this research is to generate the global schema with high quality but less user involvement. Since different users may have their own preferred global schemas, top-k ranking algorithms are utilized to obtain multiple schema integration results, so users may have more than one choice. After comparing the approach in this research with state of the art schema integration approaches, the proposed approach better preserves the hierarchical structure, and generates the global schema with higher quality.

Table of Contents

1. Introduction 1

2. Background and Rationale 3

2.1. Previous Related Work 3

2.2. Converting Relational Schema to Ontology 8

3. Challenges Motivating Proposed Work 12

3.1. Multiple Merging Results 12

3.2. Automatically Generating Global Schema 12

3.3. Preserve Hierarchical Structure 13

3.4. Relationship Conflicts 13

3.5. Constraints Conflicts 14

4. Objectives of Proposed Research 14

4.1. Scope of the Proposed Research 14

5. Approach Architecture 16

5.1. Creation of Ontologies 17

5.2. Similarity between Entities (*Hausdorff* distance) 22

5.3. Top-k Algorithm 23

6. Case Study and Prototype Implementation 24

6.1. ERD TO ERD 25

6.1.1. The Schema Integration Approach Posed by Radwan, et. al. (2009) 26

6.1.2. The Proposed Ontology-based Schema Integration Approach 31

6.2. ERD TO EERD 35

7. How the Ontology Integration Approach Enhance the System 44

7.1. Hierarchical Structure Analysis Procedure 45

7.1.1. Hierarchical Structure Analysis Procedure (EERD TO EERD) 50

7.2. Utilizing Object Properties to Enhance the Merging System 54

7.3. EERD TO EERD in Enhanced Ontology Integration Approach 59

8. Conclusion and Future Work 62

References 63

Appendix A 65

Appendix B 67

Appendix C 70

Appendix D 74

Appendix E 76

LIST OF TABLES

TABLE Page

I Ontology conversion rules 10

LIST OF FIGURES

FIGURE Page

1 The concept graph example 5

2 The merging results effected by λ 7

3 The result of top k algorithm 8

4 Rule for Translating EER to OWL 12

5 Ontology-based Schema Integration 16

6 Rule C1, Fragmentation class rule 18

7 Rule C2, hierarchy class rule 19

8 Top-k algorithm 23

9 UniversityOneERD ER Diagram 25

10 UniversityTwoERD ER diagram 26

11 ERD to ERD schema matching 27

12 ERD to ERD schema matching text format 28

13 Top 2 merging assignment 28

14 Merging information 29

15 Integrated university ERD 30

16 Ontology merging information 32

17 Integrated university ERD by using ontology 34

18 Modified UniversityOneERD ERD 36

19 UniversityThreeEERD EER diagram 37

20 The matching of UniversityOne and UniversityThree..... 38

21 ERD TO EERD merging information..... 39

22 Integrated EERD of university schema..... 40

23 Refuse to generate the result..... 41

24 Assignment no breaking the hierarchy structure 42

25 Ontology approach merging result of ERD to EERD..... 43

26 Candidate assignment contains the hierarchical structure 44

27 Merging class *A* with *B*..... 46

28 a) Merge class *A* with *C*..... 47

28 b) Merging class *A* with *D* 47

29 Insert *SubClassOf* between new merged classes 48

30 Merged parent and child Classes with redundant properties 49

31 a) Remove properties in subclass..... 49

31 b) Remove original properties 49

31 c) Remove redundant properties in super class..... 50

32 Result generated by ontology approach..... 50

33 a) Two ontologies with hierarchical structure 51

33 b) Merge A with D..... 52

33 c) Merge B with E..... 52

33 d) Merge C with F 53

33 e) Merging result (EERD to EERD) 53

34 Many-to-many relationship in concept graph 54

35 Many-to-many relationship with additional attribute to ontology 55

36 a) Two schemas before Merging 56

36 b) Concept graph 56

36 c) Schemas after merging 57

37 Merging ontologies without considering object property 58

38 Ontologies before merging 59

39 ERD after merging by considering object property 59

40 UniversityFourEERD EERD 60

41 Top 2 assignment 60

42 EERD to EERD by ontology approach 61

1. Introduction

Database integration has been studied by researchers for a long period of time. A vast amount of data is stored in the distributed heterogeneous data sources which cause data redundancy and management difficulty. Database integration could resolve these problems by querying global sources. Users can then gather shared distributed data by transparently accessing local databases. Therefore, one major part of database integration is to design a global schema, but it's always designed by domain experts, which may cause problems due to time consumption and labor intensity especially for integrating large-scale databases. The problem of automatically generating global schema from users' feedback challenges researchers. Global schema generation refers to the problem of integrating different existing local source schemas to a unified schema based on their matching. According to Chiticariu, Kolaitis, & Popa (2008), "By providing a standard representation of the data, the integrated target schema can be viewed as a means for dealing with heterogeneous data sources" (p. 1). After a set of local schemas and correspondence of each local schema are given as input, how to output the global schema without redundant semantic elements and keep the inheritance relationship is the goal of this research.

Semantic redundant problems may happen during the schema integration. For example, two entities may be named "table"; however, the first "table" represents the grid table in the database, but the second "table" represents a wooden table made in China. Integrating the two tables can lose the meaning of the actual entity.

Schema structure could represent semantics such as is-a-kind-of, is-a-part-of, has-a, and has-many. For example, an apple tree is a kind of tree, an engine is a part of car, a student has a professor, and a professor has many students, and these semantics can be modeled from an entity relationship diagram (ERD) or an extended entity relationship diagram (EERD). Structure problems may happen when one entity or class matches another sub-entity or subclass. For example, if the majority of the attributes in one local schema whose entity is named "tree" match the majority of attributes in another local schema whose entity is named as "apple tree", without metadata and semantics, there is no clue of how to merge those two entities to represent the super-sub type relationship (An apple tree is a kind of tree).

Fahad (2008) states "ontology is regarded as the formal specification of the knowledge of concepts and the relationships among them" (p. 28). In the past decade, with the rise of the Semantic Web, ontologies have been used in the domain of database integration. As an approach of data description and sharing, ontology can not only solve the semantic redundancy problem, but it also supports the searching in the Deep Web. The building of the web ontologies could utilize the technique of database integration, which constructs one major part of Deep Web, integrating the database located behind the internet, to support using key words to query data sources. Therefore, utilizing ontology to build global schema is a good idea for solving database integration and the Deep Web problem.

Radwan et. al. (2009) argues that "schema integration is a long-standing research problem and continues to be a challenge in practice" (p. 1). Most of the current approaches of schema integration requires a substantial amount of user feedback during the integration process and

generates only one possible integration result. Different users may prefer different global schemas, obviously only one integrated result cannot meet all the users' requirements. Top-k integrated global schema provides a solution to satisfy users' special needs. In this research, top-k schema integration algorithms and ontology techniques are utilized to generate global schemas without conflicts.

2. Background and Rationale

2.1. Previous Related Work

Pottinger and Bernstein (2003) examined the problem of merging two models with given correspondences. In their paper, Pottinger and Bernstein (2003) introduced the "equivalent" model, Map_{AB}, which represents how two models should be merged based on correspondences. An equivalent mapping element in Map_{AB} model directly points to two elements in two different models. Pottinger & Bernstein (2003) defined "each mapping element also has a property *HowRelated*, with value *Equality* or *Similarity* to distinguish the two kinds of mapping elements" (p.3). Complex relationships can be defined between elements, for example, "First Name" and "Last Name" are sub-elements of "Name". In this research, Pottinger and Bernstein(2003) defined representation conflicts, meta-model conflicts, and fundamental conflicts, and gave solutions to solve conflicts with "Vanilla", an extended entity-relationship-style meta-meta model that includes semantic modeling constructs. In this "equivalent" model, semantic redundant problems could be solved by equivalent mapping, and hierarchical structure could be obtained by the relationship of *Associates*, *Contains*, *Has-a*, *Is-a*, and *Type-of*. One problem of this "equivalent" is that "similar" concepts could only be merged by user feedback and it's difficult to define them.

Chiticariu, Kolaitis, and Popa (2008) develop a method and a design tool that provide "adaptive enumeration of multiple interesting integrated schemas and easy-to-use capabilities for refining the enumerated schemas via user interaction" (p. 1). In their research, Chiticariu, Kolaitis, and Popa (2008) use concept graphs to represent both relational and XML schemas, and they gave the definition of concept graphs as follow.

Chiticariu, Kolaitis, & Popa (2008):

> "A concept is a relation name C associated with a subset att(C) of U (these are the attributes of C). A concept graph is a pair (*V*; *HasA*) where *V* is a set of concepts and *HasA* is a set of directed edges between concepts, such that each edge has a label *L*. We write A HasA B [L] whenever there is a HasA edge with label L from concept A to concept B" (p. 3).

For example, in figure 1, concept *dept*, *manager*, and *emp* are three concepts converted by entities. In the relational schema, relationships between "dept" and "manager", and "dept" and "emp" are one-to-many. This relationship can be represented in concept graphs as the "manager" has a "dept" and "emp" has a "dept". Chiticariu, Kolaitis and Popa (2008) defined the condition when two concepts may be merged. If there is any correspondence between two concepts, then these two concepts may be merged. Therefore, if there are *n* correspondences existing between concepts in different schemas, then 2^n possible integrated global schemas may be created. Even though duplicates and cycles were defined to avoid a full enumeration, having the user pick a global schema in the average of 2^n integrated results is infeasible and time consuming.

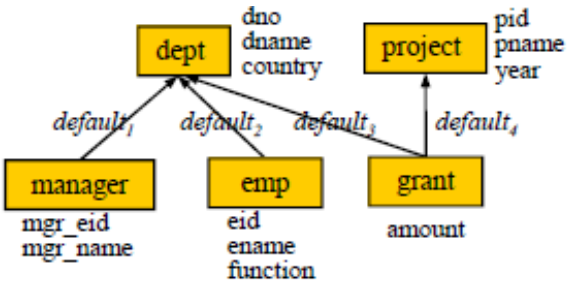


Figure 1: The concept graph example (Chiticariu, Kolaitis, & Popa, 2008, p. 4).

Radwan, Popa, Stanoi, and Younis (2009) introduced top-k algorithm for schema integration based on the directed weight correspondences. They proposed "a more automatic approach to schema integration that is based on the use of directed and weighted correspondences between the concepts that appear in the source schemas" (Radwan et. al., 2009, p. 1). In the research, *Hausdorff* distance is first introduced to evaluate the similarity between two entities. This similarity measure varies in the range [0, 1], where 1 denotes all the attributes in concept *A* that are covered by concept *B*. *Hausdorff* distance $\hat{d}(A, B)$ is computed as follow. The algorithm denotes $\alpha(X)$ as the collection of attributes in a concept *X*, and $\beta(X)$ as the collection of concepts to which *X* has direct references. N_α and N_β are the numbers of attributes. If there is a correspondence between the attributes *a* and *b*, the distance is 0, otherwise, the distance is 1.

$$\hat{d}(a, B) = \min\left(min_{b \in a(B)} d(a, b), min_{B' \in \beta(B)} \hat{d}(a, B')\right) \quad (1)$$

$$\hat{d}(A, B) = \frac{1}{N_\alpha + N_\beta}\left(\sum_{a \in \alpha(A)} \hat{d}(a, B) + \sum_{A' \in \beta(A)} \hat{d}(A', B)\right) \quad (2)$$

(Radwan et. al., 2009, p. 10)

According to the similarity value *d (A, B)*, top-k algorithm is used to rank 2^n possible integrated global schemas. The idea of the top-k ranking algorithm is to calculate the lowest cost value. If similarity $\hat{S}(A,B)$ is bigger than dissimilarity $\widehat{D}(A,B)$, then the cost of merging the two concepts is less than not merging the two concepts, therefore it would be better to merge two concepts. If similarity is smaller than dissimilarity, then the cost of merging two concepts is more than not merging the two concepts, therefore it would be better not to merge the two concepts. The cost is calculated as follows. K_1 is the set of used correspondences, and K_2 is the set of unused correspondences.

$$cost(A) = \frac{1}{n}\left(\sum_{e\in K_1}\widehat{D}\,(A,B\,) + \sum_{e\in K_2}\hat{S}(A,B)\right) \quad (3)$$

(Radwan et. al., 2009, p. 6)

Radwan, Popa, Stanoi, and Younis (2009) used parameter λ, which is set by the user, to control how to combine concepts once the top-k assignments have been generated. If the similarity between the two concepts is higher than λ, then the two concepts should be merged. Otherwise, one concept turns into an extended concept of the other concept. For example, in figure 2, the left graph shows the merging result that the similarity between the two concepts is higher than λ. The "dependent-member" concept and the "householder" concept merged to one concept. The right graph shows the merging result that the similarly is lower than λ. The "householder" concept is the extended concept of the "dependent-member".

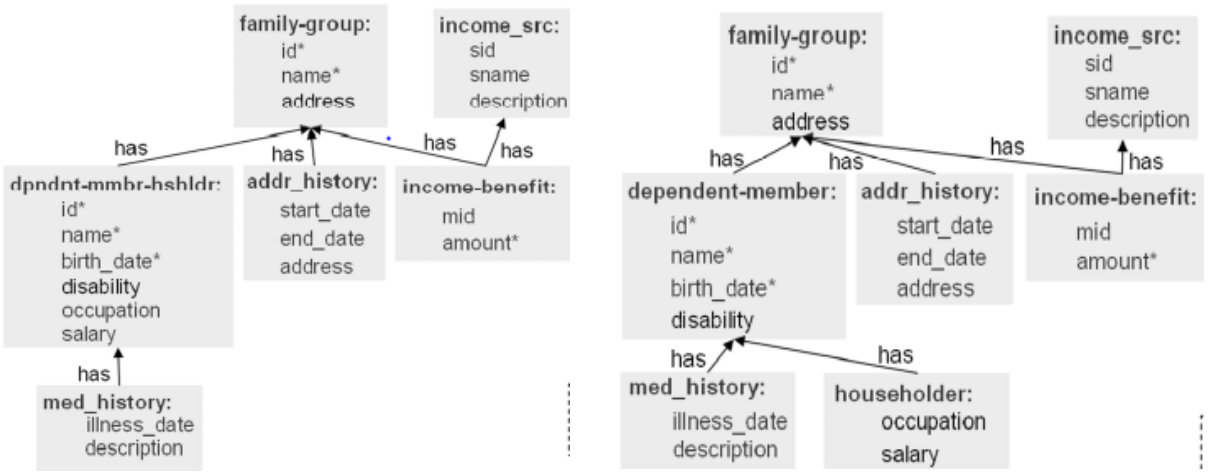


Figure 2: The merging results effected by λ (Radwan et. al, 2009, p. 9).

Guohui, Guoren, and Bin (2010) proposed a different top-k ranking algorithm. In this algorithm, the top-k integrated global schemas are generated by enumerating 2^n possible assignments and attaching the pruning technique to the enumeration. The theorem of pruning enumeration is provided as follows and it was proven by Guohui, Guoren, and Bin (2010).

Theorem provided by Guohui, Guoren, and Bin: "Let $X = [Xn...Xi...X0]$ be an assignment, and Xi is any bit of X for $n \geq i > 0$. Let $dec(X)$ denote the decimal number that X corresponds to. If $Xi = 1$, $Xj = 0$ for $i - 1 \geq j \geq 0$, and $score(X) > \lambda$, then we can infer that all the scores of the next sequential $2i - 1$ assignments of X are greater than λ, formally described as $score(A) > \lambda$, where A is any assignment of X and satisfies the constraint: $dec(X) + (2i - 1) \geq dec(A) > dec(X)$" (Guohui, Guoren, & Bin, 2010, p 150).

In figure 3, it displays the result of Top-K algorithm designed by Guohui, Guoren, and Bin (2010). In this example, *A5* to *A7*, *A9* to *A15*, and *A17* to *A31* are pruned based on the theorem during the top 3 global schemas that are generated.

The problems of schema integration are given in detail in previous approaches. The framework has been built for interactive generation of integrated schemas which are proposed by Chiticariu, Kolaitis, and Popa (2008). In contrast, the more automatic system of schema integration that utilizes weighted correspondences to build top k schema integration results is proposed by Radwan, Popa, Stanoi, and Younis (2009). However, the approaches proposed so far do not deal well with the inheritance in an advanced schema model such as EER Diagram, and many conflicts arise in the merged schema, and still, lots of work is needed from domain experts.

X	X_5	X_4	X_3	X_2	X_1	X_0	score(**X**)	Top-3 Buffer
A_0	0	0	0	0	0	0	0 .308	A_0
A_1	0	0	0	0	0	1	0 .325	A_0, A_1
A_2	0	0	0	0	1	0	0 .348	A_0, A_1, A_2
A_3	0	0	0	0	1	1	0 .365	A_0, A_1, A_2
A_4	0	0	0	1	0	0	0 .392	A_0, A_1, A_2
A_5 - A_7				*			*	A_0, A_1, A_2
A_8	0	0	1	0	0	0	0 .408	A_0, A_1, A_2
A_9 - A_{15}				*			*	A_0, A_1, A_2
A_{16}	0	1	0	0	0	0	0 .418	A_0, A_1, A_2
A_{17} - A_{31}				*			*	A_0, A_1, A_2
A_{32}	1	0	0	0	0	0	0 .342	A_0, A_1, $\mathbf{A_{32}}$

Figure 3: The result of top-k algorithm in (Guohui, Guoren, & Bin, 2010, p 156).

2.2. Converting Relational Schema to Ontology

Gruber (1993) gave the definition of ontology early on as “an explicit specification of a conceptualization” (Gruber, 1993). Since then, the definition has been amended to express other

features. The new definition of ontology is: a formal explicit specification of a shared conceptualization (Mihoubi et. al, 2000). Since the objective of ontology is for sharing and reusing resources and schemas is also a kind of resource, we can utilize it to describe the merging models and generate global schemas.

Alawan (2011) "ontology consists of four principal elements: concepts, relations, axioms, and instances" (p. 40). A concept (also known as a class) is the essential abstract component of a domain. Concepts may be arranged in hierarchical graphs on two levels: parent concepts and child concepts. A relation is used to declare the relationships between concepts in a specific domain. Alawan (2011) "in order to specify the two classes involved in a particular relationship, one of them will be described as a Domain and the other one as a Range" (p. 40). An Axiom (also known as role restriction) is utilized to express constraints. An instance is the real value of a concept.

There are many languages proposed to build ontology such as RDF, RDFS, OIL, DAML+OIL, and OWL. RDF is a basic foundation language, and other languages are built based on RDF. The limit of RDF is that it is unable to describe the relation in ontology such as cardinality, domain and range constraints, existence descriptions, and property characteristic. RDFS overcomes these limitations. RDFS supports hierarchies and property hierarchies, while enabling relationship construction and restricting the domain and range. The limit of RDFS is that it is unable to describe the equality or inequality between properties. Alawan (2011) "union, intersection, unique, symmetric, transitive and inverse are relation characteristics which cannot be expressed by RDFS" (p. 48). DAML+OIL is designed to overcome the limits of RDFS, but it has its own

limitations. For instance, it lacks property descriptions. Fahad (2008) "in 2004, W3C made OWL the standard used to build ontologies because of its decidability and high level of expressivity" (p. 28). OWL is proposed based on RDF vocabularies and XML syntax and it overcome the drawbacks of RDF, RDFS, and DAML+OIL. OWL Lite, OWL Description Logic (OWL DL) and OWL Full are three sublanguages of OWL to supply different goals.

Fahad (2008) proposed ERD to OWL-DL ontology transformation rules at a concrete level. In the research, Fahad assumed ER Diagrams or Extended ER Diagrams were documented and available in most legacy systems. Fahad (2008) "the framework provides OWL ontology for the semantic web component from old legacy systems and enables them to upgrade and become a part of emerging semantic webs" (p. 1). The rules of converting ERD to OWL proposed by Fahad are listed in Table 1. This research also proposed the way in which relationships between entities, such as Bi-Directional Relationship, Unary (recursive) 1:N Relationship, Unary M:N Relationship, Associative Entity, 1 to Many Relationship, and Many to Many Relationship, are mapped to OWL.

Table 1. Ontology conversion rules (Fahad, 2008)

Entity	Map each Entity in the ERD into OWL class in the OWL Ontology.
Simple Attribute	Map Simple Attribute of entity into datatype property of corresponding OWL class. Domain of the datatype property is the Entity, and range is the actual datatype of that attribute.
Composite Attribute	Map only their simple component attributes of composite attribute to datatype properties of corresponding OWL class, and ignore composite

	attribute itself.
	Map composite attribute to datatype property and then map its simple, component attributes to subproperty of corresponding datatype property.
Multi-valued Attribute	Multi-valued Attribute is mapped to datatype property like simple attribute, but without a "*functional*" tag.
Primary Key	An attribute which stands as a primary key, is transformed into datatype property and is tagged with both "*functional*" and "*inverse - functional*".
Subtype Relations	Convert subtype relations in the ERD to subClassOf in the OWL ontology.

Alalwan (2011) utilises the "reverse engineering techniques" to catch the semantics hidden in the SQL language. The logical model of the database is reproduced based on SQL-DDL. Fragmentation rules, hierarchy rules, multi-valued rules, default rules, rules for the creation of object properties, and rules for instances are proposed in this research.

The general rule for translating an EER model to an OWL model consisted of three steps and details are shown in figure 4.

1. Entities are represented by OWL classes.
2. Relationships are translated into a pair of object properties; or two pairs of inverse object properties with a class, as may apply.
3. Attributes can form datatype properties or a class with datatype properties, as may apply.

