Towards Semantic Web: An Effective Approach for Association Identification in Real Scenarios

By

Asif Sohail Abid

Submitted to the Department of Computer Engineering

in partial fulfillment of the requirements for the Degree of

Masters of Science

In

Computer Software Engineering

Thesis Supervisor

Brig Dr. Muhammad Younus Javed

College of Electrical and Mechanical Engineering

National University of Sciences & Technology

2010

ACKNOWLEDGEMENTS

All praise to the Almighty Allah, the most Merciful and the most gracious one. Without whose help and blessings, I would not have been able to complete this research. Many thanks to my project supervisor, Dr. Mohammad Younus Javed, whose constant motivation, unflagging efforts and uninvolved words of wisdom ever proved a lighthouse for me; it was earnestly felt whenever we swayed. Despite his never ending assignments of university management, student counseling, project supervision and teaching, he did never mind whenever we went for an advice, within or without the time slot allocated for us.

Acknowledgement is also due to my teachers for dedicatedly instilling and imparting enlightenment to me during the course of studies and afterwards for our project. I am also very thankful to my parents for their tacit and avowed support, patience and understanding.

I would like to thank my friends who gave me confidence to face the difficulties of life. They all gave me good company and everlasting memories. Especial thanks to my friend Mr. Sheraz Khurram.

In the Name of Almighty Allah
The Most Beneficent and Most Merciful

Dedicated to my;
caring mother, considerate teachers
&
all those who contributed towards my future

Abstract

The semantic web is an extension of current web. The concept introduced by this web is dramatically changing the world of web. This is one of the reasons that it is often named as future web. The semantic web's objective is to make web meaningful and understandable. In the last few years, this area has gained interest from many researchers, bringing the results in the form of different standards for the semantic web. The aim of this web is to make web meaningful, understandable and machine processable. Making use of information on current web for the productiveness of future web is becoming vital. Heterogeneity of relational data coupled with present web complicates utilization of information for future web. To use the data associated with current web it was required to transform it into ontology. The already presented and provided algorithms merely give the result in user required form. The thesis focuses on the problem where some of the meta data is available with relational schema and identification of associations. Protégé is an open source tool that can be used for ontology development. The DataMaster was developed in BioSTORM [10] project which supports both OWL and frame-based ontologies. It works as a plug-in for protégé [11]. Many of these approaches merely give the results in the user required form. Moreover, the weak entities of the database are also mapped into classes (for example in datamaster, DataGenie etc). It will be a tough job to find the relationship in the extracted classes (i.e. ontologies), with a little domain knowledge. Moreover it will become very tough when the relational schema is of large size. Need of an automated or semi automated approach for discovering the relationships in the relational schemas was vital. This presents a scheme for the identification of association in real scenarios, where there can be very little metadata availability.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

OWL	Web Ontology Language
RDF	Resource Description Frame Work
3NF	Third Normal Form
ERD	Entity Relationship Diagram
EERD	Extended Entity Relationship Diagram
RID	Relational Intermediate Directed
R2O	Relational to Ontology
DL	Description Logic
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
WWW	World Wide Web

Chapter 1

Introduction

In this chapter, the overview of the thesis is presented. This chapter holds its importance by providing the basis for research. It includes motivation, problem definition, objectives and goals of research.

1.1. Motivation

The semantic web can be considered as a mesh of information that makes information processable by the machines on global scale. The aim of semantic web is not only to make information processable but, understandable by the machines.
The concept of semantic web was introduced by Tim Berners-Lee, the inventor of WWW, URIs, HTTP and HTML [7]. The concept introduced by this web is dramatically changing the world of web. This is one of the reasons that it is often named as future web. The semantic web's objective is to make web meaningful and understandable. In the last few years, this area has gained interest from many researchers, bringing the results in the form of different standards for the semantic web. Some of these standards include: global naming scheme (i.e. URIs), Resource Description Framework (RDF), the data interchanges formats, and notations such as RDF Schema (RDFS) for describing the properties of the data and the Web Ontology Language (OWL) for describing the relationships of the data.

The ontology is defining the explicit concepts of the domain of discourse where each of its characteristics is called as attribute. In the process of creation of ontologies for the database, three techniques are used: analysis of database schema, tuples or analysis of user queries [2]. From different schemes of transformation of relational schemas to ontology, the following points are observed: the database tables are mapped into the classes (i.e. ontologies, attribute of the tables are mapped as the attributes of the ontologies) and identifications of foreign keys in a database schema. However, the types of the relationship can be identified by the entries in the database.

The ontology is one of the pillars of the semantic web. Different tools for ontology creations and management are developed to implement the concept of ontology like protégé, OntoGen etc. Protégé is an ontology editor and knowledge based frame work [3] whereas OntoGen is said to be a semi automated and data driven tool that combines text mining approaches[8][9].

The semantic web promises for advantages to the current web would compel user to think for extending their current web to the semantic web. There would be a lot of transition steps that will be required for this conversion. As there is heterogeneous type of database associated to the current web and this data is required to be converted in the ontology to become useful for the semantic web. Moreover, this relational database is not in the same normalize form. There would be relational schema to ontology conversion required in transformation. This thesis discusses the same domain (i.e. relational schema to ontology conversion). A better conversion algorithm is presented that will help to recognize relationship in relational schema by gathering already available metadata.

1.2. Problem Definition

As already discussed, after knowing the advantages of the semantic web over current web, one would definitely think for moving from current to future web. This transition will require relational schema to ontology conversion step at some stage. Moreover, the relational schemas on the internet are in heterogeneous formats and it is difficult to propose and implement a single methodology for all of them. This is the major reason that there is no standard transition algorithm, and huge efforts are required for smooth transition from the existing web to semantic web.

Researchers have worked for creating tools for transformation from relational schema to ontologies (e.g. DataGenie, DataMaster etc). DataGenie [3] is a tab plug-in for Protégé that enables Protégé to connect to database and move portions (or all) of your database into Protégé. But, it does not support OWL ontologies or schemas. For the drawbacks in the DataGenie, the DataMaster was developed in BioSTORM [10] project which supports both OWL and frame-based ontologies. It works as a plug-in for protégé [11]. Many of these approaches merely give the results in the user required form. Moreover, the weak entities of the database are also mapped into classes (for example in datamaster, DataGenie etc). It will be tough job to find the relationship in the extracted classes (i.e. ontologies), with a little domain knowledge. Moreover it will become very tough when the relational schema is of large size. Requirement of an automated or semi automated approach for discovering the relationships in the relational schemas was vital.

When it comes to execution cost of the algorithm, the algorithm that uses metadata extraction step for conversion will cost low as it will not require derivation and comparisons steps. Whereas the algorithm that works in absence of metadata will definitely cost higher. But will be the case where some metadata is not available with relational schema. The technique presented here performs extraction of the relationships in the relational schemas when there is no enriched Metadata available.

1.3. Objectives and Goals

This research focuses on providing a flexible approach for association identification in real scenarios, and to propose a technique that extracts metadata available. This research provides an algorithm that will identify ontologies and associations. This approach will be a semi-automated approach. The transformation rules will be used for the analyzing the relationship and types of relationships identified.

The evaluation criteria of the proposed methodology are; (a) performance of algorithm (b) preservation of information and (c) correct identification and transformation of relationships.

Therefore, the evaluation of the proposed scheme will be done by; (i) experimental results, (ii) mathematical proof.

As relational database is well established and widely used data model, so this research will only concentrate on relational database. Extension of proposed methodology to other data models such as semi-structured and un-structured data models will be done in future.

1.4. Outline of thesis

The outline of thesis is as follows. In Chapter 2, background study is provided. Chapter 3 focuses on the literature survey followed by the summarization of the techniques. In Chapter 4, the proposed system architecture and methodology is given. In

Chapter 5 discusses the implementation details of the prototype system are provided. The results and the evaluation of the proposed methodology are given in Chapter 6. Mathematical proof of schema equivalence and information capacity preservation is also provided in this chapter. Finally in Chapter 7 conclusion of the work and direction for future work are presented.

1.5. Summary

In this chapter, the overview of the thesis was given. Motivation, problem definition, objectives and goals of research was discussed.

Chapter 2

BACKGROUND

In this chapter the background study is discussed that will be helpful in understanding this research thesis. In this chapter explanation of semantic web, its approaches and the main components of semantic web are discussed. The relational model, ontology and its language is also presented in this chapter. Latter in this chapter detail of model transformation is provided.

2.1. Semantic Web

The semantic web is often named as future web. It is an extension of the World Wide Web, its aim is to make web meaningful. The semantic web defines semantics of information and services on the web, making it possible for the web to understand people and machines to use the web content [12][13]. It derives from World Wide Web Consortium director Sir Tim Berners-Lee's vision of the Web as a universal medium for data, information, and knowledge exchange [13].

The Semantic Web is a web of data that is globally shared. There is a lot of useful information every day from the internet. Most of this information are not a part of WWW. But can I see my photos in a calendar to see what I was doing when I took them? Can I see bank statement lines in a calendar?

Why not? Because current web is no a web of data. Because data is controlled by applications, and each application keeps it to itself.

The Semantic Web is about two things. It is about common formats for integration and combination of data drawn from diverse sources, where as the original Web mainly concentrates on the interchange of documents. It is also about language for recording how the data relates to real world objects. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases which are connected not by wires but by being about the same thing. [15].

2.1.1. Semantic web approach

The approach of semantic web is to represent web contents in a form that is more easily machine-processable. It uses intelligent techniques to take advantage of these representations. The Semantic web will gradually evolve out of the current web.

2.2. Semantic Web Architecture

A set of standards and technologies has been given by semantic web that acts as an infrastructure to support its vision. These series of standards are interlinked and organized. These standards are shown in the Figure 2.1.

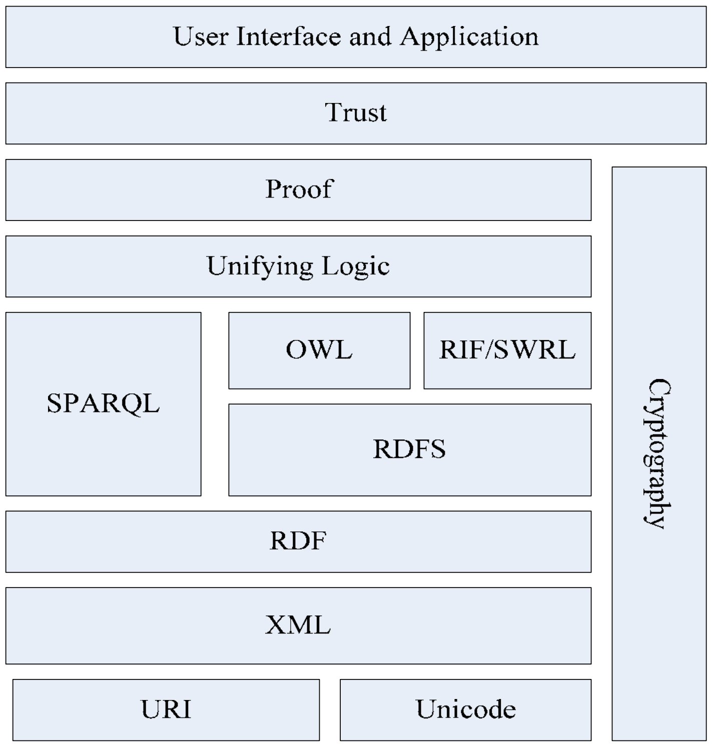


Figure 2.1 Semantic Web Architecture

2.2.1. Universal Resource Identifier (URI)

A universal resource identifier (URI) identifies an abstract or physical resource. A URI can be further classified as a locator, a name, or both. Uniform resource locator (URL) is a subset of URI. A uniform resource name (URN) refers to the subset of URI, which is required to remain globally for example:

a. The URL http://ppclub.org/index.php, identifies the location from where a Web page can be retrieved
b. The URN urn:isbn:3-540-24328-3 identifies a book using its ISBN.

2.2.2. Unicode

Unicode provides a unique number for every character, independently of the underlying platform, program, or language. Before the creation of Unicode, there were various different encoding systems.

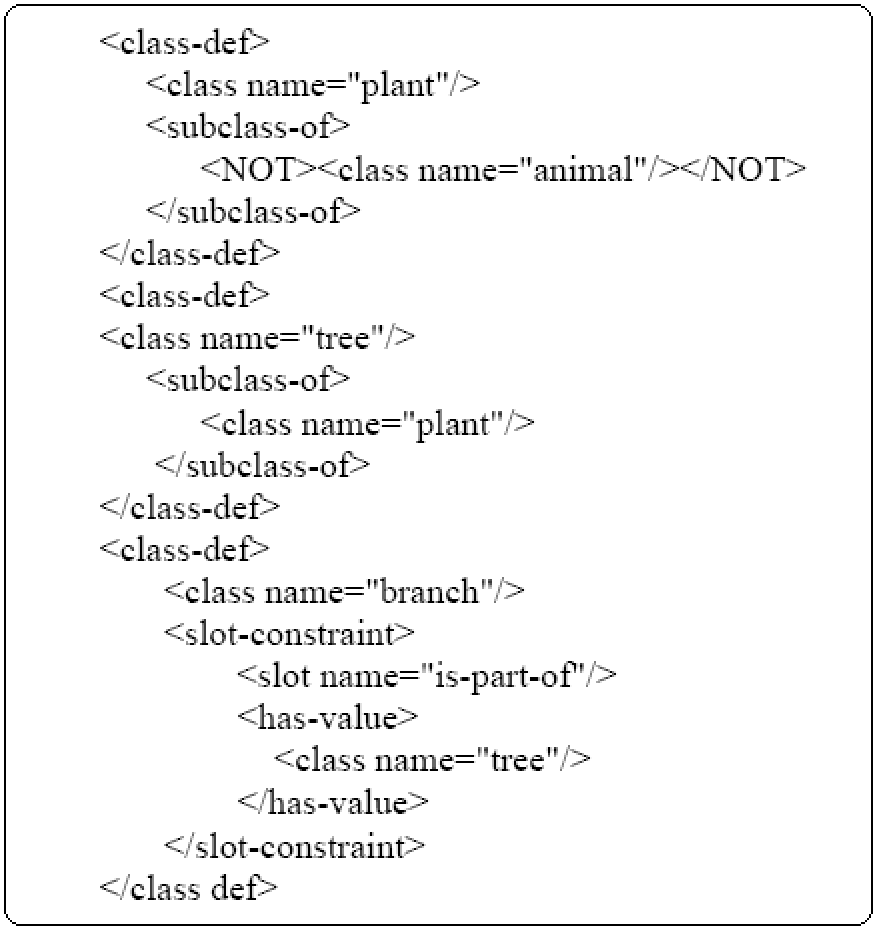
2.2.3. XML

XML stands for External Markup Language. It was designed for the storing and transportation of the data. Is keeps its importance and being easy it is user friendly. XML is verbose by design and it is a family of technologies. It has lead HTML to XHTML. XML is in the form of open and close tags.

2.2.4. Resource Description Frame work (RDF)

The RDF was built for web. It is a framework for describing and interchanging metadata. It is built on the following rules: A Resource can be anything having a URI (e.g. world's Web pages, as well as individual elements of an XML document). An example of a resource http://www.ppclub.org/index.php

1. A PropertyType is a Resource that has a name and can be used as a property, for example Author or Title.
2. A Property is the combination of a Resource, a PropertyType, and a value. An example would be: "The Author of http://www.ppclub.org/index.php is Asif Sohail. The RDF example is shown in Figure 2.2.



```
<class-def>
    <class name="plant"/>
    <subclass-of>
        <NOT><class name="animal"/></NOT>
    </subclass-of>
</class-def>
<class-def>
<class name="tree"/>
    <subclass-of>
        <class name="plant"/>
     </subclass-of>
</class-def>
<class-def>
     <class name="branch"/>
     <slot-constraint>
           <slot name="is-part-of"/>
           <has-value>
              <class name="tree"/>
           </has-value>
     </slot-constraint>
</class def>
```

Figure 2.2 RDF Example

2.2.5. RDF schema

The RDF schema provides a type system for RDF. It can be said as enriched form of RDF. It provides a way of building an object model from which the actual data is referenced and which tells us what things really mean [6].