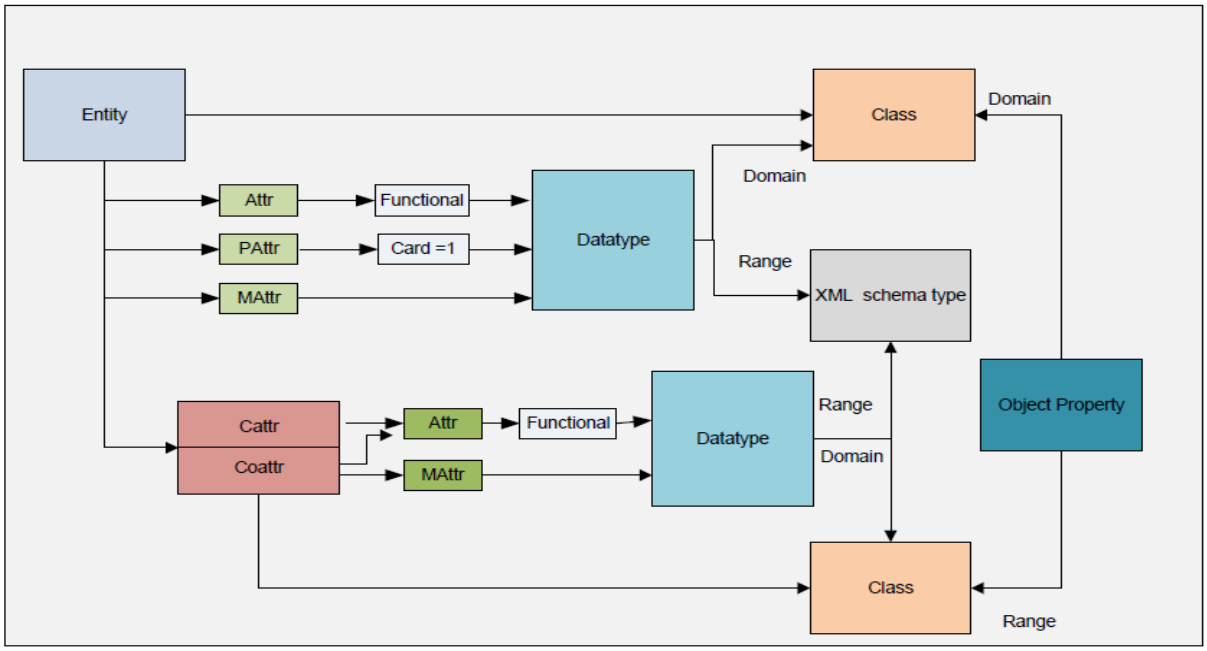


Figure 4: Rule for Translating EER to OWL (Alalwan, 2011, p. 144).

3. Challenges Motivating Proposed Work

3.1. Multiple Merging Results

Existing schema integration approaches do not generate a perfect global schema because there is no perfect schema matching approach while schema integration is a technique based on schema matching. Therefore, multiple interest schema integration results should be obtained after schema integration and let the user pick one of them. This is the best way to reduce the user's involvement and still achieve high quality results from schema integration. Determining how to get the most interesting top-k global schemas is a major challenge noted in the research of global schema generation.

3.2. Automatically Generating Global Schema

Radwan, et. al. (2008) introduce that:

> "Although today the process of integrating schemas is partially automated, it is still labor-intensive. In order to reduce the amount of manual intervention that is required from users, we need to modify or avoid parts of the integration process that unnecessarily increase the load on users" (p. 1).

Investigating the semantics from relational schema to build the machine readable model, and follow the merging algorithms to build the global schema is another challenge within the research of global schema generation.

3.3. Preserve Hierarchical Structure

Hierarchical structure should be retained during schema integration. In a local schema, hierarchical structure could be gathered by schema constraints such as prime key and foreign key. Therefore, parent entities and child entities could be defined by SQL-DDL. However, global constraints are missing during schema integration. Defining whether one entity in source A is the parent class or child class or equivalent class in source *B* is another challenge within the research.

3.4. Relationship Conflicts

Relationship conflicts may happen during schema integration. For example, *A* and *B* are two entities in schema one and the relationship between them is many-to-many. *C* and *D* are two entities in schema two and the relationship between them is one-to-many. After schema matching, if we gather the matching result as entity *A* matches entity *C* and entity *B* matches entity *D*, then a relationship conflict happens. Defining the relationship between two merging concepts is a challenge within the research of schema integration.

3.5. Constraints Conflicts

Constraint conflicts may happen during schema integration. For example, in the merging model, if attribute *A* and attribute *B* have to be merged after running the schema matching algorithms, but the data type of attribute *A* is defined as an integer and the data type of attribute *B* is defined as a string, then constraint conflicts occur. Deciding the kind of data type when constraint conflicts occur is one challenge of schema integration.

4. Objectives of Proposed Research

4.1. Scope of the Proposed Research

In this research, the challenge of global schema generation was introduced in the previous section. Global schema generation is one sub-category in the research of database integration. Since the quality of global schema could directly affect the quality of database integration, global schemas are always generated by domain experts. However, for uncertain or large-scaled local schemas, designing a global schema only by experts is infeasible and time consuming. A machine understandable merging model could help solve the problems introduced in the previous section. In this machine understandable merging model, merging rules and algorithms would be designed to reduce human's involvement as much as possible.

In this thesis, we assume that schema matching results are available based on some existing schema matching approaches, such as *Cupid* or *COM++*. The schema matching is not considered in the research. The question of how to preserve hierarchical structure during schema integration is the first concern in the research. Problems in schema integration will be studied in future research.

ER diagrams or Extended ER diagrams from local sources are not available in this research. The only information we can gather is from SQL-DDL, which can be easily gathered from any database.

5. Approach Architecture

The architecture of the proposed approach is shown in figure 5. First, local relational schemas are converted to ontologies. Based on some factors of the ontology, top-k ontologies are generated. After users choosing any top n candidate merged ontology, system gets into hierarchical structure analysis procedure to preserve the hierarchical structure. At the end, integrated ontology will be converted back to the relational schema.

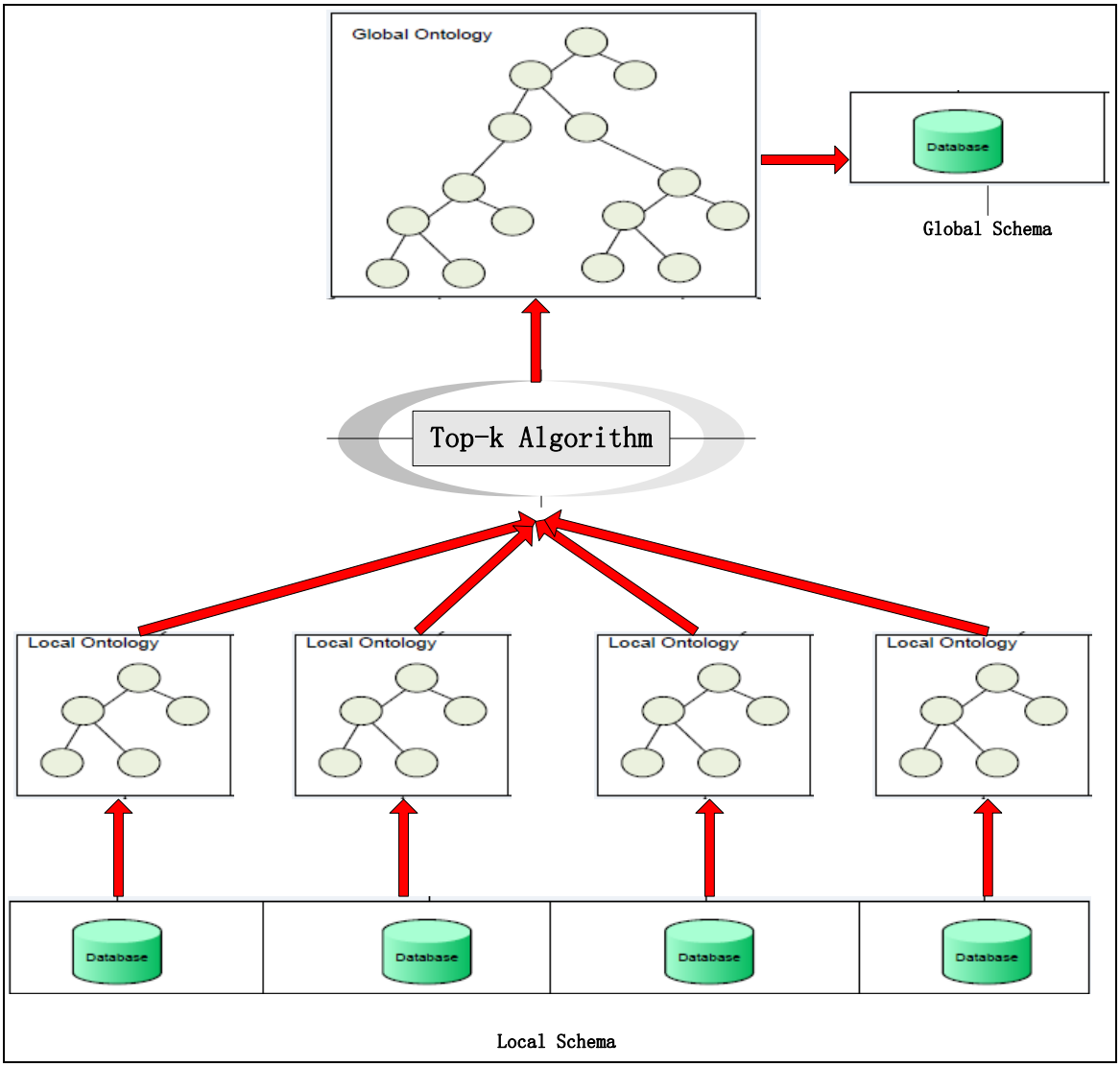


Figure 5: Ontology-based Schema Integration.

5.1. Creation of Ontologies

First, this schema integration approach should convert relational schema to ontology, which is represented by OWL. Since OWL is a machine readable language and it could provide more information for schema integration, such as property, hierarchical structure, domain, range, data type, etc. Concept graph which is proposed by Chiticariu, Kolaitis, and Popa (2008) is not concerned in this thesis since it is less representable for schema integration than the ontology. From the definition of concept graph which is introduced in section 2.1, the meaning behind the *has* edge can refer to both *has-a* and *is-a*. Radwan (2009) "As an example, a *student* concept may have two *has* edges to a *person* concept, one reflecting that *student* "is-a" *person*, and the other reflecting that *student* "has-a *person* as an advisor. Thus, in this model, *has* edges can encode both *has-a* and *is-a* type of relationships that exist in conceptual models" (643). Therefore, many-to-one and parent-child relationship are blurred in the concept graph model, since they are all be represented by *has* edge. This can confuse users when they convert a concept graph back to relational schema.

In ontology, the relationships between two classes are more specific since ontology supports the *supper class of*" and *sub class of* relationships between two classes, and ontology uses object properties and restrictions such as "some values from" and "cardinality" to represent one-to-many and one-to-one relationship in relational schema. Using ontology as a schema merging model, more information can be provided to users, such as properties. If ER diagram is available, then the relationship between two entities can be supported with semantic. For example, information about teacher teaches students can be gathered. Then, the relationship between class, teacher, and class student can be represented by object property with the name of teaches. The domain of property "teaches" is the teacher class, and the range of property "teaches" is the

student class. The reverse object property of "teachers" can be set and named with *is-taught*. The domain of *is-taught* object property is student class, and the range is student class. After setting object property and reverse object property, restrictions, such as *someValueFrom* and *cardinality*, can be set on those properties. *SomeValueFrom* restriction means that the instance of domain class references to zero or more instances of range class. "Cardinality" restriction means that the instances of domain class reference to exactly number of instances of range class. In this research, the cardinality value is set to 1 to represent *one-to-many* in relational schema.

The rules of converting relational schemas to OWL are represented in Alalwan (2011) and Fahad (2008). In both examples of convert relational schema to OWL in Alalwan (2011) and in Fahad (2008), a set of rules are developed to build ontology using ER/EER Diagram. Therefore, both ER Diagram and Extended ER Diagram are taken as available resource in Alalwan (2011) and Fahad (2008). However in this research, the assumption is that only Data Description Language (DDL) is available resource for building ontology, since most of ER or EER diagrams are remained by database designers and those are not easy to be obtained from the internet. Therefore, only partial rules are adopted from Alalwan (2011) and Fahad (2008) to build the ontology in the thesis.

- Rule C1, Fragmentation class rule, is utilized to defined fragmentation relationship between two entities. All fragmentation entities should be taken as one class in the ontology. The rule is shown in the figure 6. For example, according to the SQL-DDL as follow, *staff-details* is the fragmentation class of *staff.*

```
Create table staff (
        staff_id int, staff_ family_name varchar(50) not null,dept_id int,
        manger_id int, n_id int not null unique,
        primary key(staff_id),
        foreign key (dept_id) references dept(dept_id)
        foreign key (manger_id references staff(staff_id)
);

Create table staff-details (
        staff_id int,staff_first_name varchar(10), staff_mid_name varchar(15),
        DOB date, address  varchar(50),email varchar(50),ext phone varchar(10),
        homephone varchar(20)
        primary key(staff_id),
        foreign key (staff_id) references staff(staff_id));
        foreign key (staff_id) references staff-details (staff_id)
);
```

- Rule C2, Hierarchy class rule which is also provided by Alalwan is utilized in this research to define parent and child relationship between two entities. Rule C2 is shown in the figure 7. For example, according to SQL-DDL as follow, *graduate-student* is the subclass of *student*.

```
Create table student (
        st_id int, st_name varchar(70) not null, sex char(1),
        module-name varchar(70) unique ,dept_id int not null,
        n-id unique not null,
        primary key(st_id),
        foreign key (dept_id) references department (dept_id)
);
```

```
Create table graduate-student (
    st_id int, research_area varchar(70),
    primary key(st_id),
    foreign key (st_id) references student(st_id)
);
```

$$Rule_{C_{1cc}} = \begin{cases} -Class\ condition: \\ \exists x, y: \begin{pmatrix} (a) PK(x, R_1)\ \wedge\ PK(y, R_2) \wedge \\ (b) FK(y, R_2, x, R_1)\ \wedge \\ (c) FK(x, R_1, y, R_2) \end{pmatrix} \\ -Class\ Action: \\ Class\ (C)\ \leftarrow\ (R_1 \cup\ R_2) \end{cases}$$

Figure 6: Rule C1, Fragmentation class rule (Alalwan, 2011, p 96).

$$Rule_{C_2} = \begin{cases} -Class\ condition: \\ \exists x, y: \begin{pmatrix} (a) PK(x, R_1)\ \wedge\ PK(y, R_2) \wedge \\ (b) FK(y, R_2, x, R_1) \end{pmatrix} \\ -Class\ Action: create \\ \begin{pmatrix} 1)\ Class\ (C_1)\ \leftarrow R_1 \\ 2)\ Class\ (C_2)\ \leftarrow R_2 \\ 3)\ SubClass\ (C_2,\ C_1) \end{pmatrix} \\ where \\ C_1\ and\ C_2\ are\ not\ already\ created \end{cases}$$

Figure 7: Rule C2, hierarchy class rule (Alalwan, 2011, p 99).

After gathering all the relationships between entities, ERD/EERD to OWL Mapping Rules which are represented in the research of Fahad (2008, p35) are utilized to build the ontology in the thesis. Fahad (2008) uses Chen's model, and the name of relationship between two entities is given, but in the thesis only DDL is available. As a result, the relationship between two entities is one to many, the name of object property is created and named with "C1hasManyC2", *C1* is the domain name of the object property, and *C2* is the range name of the object property. The reverse object property can be set as "C2haveManyC1", which *C2* is the domain class of this object property and *C1* is the range class of the property. For example, the one-to-many relationship between *Department* and *Instructor* is represented as OWL as follow.

```
<owl:ObjectProperty rdf:ID=" DepartmentHasManyInstructor ">
        <rdfs:domain rdf:resource="#Department"/>
        <rdfs:range rdf:resource="#Instructor"/>
        <owl:inverseOf rdf:resource="# InstructorsHaveADepartment "/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID=" InstructorsHaveADepartment ">
        <rdfs:domain rdf:resource="# Instructor "/>
        <rdfs:range rdf:resource="# Department "/>
        <owl:inverseOf rdf:resource="# DepartmentHasManyInstructor "/>
</owl:ObjectProperty>
```

5.2. Similarity between Entities (*Hausdorff* distance)

The algorithm of calculating similarity between entities will be utilized for schema integration. For simplicity, *Hausdorff* distance is utilized to calculate similarity between two entities to decide if two or more entities should be merged.

The concept similarity computation algorithm proposed by Guohui, Guoren, and Bin (2010) is adopted in the research. The algorithm is shown as follow.

Definition Let $Let\ C_a\ and\ C_b\ be\ two\ concepts, C_a = \{a_1, a_2, \ldots, a_i, \ldots, a_n\} and\ C_b = \{b_1, b_2, \ldots, b_j, \ldots, b_m\}. We\ define\ the\ similarity\ between\ C_a\ and\ C_b\ to\ be$:

$$S(C_a, C_b) = 1 - \frac{1}{2}\left(\frac{1}{n}\sum_{i=1}^{n} d(a_i, C_b) + \frac{1}{m}\sum_{j=1}^{m} d(b_j, C_a)\right)$$

(Guohui, Guoren, & Bin, 2010, p 151).

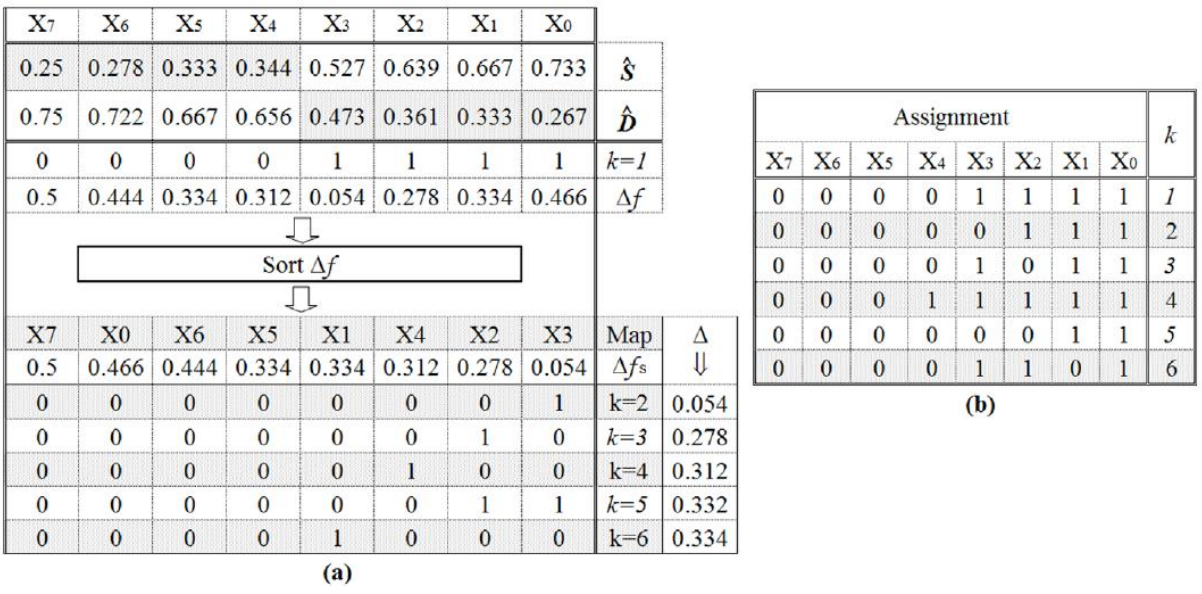
The notation $d\ (a_i, C_b)$ denotes the distance from the element a_i to the set C_b. If a_i does not have any correspondences to any attribute in C_b, then the value of $d\ (a_i, C_b)$ is 1, else is 0. Since the distance C_a to C_b is different with distance C_b to C_a, the algorithm take the average of two directed distance.

In the ontology based schema integration system, the similarity computation algorithm is enhanced for preserving the hierarchical structure and the weak entities, which is introduced in section 7. *Hausdorff* distance can be substituted by other algorithms which can also be utilized to calculate the similarity between two entities.

5.3. Top-k Algorithm

After gathering similarities of classes between two different ontologies, the top k algorithm, which is shown as follow in the figure 8, is adopted from the approach proposed by Radwan, Popa, Stanoi, and Younis (2009). The algorithm can be expressed as several steps:

1. The similarities of the classes are inserted into the array if any correspondences exist between them.
2. Initial the assignment. If the similarity is higher than dissimilarity, then set the bit in the assignment to 1, otherwise, set to 0. Assignment is a set of vectors that decide if two concepts should be merged.
3. Get delta array. Delta equals to *ABSOLUTE (Sim - Dis).*
4. Sort the delta array.
5. Flip the bits in assignment if the bit or sum of the bits in delta array has the lowest value.



X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	
0.25	0.278	0.333	0.344	0.527	0.639	0.667	0.733	$\hat{S}$
0.75	0.722	0.667	0.656	0.473	0.361	0.333	0.267	$\hat{D}$
0	0	0	0	1	1	1	1	*k=1*
0.5	0.444	0.334	0.312	0.054	0.278	0.334	0.466	Δf

Sort Δf

X7	X0	X6	X5	X1	X4	X2	X3	Map	Δ ⇓
0.5	0.466	0.444	0.334	0.334	0.312	0.278	0.054	Δf_S	
0	0	0	0	0	0	0	1	k=2	0.054
0	0	0	0	0	0	1	0	*k=3*	0.278
0	0	0	0	0	1	0	0	k=4	0.312
0	0	0	0	0	0	1	1	*k=5*	0.332
0	0	0	0	1	0	0	0	k=6	0.334

(a)

Assignment								*k*
X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	
0	0	0	0	1	1	1	1	*1*
0	0	0	0	0	1	1	1	2
0	0	0	0	1	0	1	1	*3*
0	0	0	1	1	1	1	1	4
0	0	0	0	0	0	1	1	*5*
0	0	0	0	1	1	0	1	6

(b)

Figure 8: Top-k algorithm (Radwan, Popa, Stanoi, & Younis, 2009, p 647).

Before generating the result of the merging ontology, the developed ontology merging approach detects each merged class. If any merged class is composed of classes that has hierarchical relationships between them, for instance, student class, graduate student class, and other classes are ready to be merged, but student class is the supper class of graduate student class, system will extract the parent class, examining if other class should merged to parent class or its child class.