Some important things to know about RDF Schema are as follows:

a. rdfs:Literal is the class of literal values such as strings and integers.
b. rdfs:subClassOf is a transitive property that specifies a subset-superset relation between classes.
c. rdfs:subPropertyOf is an instance of rdf: Property used to specify that one property is a specialization of another.
d. rdfs:comment is a human-readable description of a resource.
e. rdfs:label is a human-readable version of a resource name and it can only be a string literal.

f. rdfs:seeAlso specifies a resource that might provide additional information about the subject resource.
g. rdfs:isDefinedBy is a subproperty of rdfs: seeAlso and indicates the resource defining the subject resource.
h. rdfs:member is a super-property of all the container membership properties
i. rdfs:range indicates the classes that the values of a property must be members of.

2.2.6. Owl

OWL stands for web ontology language that provides extended impressibility to RDF. This language is divided into three parts [17], which are described as follows

2.2.6.1. OWL lite

OWL Lite is a kind of the OWL ontology language. It is a sublanguage of OWL DL and was originally designed to provide a simpler formalism. This simpler formalism was in terms of computational complexity. This, however, has largely failed, and OWL Lite is almost as complex as OWL DL. The practically relevant species of OWL therefore today are OWL DL and OWL Full.

2.2.6.2. OWL DL

The name OWL DL is due to its correspondence with description logics. OWL DL has been designed to support the existing description logic business segments. OWL DL provides maximum expressiveness without losing computational completeness and decidability of reasoning system. Computational completeness means that all entailments are guaranteed to be computed, whereas decidability means that all computations finish in a finite time. OWL DL has all OWL language constructs along with the restrictions. For example, type separation restriction which means that a class cannot also be an individual or a property simultaneously. Similarly a property cannot also be an individual or class at the same time.

2.2.6.3. OWL Full

OWL Full provides maximum expressiveness with the syntactic freedom of Resource Description Framework (i.e., RDF) but it does not provide computational guarantee.

Unlike OWL DL, in OWL Full a class can be treated as a collection of individuals and as an individual. Another major difference from OWL DL is that owl:DatatypeProperty can also be declared as an owl:InverseFunctionalProperty. owl:DatatypeProperty and owl:InverseFunctioanlProperty are discussed later in detail. OWL Full allows an ontology to increase the meaning of the pre-defined (RDF or OWL) vocabulary.

2.3. Ontologies

The word "ontology" seems to generate a lot of controversy in discussions about AI. It has a long history in philosophy, in which it refers to the subject of existence. It is also often confused with epistemology, which is about knowledge and knowing.

In the context of knowledge sharing, I use the term ontology to mean a *specification of a conceptualization*. That is, ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set-of-concept-definitions, but more general. And it is certainly a different sense of the word than its use in philosophy.

What is important is that what ontology is *for*. My colleagues and I have been designing ontologies for the purpose of enabling knowledge sharing and reuse. In that context, ontology is a specification used for making ontological commitments. The formal definition of ontological commitment is given below. For pragmatic reasons, it was preferred to write ontology as a set of definitions of formal vocabulary. Although this isn't the only way to specify a conceptualization, it has some nice properties for knowledge sharing among AI software (e.g., semantics independent of reader and context). Practically, an ontological commitment is an agreement to use a vocabulary (i.e., ask queries and make assertions) in a way that is consistent (but not complete) with respect to the theory specified by an ontology.

2.3.1. Components of Ontologies

The main components of ontology are called as classes. They are the concepts of the domain of discourse. For example the class “Person” represents all kinds of person. Classes can have sub-classes that are more specific concepts than the super-class. For example the “animal” class can have sub-class “tiger”, “elephant”, “cat”. Multiple inheritances are allowed in ontology. The various features and attributes of a class are described by properties (also called slots or roles). For example cat may belong to special area, can have different color, height etc.

Restrictions (also called facets or role restrictions) can be applied on properties. Ontology together with its instances is called a knowledge base. Noy and McGuinness [16] provided information about how to develop ontology for declarative frame-based systems.

The manual ontology development includes the following steps:

a. Identifying and defining the classes in ontology.
b. Taxonomically (i.e., subclass, superclass) arranging classes to form hierarchy.
c. Defining slots and describing allowed values for these slots.
d. Creating instances by filling in the values for slots.

The fundamental rules for ontology design are: (1) there is no single correct way to develop ontology; there exist alternatives to model a domain. The best solution depends on the application and the intended use of ontology. (2) Ontology development is an iterative process. (3) The quality of ontology can only be accessed by the response of the application for which it is designed. (4) Classes and properties of ontology should be close to objects (physical or logical) and relationships of domain of interest.

2.3.2. Classes and Individuals

Ontology is mostly used to reason about individuals; therefore the classes, individuals and properties must be defined. The most powerful feature of ontology is due to its class-based reasoning. A class definition has a name introduction or reference and a list of restrictions. The list of expressions / restrictions in a class definition restricts instances of the class. The instances of a class belong to the intersection of all the restrictions.

In OWL every individual is a member of class owl:Thing. Hence, every user defined class is implicitly a member of class owl:Thing. The class owl:Nothing is an empty class. The classes are defined by declaring the named class. For example, in a hospital domain the three classes can be “Employee”, “Patient” and “Ward”. The OWL code for these named classes is shown in Figure 2.3:

```
<owl:Class rdf:ID="Employee"/>
<owl:Class rdf:ID="Patient"/>
<owl:Class rdf:ID="Ward"/>
```

Figure 2.3 OWL Example

Class definitions can be extended later. Within this ontology the above declared classes can be referenced with the “#” preceding the name of the class. For example the “Employee” class can be referenced by #Employee. Derived ontologies can be created by importing and augmenting the ontologies. The class definition can be incremental and distributed. A class can have subtypes. For example, in hospital domain, “Employee” can be a “Physician” or a “Nurse”. The example is shown below in Figure 2.4.OWL classes.

```
<owl:Class rdf:ID="Physician">
<rdfs:subClassOf rdf:resource="#Employee"/>
.
.
.
</owl:Class>
```

Figure 2.4 OWL Classes

The “rdfs:subClassOf” relates to a more specific class to a general class. This example relates more specific class “Physician” to the more general class “Employee”. Every instance of the “Physician” class is also an instance of the “Employee” class i.e., if X is a subclass of Y, then every instance of X is also an instance of Y [19]. The rdfs:subclass of relation is transitive. If X is a subclass of Y and Y a subclass of Z then X is a subclass of Z [16]. Transitive property is discussed in detail later in this chapter. rdf:type is an RDF property that ties an individual to a its class. For example “Dr Kamran” is an individual of the “Physician” class. An individual is a member of a class.

2.3.3. Property

A property is a binary relation. Two types of important properties are (1) objectproperty and (2) datatypeproperty. Datatypeproperty is a relation between instance of class and RDF literal or XML schema data type. Objectproperty is the a relation between instances of two classes [19]. Figure 2.5 shows the object and the data property.

```
<owl:ObjectProperty rdf:ID="works">
<rdfs:domain rdf:resource="#Physician"/>
<rdfs:range rdf:resource="#Ward"/>
.
.
.
</owl:ObjectProperty>
```

Figure 2.5 OWL Objects

The domain and range of these properties can be defined to restrict the relations. For example a Physician works in a Ward, the code to define the domain and range of the “works” objectproperty is shown below.

Multiple domains of a property mean domain of property is intersection of the identified classes. The same is true for multiple ranges. Properties, like the classes can also be arranged in a hierarchy. Figure 2.6 shows the properties hierarchy [16].

```
<owl:Class rdf:ID="WineDescriptor" />
<owl:Class rdf:ID="WineColor">
<rdfs:subClassOf rdf:resource="#WineDescriptor" />
</owl:Class>
<owl:ObjectProperty rdf:ID="hasWineDescriptor">
<rdfs:domain rdf:resource="#Wine" />
<rdfs:range rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasColor">
<rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
<rdfs:range rdf:resource="#WineColor" />
.
.
.
</owl:ObjectProperty>
```

Figure 2.6 OWL Object Property

2.4. Relational model

The first database systems were based on the network and hierarchical models. These models faced many problems when used for scalable databases. To faults gives birth to relational model. The relational model was first proposed by E.F. Codd in 1970 and the first such systems (notably INGRES and System/R) were developed in 1970s. The relational model is now the dominant model for commercial data processing applications. The relational database consists of a collection of tables, each having a unique name. A row in a table represents a relationship among a set of values. Thus a table represents a collection of relationships. There is a direct correspondence between the concept of a table and the mathematical concept of a relation. A substantial theory has been developed for relational databases.

2.4.1. Keys

A relation can have different types of keys. The keys Ri selected from the attribute of the relation Ri can be described as [20]:

$$K(Ri) \subseteq A(Ri) \tag{1}$$

There can be three types of keys

(a) Candidate key

A candidate key is that attribute that qualifies for all the required properties of a primary key (i.e. cannot be null and cannot be repeated)

(b) Primary key

A primary key is an attribute of relation, which cannot be null or cannot be repeated. It identifies a record uniquely.

(c) Foreign key

This key identifies parent and child entities of a relationship. Parent entity posts a copy of its primary key to the child entity, where it acts as a foreign key. The name of the foreign key can be different from the name of the primary key. Foreign key can have the same value of primary key for a set of tuples (i.e., record) [20]. NULLs are also allowed in foreign keys.

2.5. SuperClass and SubClass

Superclass and subclass help to form hierarchy (i.e., EERD) and identifies attribute inheritance. Entity with its subclass and their subclass is called type hierarchy, specialization hierarchy, generalization hierarchy and IS_A hierarchy [20]. These terms can be used alternatively. For example, the “Person” entity is the superclass of the “Student” entity and “Professor” entity whereas “Professor” and “Student” entity are the subclass of “Person” entity.

2.6. Transforming Relational Model into Ontology

Astrova1 and colleagues [18] explain the difference between transformation and mapping. Mapping assumes the existence of both ontologies and relational databases and produces set of correspondence between these two. Transformation on the other hand assumes that only relational database exists and builds ontology from it. Both mapping and transformation is shown in the Figure 2.7 and Figure 2.8.

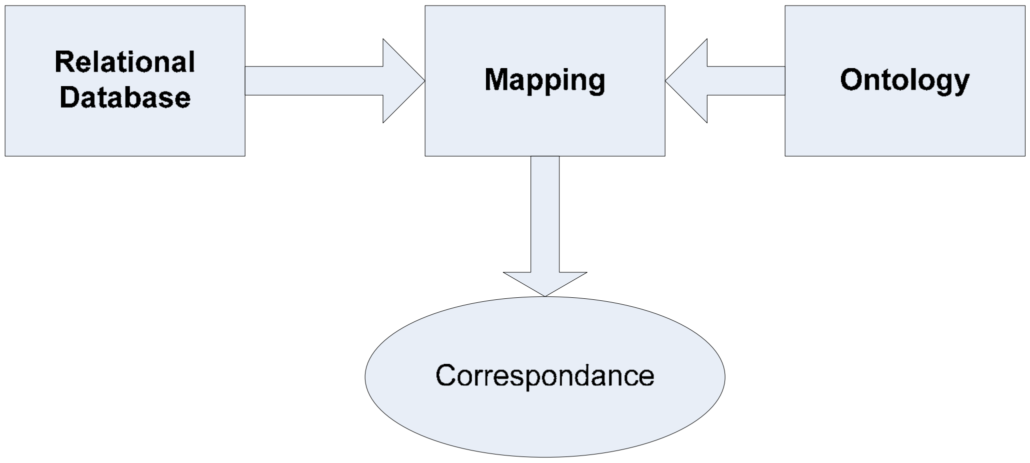


Figure 2.7 Relational Database to Ontology Mapping

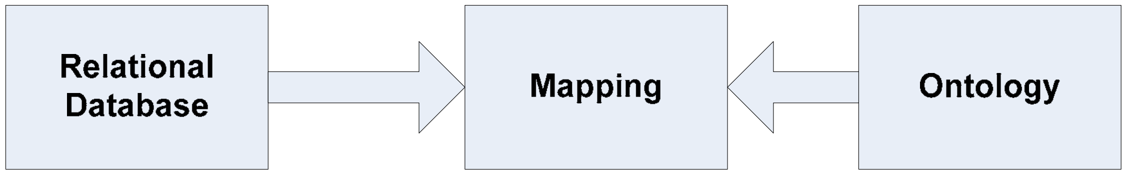


Figure 2.8 Transformation of Relational Database to Ontology

2.7. Schema Translation

P.Martin, J.R.Cordy, R.A.Hamdeh [20] define that schema translation is the process of transforming a schema in one data model into a corresponding schema in a different data model. Structural transformation recognizes structures in a source object (schema, program, etc.) and transforms them into other structures in a target language to produce a translation of the original object. Structural transformation can be applied to the problem of schema translation by searching for the structures in source schema and then transforming the structures in the translated schema.

2.8. Summary

In this chapter, the background knowledge of the research was presented. Describing the basis of semantic web, ontologies and relational schema the ground for the understanding of research was created. The next chapter will provide the literature survey that was carried out for the research.

Chapter 3

LITERATURE SURVEY

This chapter will provide the literature survey for the thesis. The relational schema to ontology transformation techniques will be discussed in detail. Moreover, the critical analysis of the techniques will also be made in the end of the chapter.

3.1. Related Work

As future web promises for many advantages including meaningful and understandable web [7], to use the information associated with current web with future web transition will be required. This transition will require relational schema to ontology conversion step at some stage. Moreover, the relational schemas on the internet are in heterogeneous formats and it is difficult to propose and implement a single methodology for all of them. This is the major reason that there is no standard transition algorithm, and huge efforts are required for smooth transition from the existing web to semantic web.

Researchers have worked for creating tools for transformation from relational schema to ontologies (e.g. DataGenie, DataMaster etc). DataGenie [3] is a tab plug-in for Protégé that enables Protégé to connect to database and move portions (or all) of your database into Protégé. But, it does not support OWL ontologies or schemas. For the drawbacks in the DataGenie, the DataMaster was developed in BioSTORM [10] project which supports both OWL and frame-based ontologies. It works as a plug-in for protégé [11]. Many of these approaches merely give the results in the user required form. Moreover, the weak entities of the database are also mapped into classes (for example in datamaster, DataGenie etc). The technique presented here, performs extraction of the relationships in the relational schemas when there is no enriched Meta data available.

There is an approach for creating semantic metadata from relational database data [1]. When ontology-based information systems are created it is often required to convert or replicate data from existing information systems (such as databases) to the ontology based information systems, if it is desire for ontology-based systems to work with real data. RDB2Onto tool converts selected data from a relational database to a RDF/OWL ontology document based on a defined template. Such filled in templates can be then stored to the ontology-based knowledge memory. In the paper they also evaluate the tool against existing solutions, such as RDQUERY or D2RQ. The architecture they present for RDB2Onto is shown in Figure 3.1- RDB2Onto Architecture.