The inference of ontology can be used during merging two ontologies to reduce the size of final merged ontology. For example, *A*, *B*, and *C* are three classes in ontology. If class *A* equals to class *B*, which can be represented by axiom of *SameAs* in ontology, and B is the sub class of *C*, *A* is sub class of *C* can be inferred. In the thesis, the developed approach utilizes this feature to preserve schema hierarchical structure and prevent the redundant attributes in the hierarchy structure. For example, as mentioned above, *A* equals to *B*, and *C* is supper class of *B*, then, *A* is sub class of *C* can be inferred. When doing the schema integration, if any top-k schema merging information contains the information such that *A* needs to merge with *B*, then the developed approach will search the list of *B*'s supper classes. If any attribute matching exit between *A* and *C*, since *C* is *A*'s supper class, attributes which matches to *A* in *C* will be removed by the system. The same idea can be used in the case of that class *A* equals to *B*, and *B* is super class of *C*. Merging system will remove attributes in *A* that matches to *C*.

6. Case Study and Prototype Implementation

In the thesis, different comparisons are tested. The schema integration system which proposed by Radwan, Popa, Stanoi, and Younis (2009) is implemented. From integrate the ER Diagram to ER Diagram, ER Diagram to EER Diagram, and EER Diagram to EER Diagram, various merging

result are presented in this section, and the enhanced top-k schema integration approaches are introduced in details as well.

6.1. ERD TO ERD

UniversityOneERD ER Diagram and UniversityTwoERD ER Diagram are shown in the figure 9 and figure 10. ER Diagrams are generated by SQL-DDL. SQL-DDL of UniversityOneERD and UniversityTwoERD are in Appendix A.

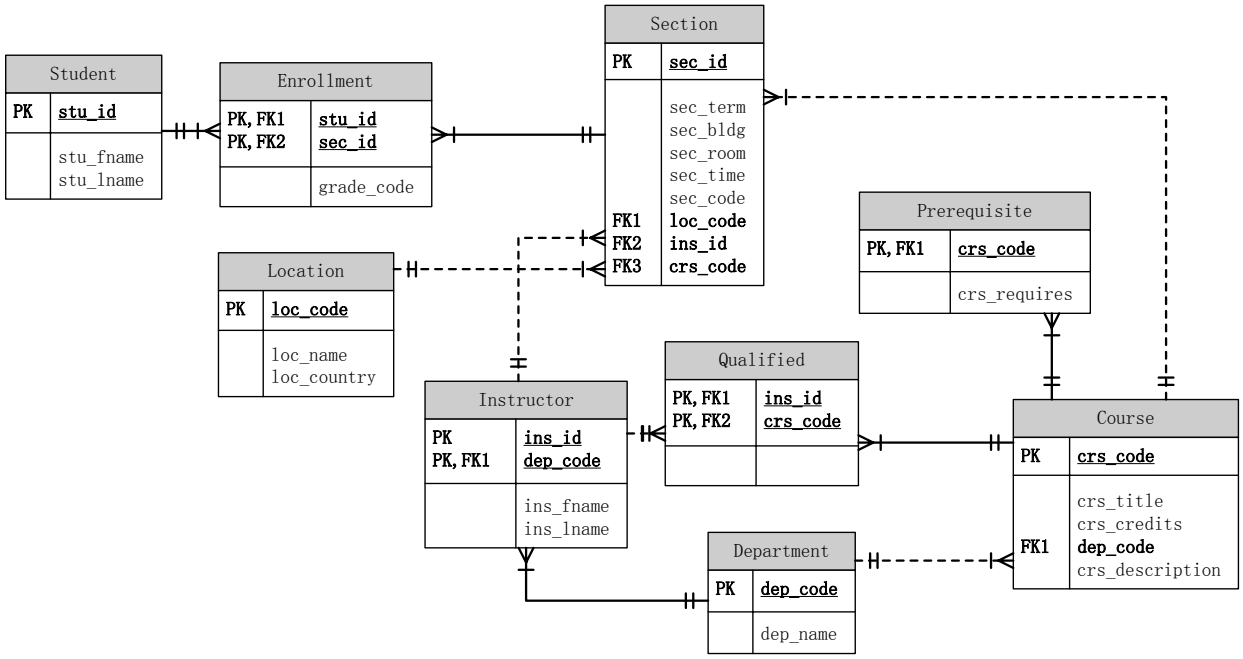


Figure 9: UniversityOneERD ER Diagram.

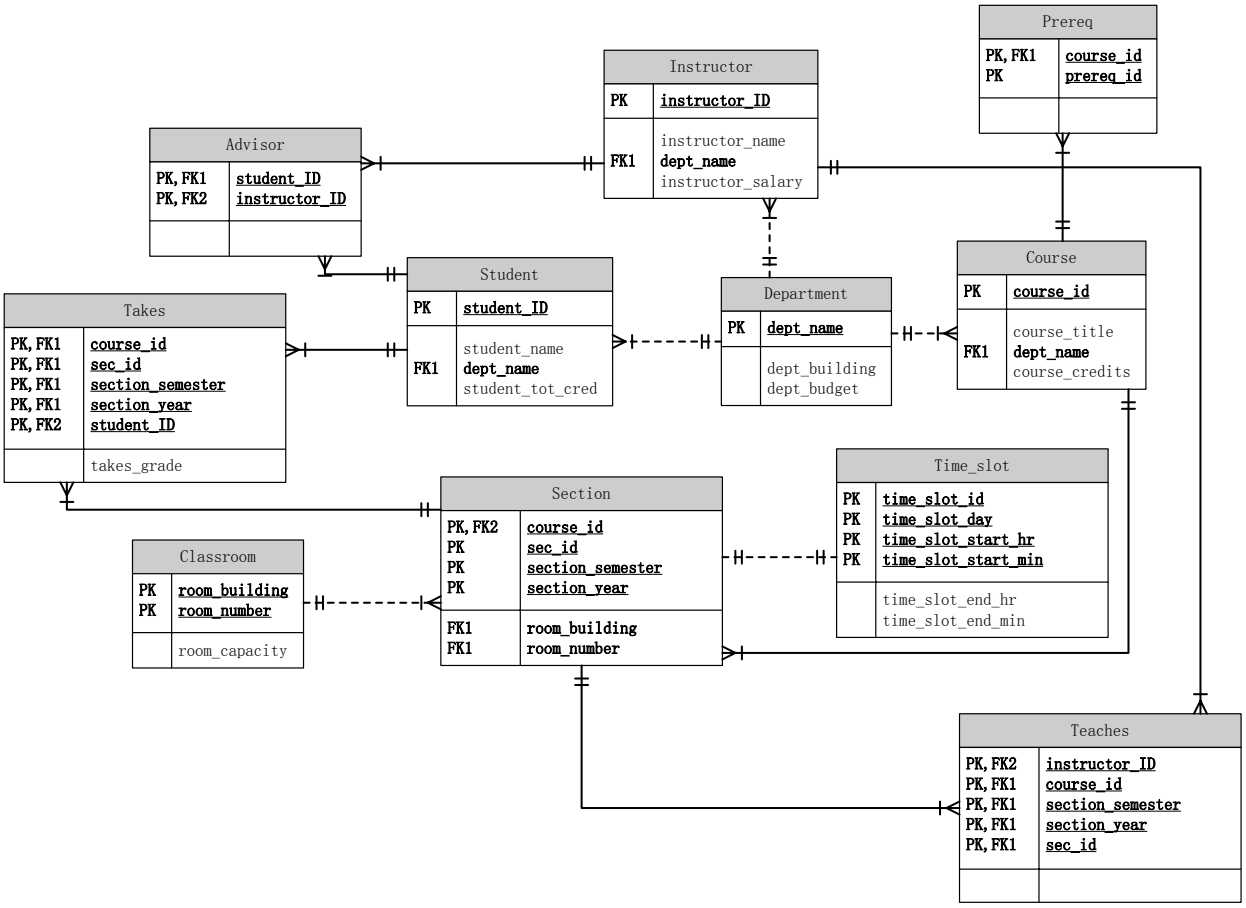


Figure 10: UniversityTwoERD ER diagram.

6.1.1. The Schema Integration Approach Posed by Radwan, et. al. (2009)

Follow the approach proposed by Radwan, Popa, Stanoi, and Younis(2009), the nested schema format needs to be generated first. The nested format of UniversityOneERD schema and UniversityTwoERD schema are in Appendix C.

Since the thesis focuses on schema integration, we assume the schema matching results are available through some existing schema matching algorithm or can be manually created by users. Figure 11 shows the graphical matching results, and figure 12 presents the results in textual format.

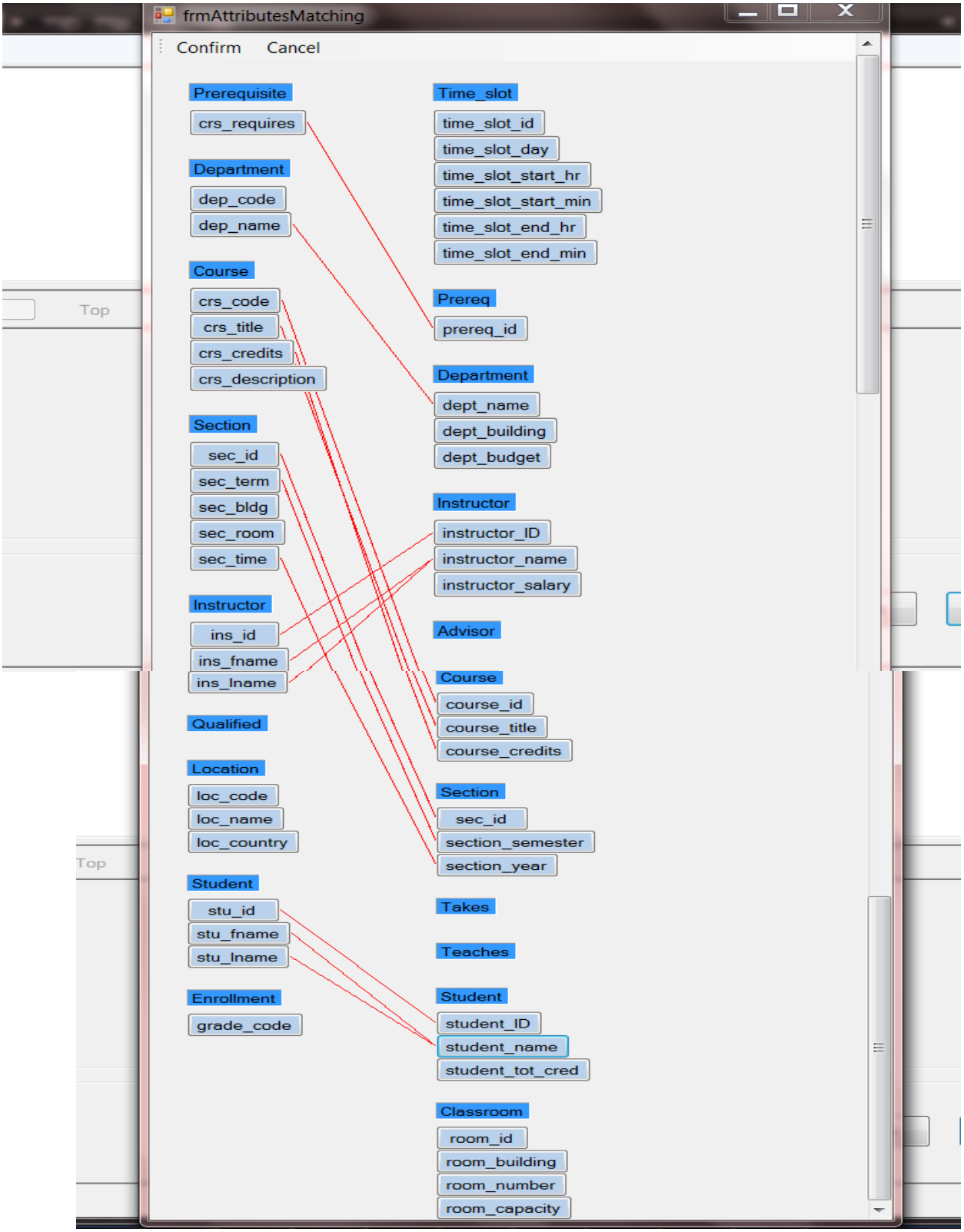


Figure 11: ERD to ERD schema matching.

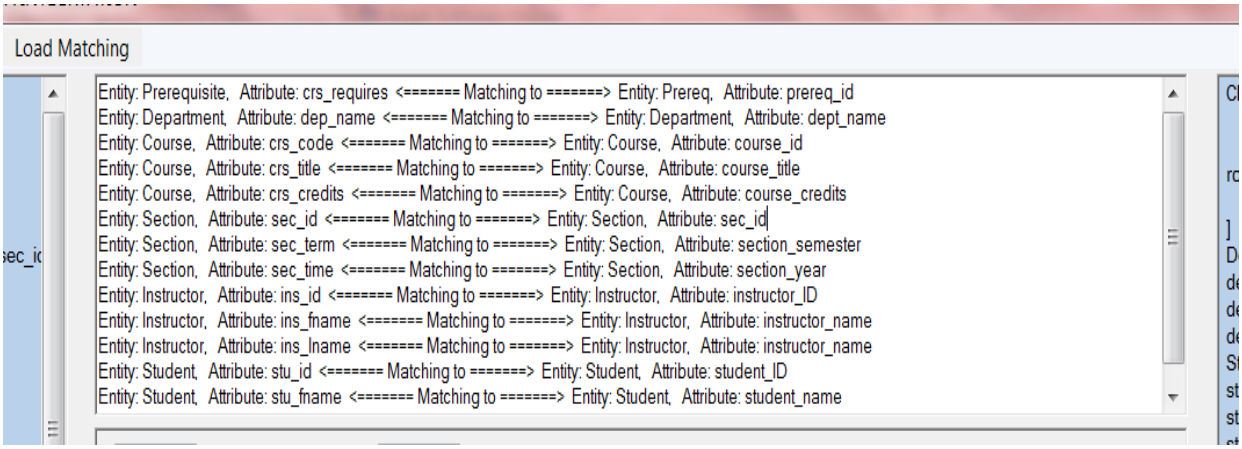


Figure 12: ERD to ERD schema matching text format.

After matching, top n possible schema integrations are given by system. In this case, top 2 is picked since it is the most likely the merging result. It is shown in figure 13.

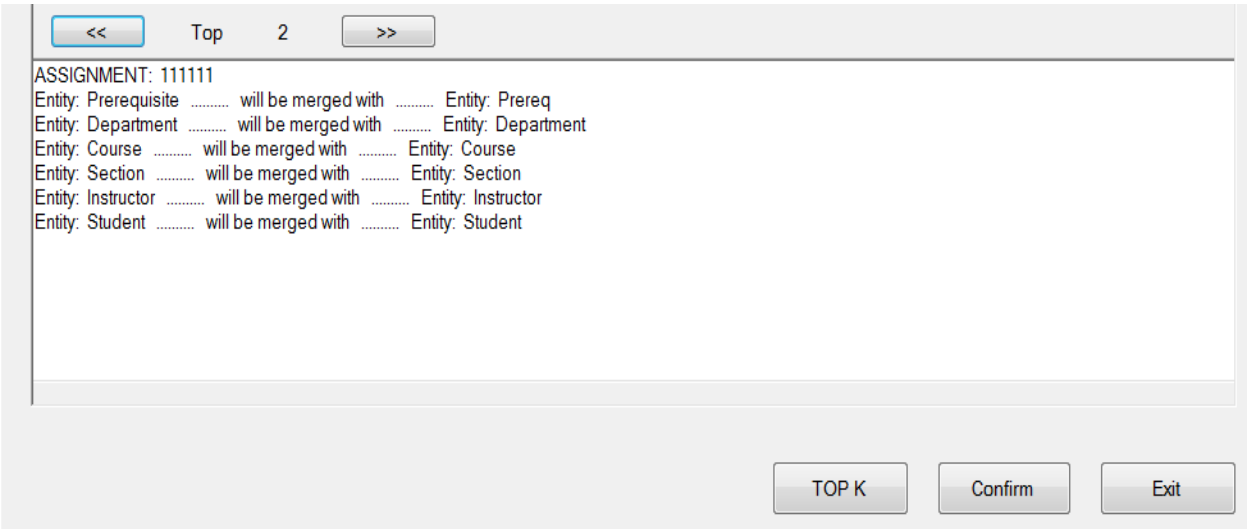


Figure 13: Top 2 merging assignment.

Merging information will be given to users to avoid conflicts during the merging. It is shown in figure 14. For example, after representing merging information, if the relationship between *A* and *B* are both one to many and many to one, then, from user's feedback, one relationship will be deleted to avoid relationship conflict. The list box on the left top shows *Class* merging information, the list box on the right top shows *Attributes* merging information, and the list box on the bottom shows *Relationship* merging information.

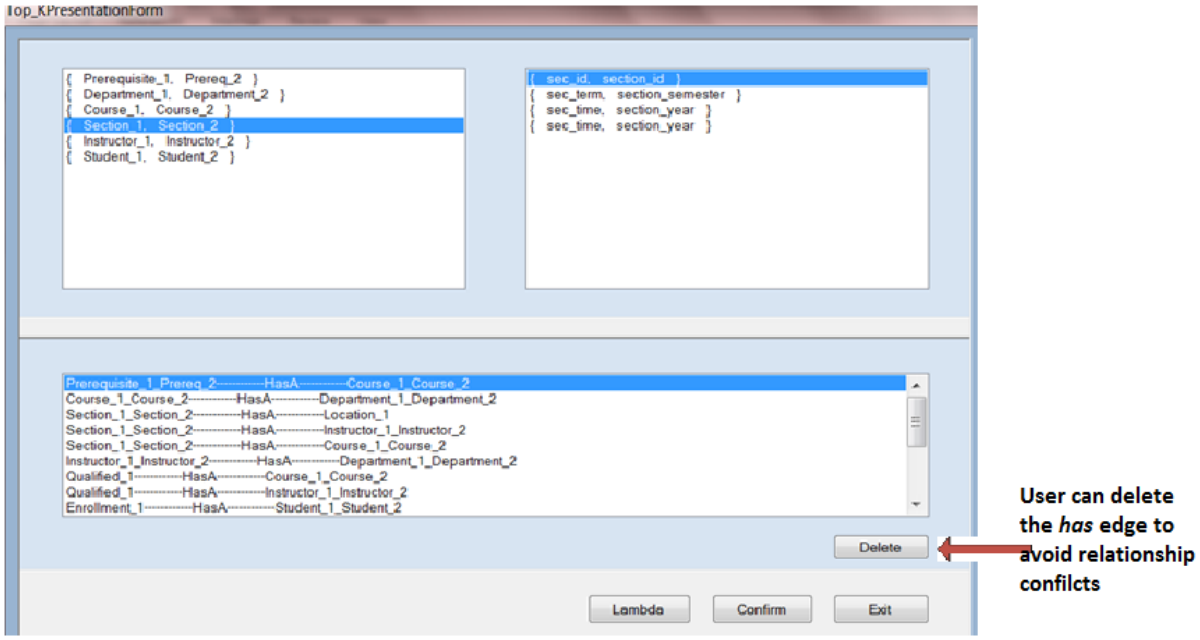


Figure 14: Merging information.

After clicking the confirm button, the result will be displayed. The nested format integrated schema will be displayed. The whole content of the schema is in Appendix D. The ER diagram of the final integrated schema is represented in figure 15.

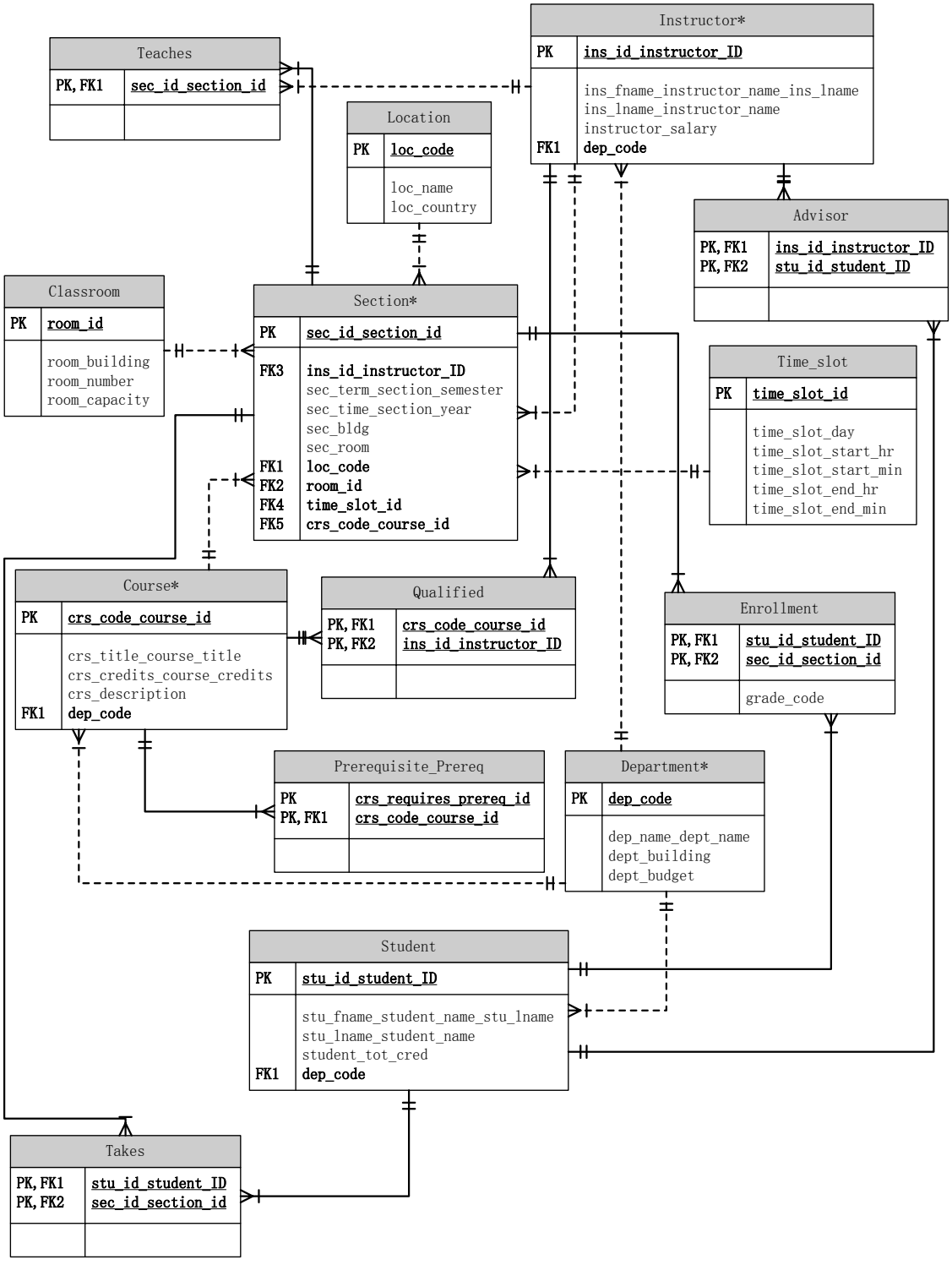


Figure 15: Integrated university ERD.

6.1.2. The Proposed Ontology-based Schema Integration Approach

In the thesis, the proposed approach utilizes ontology as a merging model. As introduced above utilizing ontology can support more information to the user, the object properties, restrictions, and parent and child frames are adding in this system to represent merging information to users. The tool named *Protégé-OWL* which is designed by Stanford University is used to convert relational schemas to ontologies. The ontologies of UniversityOneERD and UniversityTwoERD are in Appendix E.

The proposed approach adopted the top-k ranking algorithm in Radwan, Popa, Stanoi, and Younis's approach (2009). The system allows users to modify the merging information, for example, renaming the merging class, to build more accurate final integrated schema. The part of displaying merging information in the schema integration system is shown in figure 16a.

Relational conflicts and constraint conflicts can be manually prevented by users' feedback. For example, in figure 16b, the system allows users to edit, add, or remove relationships between two classes. The object properties can be modified, and *subclassOf* axiom could insert and delete between two classes. In figure 16c, it shows that the range of datatype properties can be changed manually by domain experts.

The more enhanced approach to avoid relational conflicts happening is introduced in section 7.2.

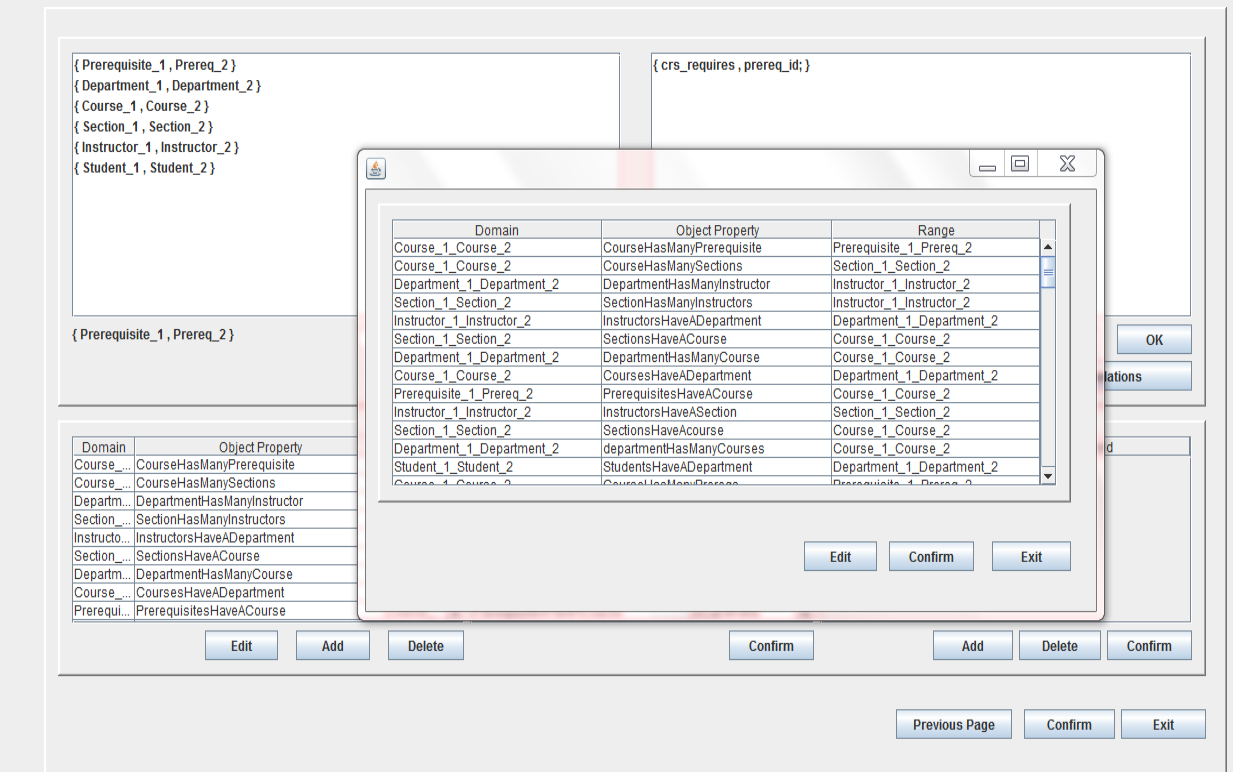


Figure 16: a) Ontology merging information (object properties).

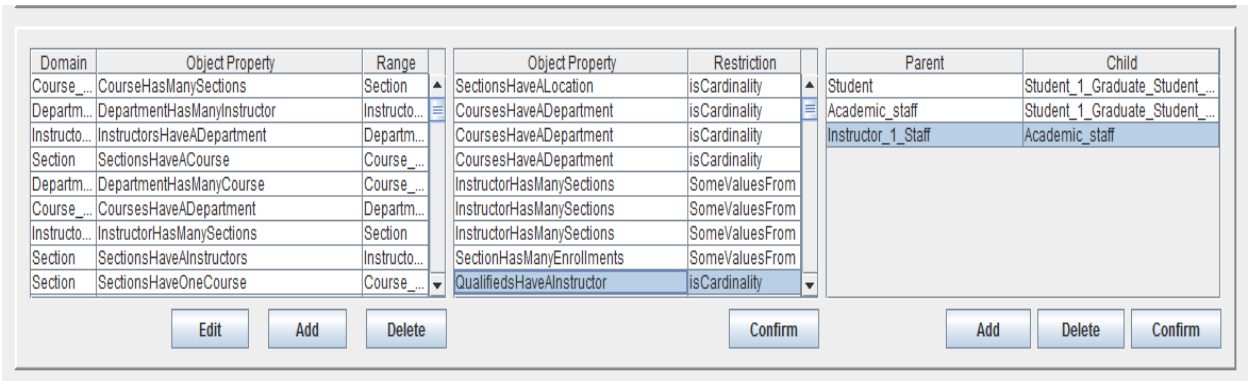


Figure 16: b) Ontology merging information (subclassOf).

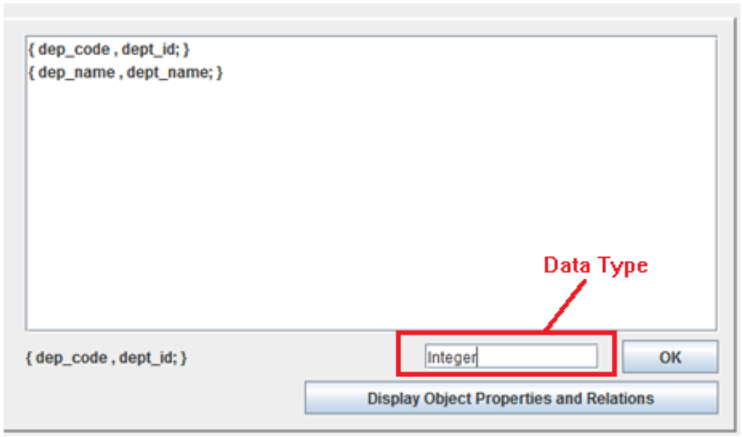


Figure 16: c) Set new data type to the datatype property.

The merging owl file is in Appendix E and ERD representation of merging result is shown in figure 17.

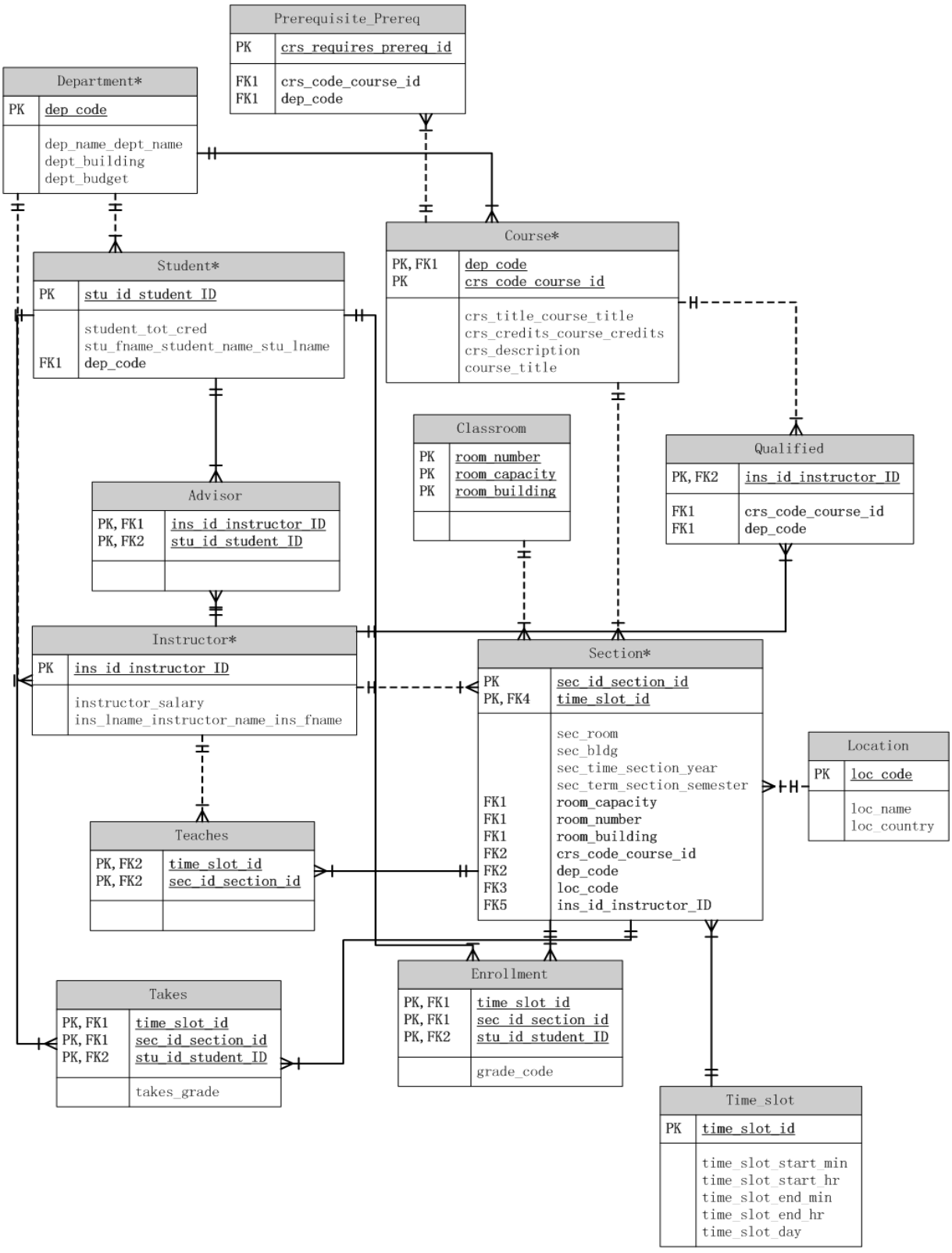


Figure 17: Integrated university ERD by using ontology.

The two schema integration approaches, the concept graph approach (CGA) and the ontology-based approach (OBA) generate very similar results when merging two relational schemas without any inheritance relations. The two merging results in figures 15 and 17 have no difference since the ontology-based schema integration system is built up on previous top-k schema integration systems. Since the main purpose in the thesis is to preserve the hierarchical structure/inheritance relationships in local schemas, if input schema does not contain hierarchical structure, the final result will be the same. The result that merging ERD to EERD is shown in the next section.

6.2. ERD TO EERD

UniversityOneERD ER Diagram is used in this section as local schema. For the reason to demonstrate the enhanced performance of the new OBA approach in the thesis, UniversityOneERD ER is modified a little bit. More attributes are inserted into the *Student* entity. The modified UniversityOneERD ERD is shown in figure 18. The EER Diagram of UniversityThreeEERD is shown in figure 19.

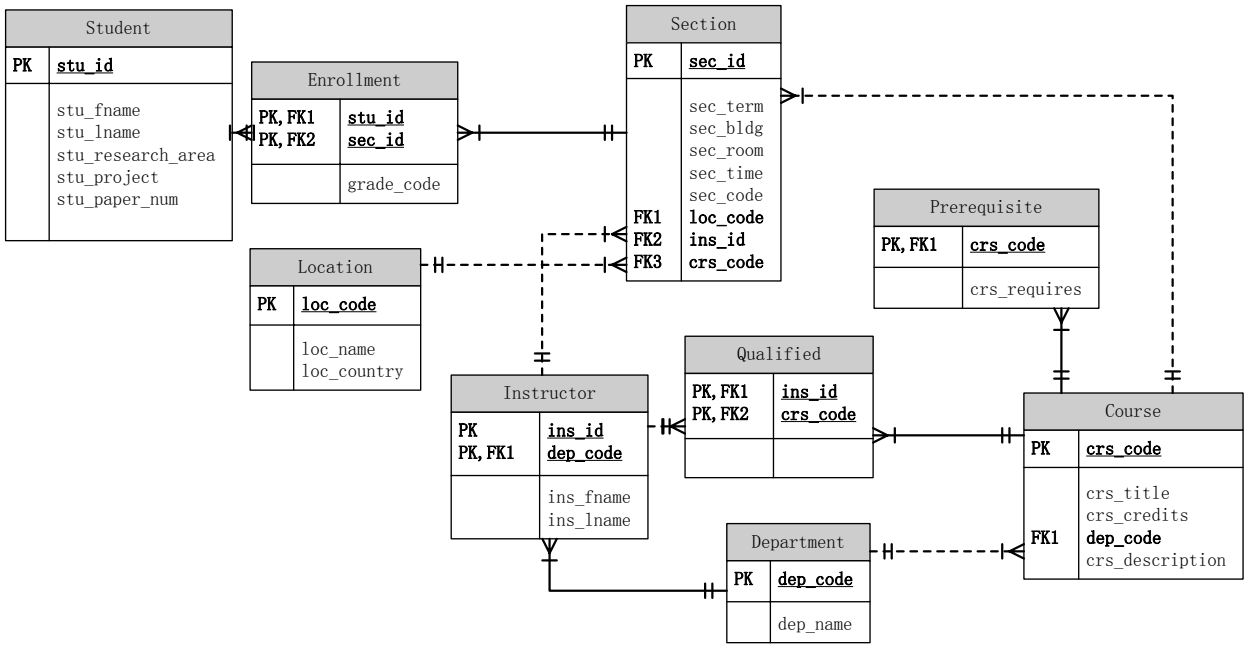


Figure 18: Modified UniversityOneERD ERD.

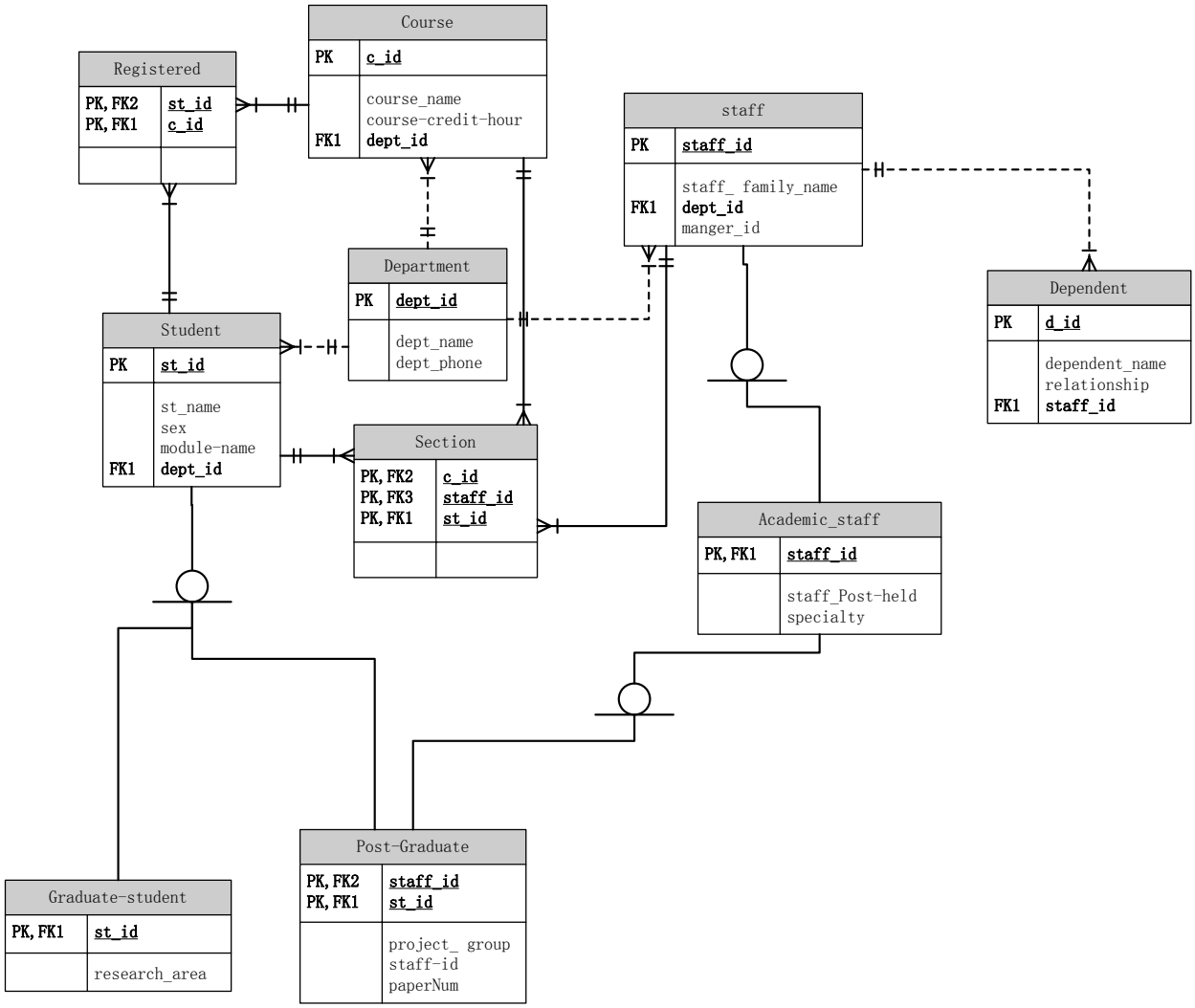


Figure 19: UniversityThreeEERD EER diagram.

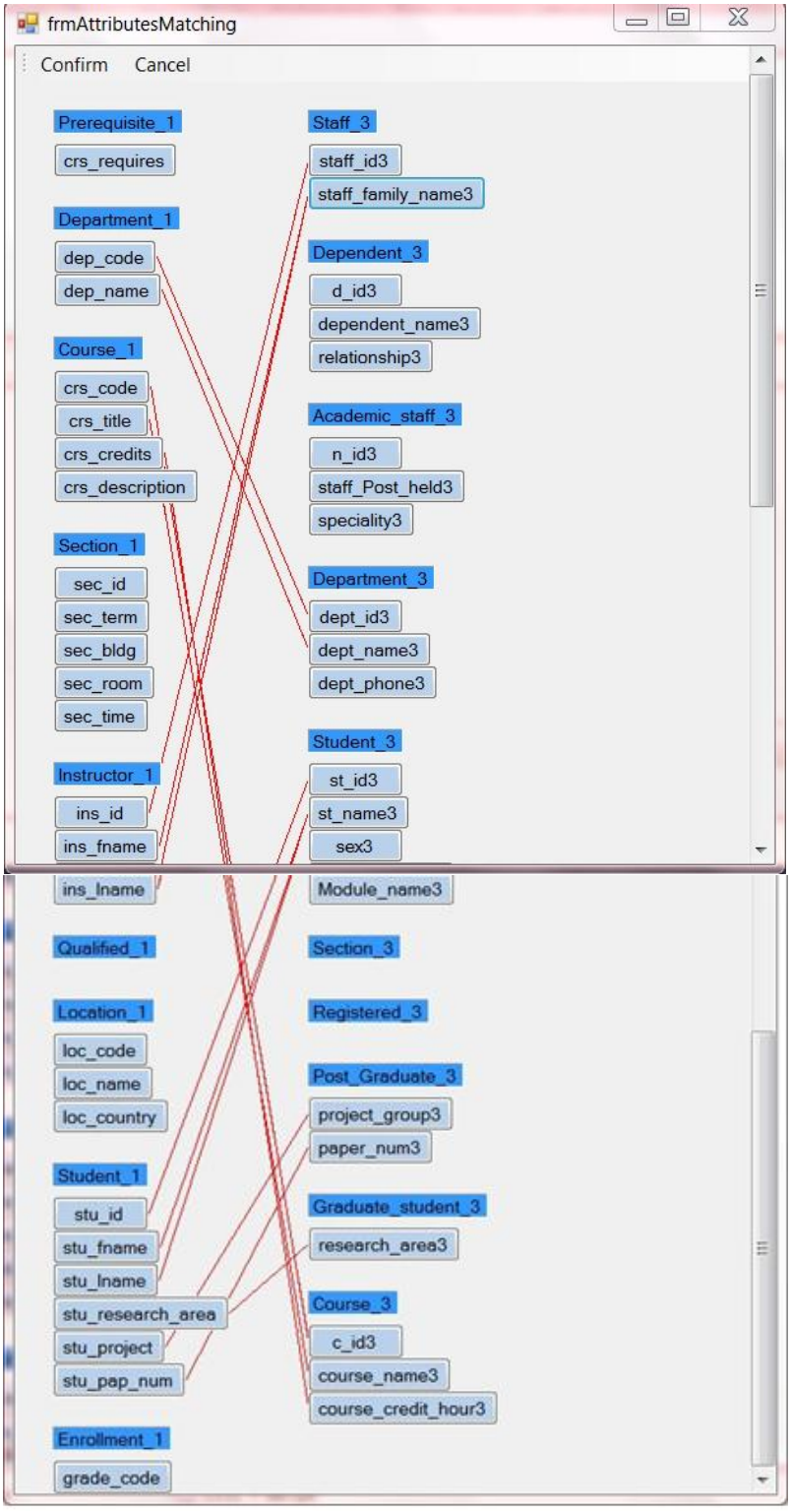


Figure 20: The matching of UniversityOne and UniversityThree.