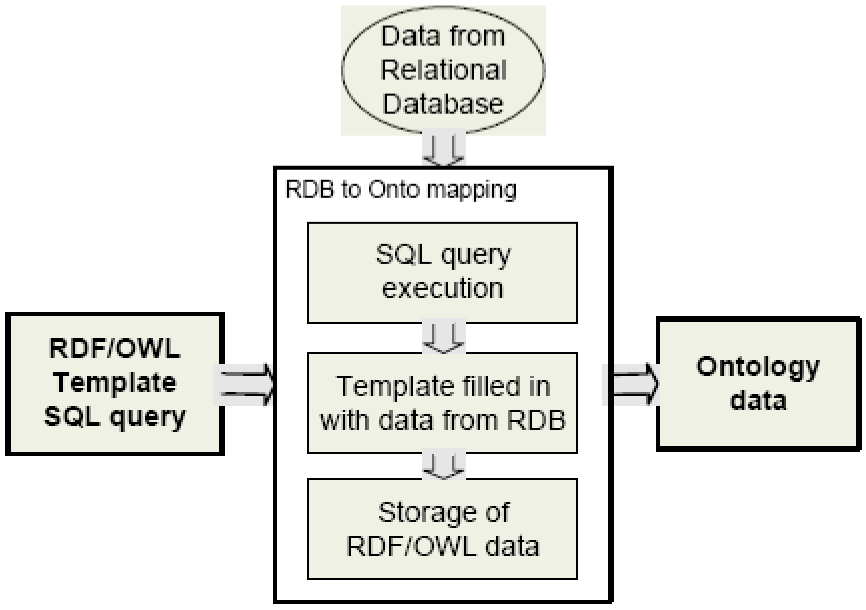

Data from Relational Database
RDB to Onto mapping
SQL query execution
Template filled in with data from RDB
Storage of RDF/OWL data
RDF/OWL Template SQL query
Ontology data

Figure 3.1 RDB2Onto Architecture

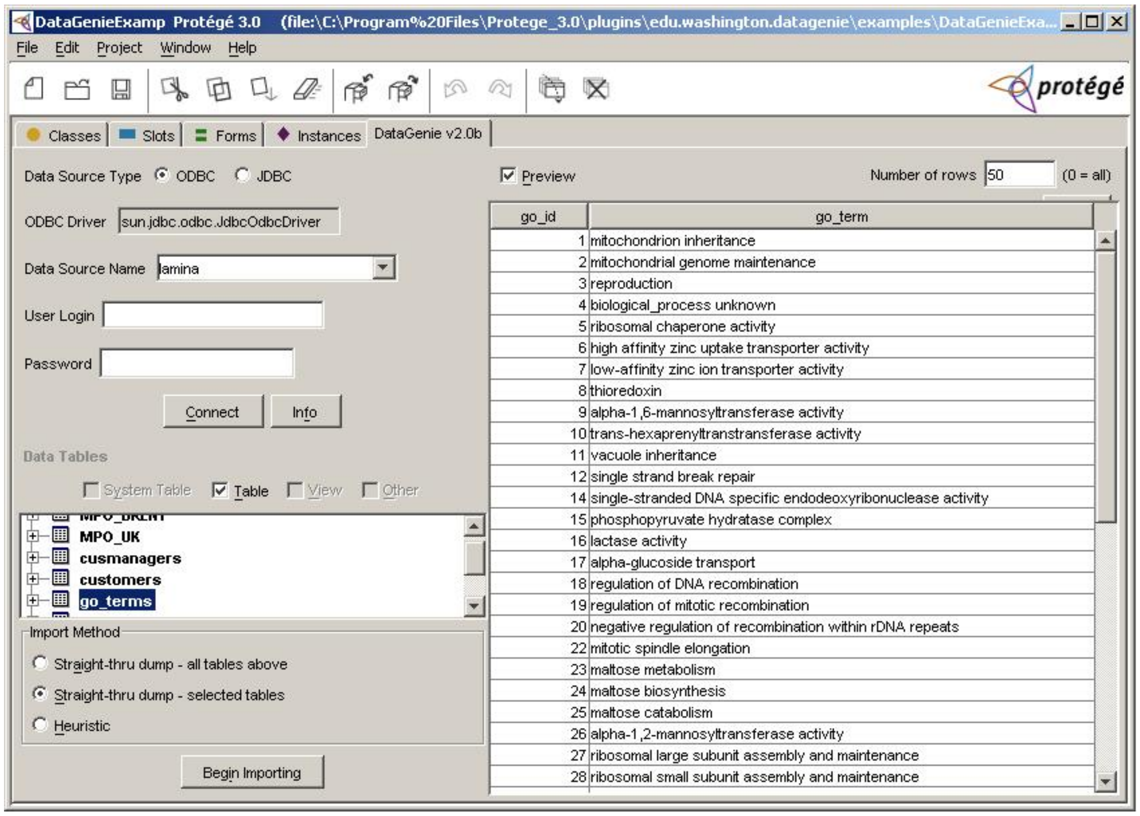
The goal of the tool is to provide Relational Database Data to Ontology Individuals Mapping. The tool works on a domain ontology model and a relational database. The overall idea is to map SQL query to RDF/OWL XML template. Such OWL data are then sent to an ontology model. The tool is being implemented in Java using Jena [21] or Sesame [22] library for ontology manipulation and MySQL database for testing but it is possible to use any other relational database using JDBC connector.

An Ontology matching tool utilizes a composite approach to combine different match algorithms [23]. COMA++ implements significant improvements and offer a comprehensive infrastructure to solve large real-world match problems. It comes with a graphical interface enabling a variety of user interactions. Using a generic data representation, COMA++ uniformly supports schemas and ontologies, e.g. the powerful standard languages W3C XML Schema and OWL. COMA++ includes new approaches for ontology matching, in particular the utilization of shared taxonomies. Furthermore, different match strategies can be applied including various forms of reusing previously determined match results and a so-called fragment based match approach which decomposes a large match problem into smaller problems. Finally, COMA++ cannot only be used to solve match problems but also to comparatively evaluate the effectiveness of different match algorithms and strategies.

An Approach Based on an Analysis of HTML Forms was being proposed [2]. According to this approach, semantics of a relational database can be inferred, without an explicit analysis of relational schema, tuples and user queries. Rather, these semantics can be extracted by analyzing HTML forms, which are the most popular interface to communicate with relational databases for data entry and display on the Web. The semantics are supplemented with the relational schema and user "head knowledge" to build ontology. Our approach can be applied to migrating data-intensive Web pages, which are usually based on relational databases, to the ontology based Semantic Web. There is a technique that builds OWL based initial source ontology to integrate data sources [24]. The technique analyzes the SQL/DDL code, used to build database and divides the tables into two categories. (1) Tables without foreign keys: OWL class for each table and DataType properties (created as functional properties) for respective attributes are created. The specific range of the properties is assigned with owl:allValuesFrom restriction. Cardinality restriction is applied to all NOT NULL attributes. (2) Tables with foreign keys: Tables in this category are further divided into two groups based on the number of foreign keys; (2a) tables with one foreign key (table correspond to 1:M relationship): OWL class, DataType properties and functional Object property are created for tables. (2b) tables with more than one foreign key (table correspond to M:M relationship): In this group the tables are of two types; (i) table without additional attributes: Object property and inverse of this property is created and domains and ranges are specified for both properties. The cardinalities cannot be inferred from the DDL code. (ii) Table with additional attributes: OWL class is created in addition to the properties specified above.

When a table has more than two foreign keys, OWL class is necessarily created because OWL does not allow properties with degree greater than 2. The positive contribution of the paper are (1) it converts strong entities, weak entities, binary M:M relationship with and without attributes and ternary relationships to OWL language. (2) It provides the advantages and disadvantages of building ontology from ERD, table and DDL code. The deficiencies of the technique are (1) it may identify multi-valued attributes and cardinalities. (2) It may not consider the hierarchies, which are built in OWL to identify specialized concepts and generalized concepts. The technique is dependent on the availability of DDL script. It makes it inappropriate be used in distributed environment. (4) The results of the technique are not provided. There is an approach consists of methods and techniques for generating data transformation rules needed for the data structure normalization. One important problem with modern organizations is the existence of non-integrated information systems, inconsistency and lack of suitable correlations between legacy and modern systems. One main solution is to transfer the local databases into a global one. In this regards there is a need to extract the data structures from the legacy systems and integrate them with the new technology systems. In legacy systems, huge amounts of a data are stored in legacy databases. They require particular attention since they need more efforts to be normalized, reformatted and moved to the modern database environments. Designing the new integrated (global) database architecture and applying the reverse engineering requires data normalization. Their paper proposes the use of database reverse engineering in order to integrate legacy and modern databases in organizations. There is some work on the scalable data integration [20]. It is significant in distributed environment and requires the generation and formal representation of conceptual model of source description. Ontology is a formal, shared and common understanding of a domain. It can solve the heterogeneity among the sources. The R2O transformation system provided in this paper transforms database relations to OWL based ontology for a source. Compared with existing techniques, the distinguished feature of this technique is to build ontology in the absence of necessary metadata from relations. It minimizes the effort and errors involved in manual ontology building. Results of the proposed methodology are provided to show the transformation is correct (i.e., total, injective). DataGenie is a tab plug-in that allows Protégé to read from arbitrary database. DataGenie uses either JDBC or ODBC/JDBC to connect to a specified database, and then allows the user to move portions (or all) of the database into Protégé classes. Generally, each table becomes a class, and each attribute becomes a slot. In addition, if the relational database table has foreign key references to other tables, these can be replaced by Protégé instance pointers when the database is converted into a knowledge base.

This plug-in is NOT a database back-end. We expect this plug-in to be used when there exists legacy data that one wants to dump into Protégé, before doing additional knowledge acquisition or knowledge modeling. This plug-in (as written) does not include any capability for moving data in the opposite direction (from Protégé classes and instances into a relational database). Another use for this plug-in might be as a database viewer. For efficiency, a database might be stored as a set of custom-designed database tables, but then the DataGenie could be used to view portions of this schema in the Protégé frame-based UI. Figure 3.2 shows the snapshot of the DataGenie


DataGenieExamp Protégé 3.0 (file:\C:\Program%20Files\Protege_3.0\plugins\edu.washington.datagenie\examples\DataGenieExa...
File
Edit
Project
Window
Help
protégé
Classes
Slots
Forms
Instances
DataGenie v2.0b
Connect
Info
Begin Importing

Data Source Type [✓ ODBC] JDBC

ODBC Driver: sun.jdbc.odbc.JdbcOdbcDriver

Data Source Name: lamina

User Login:

Password:

Data Tables

System Table [✓ Table] View Other

MPO_BRENT
MPO_UK
cusmanagers
customers
go_terms

Import Method

Straight-thru dump - all tables above
[✓ Straight-thru dump - selected tables]
Heuristic

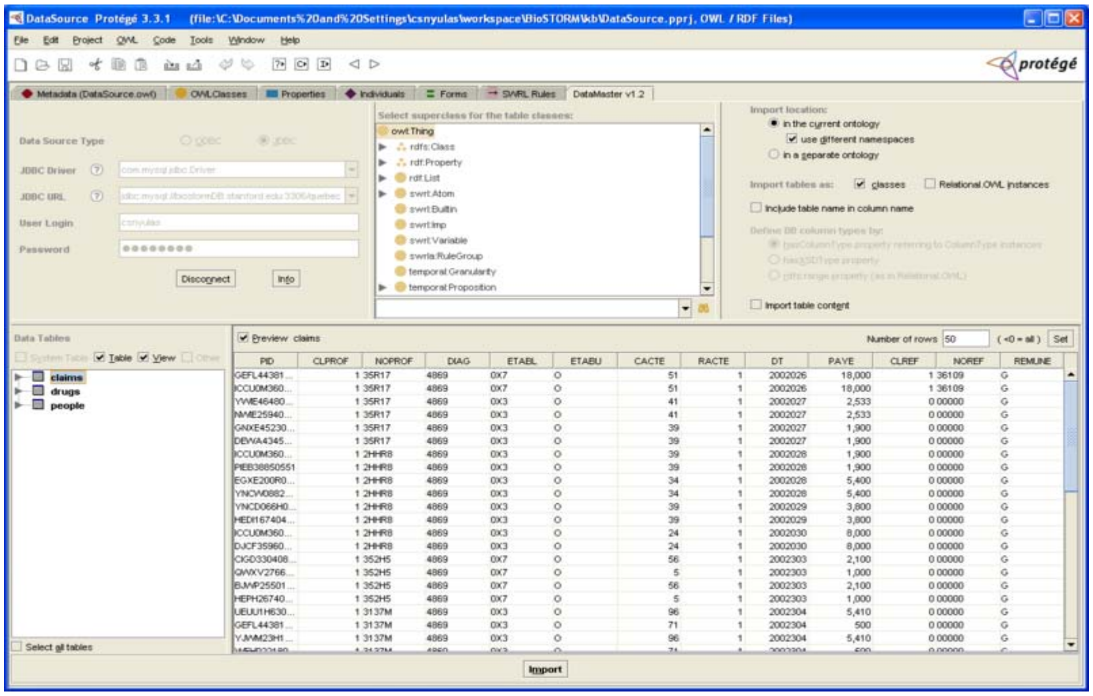
[✓ Preview] Number of rows 50 (0 = all)

go_id	go_term
1	mitochondrion inheritance
2	mitochondrial genome maintenance
3	reproduction
4	biological_process unknown
5	ribosomal chaperone activity
6	high affinity zinc uptake transporter activity
7	low-affinity zinc ion transporter activity
8	thioredoxin
9	alpha-1,6-mannosyltransferase activity
10	trans-hexaprenyltranstransferase activity
11	vacuole inheritance
12	single strand break repair
14	single-stranded DNA specific endodeoxyribonuclease activity
15	phosphopyruvate hydratase complex
16	lactase activity
17	alpha-glucoside transport
18	regulation of DNA recombination
19	regulation of mitotic recombination
20	negative regulation of recombination within rDNA repeats
22	mitotic spindle elongation
23	maltose metabolism
24	maltose biosynthesis
25	maltose catabolism
26	alpha-1,2-mannosyltransferase activity
27	ribosomal large subunit assembly and maintenance
28	ribosomal small subunit assembly and maintenance

Figure 3.2 DataGenie

Importing data from relational databases into ontologies is frequently required, especially when ontology is used to describe semantically the data used by a software application. Another growing category of applications needs database-ontology integration and/or interoperation, where a mapping between the database schema structure and ontology concepts is the main focus. In the latter cases the import of the data residing in relational databases may not be necessary or desired. To meet these requirements, a Protégé plug-in that allows the user to import in a configurable way a relational database structure into a Protégé-OWL or Protégé-Frames ontology. The plug-in also supports the optional importing of table contents.

The development of DataMaster was necessary, because existing plug-ins developed for importing data from relational databases into Protégé, such as DataGenie [1], do not support Protégé-OWL, schema-only import, and other import configurations available in DataMaster. The DataMaster plug-in has been developed in the BioSTORM [2] project, which aims to develop a computational test bed that can draw on real-world data sources and that will allow users to configure, run, and evaluate alternative surveillance methods. The plug-in represents an important part of the semantic data-access layer, which annotates and integrates disparate data sources into a semantically uniform data stream. Figure 3.3 shows the DataMaster.


DataSource Protégé 3.3.1 (file:\C:\Documents%20and%20Settings\csnyulas\workspace\BioSTORM\kb\DataSource.pprj, OWL / RDF Files)
File Edit Project OWL Code Tools Window Help
protégé
Metadata (DataSource.owl) OWLClasses Properties Individuals Forms SWRL Rules DataMaster v1.2
Disconnect Info
Import

Data Source Type: ○ ODBC ● JDBC
JDBC Driver: com.mysql.jdbc.Driver
JDBC URL: jdbc:mysql://biostormDB.stanford.edu:3306/quebec
User Login: csnyulas
Password: ●●●●●●●●

Select superclass for the table classes:
owl:Thing
rdfs:Class
rdf:Property
rdf:List
swrl:Atom
swrl:Builtin
swrl:Imp
swrl:Variable
swrla:RuleGroup
temporal:Granularity
temporal:Proposition

Import location:
● in the current ontology
☑ use different namespaces
○ in a separate ontology
Import tables as: ☑ classes ☐ Relational OWL instances
☐ Include table name in column name
Define DB column types by:
● hasColumnType property referring to ColumnType instances
○ hasXSDType property
○ rdfs:range property (as in Relational OWL)
☐ Import table content

Data Tables
☐ System Table ☑ Table ☑ View ☐ Other
claims
drugs
people
☐ Select all tables

☑ Preview claims   Number of rows 50 (<0 = all) Set

PID	CLPROF	NOPROF	DIAG	ETABL	ETABU	CACTE	RACTE	DT	PAYE	CLREF	NOREF	REMUNE
GEFL44381...	1	35R17	4869	0X7	O	51	1	2002026	18,000	1	36109	G
ICCU0M360...	1	35R17	4869	0X7	O	51	1	2002026	18,000	1	36109	G
YWIE46480...	1	35R17	4869	0X3	O	41	1	2002027	2,533	0	00000	G
NWIE25940...	1	35R17	4869	0X3	O	41	1	2002027	2,533	0	00000	G
GNXE45230...	1	35R17	4869	0X3	O	39	1	2002027	1,900	0	00000	G
DEWA4345...	1	35R17	4869	0X3	O	39	1	2002027	1,900	0	00000	G
ICCU0M360...	1	2HHR8	4869	0X3	O	39	1	2002028	1,900	0	00000	G
PIEB38850551	1	2HHR8	4869	0X3	O	39	1	2002028	1,900	0	00000	G
EGXE200R0...	1	2HHR8	4869	0X3	O	34	1	2002028	5,400	0	00000	G
YNCW0882...	1	2HHR8	4869	0X3	O	34	1	2002028	5,400	0	00000	G
YNCD066H0...	1	2HHR8	4869	0X3	O	39	1	2002029	3,800	0	00000	G
HEDI167404...	1	2HHR8	4869	0X3	O	39	1	2002029	3,800	0	00000	G
ICCU0M360...	1	2HHR8	4869	0X3	O	24	1	2002030	8,000	0	00000	G
DJCF35960...	1	2HHR8	4869	0X3	O	24	1	2002030	8,000	0	00000	G
CIGD330408...	1	352H5	4869	0X7	O	56	1	2002303	2,100	0	00000	G
QWXV2766...	1	352H5	4869	0X7	O	5	1	2002303	1,000	0	00000	G
BJWP25501...	1	352H5	4869	0X7	O	56	1	2002303	2,100	0	00000	G
HEPH26740...	1	352H5	4869	0X7	O	5	1	2002303	1,000	0	00000	G
UEUU1H630...	1	3137M	4869	0X3	O	96	1	2002304	5,410	0	00000	G
GEFL44381...	1	3137M	4869	0X3	O	71	1	2002304	500	0	00000	G
YJWM23H1...	1	3137M	4869	0X3	O	96	1	2002304	5,410	0	00000	G

Figure 3.3 DataMaster

A work was carried out that provides a technique that builds OWL based initial source ontology to integrate data sources [24]. The technique analyzes the SQL/DDL code, used to build database and divides the tables into two categories. (1) Tables without foreign keys: OWL class for each table and DataType properties (created as functional properties) for respective attributes are created. The specific range of the properties is assigned with owl:allValuesFrom restriction. Cardinality restriction is applied to all NOT NULL attributes. (2) Tables with foreign keys: Tables in this category are further divided into two groups based on the number of foreign keys; (2a) tables with one foreign key (table correspond to 1:M relationship): OWL class, DataType properties and functional Object property are created for tables. (2b) tables with more than one foreign key (table correspond to M:M relationship): In this group the tables are of two types; (i) table without additional attributes: Object property and inverse of this property is created and domains and ranges are specified for both properties. The cardinalities cannot be inferred from the DDL code. (ii) Table with additional attributes: OWL class is created in addition to the properties specified above. When a table has more than two foreign keys, OWL class is necessarily created because OWL does not allow properties with degree greater than 2.

The positive contribution of the paper are (1) it converts strong entities, weak entities, binary M:M relationship with and without attributes and ternary relationships to OWL language. (2) It provides the advantages and disadvantages of building ontology from ERD, table and DDL code. The deficiencies of the technique are (1) it may not identify multi-valued attributes and cardinalities. (2) It may not consider the hierarchies, which are built in OWL to identify specialized concepts and generalized concepts. (3) The technique is dependent on the availability of DDL script. It makes it inappropriate to be used in distributed environment. (4) The results of the technique are not provided.

There is also a work which provides an ontology learning framework [25] and group of learning rules for (1) classes, (2) properties, (3) hierarchy, (4) cardinalities, and (5) instances to learn ontology from relational database. The technique has an assumption that all tables should be in third normal form (3 NF). The rules for learning classes build OWL class in two cases; (1a) the information about an entity is spread across various tables, (1b) the table represents a real world entity instead of a relationship. The rules for learning properties are; (2a) two object properties is_part_of and has_part are created for each weak entity. (2b) If a table is a M: M binary relationship between two entities, then two object properties are created. (2c) Complex (n-ary) relationships between entities are broken into groups of binary tables and object properties are created. (2d) Datatype properties are created for allattributes except foreign keys. The domain and range of all properties are specified. The rule for learning hierarchy learns OWL hierarchies from IS_A relationships. Rules for learning cardinalities are; (4a) for each primary key, minCardinality and maxCardinality of the property are set to one (01). (4b) If an attribute has NOT NULL constraint, then minCardinality of the property is set to one (01). (4c) If an attribute has UNIQUE constraint, then max Cardinality is set to one (01). The rule for learning instances converts the relational records to OWL instances. The positive aspect of the technique is it provides an ontology learning framework followed by a set of rules that covers OWL classes, properties, restrictions and instances. The deficiencies suffered by the technique are; (1) the proposed rules do not cover relationships with attributes and multi-valued attributes. (2) OWL class is not created for n-array relationship in the proposed technique; however OWL class is built for n-array relationships.

3.2. Summary

In this chapter the discussion on some techniques that are used for the relational schema to ontology transformation and ontology matching are discussed. Some techniques architecture presented in this chapter, these technique either go for meta data extraction or completely ignore the meta data.

Chapter 4

PROPOSED METHODOLOGY

Proposed methodology is discussed in this chapter. The architecture of the proposed scheme and its working is presented. The rules that were considered in identification of relationships in real scenarios are also in the focus.

4.1. Proposed Scheme Architecture

To extract the information from a relational schema, either metadata is considered or in case it is not available then desire information through algorithms should be driven. But, what will be the case if some metadata is available? A relational schema was given to DataMaster, being a metadata extractor it fails to gather the foreign key information from the MySQL as the Metadata of MySQL was not enriched. In this case, if algorithm that will start deriving information from zero is selected then this approach will increase the execution cost of the algorithm. Figure 4.1 shows the proposed architecture of the system.

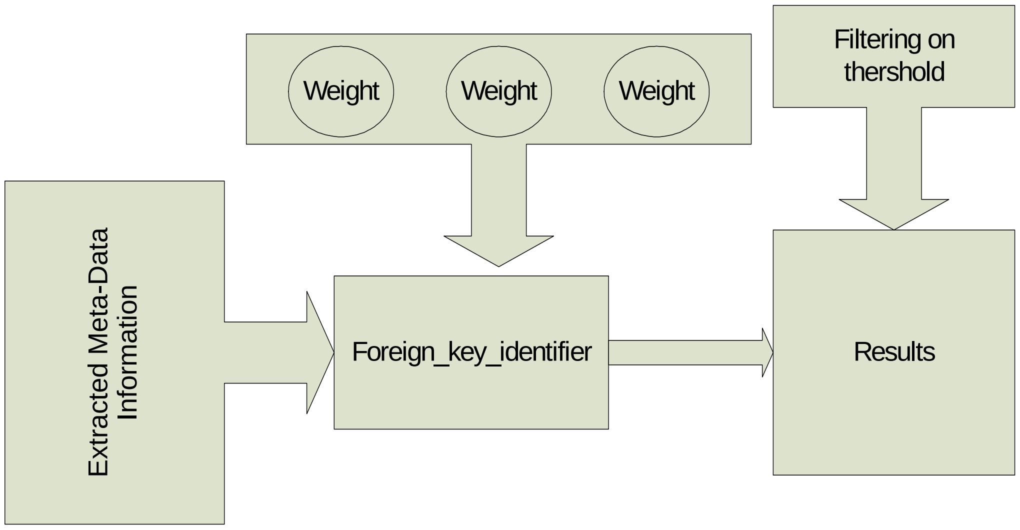


Figure 4.1 Proposed Architecture

4.2. The Flowchart for Proposed Scheme

At this point, one would definitely think to go for an efficient approach. The proposed scheme works with the gathered information from metadata and resolves foreign keys identification problem based on some weights assigned as parameters by the user or the default adjusted weights can be used. As shown in Figure 4.2.

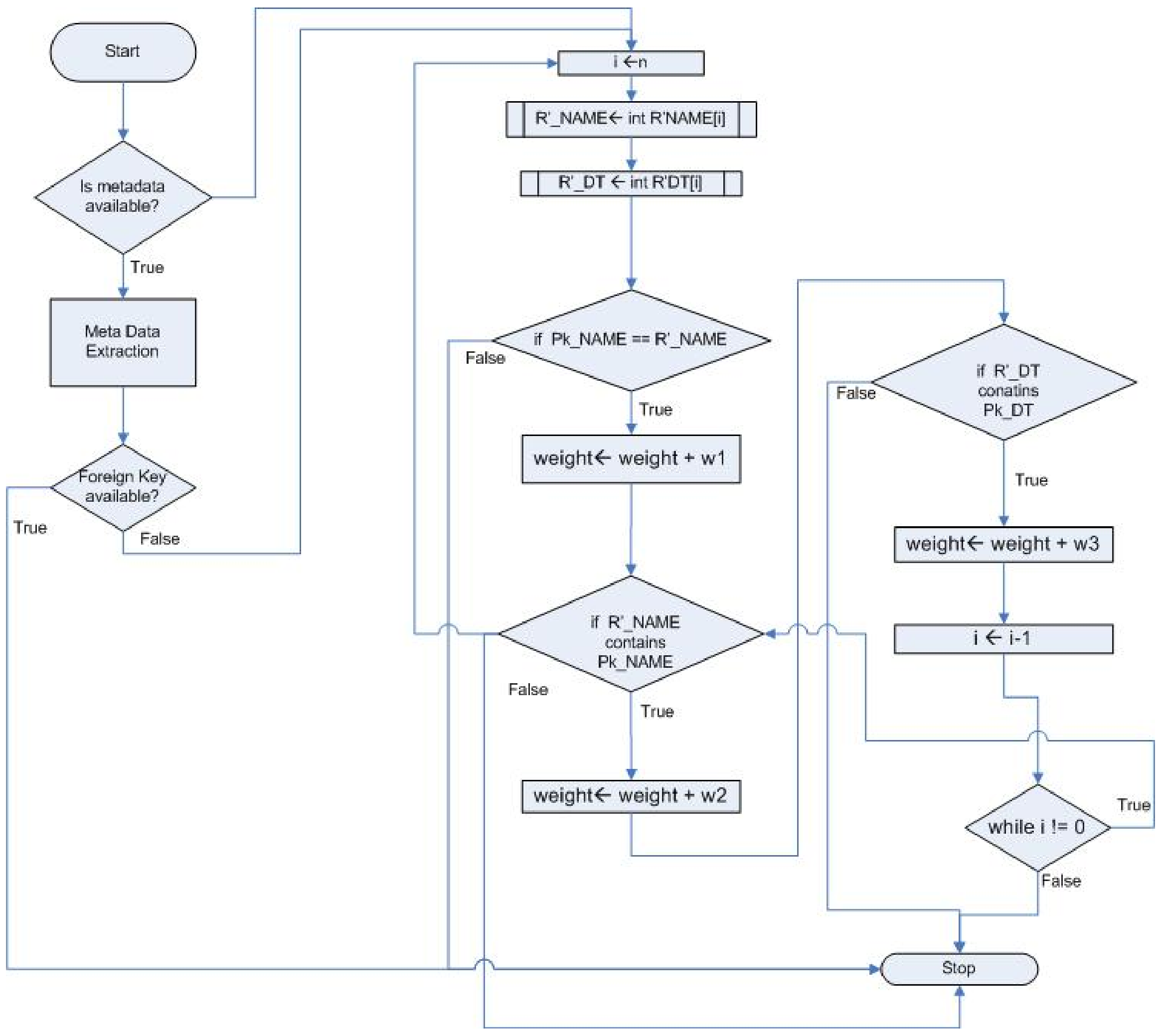


Figure 4.2 Flowchart of Proposed Scheme

4.3. Working of Proposed Scheme

Firstly the metadata available with the database is considered. The information that is available is extracted and algorithm for gathering remaining unavailable information is evoked.

In absence of metadata the identified primary keys are passed as a parameter to algorithm i.e. Algorithm_FK1. This algorithm compares the attributes of a relation with all other attributes in the relational schema. If name of the attribute is equal to any other attribute the weight (w1) is added, and if it is not matched then remaining condition are not compared, and comparison to other attributes are made in the same way. If the names of the attributes match with the other then its datatype and constraints are considered and weights are added so that the results should be more precise.

The identification of foreign keys from the relational schema in absence of the metadata will be perform on the basis of three weights w1, w2 and w3. These weights can be increased to have more accuracy in the results. The resultant dataset is taken and threshold is applied to it. Identification of the foreign keys are made with the highest match values in the dataset. The thing to remember here is that higher the threshold value, higher is the accuracy of the algorithm.

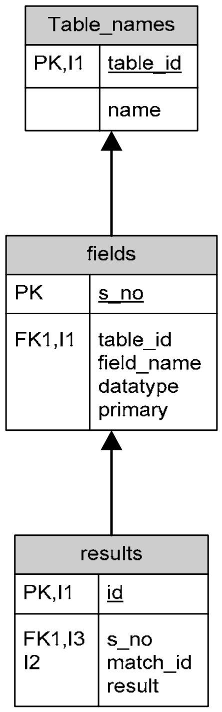


Figure 4.3 Meta Collectors

Figure 4.3 shows the Meta collector, which have three tables are used to gather the already available metadata information of the relational schema. Tablename table is used to collect the information about the tables in the schema. Fields table is used to collect the information of the fields/attributes of the relations. The result table holds the match values which are filtered by the threshold.

4.4. DataMaster Architecture

Let us consider the architecture of the datamaster. DataMaster may be used to import a relational database structure and the table data into a Protégé-Frames ontology. The ontology for describing the database structure (Figure 1) is the same as the one used by the DataGenie plug-in.

All imported database tables are defined as Protégé classes that are instances of the Table Metaclass meta-class. Each column of the database table is represented by a template slot added to the newly created table classes. The column slots will have data types corresponding to the SQL types associated to the database columns. If there are foreign keys defined between the database tables, for each foreign key an instance of the Foreign Key class will be created that will be used to link the corresponding ontology classes.

It is also possible to import the data from the tables in the database: for each row in the table an instance of the table class will be created, and the values of the own slots at these instances will be set with values contained in the table row corresponding to the table columns. An extra slot of type instance will be created for each foreign key defined in the table that will point to the instances corresponding to the referred rows.

The basic algorithms from the datamaster shown in Figure 4.4 are considered. The proposed algorithm described in appendix- A.

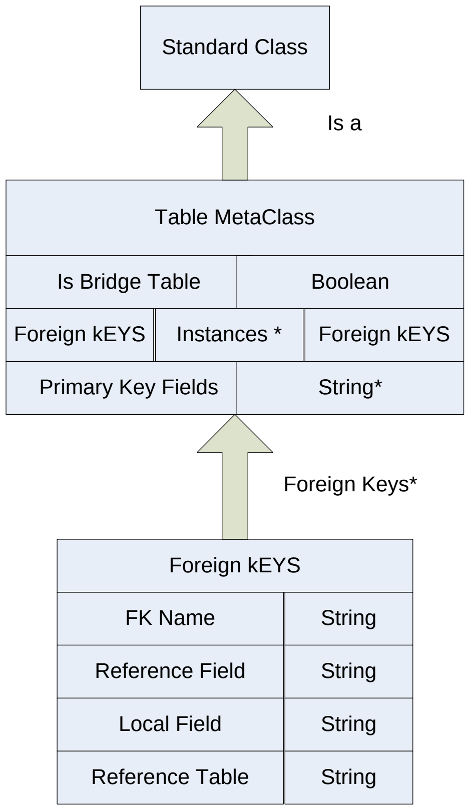


Figure 4.4 DataMaster

4.5. Identification of Foreign Key

This section provides the details of the second part of the proposed methodology. The identified relations in the extracted metadata can be classified into five categories based on the type and presence of keys.

The techniques presented in [20] are used and extensions of these categories are made. Following rules are necessary to understands the background[8, 9, 17, 20]. A relation scheme represents the relation names and the distinct attribute names denoted as

$$R = \{A_1, A_2 \ldots\ldots.A_n\} \text{ or } R(X) \qquad \text{---------------} \qquad (4.1)$$

A Relation instance of relation scheme R is a finite set of tuples denoted as

$$R = \{t_1, t_2\ldots\ldots t_m\} \qquad \text{----------------------} \qquad (4.2)$$

A database schema D for relational model is pair <R, C> where R = Relation schemes with distinct names, C = a set of integrity constraints. Database instance of database schema D with

$$R = \{R_1(X_1), R_2(X_2),\ldots.R_n(X_n)\} \quad \text{---------------------------} \quad (4.3)$$

R is set of relational instances R={r1, r2…..rn}where each instance ri is defined on corresponding schema Ri and satisfies integrity constraint C. The foreign keys of a relation scheme R(X) is defined as a set of all foreign key attributes and denoted as

$$FK= fk_1 \text{ U } fk_2 \text{ U } fk_3 \text{ U } \ldots\ldots \text{U } fk_n \quad \text{-------------------------------} \quad (4.4)$$

where n is the number of distinct foreign keys. The primary key (i.e., simple or composite) is denoted as PK. Let X denotes the entire attributes of relation scheme R(X).