After confirming the top merging result, in this case, Student of schema one will merge to *Student*, *Graduate_student*, and *Post_Graduate student* in the schema two. This is shown in figure 21. The ERD of integrated schema is shown in figure 22.

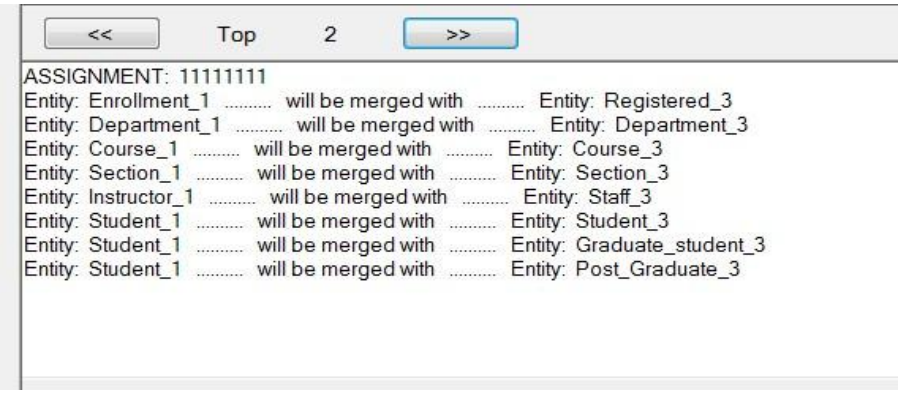


Figure 21: ERD TO EERD merging information.

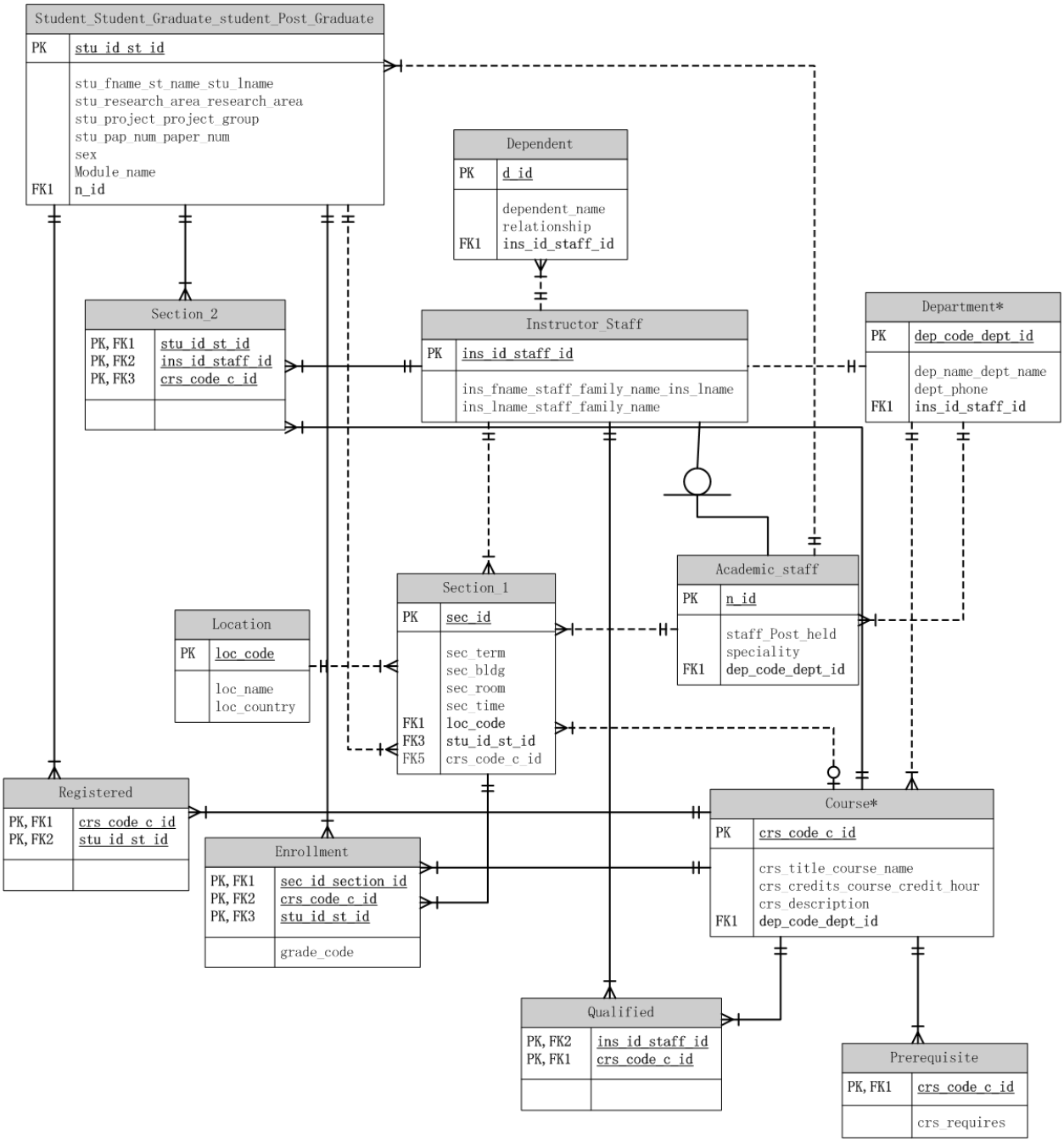


Figure 22: Integrated EERD of university schema.

Using the same *assignment* in ontology schema integration approach, hierarchy structure information is displayed to the user. If users confirm to generate integrated ontology, since the merging result can break the hierarchy structure, system will deny generating the merging result. This is shown in figure 23.

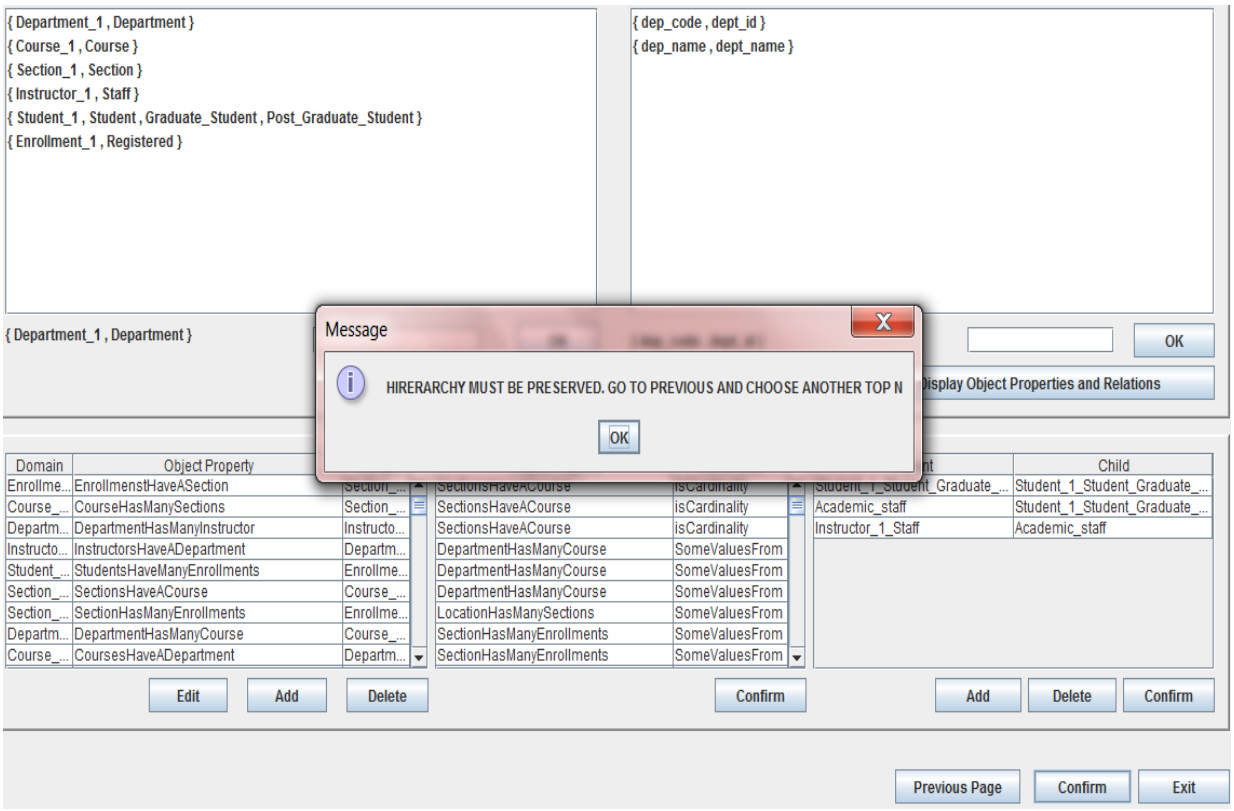


Figure 23: Refuse to generate the result.

Users can go to previous pages to select another top-k merging assignment. If the assignment that user picked does not break the hierarchical structure, for example, user picks the assignment in figure 24. Then system will generate the final integrated ontology. After converting OWL to relational schema, integrated EER Diagram is shown in figure 25.

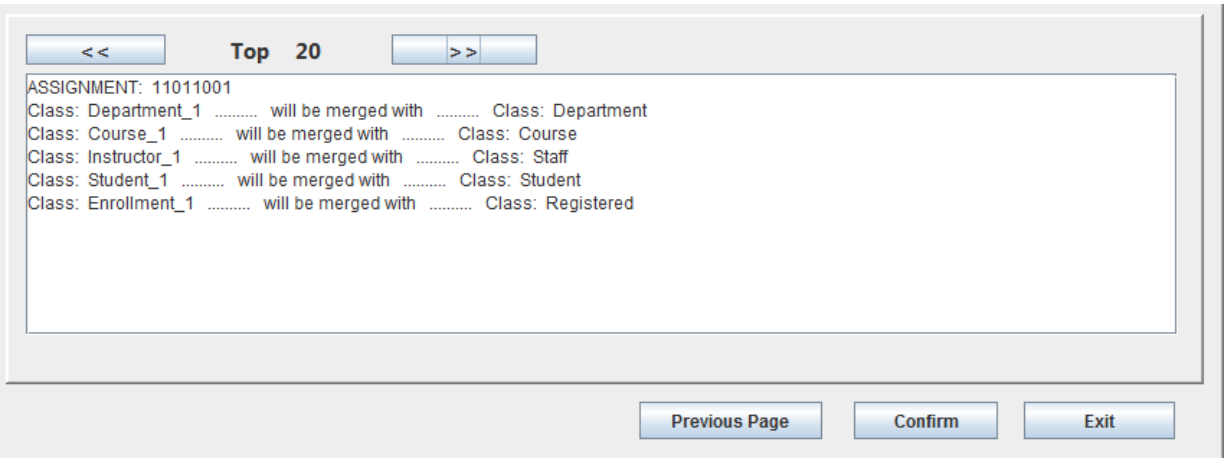


Figure 24: Assignment no breaking the hierarchy structure.

The integrated schema in figure 25 displays the merging expected result, the hierarchy structure is preserved in this schema integration system. Both *Graduate Student* and *Post Graduate Studen*t entities only have key attributes since other attributes which exist in original local schema are matching to the attributes of the entity that merged to the parent entity of them. For example, attribute *stu_research_area* of the *Student* entity in schema one is matching to attribute *research_area* of the *Graduate_student* entity in schema two, and the *Student* entity in schema one is defined as equals to *Student* entity in schema two. *Student* entity in schema one can be inferred that it is also the parent entity of the *Graduate_student* entity in schema two. For the simplicity, if two attributes are defined equivalent if they are matching.

Therefore, Child only has key attribute in *Graduate_student* entity since redundant attributes are removed by the system.

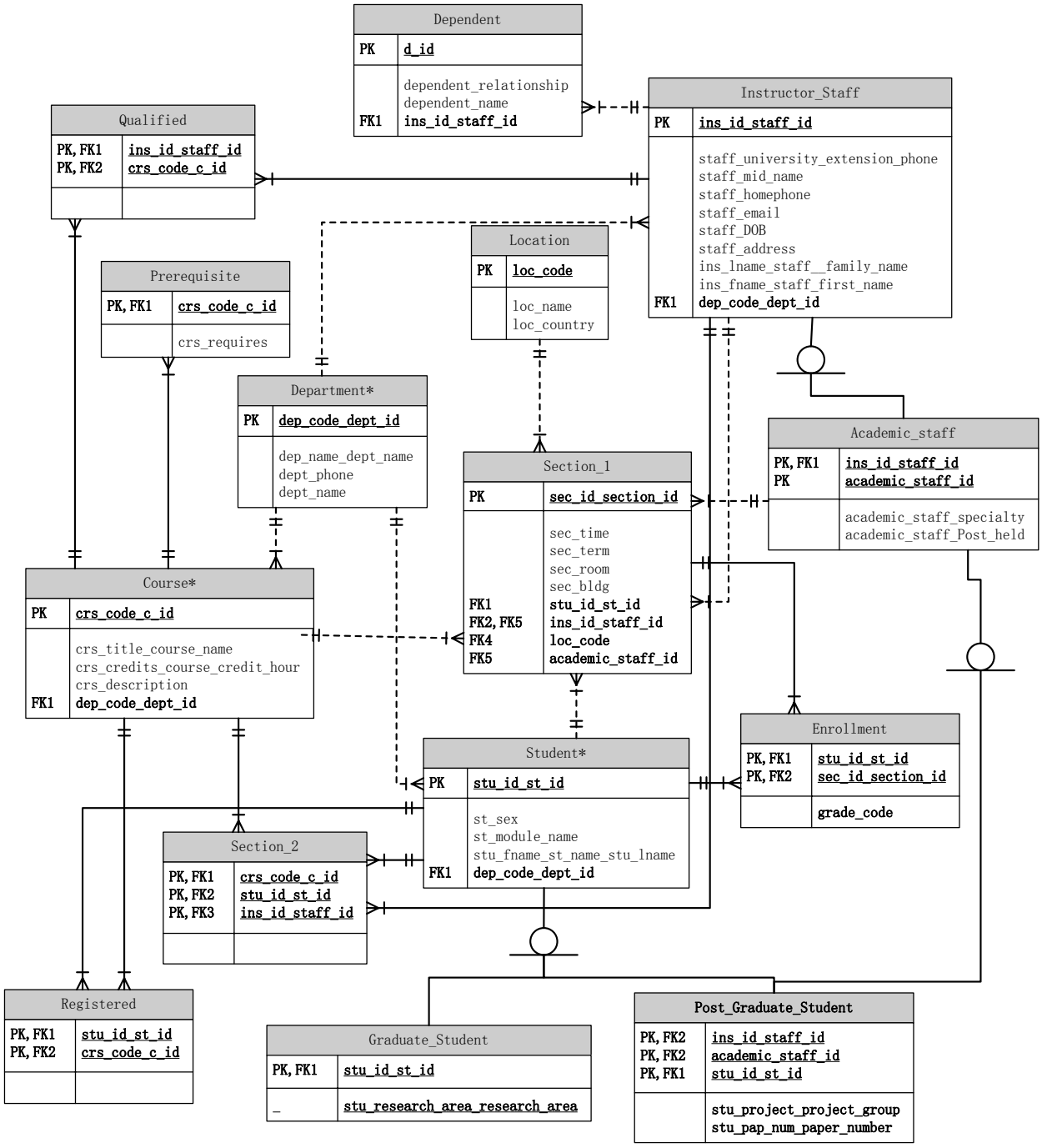


Figure 25: Ontology approach merging result of ERD to EERD.

7. How the Ontology Integration Approach Enhance the System

From the experimental evaluation in section 6, it is shown that the hierarchical structure is preserved manually by users. In first part of this section, a better way to automatically preserve the hierarchical structure is proposed by using the factors of the ontology, such as *datatype properties*, and *subClassOf* axioms. How the other factors of the ontology, such as *object properties*, *domain*, and *range*, enhance the integration system is explained in the second part of this section.

After generating the top-k assignment (for example, see the assignment in figure 20), the schema integration system detects that the candidate merging assignment contains hierarchy structure. If the assignment contains hierarchy structure, which is shown in figure 26, the system will go to the hierarchical structure analysis procedure. First, similarity between two classes is recalculated by the Ontology approach. Then, the system merges the matching classes with the highest similarity. New hierarchical structure will be built base on original hierarchical structure, and redundant datatype properties will be removed by the system at last.

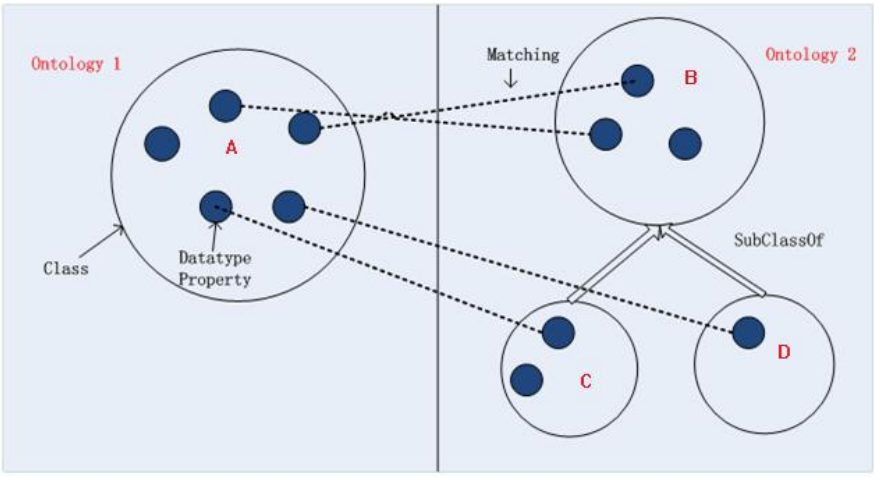


Figure 26: Candidate assignment contains the hierarchical structure.

7.1. Hierarchical Structure Analysis Procedure

First, the new similarity computation algorithm is given as follow.

Definition 1: Let C_a and C_b be two classes, *Mp* be the properties in C_A matches properties in C_B. Dp_{C_k} is datatype property which has the domain class as class C_k, n is the number of subclass of the C_k. We define the similarity between C_A and C_B to be:

$$Sim_{AB} = Num(MP) \,/((Num(Dp_A) + \sum_{i=1}^{n} Num\left(Dp_{subclassOf(A)}\right))$$

$$Sim = \frac{(Sim_{AB} + Sim_{BA})}{2}$$

The definition of *subClassOf* axiom given by W3C is that "if the class description *C1* is defined as a subclass of class description *C2*, then the set of individuals in the class extension of *C1* should be a subset of the set of individuals in the class extension of *C2*. A class is by definition a subclass of itself" (Smith, et. al., 2004). From the definition, the meaning of that *C1* inherits all the properties of *C2* can be obtained. Therefore, from the case in figure 26, both subclass *C* and *D* inherit properties in the super class *B*, and class *B* contains properties of *B*, *C*, and *D*.

In figure 26, class A has 5 attributes, class *B* has 6 attributes by combining attributes from the subclasses *C* and *D*, class C has 5 attributes by inheritance from class *B*, and class *D* has 4 attributes by inheritance from class *B*. Consider *B*, *C*, and *D* as a group, which has a total of 6 attributes.

(1) Calculate the similarity from *A*→*B*, *A*→*C*, and *A*→*D*:

The similarity from *A*→ *B* is *4/5*, from *A*→ *C* is *3/5*, and from *A*→ *D* is *3/5*.

(2) Calculate the similarity from *B*→ *A*, *C*→*A*, and *D*→ *A*:

The similarity from *B* → *A* is 4/6, from *C* → *A* is 3/5, and from *D* → *A* is 3/4.

By taking the average of bi-direction similarity values, the similarity between *A* and *B* is the highest, so we merge *A* with *B*. Using the same approach, we can get all pairs of the matching classes. For example, in the figure 27, merging *C* and *A*, and merge *D* and *A* can be obtained by the ontology similarity computation algorithm.

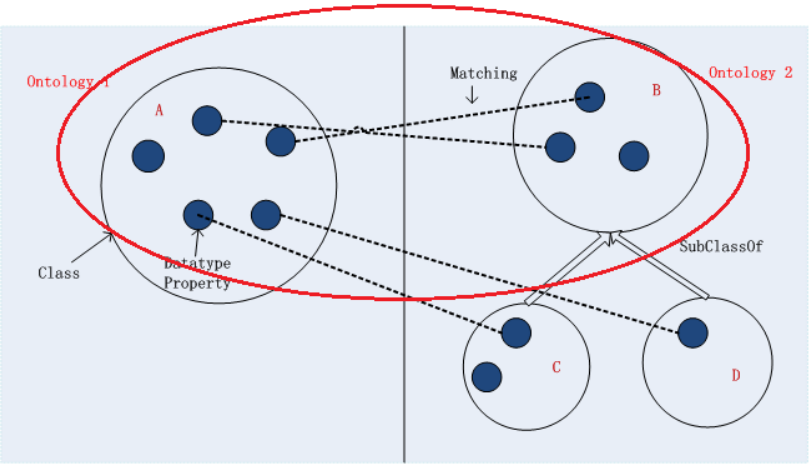


Figure 27: Merging class *A* with *B*.