4.6. Example Scenario

An inventory system has been selected for implementation. The customers and employees are identified by the numbers called as “customerNumber” and “employeeNumber” respectively. All offices of the company are identified by the “officeCode”. The customer can place one or many orders “Payements” of the respective orders are stored in payments table. “Productlines” tables contain the different orders from a customer, “orders” contain the details of the orders placed. The detail of each order is stored in the “order details” table. One or more employees may be associated with a particular customer’s order. The product table contains the product information available for sale. Each product is assigned a “productCode”. One or more products can be in one product line.

To create an environment for the algorithm, foreign key information or any linking in the database (i.e. the tables were stored in the MySQL without foreign key information) were not considered. The Entity relationship diagram of selected relational schema is shown in Figure 4.6.

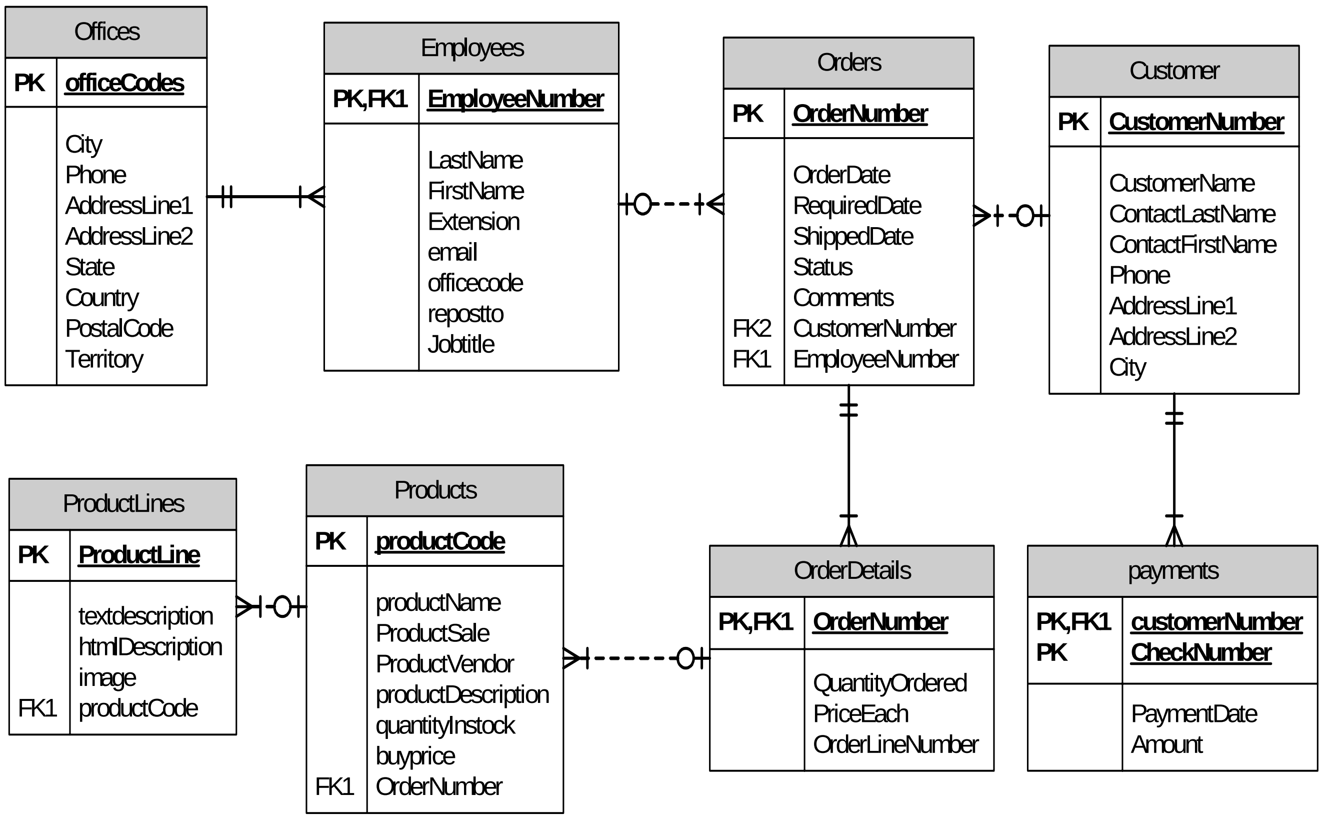


Figure 4.5 ERD of Schema

4.7. Entities and Attributes

First of all description of the different cases of metadata availability for data sources in a distributed environment are presented. Then the steps of methodology are described to extract metadata from data source relations in order to develop an EERD. In a distributed environment the metadata (i.e., data dictionaries, documentation) can be available, incomplete or unavailable. Table 4.1 shows the attributes of the entities in the relational schema.

Entity Name	Attribute
Office Code	Officecode, City, phone, addressLine1, AddressLine2, State, Country, PostalCode, Territory
Employee	EmployeeNumber, Lastname, firstname, Extension, email, officecode, repostto, Jobtitle
Orders	Orderenumber, orderDate, RequiredDate, Shippeddate, Status, Comments, CustomerNumber, EmployeeNumber
CustomerNumber	CustomerNumber, CustomerLastName, CustomerFirstName, Phone, addressLine1, AddressLine2,City
ProductLine	ProductLine, TextDescription,HtmlDescription, Image, ProductCode
Product	ProductCode, Productsale, ProductVendor, ProductDescription, Quantityinstock, buyprice, ordernumber
Orderdetails	Ordernumber, quantityordered, priceeach, orderedLineNumber
Payments	CustomerNumber, checknumber, parmentdate, amount

Table 4.1 Entities in the Selected Schema

An important thing is the association in relational schema. Foreign keys in the relational schema are shown in Table 4.2. “orderNumber” from the “order” table, “OrderNumber” from the “order” table, “CustomerNumber” from the “customer” table, “Productcode” from the “product” table , “Officecode” from “office” table, “CustomerNumber” from the “customer” table and “employee Number” from “Employee table”.

Relations	Foreign Keys	Number of Relationships
Office (<u>Officecode</u>, City, phone, addressLine1, AddressLine2, State, Country, PostalCode, Territory)	Nil	0
Employee (<u>EmployeeNumber</u>, Lastname, firstname, Extension, email, officecode, repostto, Jobtitle)	Officecode from office table	1
Orders (<u>Orderenumber</u>, orderDate, RequiredDate, Shippeddate, Status, Comments, CustomerNumber, EmployeeNumber)	The CustomerNumber from the customer table and employee Number from Employee table	2
Customer (<u>CustomerNumber</u>, CustomerLastName, CustomerFirstName, Phone, addressLine1, AddressLine2,City)	Nil	0
ProductLine(<u>ProductLine</u>, TextDescription,HtmlDescription, Image, ProductCode)	Productcode from the product table	1
Product(<u>ProductCode</u>, Productsale, ProductVendor, ProductDescription, Quantityinstock, buyprice, ordernumber)	orderNumber from the order table	1
Orderdetails(<u>Ordernumber</u>, quantityordered, priceeach, orderedLineNumber)	OrderNumber from the order table.	1
Payments(<u>CustomerNumber,</u> <u>checknumber</u>, parmentdate, amount)	The customerNumber from the customer table	1

Table 4.2 Associations Details in the Relational Schema

4.8. Summary

An algorithm is being proposed for converting relational schema to ontology. There some techniques that are already presented, but they work either in absence of meta data or in presence of meta data. This technique is applicable in real scenarios where some meta data may be unavailable currently focus of this work was associations in a relational schema and MySQL DBMS is used. As MySQL is largely used with the website since it was developed. In this chapter the architecture of proposed scheme and proposed algorithm to discover the association is presented.

Chapter 5

IMPLEMENTATION

5.1. Database

The database was created in MySQL, which is light weight Database Management System. It was especially developed for web. It was written in C and C++. It was tested with a broad range of different compilers. It can work on many different platforms. It uses GNU Automake, Autoconf, and Libtool for portability. It was Tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool. It uses multi-layered server design with independent modules, designed to be fully multi-threaded using kernel threads, to easily use multiple CPUs if they are available. It provides transactional and nontransactional storage engines. It uses very fast B-tree disk tables (MyISAM) with index compression. It was designed to make it relatively easier to add other storage engines. This is useful if you want to provide an SQL interface for an in-house database. it implements in-memory hash tables, which are used as temporary tables, and SQL functions using a highly optimized class library that should be as fast as possible. Usually there is no memory allocation at all after query initialization. It provides the server as a separate program for use in a client/server networked environment, and as a library that can be embedded (linked) into standalone applications. Such applications can be used in isolation or in environments where no network is available.

5.1.1. Data Types

Many data types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long, FLOAT, DOUBLE, CHAR, VARCHAR, BINARY, VARBINARY, TEXT, BLOB, DATE, TIME, DATETIME, TIMESTAMP, YEAR, SET, ENUM, and OpenGIS spatial types. Fixed-length and variable-length string types.

5.1.2. Security

A privilege and password system that is very flexible and secure, and that allows host-based verification. It provides password security by encryption of all password traffic when you connect to a server.

5.1.3. Scalability and Limits

Support for large databases. MySQL Server was used with databases that contain 50 million records. It support for up to 64 indexes per table (32 before MySQL 4.1.2). Each index may consist of 1 to 16 columns or parts of columns. The maximum index width is 1000 bytes (767 for InnoDB); before MySQL 4.1.2, the limit is 500 bytes. An index may use a prefix of a column for CHAR, VARCHAR, BLOB, or TEXT column types.

5.1.4. Connectivity

Clients can connect to MySQL Server using several protocols, Clients can connect using TCP/IP sockets on any platform. On Windows systems in the NT family (NT, 2000, XP, 2003, or Vista), clients can connect using named pipes if the server is started with the --enable-named-pipe option. In MySQL 4.1 and higher, Windows servers also support shared-memory connections if started with the --shared-memory option. On Unix systems, clients can connect using Unix domain socket files. MySQL client programs can be written in many languages. A client library written in C is available for clients written in C or C++, or for any language that provides C bindings. APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available, allowing MySQL clients to be written in many languages. The Connector/ODBC (MyODBC) interface provides MySQL support for client programs that use ODBC (Open Database Connectivity) connections. For example, you can use MS Access to connect to your MySQL server. Clients can be run on Windows or Unix. MyODBC source is available. All ODBC 2.5 functions are supported, as are many others. The Connector/J interface provides MySQL support for Java client programs that use JDBC connections.

5.2. Example Database Implementation

The localhost server snapshot in Figure 5.1 shows the relational schema, which has been used for the prototype testing of the proposed scheme. There are eight tables shown i.e. customers, employees, offices, orderdetails, orders, payments, productlines and product.

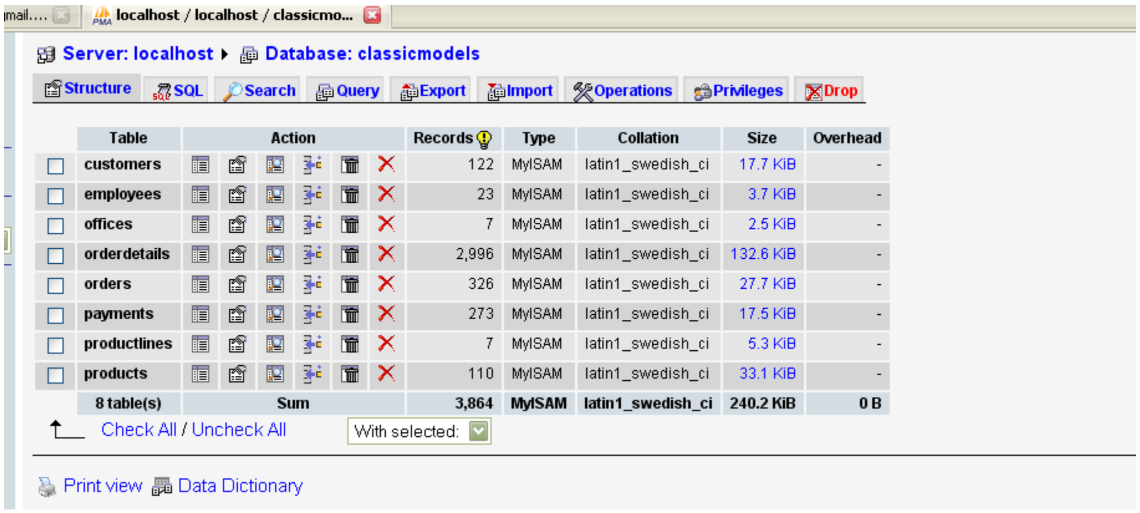
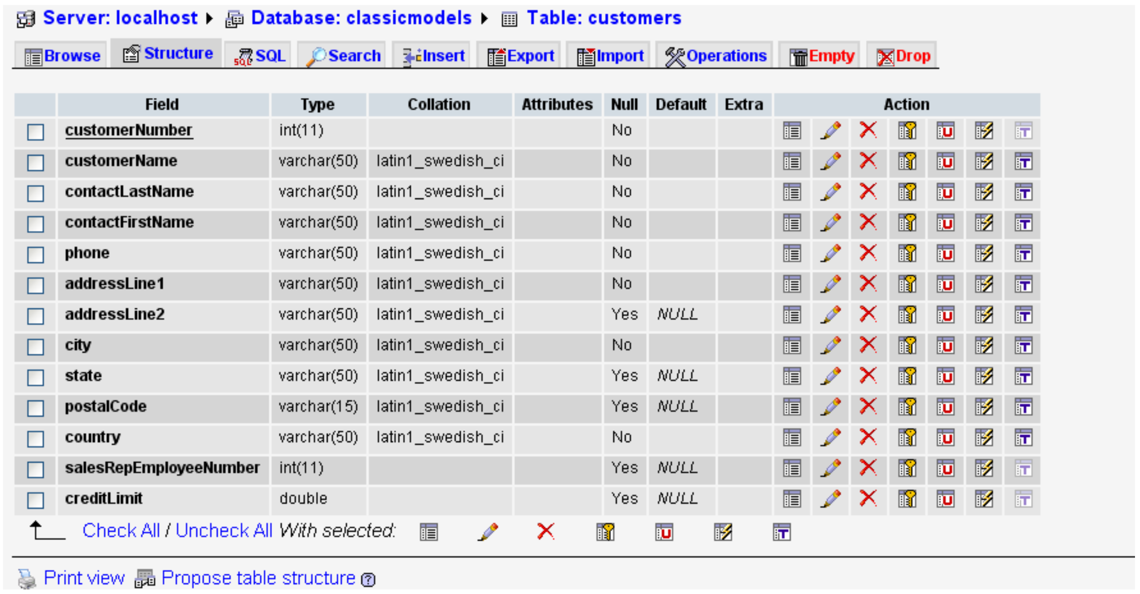


Table	Action	Records	Type	Collation	Size	Overhead
customers		122	MyISAM	latin1_swedish_ci	17.7 KiB	-
employees		23	MyISAM	latin1_swedish_ci	3.7 KiB	-
offices		7	MyISAM	latin1_swedish_ci	2.5 KiB	-
orderdetails		2,996	MyISAM	latin1_swedish_ci	132.6 KiB	-
orders		326	MyISAM	latin1_swedish_ci	27.7 KiB	-
payments		273	MyISAM	latin1_swedish_ci	17.5 KiB	-
productlines		7	MyISAM	latin1_swedish_ci	5.3 KiB	-
products		110	MyISAM	latin1_swedish_ci	33.1 KiB	-
8 table(s)	Sum	3,864	MyISAM	latin1_swedish_ci	240.2 KiB	0 B

Figure 5.1 Example Schema

5.2.1. Customers Table

The structure of customer's table is described in the Figure 5.2. The customer table has "customernumber" as primary key. The other attributes are customername, contactfirstname, contactlastname, phone, addressline1, addressline2, city, state, postalcode, country, salesrepemployeenumber and creditlimit.