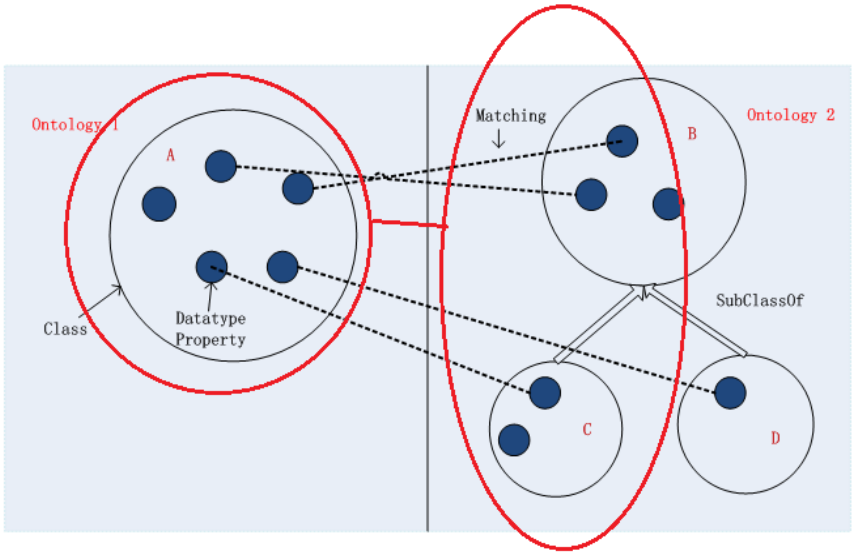


Figure 28: a) Merge class *A* with *C*.

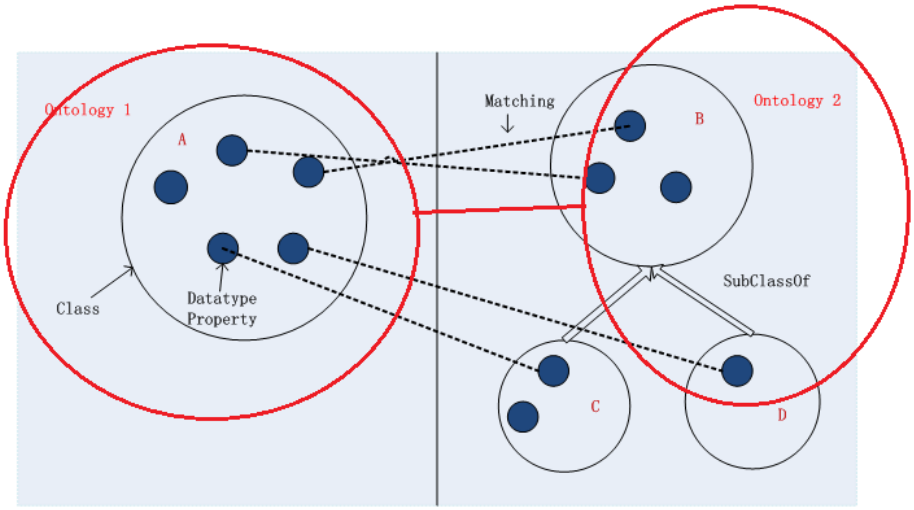


Figure 28: b) Merging class *A* with *D*.

The third of the new hierarchical structure analysis procedure is to build hierarchical structure based on original hierarchical structure. For example, in figure 28, class *A*-*D* consists of

A and *D*, class *A-B* consists of A and *B*, since *D* is subclass of *B*, the *subClassOf* axiom needs to be added between *A-B* and *A-D*, and express as *A-D subclass A-B*.

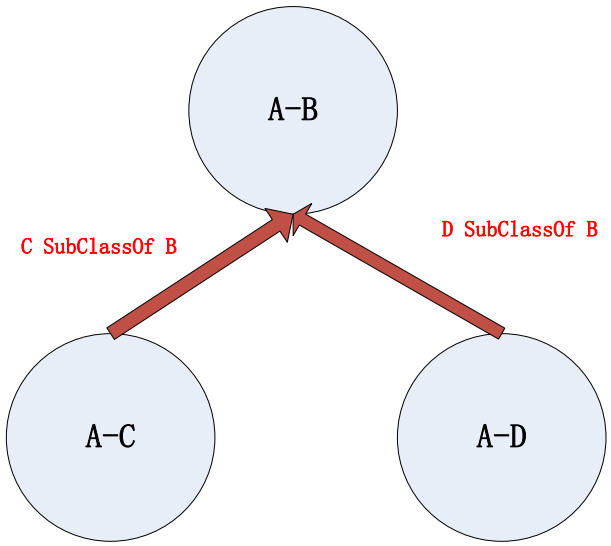


Figure 29: Insert *SubClassOf* between new merged classes.

The Fourth step is to remove the redundant properties between super classes and subclasses. Figure 30 displays the merged classes with redundant properties. The rules remove redundant properties and are given as follow:

1. If the list of the domain classes of the property contains both supper class and all subclasses, then remove the subclass from the domain list of that property. It is shown in figure 31 a.
2. If any original datatype properties exist, for example, in figure 31 a, *d* is original datatype property since datatype property *d* merged with datatype property *i* already, then remove the original datatype as shown in figure 31 b.
3. Remove the datatype properties in super class if the properties exist in both super class and sub class. This is shown in figure 31 c.

   The result generated by ontology integration approach is shown in figure 32.

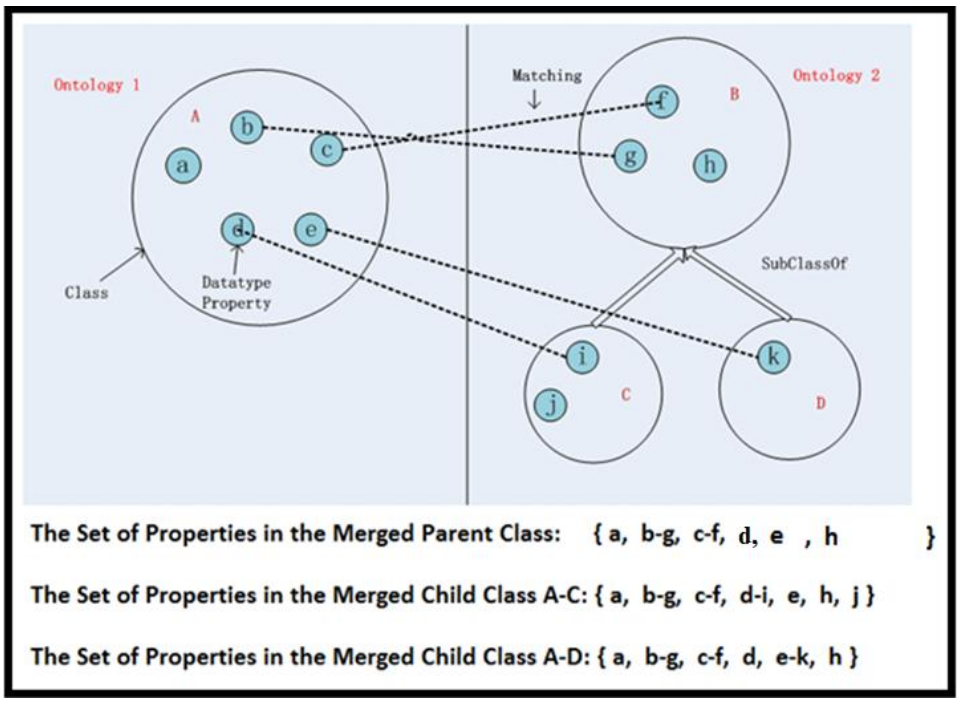


Figure 30: Merged parent and child Classes with redundant properties.

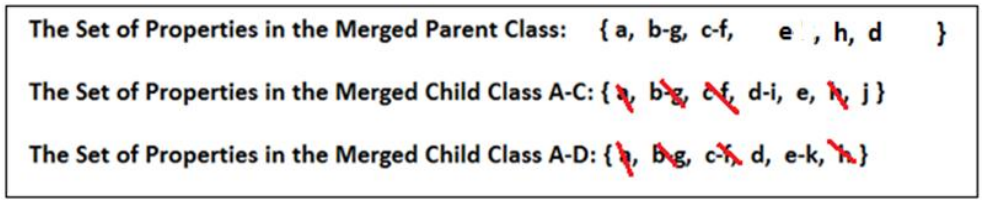


Figure 31 a) Remove properties in subclass.

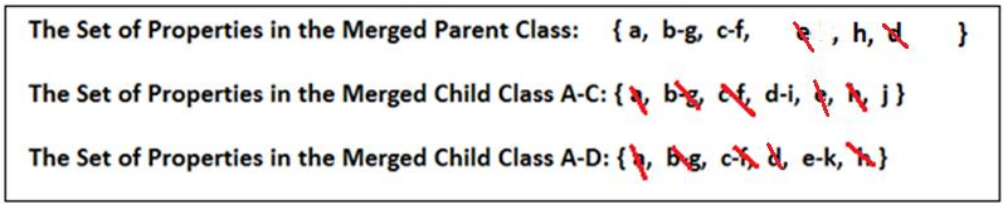


Figure 31 b) Remove original properties.

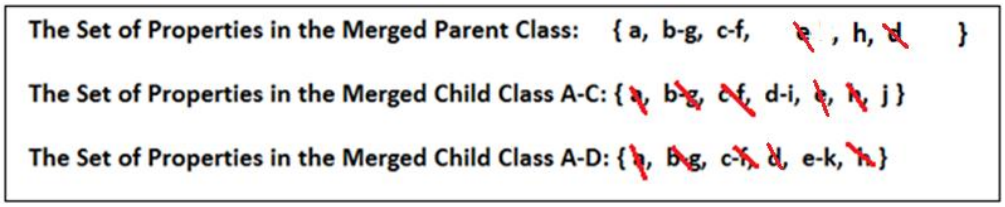


Figure 31 c) Remove redundant properties in super class (In this case, there is no redundant properties).

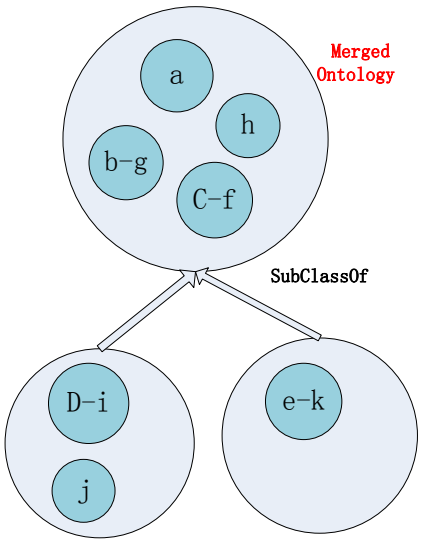


Figure 32: Result generated by ontology approach.

7.1.1.Hierarchical Structure Analysis Procedure (EERD TO EERD)

Figure 33a shows two ontologies with hierarchical structure. Correspondence is given. The list of candidate merging class is represented as *{A, B, C, D, E, F}*. *B, C* are the subclass of *A*, and *E, F* are the subclass of *D*. According to new similarity computation given by *definition 1*, the similarity between *A* and *D* is the highest, so we merge *A* with *D*, which is shown in figure 33b. Then the list of candidate merging class is changed to *{B, C, E, F}*. Since the similarity between *B* and *E* has the highest value in the new candidate merging class list, so we merge *B* with *E*. Then the list has only two classes, so we merge *C* with *F*. Merging *B* with *E* is shown in figure

33c, and merging *C* with *F* is shown in figure 33d. Three new classes are generated and named with AD, BE, and CF. The second step is rebuild the hierarchical structure. Since *B* and *C* are the subclass of *A*, and *E* and *F* are the subclass of *D*, we set BE and CF as subclass of AD. The third step is to remove redundant properties, and the result after removing redundant properties is shown in figure 33e.

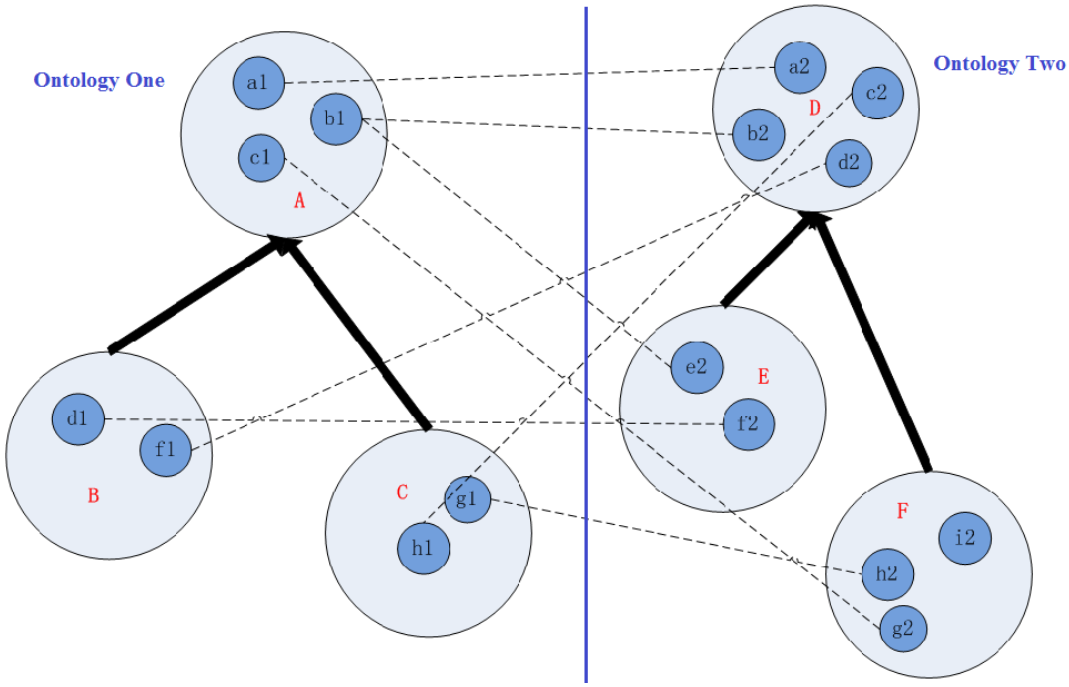


Figure 33a: Two ontologies with hierarchical structure.

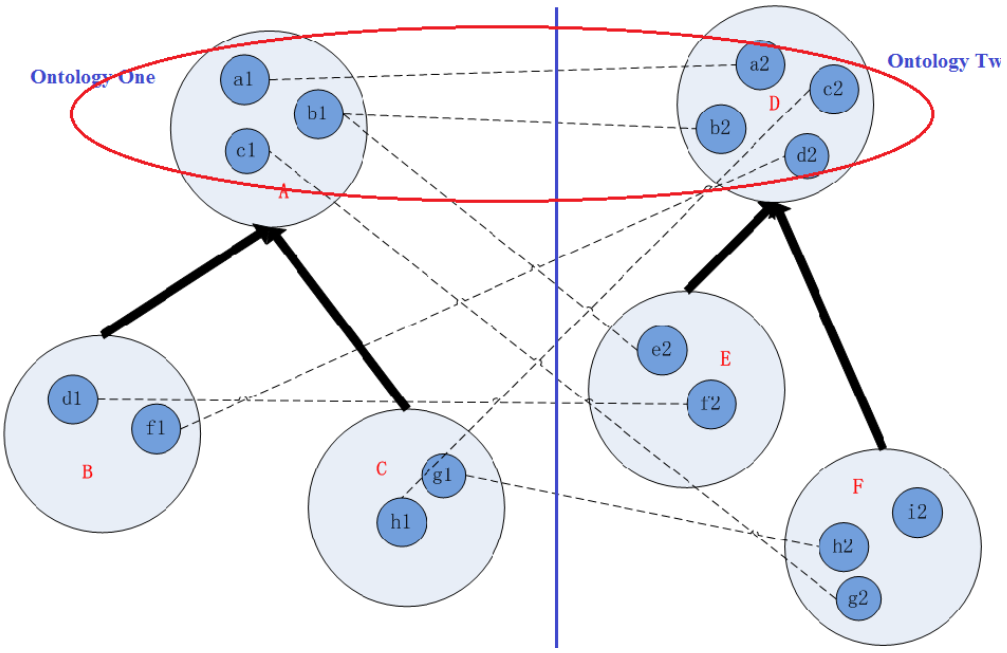


Figure 33b: Merge A with D.

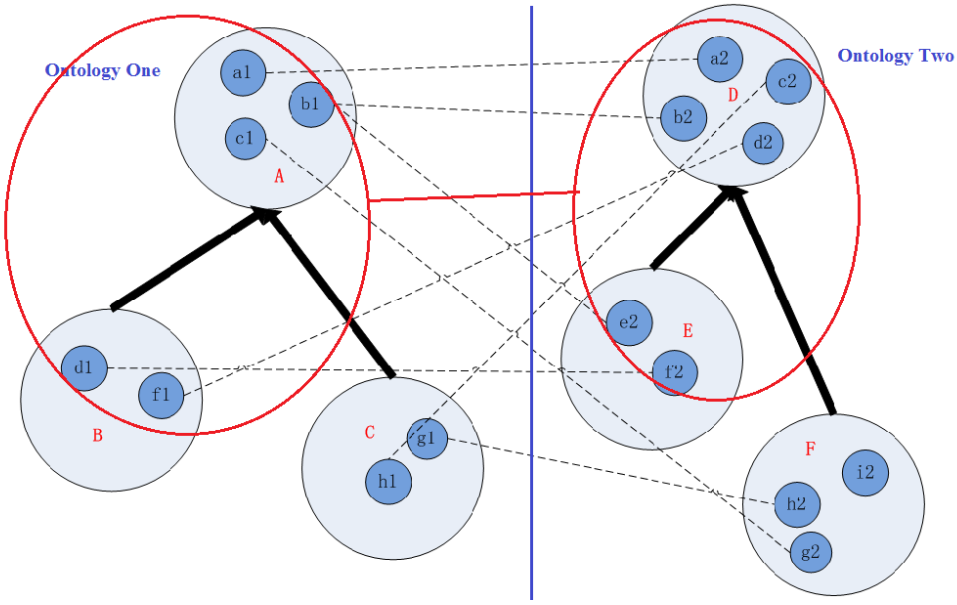


Figure 33c: Merge B with E.

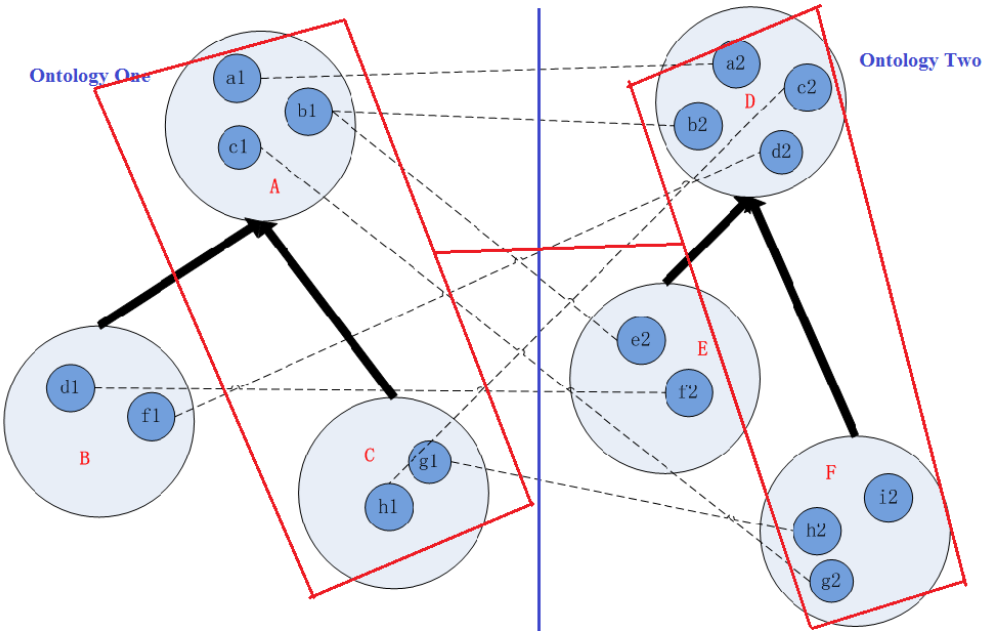


Figure 33d: Merge C with F.

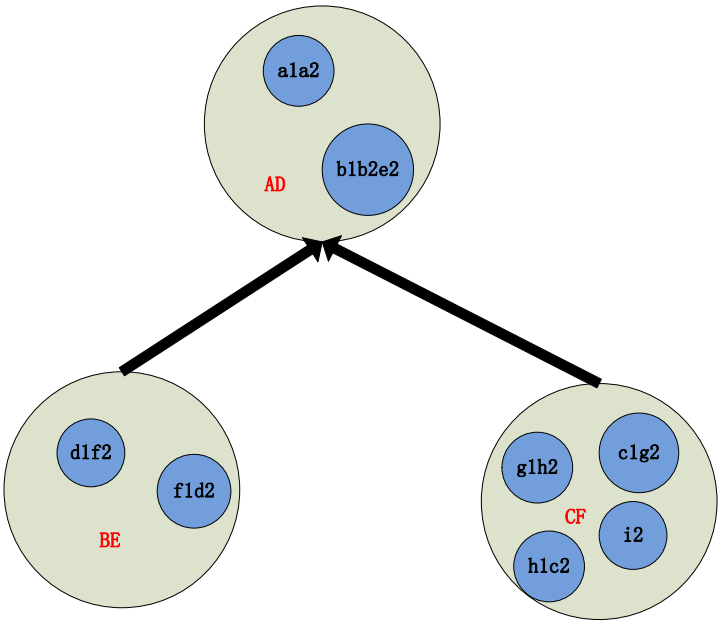


Figure 33e: Merging result (EERD to EERD).

7.2. Utilizing Object Properties to Enhance the Merging System

When converting the relational schema to ontology or concept graph, foreign key in the entities need to be removed away. For example, in figure 34, *family* and *income_src* are many-to-many relationship and linked by the weak entity *income*. When convert *income* entity to concept in the concept graph, the foreign key *fid* and *sid* are taken out from the concept *income*.