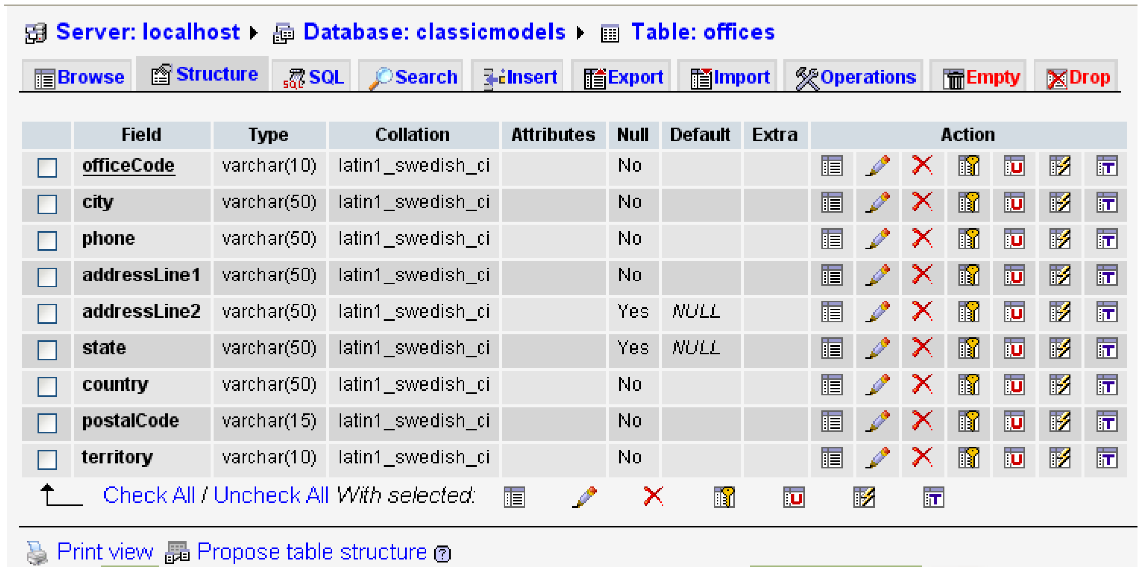
Server: localhost ▸ Database: classicmodels ▸ Table: customers
Browse
Structure
SQL
Search
Insert
Export
Import
Operations
Empty
Drop

	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	customerNumber	int(11)			No			
☐	customerName	varchar(50)	latin1_swedish_ci		No			
☐	contactLastName	varchar(50)	latin1_swedish_ci		No			
☐	contactFirstName	varchar(50)	latin1_swedish_ci		No			
☐	phone	varchar(50)	latin1_swedish_ci		No			
☐	addressLine1	varchar(50)	latin1_swedish_ci		No			
☐	addressLine2	varchar(50)	latin1_swedish_ci		Yes	*NULL*		
☐	city	varchar(50)	latin1_swedish_ci		No			
☐	state	varchar(50)	latin1_swedish_ci		Yes	*NULL*		
☐	postalCode	varchar(15)	latin1_swedish_ci		Yes	*NULL*		
☐	country	varchar(50)	latin1_swedish_ci		No			
☐	salesRepEmployeeNumber	int(11)			Yes	*NULL*		
☐	creditLimit	double			Yes	*NULL*		

Figure 5.2 Customer Table

5.2.1. Offices Table

The structures of "offices" table with all the data types is presented in Figure 5.3. Here "officecode" is the primary key, other attributes are city, phone, addressline1, addressline22, state, country, postalcode and territory.

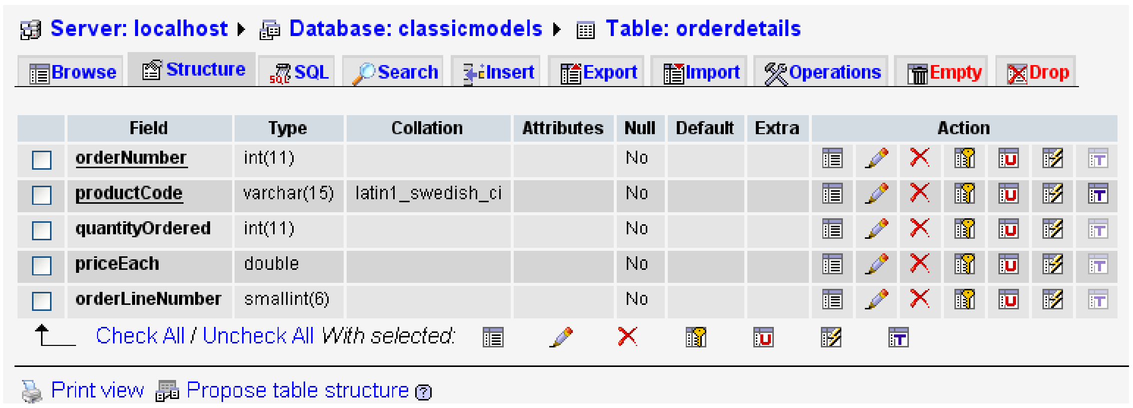


	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	officeCode	varchar(10)	latin1_swedish_ci		No			
☐	city	varchar(50)	latin1_swedish_ci		No			
☐	phone	varchar(50)	latin1_swedish_ci		No			
☐	addressLine1	varchar(50)	latin1_swedish_ci		No			
☐	addressLine2	varchar(50)	latin1_swedish_ci		Yes	*NULL*		
☐	state	varchar(50)	latin1_swedish_ci		Yes	*NULL*		
☐	country	varchar(50)	latin1_swedish_ci		No			
☐	postalCode	varchar(15)	latin1_swedish_ci		No			
☐	territory	varchar(10)	latin1_swedish_ci		No			

Figure 5.3 Offices Table

5.2.1. Orders Details Table

The structure of "Orders Details Table" with all the data types is shown in Figure 5.4. here "OrderNumber" and "ProductCode" are the primary key.

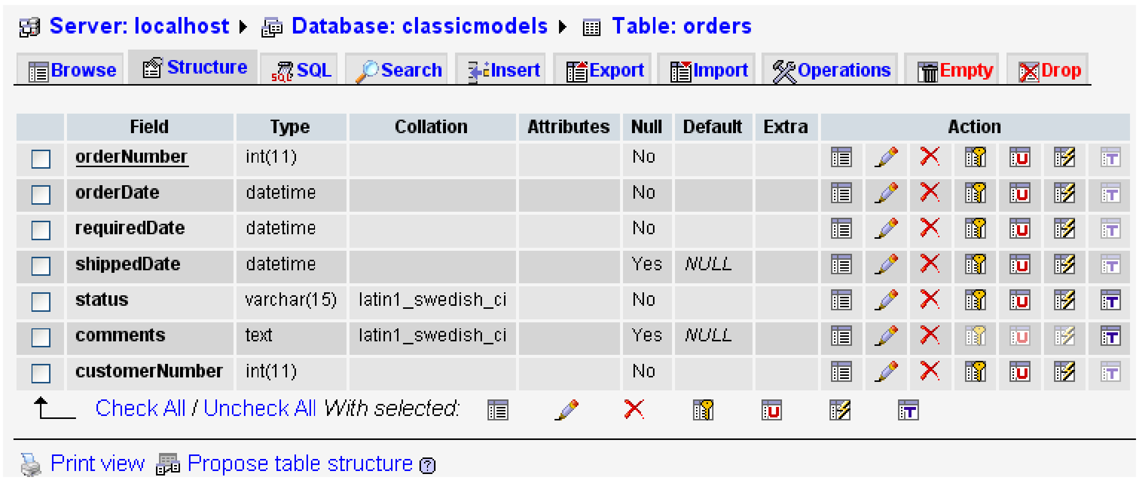

Server: localhost ▸ Database: classicmodels ▸ Table: orderdetails

	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	orderNumber	int(11)			No			
☐	productCode	varchar(15)	latin1_swedish_ci		No			
☐	quantityOrdered	int(11)			No			
☐	priceEach	double			No			
☐	orderLineNumber	smallint(6)			No			

Figure 5.4 Orders Details Table

5.2.1. Orders Table

The structure of orders table with all the data types are presented in the Figure 5.5. Here "OrderNumber" is primary key, whereas "orderdate", "requireddate", "shippeddate", "status", "comments", "customernumber" are other attributes.

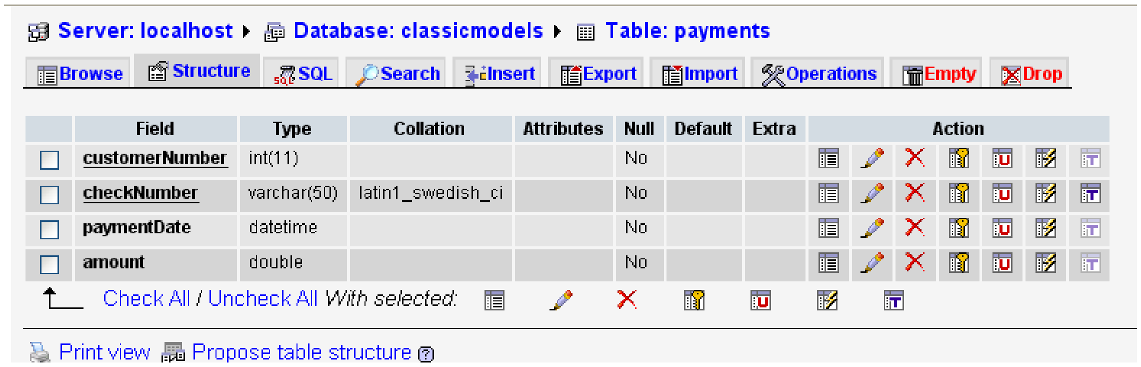

Server: localhost ▸ Database: classicmodels ▸ Table: orders

	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	orderNumber	int(11)			No			
☐	orderDate	datetime			No			
☐	requiredDate	datetime			No			
☐	shippedDate	datetime			Yes	*NULL*		
☐	status	varchar(15)	latin1_swedish_ci		No			
☐	comments	text	latin1_swedish_ci		Yes	*NULL*		
☐	customerNumber	int(11)			No			

Figure 5.5 Orders Table

5.2.1. Payments Table

The structure of payments table with all the data types is shown in Figure 5.6. Here “CustomerNumber” and “checknumber” are the composite primary keys.

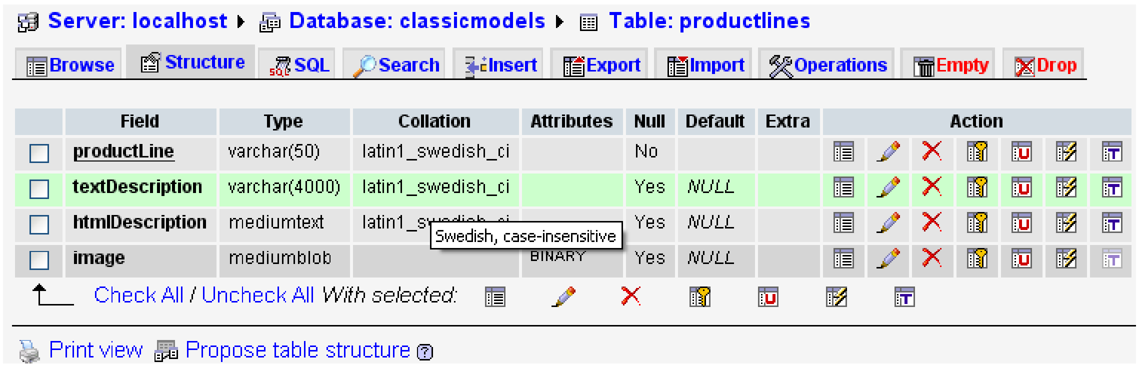


Field	Type	Collation	Attributes	Null	Default	Extra
customerNumber	int(11)			No		
checkNumber	varchar(50)	latin1_swedish_ci		No		
paymentDate	datetime			No		
amount	double			No		

Figure 5.6 Payments Table

5.2.1. Product line Table

The structure of product Line table with all the data types are presented in the Figure 5.7.

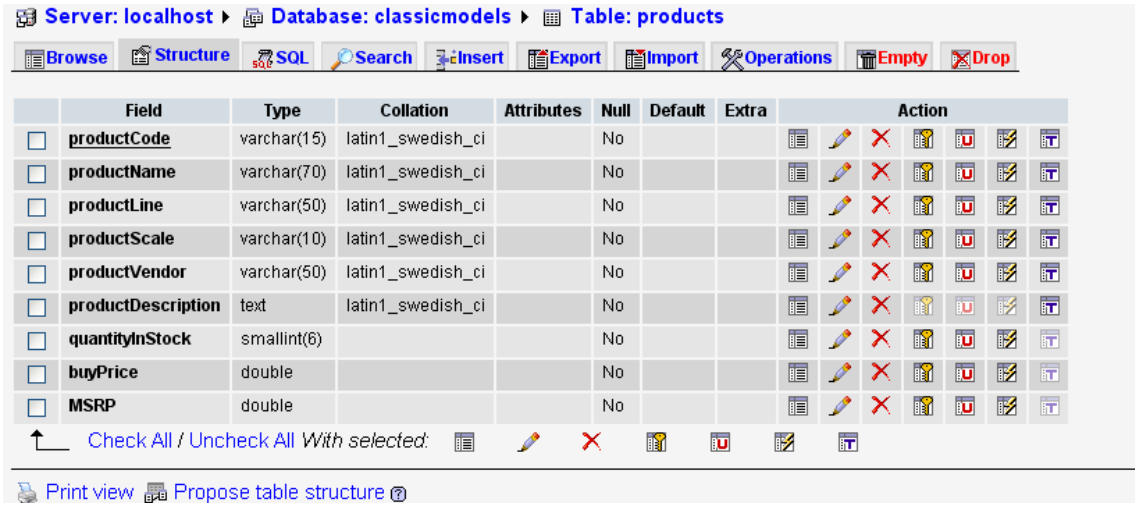


Field	Type	Collation	Attributes	Null	Default	Extra
productLine	varchar(50)	latin1_swedish_ci		No		
textDescription	varchar(4000)	latin1_swedish_ci		Yes	NULL	
htmlDescription	mediumtext	latin1_swedish_ci		Yes	NULL	
image	mediumblob		BINARY	Yes	NULL	

Figure 5.7 Product Line Table

5.2.1. Product Table

The structure of product table with all the data types is shown in Figure 5.8. “productcode” is the primary key.



Field	Type	Collation	Attributes	Null	Default	Extra
productCode	varchar(15)	latin1_swedish_ci		No		
productName	varchar(70)	latin1_swedish_ci		No		
productLine	varchar(50)	latin1_swedish_ci		No		
productScale	varchar(10)	latin1_swedish_ci		No		
productVendor	varchar(50)	latin1_swedish_ci		No		
productDescription	text	latin1_swedish_ci		No		
quantityInStock	smallint(6)			No		
buyPrice	double			No		
MSRP	double			No		

Figure 5.8 Product Table

5.3. User Interfaces

Figure 5.10 shows the application’s user interface. To create an environment, the user will first select the show tables and save table buttons. Then fields can be saved. Afterwards the comparison can be made.