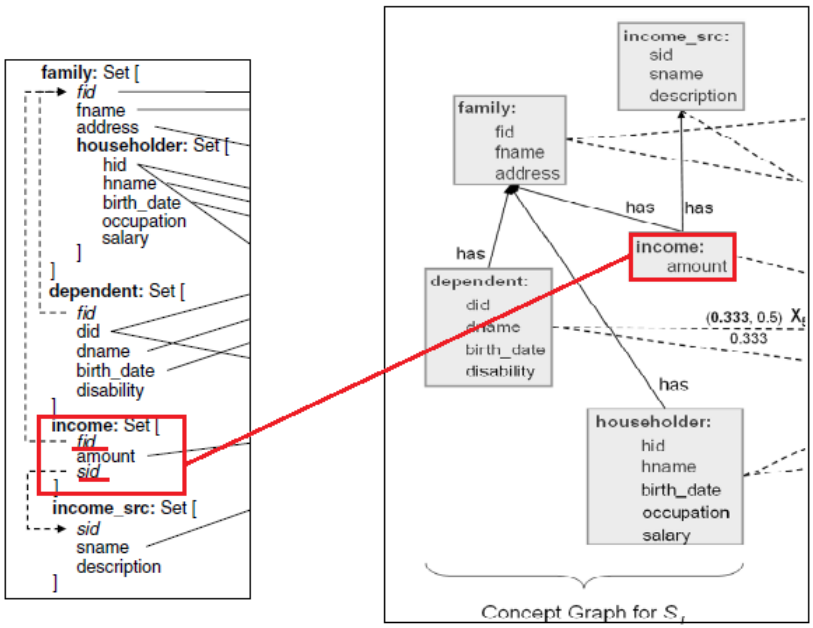


Figure 34: Many-to-many relationship in concept graph.

In the conversion to ontology proposed by Alawan, he explained the foreign key should not convert to datatype property, which is shown as follow.

$$Rule_{DP_4} = \begin{cases} -Datatype\ condition:, \\ Attr(x,R) \wedge NonFK(x,R) \\ -Datatype\ action: create \\ DP\big(x, C, type(x)\big) \end{cases}$$

(Alawan, 2011, p. 116)

Figure 35 shows how to convert the many-to-many relationship with additional attribute in ER to ontology model. In Alawan's ontology conversion approach, he also introduced the weak entities in many-to-many relationship without additional attribute should be eliminated when doing the ontology conversion (Alawan, 2011, p. 191). The relationships in ontology are expressed by object properties instead of foreign keys.

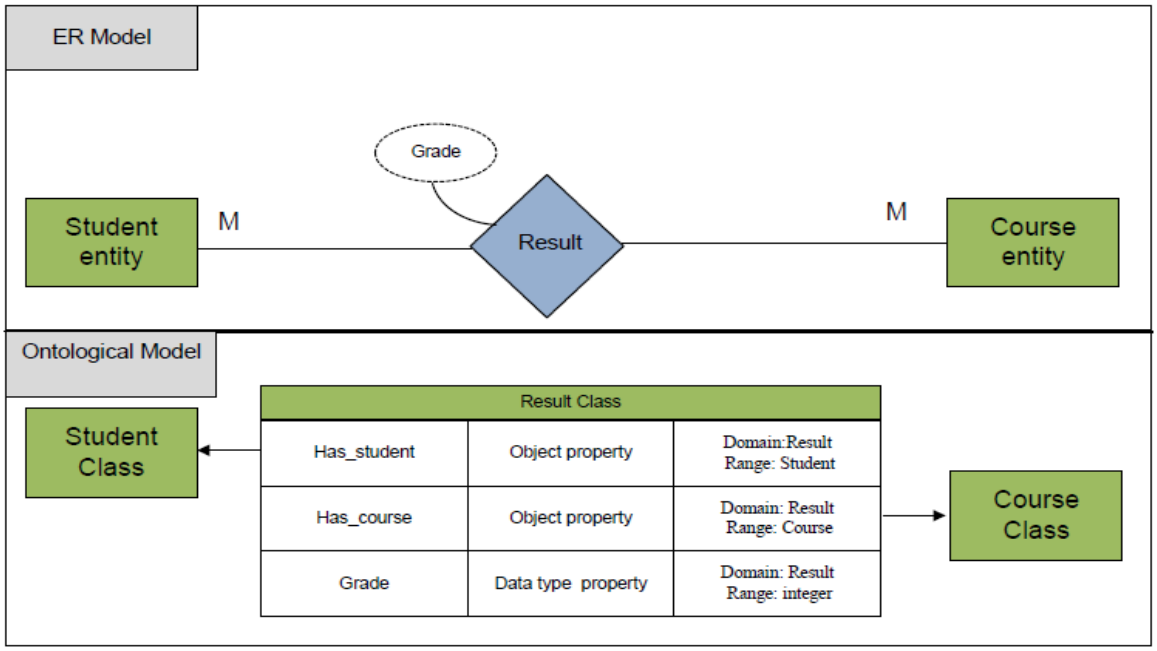


Figure 35: Many-to-many relationship with additional attribute to ontology.

In the previous schema integration system, matching attributes are the only factors to compute the *Hausdorff distance*, which may cause the redundant weak entities in the integrated schema. For example, as shown in figure 36, two weak entities should be merged, however, since there is no correspondence exists between them, both *Registered* and *Enrollment* entities exist in the merging result.

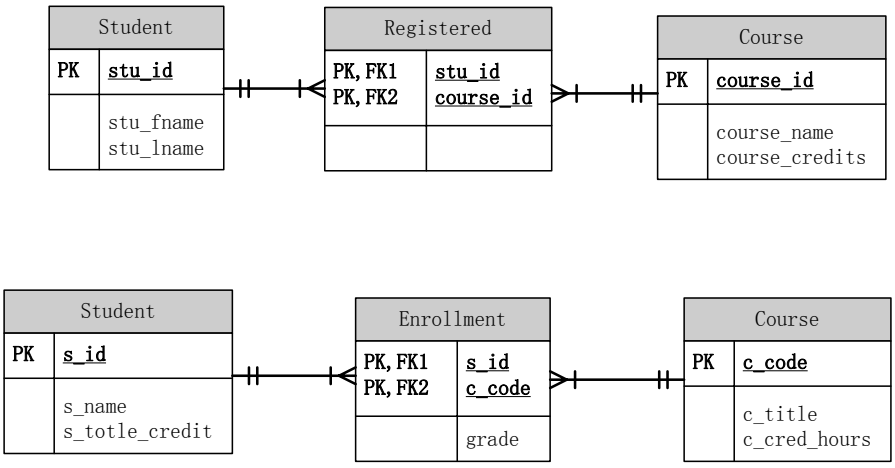


Figure 36: a) Two schemas before Merging.

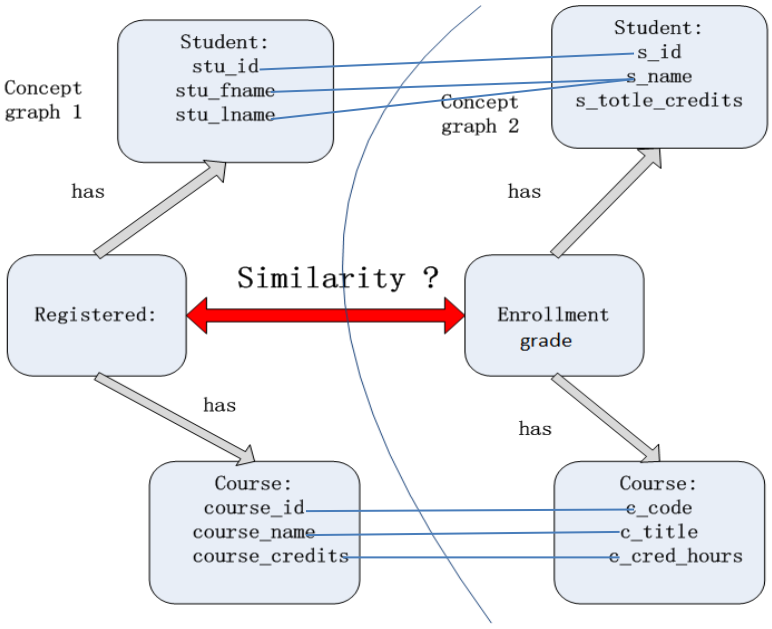


Figure 36: b) Concept graph.

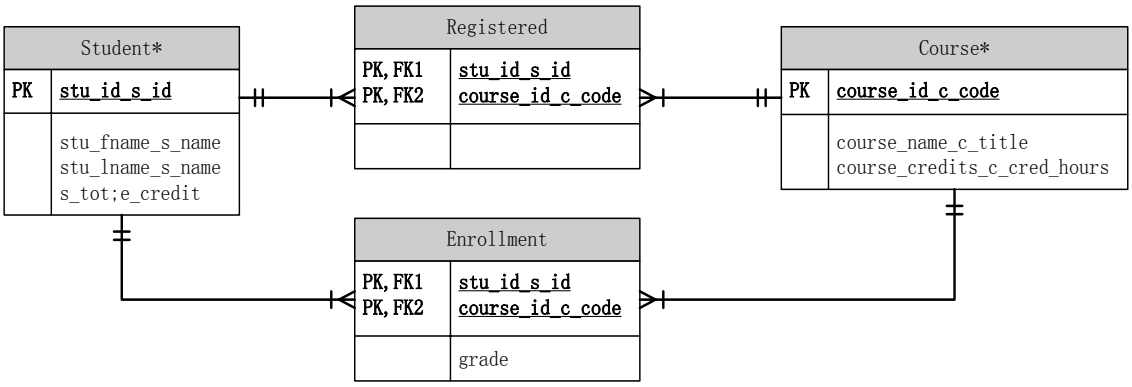


Figure 36: c) Schemas after merging.

The same problem in figure 36 could happen in ontology based schema merging approach. When converting relational schema to ontology, weak entity without additional attribute in ER Diagram will be eliminated. However, weak entity with additional attribute in ER Diagram will be converted to *Class* in ontology. When merging the ontologies, more redundant object properties will be created by the system. For example, in figure 37, the object properties *StudentsHaveManyCourses* and *CoursesHaveManyStudents* express that relationship between Student* and Course* is many-to-many. The object properties between Course* and Enrollment, and between Enrollment and Student* also express that the relationship between Student* and Course* is many-to-many. Hence, the object properties *StudentsHaveManyCourses* and *CoursesHaveManyStudents* should not exist in the merged ontology.

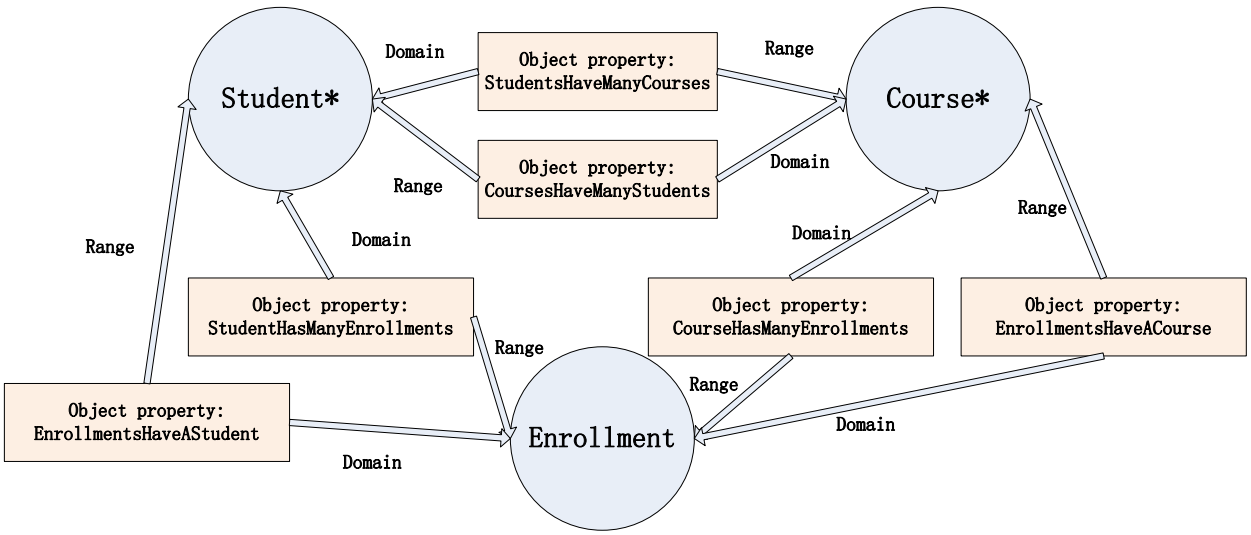


Figure 37: Merging ontologies without considering object property.

To avoid generating the redundant relationships during the ontology merging, object properties can be utilized. In the ontology based schema integration system, object property is another factor to compute the similarity between two classes.

When converting relational schema to ontology, weak entities remain and will convert to classes by the system. The object property could match each other. If two object properties match, the system will add one correspondence to both domain classes and one correspondence to both range classes. For example, in figure 38, if the object property *StudentHasManyRegistereds* matches object property *StudentHasManyEnrollments*, since the domain of both object properties are *Student1* and *Student2,* if the similarity calculated based on the matching of datatype properties is *5/6*, the new similarity between two concept after concerning about the matching of object properties is *(5+1) / (6+1).* Since the similarity between *Registered* and *Enrollment* is 0 without considering the matching of object properties, the similarity between *Registered* and

Enrollment after adding one correspondence is (0 + 1) / (0+ 1). The merging result after considering object properties is shown in the figure 39.

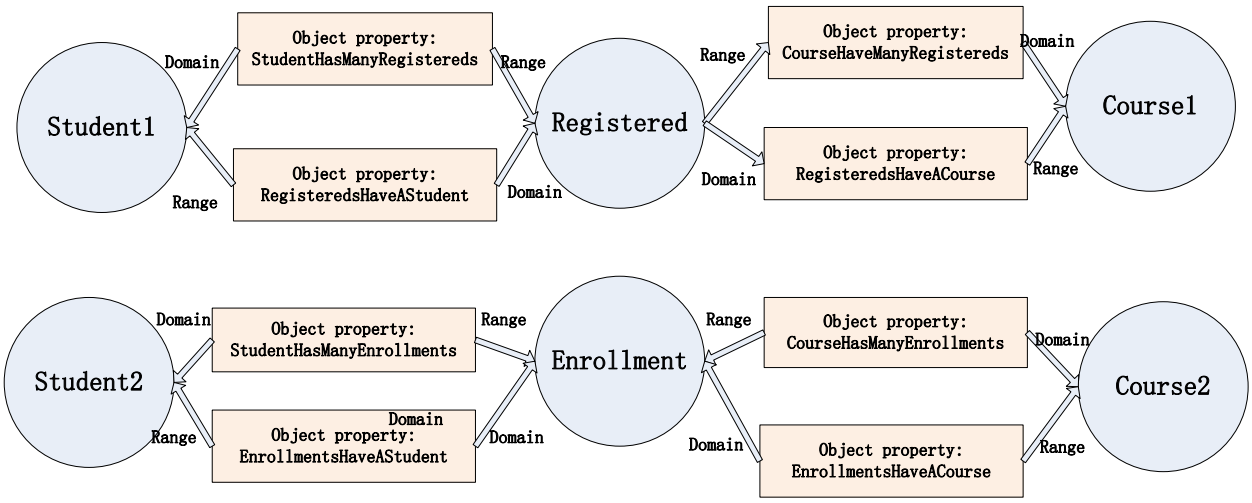


Figure 38: Ontologies before merging.

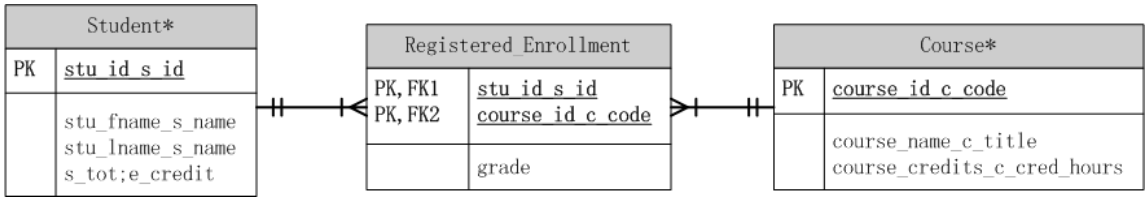


Figure 39: ERD after merging by considering object property.

7.3. EERD TO EERD in Enhanced Ontology Integration Approach

A case of EERD to EERD integration in enhanced ontology integration approach is given as follow. The two candidate EER Diagrams are UniversityThreeEERD EERD in figure 19 and UniversityFourEERD EERD in figure 40. The top 2 assignment generated by system is shown in figure 41. The final integrated EERD is shown in figure 42. The merging result shows the expecting result. First, the merged schema preserved the hierarchical structure by using the hierarchical structure analysis procedure which is introduced in section 7.1.1. Second, the weak entities are merged based on the matching of object properties which is introduced in section 7.2.

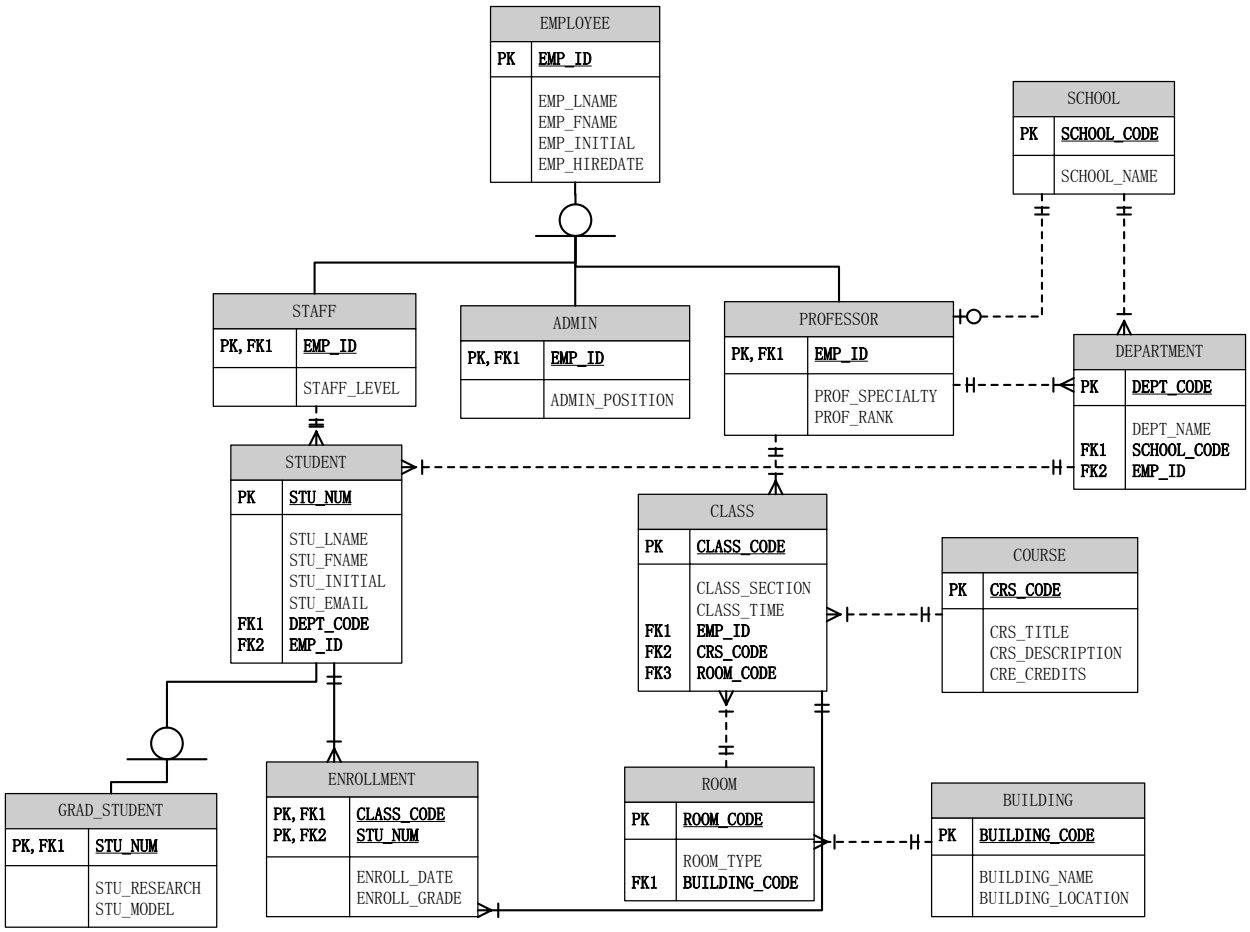


Figure 40: UniversityFourEERD EERD.

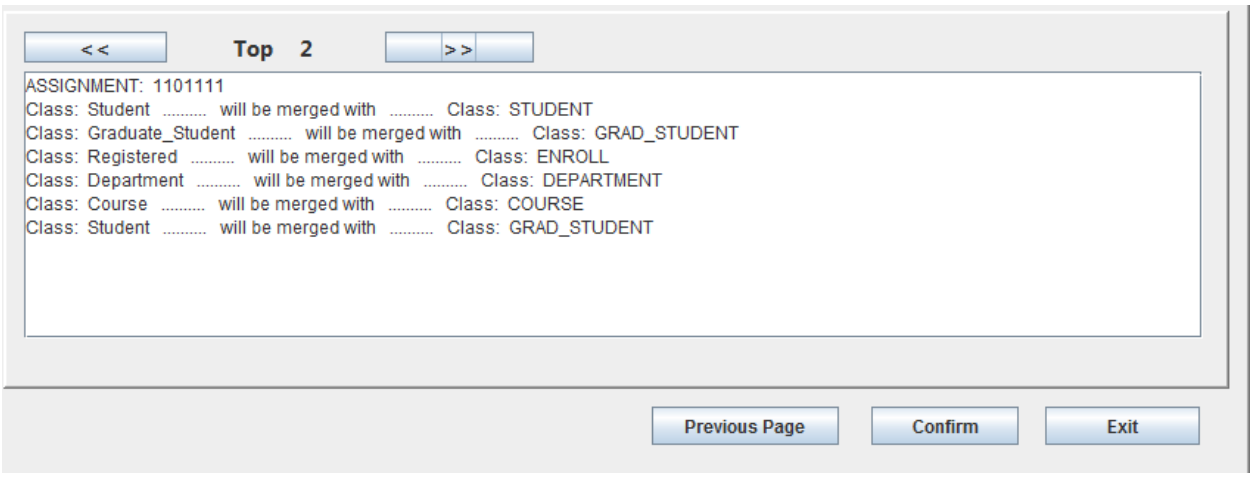


Figure 41: Top 2 assignment.