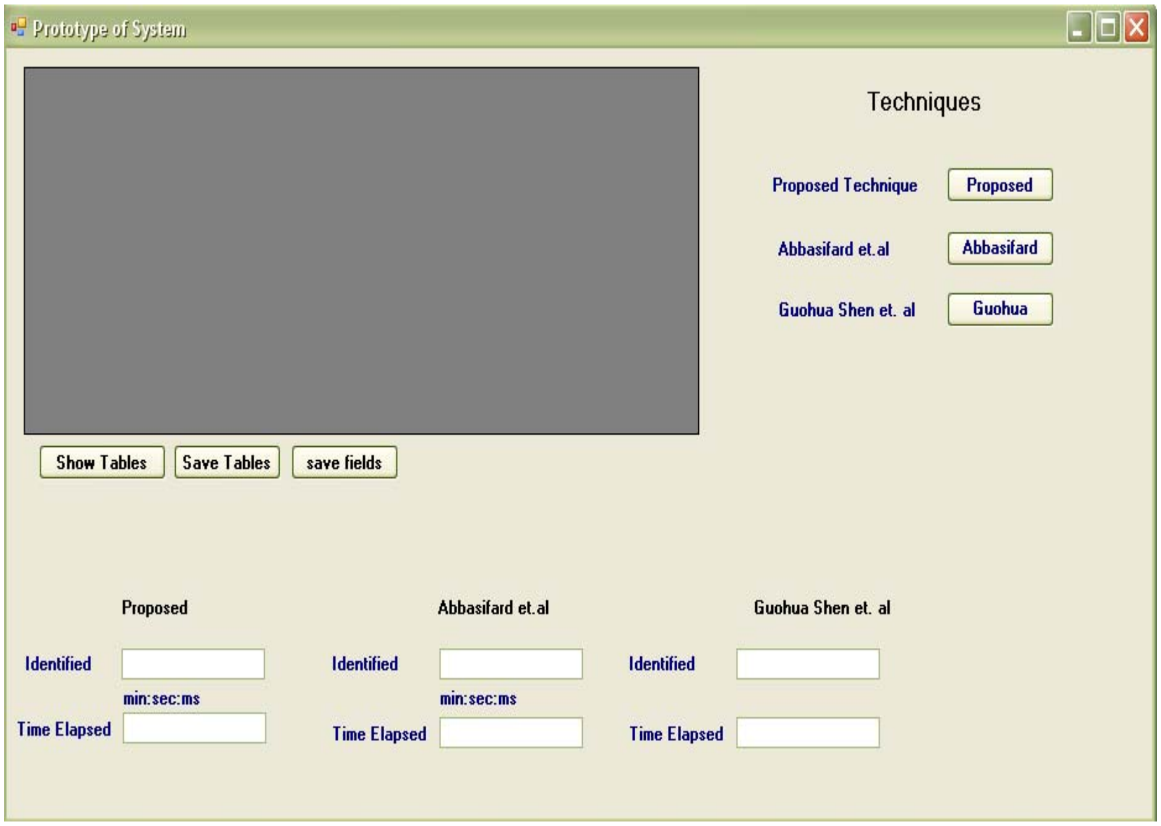


Figure 5.10 Main User Interface

5.4. Summary

In this chapter, discussion on the implementation has been made. The implementation of the algorithm and database has been illustrated with the help of the Figures.

Chapter 6

Evaluation and Results

In this chapter the results and evaluations of the proposed scheme is discussed in detail.

6.1. Criteria for evaluations

In the proposed system the correctness of identification of association is of main concern. The evaluation criteria of the proposed scheme are

(a) System performance: This criterion determines the efficiency (in terms of time) of the proposed scheme. It is required to find whether the system takes greater or lesser time than the existing systems.

(b) Preservation of information capacity: This is to finds out whether the association in relational schema were completely identified or not. It is important to find information loss (if any) as a result of transformation.

(c) Correct identification: This criterion evaluates the correctness of the transformation. The evaluation of the research is done through the experimental results. The Table 6.1 shows system specification and software requirements

System Requirement	
RAM	1GB
Hard Disk	80
Processor	1.37 Dual Core
Software Requirement	
Operating System	Windows XP
Database	MySQL
Programming Language	Java, C#

Table 6.1 System Specifications

6.2. Results and Evaluations

As discussed in the chapter 4, selection of an inventory database of an organization was chosen. Table 6.2 describes the relational schema. There are seven associations in the existing database.

Relation	Attribute	Foreign Keys	Association
Office	Officecode, City, phone, addressLine1, AddressLine2, State, Country, PostalCode, Territory	Nil	0
Employee	EmployeeNumber, Lastname, firstname, Extension, email, officecode, repostto, Jobtitle	Officecode from office Table	1
Orders	Orderenumber, orderDate, RequiredDate, Shippeddate, Status, Comments, CustomerNumber, EmployeeNumber	The CustomerNumber from the customer Table and employee Number from Employee Table	2
Customer	CustomerNumber, CustomerLastName, CustomerFirstName, Phone, addressLine1, AddressLine2,City	Nil	0
ProductLine	ProductLine, TextDescription, HtmlDescription, Image, ProductCode	Productcode from the product Table	1
Product	ProductCode, Productsale, ProductVendor, ProductDescription, Quantityinstock, buyprice, ordernumber	orderNumber from the order Table	1
Orderdetails	Ordernumber, quantityordered, priceeach, orderedLineNumber	OrderNumber from the order Table.	1
Payments	CustomerNumber, checknumber, parmentdate, amount	The customerNumber from the customer Table	1

Table 6.2 Details of the Relational Schema

The proposed algorithm-1 given in the appendix-A is used to make the comparisons. To extract the foreign key, above described relational schema was taken. The gathered information was passed to proposed algorithm and results were taken out by a threshold value working as a filter, remembering the fact that higher the threshold value, higher is the accuracy of the algorithm.

The Figure 6.1 shows the comparison values of the primary key with other attribute.

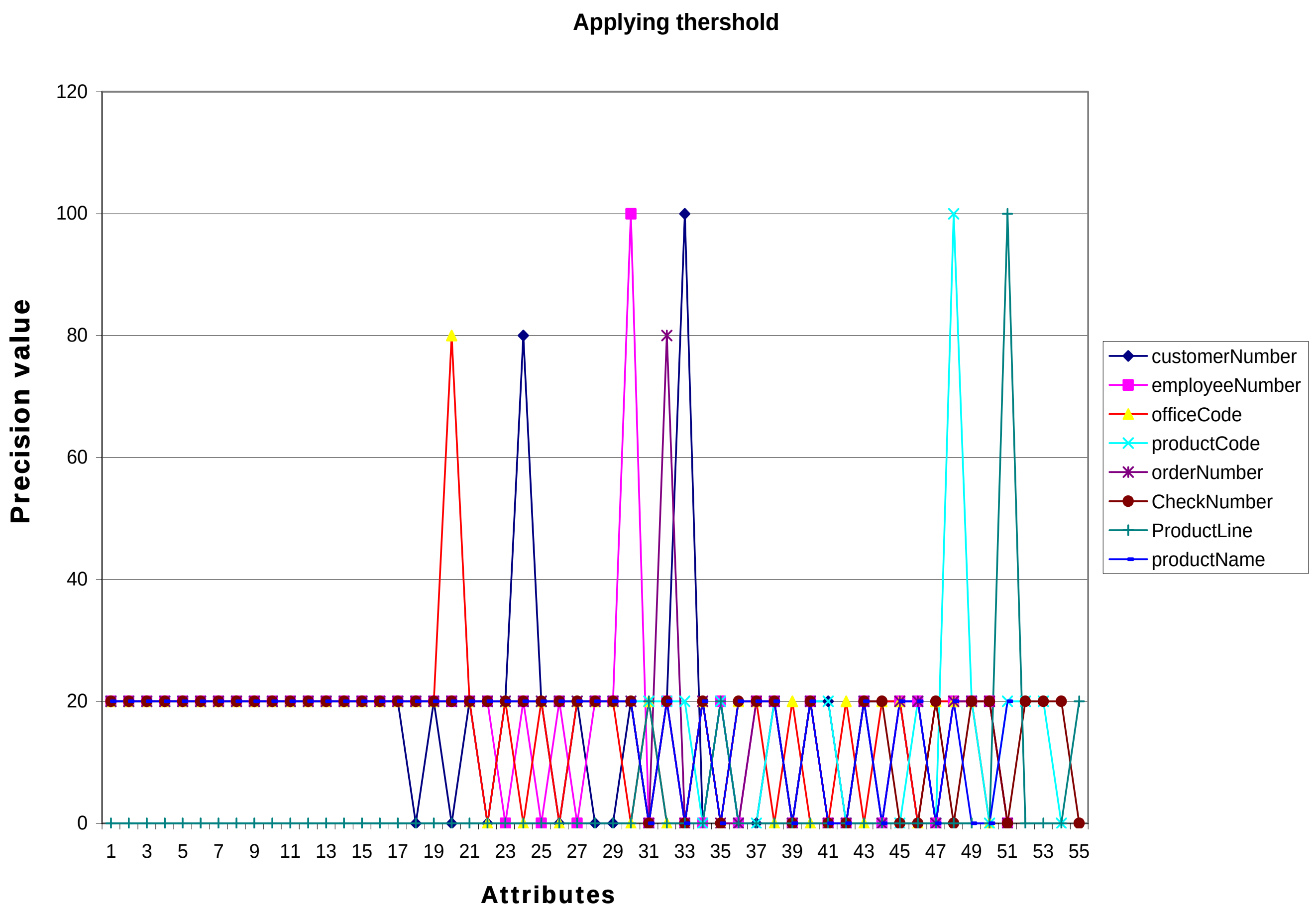


Figure 6.1 Comparison Detailed Chart for Schema-1

The schema-2 in annexure-1 is selected. It contains 28469 attributes. When the proposed algorithm is applied on the schema-2 one following is the results are shown in Figure 6.2.

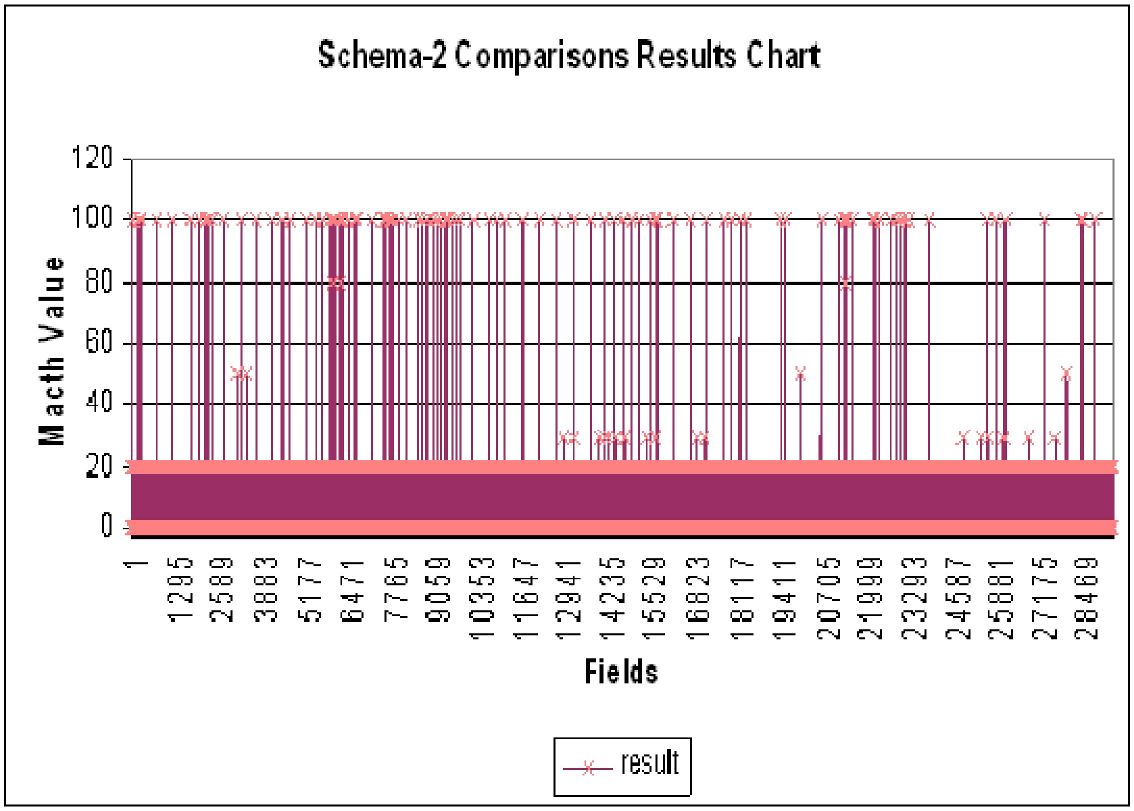


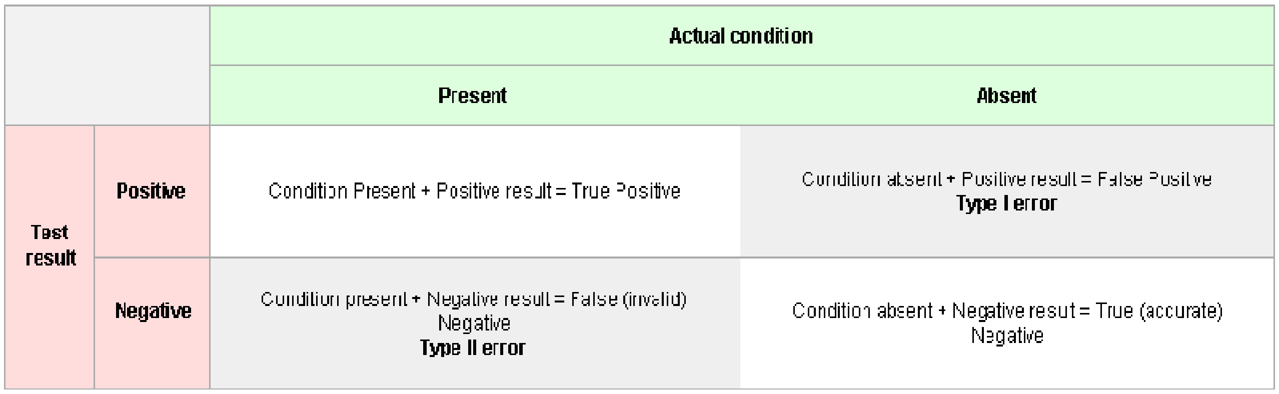
Figure 6.2 Comparison Detailed Chart for Schema-2

6.3. Type-I and type-II errors

Type I error, also known as an "error of the first kind", and α error, or a "false positive": the error of rejecting a null hypothesis when it is actually true. Plainly speaking, it occurs while observing a difference when in truth there is none, thus indicating a test of poor specificity. An example of this would be if a test shows that a woman is pregnant when in reality she is not. Type I error can be viewed as the error of excessive credulity.

Type II error also known as an "error of the second kind", a β error, or a "false negative": the error of failing to reject a null hypothesis when it is in fact not true. In other words, this is the error of failing to observe a difference when in truth there is one, thus indicating a test of poor sensitivity. An example of this would be if a test shows that a woman is not pregnant, when in reality, she is. Type II error can be viewed as the error of excessive skepticism.

These examples illustrate the ambiguity, which is one of the dangers of this wider use: They assume the speaker is testing for guilt; they could also be used in reverse, as testing for innocence; or two tests could be involved, one for guilt, and the other for innocence. The Tables 6.3 and Table 6.4 show the conditions.



		Actual condition	
		Present	Absent
Test result	Positive	Condition Present + Positive result = True Positive	Condition absent + Positive result = False Positive **Type I error**
	Negative	Condition present + Negative result = False (invalid) Negative **Type II error**	Condition absent + Negative result = True (accurate) Negative

Table 6.1 Type-I & Type-II Errors

Example, testing for guilty/not-guilty:

		Actual condition	
		Guilty	**Not guilty**
Test result	**Verdict of "guilty"**	True Positive	False Positive (i.e. guilt reported unfairly)
	Verdict of "not guilty"	False Negative (i.e. guilt not detected)	True Negative

Table 6.4 Example Type-I & Type-II Errors

- Rejecting a null-hypothesis when it should not have been rejected creates a type I error.
- Failing to reject a null-hypothesis when it should have been rejected creates a type II error.
- (In either case, a wrong decision or error in judgment has occurred.)
- Decision rules (or tests of hypotheses), in order to be good, must be designed to minimize errors of decision.

- Minimizing errors of decision is not a simple issue for any given sample size the effort to reduce one type of error generally results in increasing the other type of error.
- Based on the real-life application of the error, one type may be more serious than the other.
- (In such cases, a compromise should be reached in favor of limiting the more serious type of error.)
- The only way to minimize both types of error is to increase the sample size, and this may or may not be feasible

6.4. Weight W1, W2 and W3

There are three weight used in algorithm w1, w2 and w3. The selection of weight can affect the net result for example weight-1 is applied by selecting it different values here is the Table that will show the change in result.

Identified	**W1**	**W2**	**W3**	**FP**	**FN**
28400	**0**	0.5	0.5	YES	0
24678	**0.2**	0.4	0.4	YES	0
16034	**0.3**	0.4	0.3	YES	0
5261	**0.4**	0.3	0.3	YES	0
5255	**0.5**	0.3	0.2	YES	0
29	**0.6**	**0.3**	**0.2**	**0**	**0**
29	**0.7**	**0.2**	**0.1**	**0**	**0**
29	**0.8**	**0.1**	**0.1**	**0**	**0**
27	**0.9**	0.1	0	0	YES
27	**1**	0	0	0	YES

Table 6.5 W1 Values Selection Table

The Figure 6.3 shows the graphical representation of the Table 6.5. in graph it is evident that values of w1 below 0.6 will add false positive to the results and values of w1 greater then 0.8 will increase false negative results. Whereas, if the values of w1 are kept in between 0.6 to 0.8 then there is no false positive and no false negative in the result.