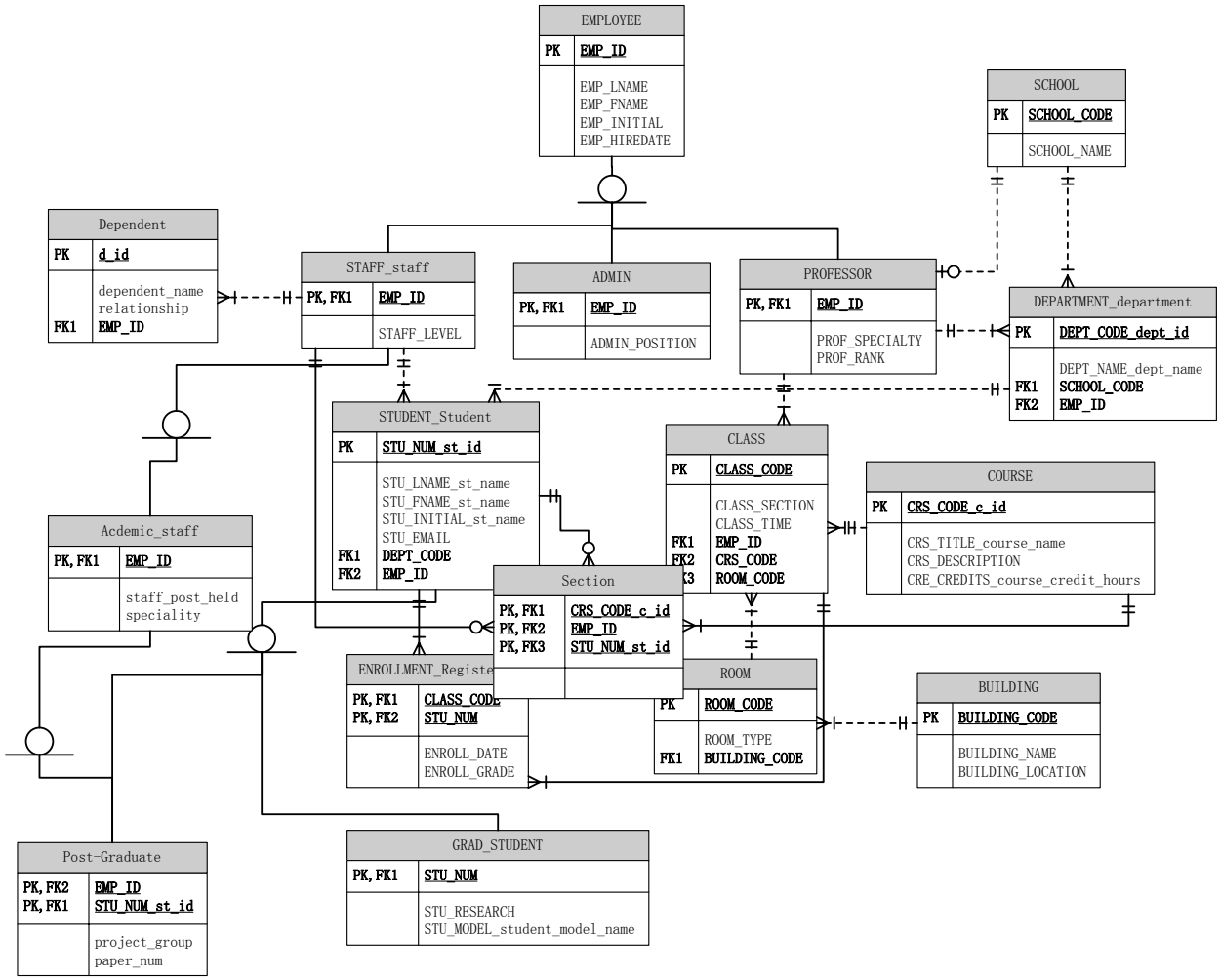


Figure 42: EERD to EERD by ontology approach.

8. Conclusion and Future Work

Since the approaches of previous schema integration systems do not mention too much about the case that hierarchy structure exists in the schema, their schema integration system are implemented and tested first. From testing, in some cases, the previous top-k schema integration system cannot preserve hierarchy structures that have been found. Based on this collected information, we observed how to keep the hierarchy structure while merging the schemas. Ontology is utilized since it is a very powerful concept in representing relationships between each class. From each case study, the results show the system in this research can preserve hierarchical structure in simple cases. More cases need to be tested in the future. Problems such as how to build hierarchical structure and how to reduce conflicts which introduced in section 3, will be researched in the future.

References

Alawan, N. A. (2011). *Ontological approach for database integration* (Doctoral thesis). Retrieved from Google Scholar.

Chiticariu, L., Kolaitis, P., & Popa, L. (2008). *Interactive generation of integrated schemas*. Paper presented at the ACM SIGMOD International Conference on Management of Data, New York, NY. Retrieved from http://dl.acm.org

Fahad. M. (2008). *ER2TOOW: Generating OWL ontology from ER diagram*. *IFIP International Federation for Information Processing, 288*, 28-37. Retrieved from http://www.springerlink.com

Guohui, D., Guoren, W., & Bin, W. (2010). Top-K Generation of Mediated Schemas over Multiple Data Sources. *Springer-Verlag Berlin Heidelberg*, 143-155.

Gottlob, G., Pichler, R., & Savenkov, V. (2009) Normalization and optimization of schema mappings. *VLDB Conference*, 2009. 2(1).

Gruber, T. R. (March, 1993). Toward Principles for the Design of Ontologies Used for Knowledge Sharing, *International Journal Human-Computer Studies*, 907-928.

Mihoubi, H., Simonet, A., & Simonet, M. (2000). An Ontology Driven Approach to Ontology Translation, *Proceedings of DEXA,* 573-582.

Nyulas, C., and Tu, S. "*DataMaster – a Plug-in for Importing Schemas and Data from Relational Databases into Protégé*", In Proceedings of 10th International Protégé Conference.

Octavian, U., Lise, G. (2007). Leveraging Data and Structure in Ontology Integration. *SIGMOD Conference*.

Pottinger, R & Bernstein, P. A. (2003). Merging Models Based on Given Correspondences. *VLDB*, 826–873.

Radwan, A., Popa, L., Stanoi, L. R., & Younis, A. (2009). Top-k generation of integrated schemas based on directed and weighted correspondences. *SIGMOD Conference*.

Smith, M. K., Welty, C., & McGuinness, D. L. (2004). OWL Web Ontology Language Guide. Retrieved from http://www.w3.org/TR/owl-guide

Sharma, A. (2006). Ontology Matching Using Weighted Graphs. *IEEE*, 2006.

Wache, H., Vögele, T., Visser, H. U., Stuckenschmidt, G. S., Neumann, H., & Hübner, S. (2001). Ontology-based integration of information - a survey of existing approaches, *Workshop: Ontologies and Information Sharing*, 108-117.

Appendix A

```
CREATE TABLE Student (
   stu_id    char(9)   PRIMARY KEY,
   stu_fname char(20)  NOT NULL,
   stu_lname char(20)  NOT NULL
);

CREATE TABLE Department (
   dep_code char(4)  PRIMARY KEY,
   dep_name char(40) NOT NULL UNIQUE
);

CREATE TABLE Instructor (
   ins_id    char(9)  PRIMARY KEY,
   ins_fname char(20) NOT NULL,
   ins_lname char(20) NOT NULL,
   dep_code  char(4)  NOT NULL REFERENCES Department(dep_code)
);

CREATE TABLE Location (
   loc_code    char(5)  PRIMARY KEY,
   loc_name    char(40) NOT NULL,
   loc_country char(2)  NOT NULL
);

CREATE TABLE Course (
   crs_code        char(10)     PRIMARY KEY,
   crs_title       varchar(100) NOT NULL,
   crs_credits     tinyint      NOT NULL,
   dep_code        char(4)      NOT NULL REFERENCES Department(dep_code),
   crs_description varchar(255) NOT NULL
);

CREATE TABLE Section (
   sec_id   int      PRIMARY KEY,
   sec_term char(8)  NOT NULL,
   sec_bldg char(6),
   sec_room char(4),
   sec_time char(10),
   crs_code char(10) NOT NULL REFERENCES Course(crs_code),
   loc_code char(5)  NOT NULL REFERENCES Location(loc_code),
   ins_id   char(9)  REFERENCES Instructor(ins_id)
);

CREATE TABLE Enrollment (
   stu_id     char(9)  REFERENCES Student(stu_id),
```

```
    sec_id     int      REFERENCES Section(sec_id),
    grade_code char(2),
    PRIMARY KEY (stu_id, sec_id)
);

CREATE TABLE Prerequisite (
    crs_code     char(10) REFERENCES Course(crs_code),
    crs_requires char(10) ,
    PRIMARY KEY (crs_code, crs_requires)
);

CREATE TABLE Qualified (
    ins_id   char(9)  REFERENCES Instructor(ins_id),
    crs_code char(10) REFERENCES Course(crs_code),
    PRIMARY KEY (ins_id, crs_code)
);
```

Appendix B

```
CREATE TABLE Classroom(
   room_building   varchar(15),
   room_number     varchar(7),
   room_capacity   numeric(4,0),
   PRIMARY KEY (room_building, room_number)
);

CREATE TABLE Department(
   dept_name       varchar(20),
   dept_building   varchar(15),
   dept_budget                numeric(12,2) check (budget > 0),
   PRIMARY KEY (dept_name)
);

CREATE TABLE Course(
   course_id                  varchar(8),
   course_title               varchar(50),
   dept_name                  varchar(20),
   course_credits             numeric(2,0) check (credits > 0),
   PRIMARY KEY (course_id),
   FOREIGN KEY (dept_name)  references department
);

CREATE TABLE Instructor(
   instructor_ID          varchar(5),
   instructor_name      varchar(20) not null,
   dept_name       varchar(20),
   instructor_salary            numeric(8,2) check (salary > 29000),
   PRIMARY KEY (instructor_ID),
   FOREIGN KEY (dept_name) references department
);

CREATE TABLE Section (
   course_id           varchar(8),
   section_id                  varchar(8),
   section_semester            varchar(6)
                               check (semester in ('Fall', 'Winter', 'Spring', 'Summer')),
   section_year                numeric(4,0) check (year > 1701 and year < 2100),
   room_building    varchar(15),
   room_number      varchar(7),
   time_slot_id       varchar(4),
   PRIMARY KEY (course_id, sec_id, section_semester, section_year),
   FOREIGN KEY (course_id) references course
   FOREIGN KEY (time_slot_id) references Time_slot
   FOREIGN KEY (room_building, room_number) references classroom
```

```
);

CREATE TABLE Teaches(
   instructor_ID              varchar(5),
   course_id                  varchar(8),
   sec_id                     varchar(8),
   section_semester                  varchar(6),
   section_year                      numeric(4,0),
   PRIMARY KEY (teaches_ID, course_id, sec_id, section_semester, section_year),
   FOREIGN KEY (course_id,sec_id, section_semester, section_year) references section
   FOREIGN KEY (instructor_ID) references instructor
);

CREATE TABLE Student (
   student_ID                 varchar(5),
   student_name               varchar(20) not null,
   dept_name                  varchar(20),
   student_tot_cred                  numeric(3,0) check (tot_cred >= 0),
   PRIMARY KEY (student_ID),
   FOREIGN KEY (dept_name) references department
);

CREATE TABLE Takes (
   student_ID                 varchar(5),
   course_id                  varchar(8),
   sec_id                     varchar(8),
   section_semester           varchar(6),
   section_year               numeric(4,0),
   takes_grade                varchar(2),
   PRIMARY KEY (student _ID, course_id, sec_id, section_semester, section_year),
   FOREIGN KEY (course_id,sec_id, section_semester, section_year) references section
   FOREIGN KEY (student _ID) references student
);

CREATE TABLE Advisor(
   student_ID           varchar(5),
   instructor_ID        varchar(5),
   PRIMARY KEY (student_ID),
   FOREIGN KEY (instructor_ID) references instructor (instructor_ID)
   FOREIGN KEY (student_ID) references student (student_ID)
);

CREATE TABLE Time_slot (
   time_slot_id       varchar(4),
   time_slot_day              varchar(1),
   time_slot_start_hr         numeric(2) check (start_hr >= 0 and start_hr < 24),
```

```
   time_slot_start_min          numeric(2) check (start_min >= 0 and start_min < 60),
   time_slot_end_hr                    numeric(2) check (end_hr >= 0 and end_hr < 24),
   time_slot_end_min                   numeric(2) check (end_min >= 0 and end_min < 60),
   PRIMARY KEY (time_slot_id, time_slot_day, time_slot_ start_hr, time_slot_ start_min)
);

CREATE TABLE Prereq (
   course_id                  varchar(8),
   prereq_id                  varchar(8),
   PRIMARY KEY (course_id, prereq_id),
   FOREIGN KEY (course_id) references course
);
```

Appendix C

UniversityOneERD nested schema:

```
Student:Set[
        stu_id
        stu_fname
        stu_lname
        Enrollment:Set[
                grade_code
                sec_id $Section.sec_id
                ]
]
Location:Set[
        loc_code
        loc_name
        loc_country
]
Department:Set[
        dep_code
        dep_name
        Instructor:Set[
   ins_id
        ins_fname
        ins_lname
                Qualified:Set[
                crs_code $Course.crs_code
                ]
        ]
        Course:Set[
                crs_code
                crs_title
                crs_credits
                crs_description
                Section:Set[
                        sec_id
                        sec_term
                        sec_bldg
                        sec_room
                        sec_time
                        loc_code $Location.loc_code
                        ins_id $Instructor.ins_id
                ]
        ]
]
```

```
Prerequisite:Set[
                    crs_code $Course.crs_code
                    crs_requires
]
```

UniversityTwoERD nested schema:

```
Classroom:Set[
   room_id
   room_building
         room_number
   room_capacity
]
Department:Set[
         dept_name
         dept_building
         dept_budget
         Student:Set[
                  student_ID
                  student_name
                  student_tot_cred
         ]
         Course:Set[
                  course_id
                  course_title
                  course_credits
                  Section:Set[
                           sec_id
                           section_semester
                           section_year
                           time_slot_id $Time_slot.time_slot_id
        room_id $Classroom.room_id
                           Teaches:Set[
                                    instructor_ID $Instructor.instructor_ID
                           ]
                           Takes:Set[
                                    student_ID $Student.student_ID
                           ]
                  ]
         ]
         Instructor:Set[
                  instructor_ID
                  instructor_name
                  instructor_salary
                  Advisor:Set[
                           student_ID $Student.student_ID
                  ]
         ]
]
Prereq:Set[
```

```
        prereq_id
        course_id $Course.course_id
]
Time_slot:Set[
        time_slot_id
        time_slot_day
        time_slot_start_hr
        time_slot_start_min
        time_slot_end_hr
        time_slot_end_min
]
```

Appendix D

```
Location_1:Set[
                    loc_code
                    loc_name
                    loc_country
                    Section_1_Section_2:Set[
                              sec_id_section_id
                              sec_term_section_semester
                              sec_time_section_year
                              sec_bldg
                              sec_room
                              room_id $Classroom_2.room_id
                              ins_id_instructor_ID
$Instructor_1_Instructor_2.ins_id_instructor_ID
                              time_slot_id $Time_slot_2.time_slot_id
                              crs_code_course_id $Course_1_Course_2.crs_code_course_id
                    Teaches_2:Set[
                              ins_id_instructor_ID
$Instructor_1_Instructor_2.ins_id_instructor_ID
                    ]
          ]
]
Time_slot_2:Set[
          time_slot_id
          time_slot_day
          time_slot_start_hr
          time_slot_start_min
          time_slot_end_hr
          time_slot_end_min
]
Classroom_2:Set[
          room_id
          room_building
          room_number
          room_capacity
]
Department_1_Department_2:Set[
          dep_name_dept_name
          dep_code
          dept_building
          dept_budget
          Course_1_Course_2:Set[
                    crs_code_course_id
                    crs_title_course_title
                    crs_credits_course_credits
                    crs_description
```

```
                Qualified_1:Set[
                        ins_id_instructor_ID
$Instructor_1_Instructor_2.ins_id_instructor_ID
                ]
                Prerequisite_1_Prereq_2:Set[
                        crs_requires_prereq_id
                ]
        ]
        Instructor_1_Instructor_2:Set[
                ins_id_instructor_ID
                ins_fname_instructor_name_ins_lname
                ins_lname_instructor_name
                instructor_salary
                Advisor_2:Set[
                stu_id_student_ID $Student_1_Student_2.stu_id_student_ID
                ]
        ]
        Student_1_Student_2:Set[
                stu_id_student_ID
                stu_fname_student_name_stu_lname
                stu_lname_student_name
                student_tot_cred
                Enrollment_1:Set[
                grade_code
                sec_id_section_id $Section_1_Section_2.sec_id_section_id
        ]
        Takes_2:Set[
                sec_id_section_id $Section_1_Section_2.sec_id_section_id
            ]
        ]
]
```

Appendix E

UniversityOneERD.owl

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
   xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
   xmlns="http://www.owl-ontologies.com/Ontology1334642318.owl#"
   xmlns:swrl="http://www.w3.org/2003/11/swrl#"
   xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1334642318.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Student_1">
   <rdfs:subClassOf>
    <owl:Restriction>
     <owl:someValuesFrom>
      <owl:Class rdf:ID="Enrollment_1"/>
     </owl:someValuesFrom>
     <owl:onProperty>
      <owl:ObjectProperty rdf:ID="StudentsHaveManyEnrollments"/>
     </owl:onProperty>
    </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:ID="Qualified_1">
   <rdfs:subClassOf>
    <owl:Restriction>
     <owl:onProperty>
      <owl:ObjectProperty rdf:ID="QualifiedsHaveAInstructor"/>
     </owl:onProperty>
     <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
     >1</owl:cardinality>
    </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
    <owl:Restriction>
     <owl:onProperty>
      <owl:ObjectProperty rdf:ID="QualifiedsHaveACourse"/>
     </owl:onProperty>
```

```
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="Course_1">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="CoursesHaveADepartment"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="Section_1"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="CourseHasManySections"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="CourseHasManyQualifieds"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#Qualified_1"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="Prerequisite_1"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="CourseHasManyPrerequisite"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
```

```
<owl:Class rdf:about="#Enrollment_1">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="EnrollmentHasAStudent"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="EnrollmenstHaveASection"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="Location_1">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Section_1"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="LocationHasManySections"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="Department_1">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="Instructor_1"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="DepartmentHasManyInstructor"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
```

```
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#Course_1"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="DepartmentHasManyCourse"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:about="#Section_1">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="SectionsHaveALocation"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="SectionsHaveACourse"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="SectionHasManyInstructors"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Instructor_1"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="SectionHasManyEnrollments"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#Enrollment_1"/>
    </owl:Restriction>
  </rdfs:subClassOf>
```

```
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:about="#Prerequisite_1">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="PrerequisitesHaveACourse"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:about="#Instructor_1">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="InstructorsHaveASection"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="InstructorsHaveADepartment"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="InstructorHasManyQualifieds"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#Qualified_1"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:ObjectProperty rdf:about="#LocationHasManySections">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#SectionsHaveALocation"/>
```

```
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#SectionHasManyEnrollments">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#EnrollmenstHaveASection"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#DepartmentHasManyCourse">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#CoursesHaveADepartment"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#PrerequisitesHaveACourse">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#CourseHasManyPrerequisite"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#SectionsHaveALocation">
  <owl:inverseOf rdf:resource="#LocationHasManySections"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#StudentsHaveManyEnrollments">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#EnrollmentHasAStudent"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#DepartmentHasManyInstructor">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#InstructorsHaveADepartment"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#EnrollmentHasAStudent">
  <owl:inverseOf rdf:resource="#StudentsHaveManyEnrollments"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#SectionHasManyInstructors">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#InstructorsHaveASection"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#QualifiedsHaveACourse">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#CourseHasManyQualifieds"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#InstructorsHaveASection">
  <owl:inverseOf rdf:resource="#SectionHasManyInstructors"/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:about="#CourseHasManyPrerequisite">
  <owl:inverseOf rdf:resource="#PrerequisitesHaveACourse"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#CoursesHaveADepartment">
  <owl:inverseOf rdf:resource="#DepartmentHasManyCourse"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#SectionsHaveACourse">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#CourseHasManySections"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#InstructorsHaveADepartment">
  <owl:inverseOf rdf:resource="#DepartmentHasManyInstructor"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#CourseHasManyQualifieds">
  <owl:inverseOf rdf:resource="#QualifiedsHaveACourse"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#CourseHasManySections">
  <owl:inverseOf rdf:resource="#SectionsHaveACourse"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#QualifiedsHaveAInstructor">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#InstructorHasManyQualifieds"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#EnrollmenstHaveASection">
  <owl:inverseOf rdf:resource="#SectionHasManyEnrollments"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#InstructorHasManyQualifieds">
  <owl:inverseOf rdf:resource="#QualifiedsHaveAInstructor"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="sec_bldg">
  <rdfs:domain rdf:resource="#Section_1"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="loc_name">
  <rdfs:domain rdf:resource="#Location_1"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="stu_fname">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Student_1"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="loc_country">
  <rdfs:domain rdf:resource="#Location_1"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
```

```
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="dep_name">
  <rdfs:domain rdf:resource="#Department_1"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="stu_lname">
  <rdfs:domain rdf:resource="#Student_1"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="grade_code">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#Enrollment_1"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="crs_title">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Course_1"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="sec_term">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Section_1"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="sec_time">
  <rdfs:domain rdf:resource="#Section_1"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="dep_code">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#Department_1"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="crs_credits">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  <rdfs:domain rdf:resource="#Course_1"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ins_fname">
  <rdfs:domain rdf:resource="#Instructor_1"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="datatypeProperty_32"/>
<owl:DatatypeProperty rdf:ID="crs_description">
  <rdfs:domain rdf:resource="#Course_1"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="sec_room">
  <rdfs:domain rdf:resource="#Section_1"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

```
  <owl:DatatypeProperty rdf:ID="ins_lname">
    <rdfs:domain rdf:resource="#Instructor_1"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:FunctionalProperty rdf:ID="crs_code">
    <rdfs:domain rdf:resource="#Course_1"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="ins_id">
    <rdfs:domain rdf:resource="#Instructor_1"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="sec_id">
    <rdfs:domain rdf:resource="#Section_1"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="stu_id">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#Student_1"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="loc_code">
    <rdfs:domain rdf:resource="#Location_1"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:FunctionalProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.4.8, Build 629)  http://protege.stanford.edu -->
```