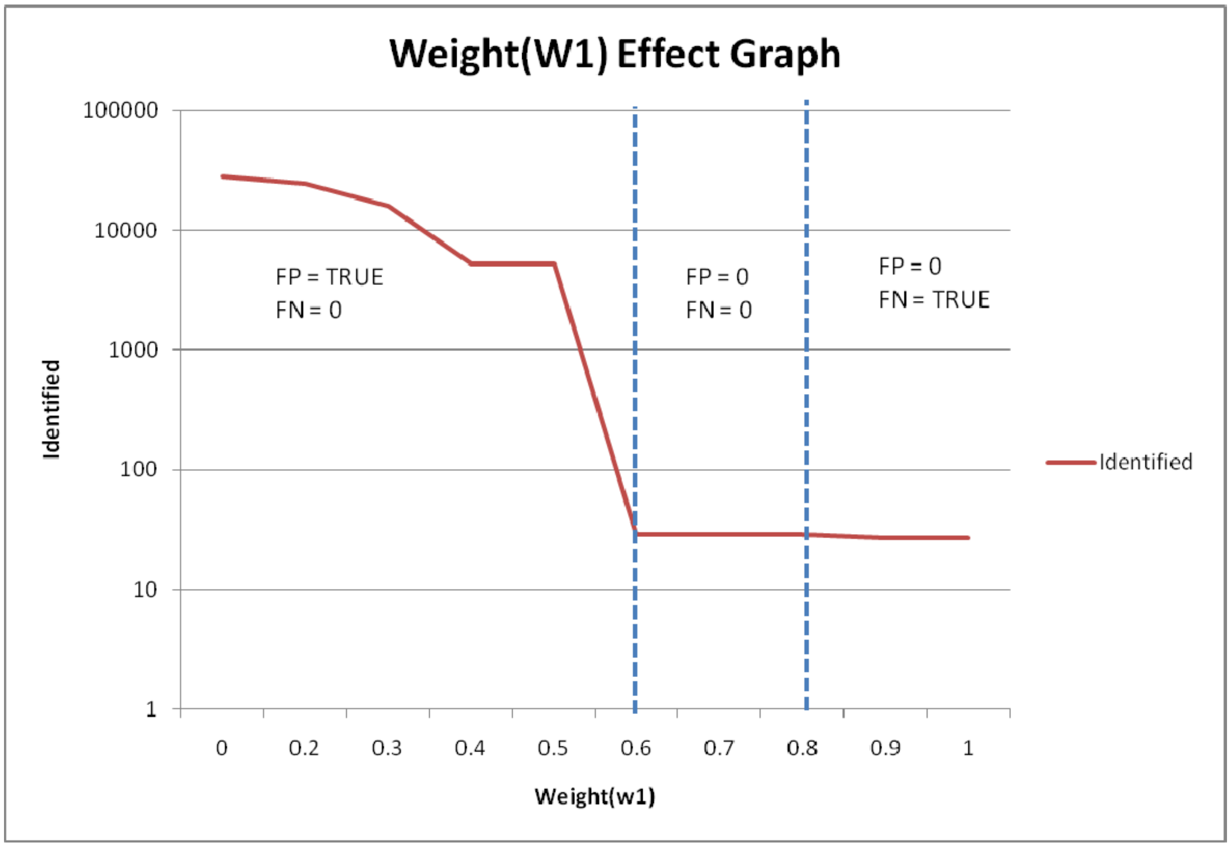


Figure 6.3 W1 Values Selection Graph

The Table 6.6 shows different values selected for w2. The left column shows the identified values and on the right most columns false positive and false negative are shown that were present in the result for the W2 values selected.

Identified	W1	W2	W3	FP	FN
30	0.5	**0**	0.5	YES	0
29	**0.4**	**0.1**	**0.4**	**0**	**0**
30	0.4	**0.2**	0.3	YES	0
30	0.3	**0.3**	0.3	YES	0
30	0.3	**0.4**	0.2	YES	0
5254	0.3	**0.5**	0.2	YES	0
5254	0.2	**0.6**	0.1	YES	0
5254	0.1	**0.7**	0.1	YES	0
5254	0.1	**0.8**	0	YES	0
5254	0	**0.9**	0.1	YES	0
5254	0	**1**	0	YES	0

Table 6.6 W2 Values Selection Table

The Figure 6.4 shows the graphical representation of the Table 6.6. In graph it is evident that values of w1 below 0.2 will add false positive to the results and values of w1 greater then 0.3 will increase false negative results. Whereas, if the values of w1 are kept in between 0.2 to 0.3 then there is no false positive and no false negative in the result.

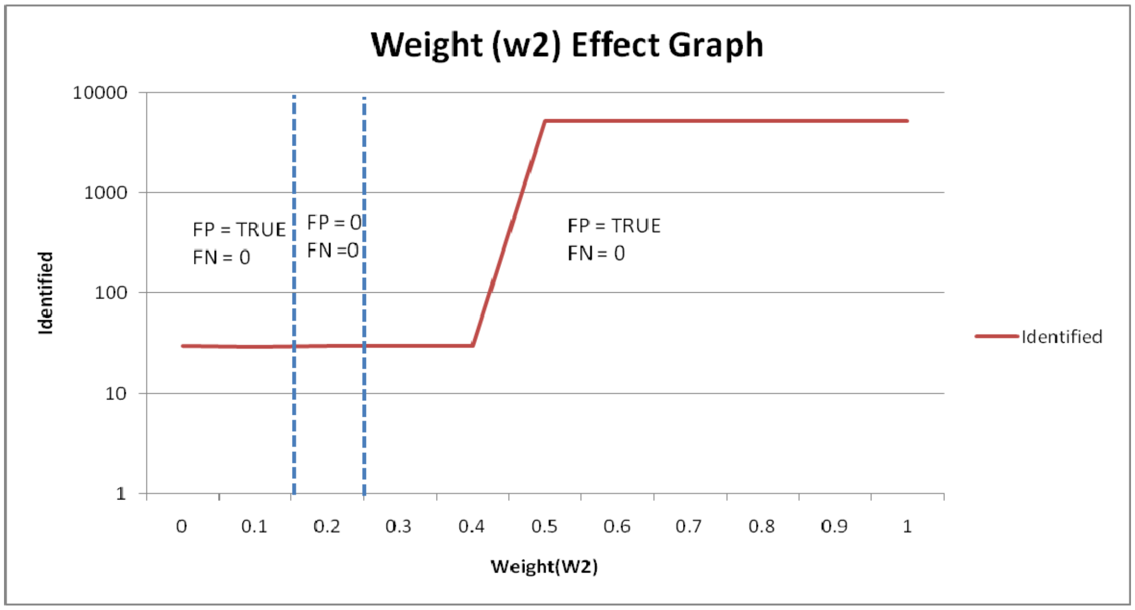


Figure 6.4 W2 Values Selection Graph

The Table 6.7 shows different values selected for w2 shown in graph in Figure 6.4. The left column shows the identified values and on the right most columns false positive and false negative are shown that were present in the result for the W2 values selected.
The graphical representation of the Table 6.7 is shown in the Figure 6.5. In graph it is evident that values of w1 below 0.6 will add false positive to the results and values of w1 greater then 0.9 will increase false negative results. Whereas, if the values of w1 are kept in between 0.6 to 0.9 then there is no false positive and no false negative in the result.

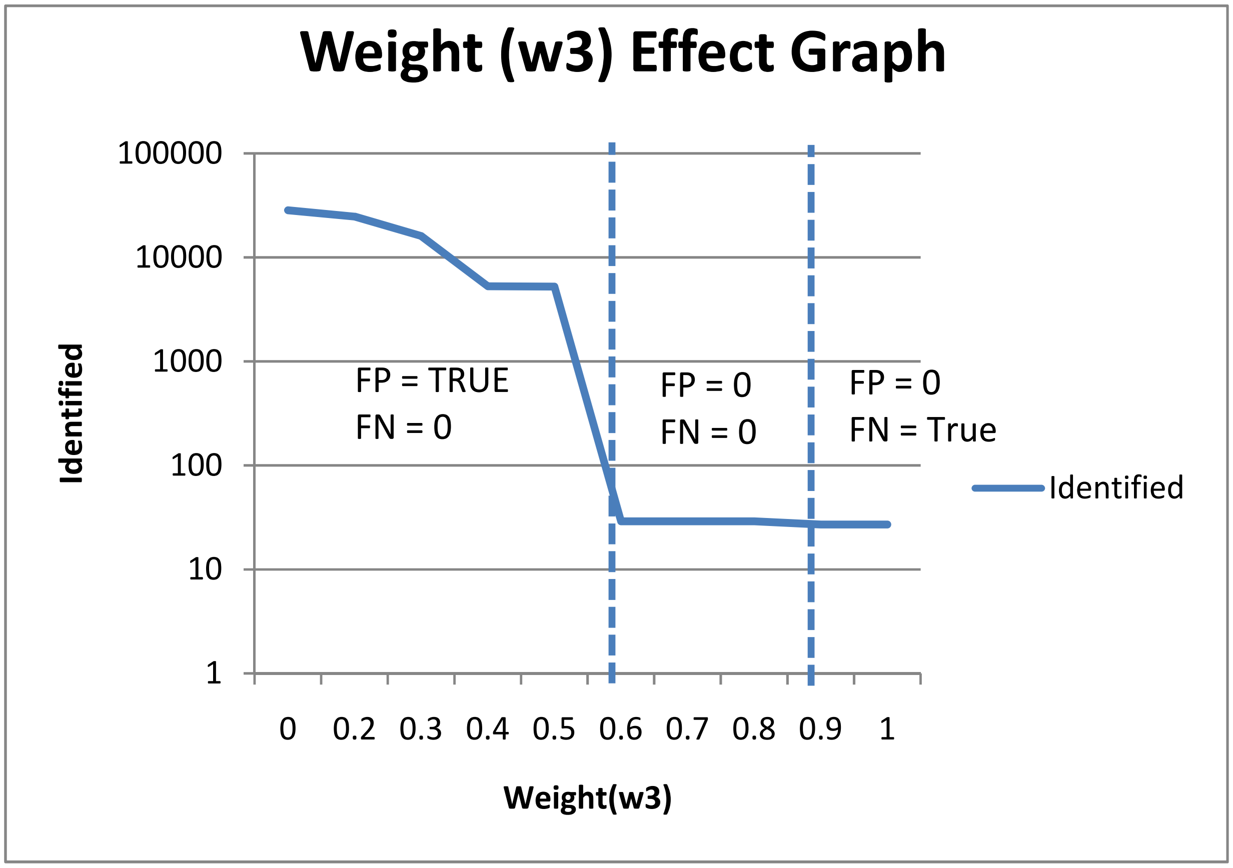


Figure 6.5 W3 Values Selection Graph

Identified	W3	W2	W1	FP	FN
28400	**0**	0.5	0.5	YES	0
24678	**0.2**	0.4	0.4	YES	0
16034	**0.3**	0.4	0.3	YES	0
5261	**0.4**	0.3	0.3	YES	0
5255	**0.5**	0.3	0.2	YES	0
29	**0.6**	**0.3**	**0.2**	**0**	**0**
29	**0.7**	**0.2**	**0.1**	**0**	**0**
29	**0.8**	**0.1**	**0.1**	**0**	**0**
27	**0.9**	**0.1**	**0**	**0**	**0**
27	**1**	0	0	0	YES

Table 6.7 W3 Values Selection Table

The three weights can be adjusted better after considering the graph 6.6.

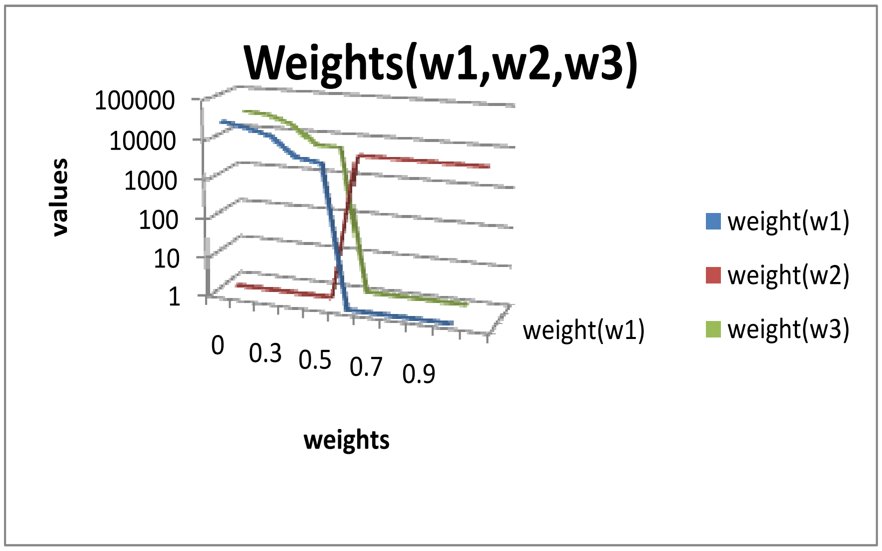


Figure 6.6 Weight (W1,W2 and W3) Selection Graph

6.4. Comparison with other schemes

The proposed scheme is compared with abbasifard et al and Guohua Shen et al schemes. Both techniques that are considered for the comparison are selected because of their addressing to same problem to which proposed solution refers.

The precision and recall on the selected schema is shown in the following Table 6.8.

Relations	Proposed Technique		Abbasifard et.al		Guohua Shen et. al	
	Precision	Recall	Precision	Recall	Precision	Recall
29	1	1	1	1	0.96	0.96

Table 6.8 Precision and Recall

The graph shown in Figure 6.7 will help to understand the precision and recall

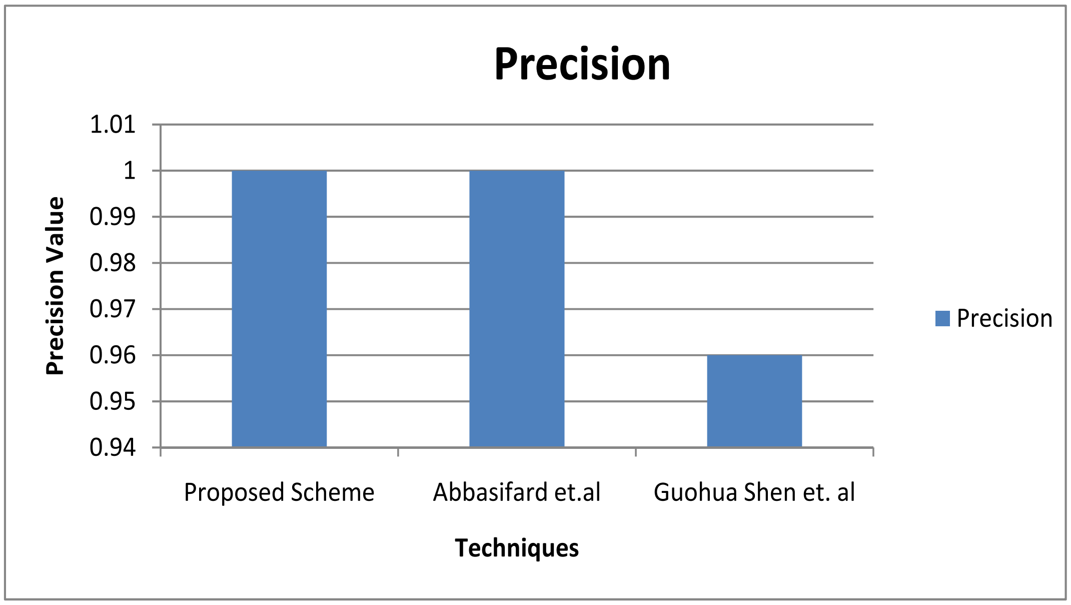


Figure 6.7 Precision Graph

Similarly the recall of the proposed system with other system are shown in Figure 6.8

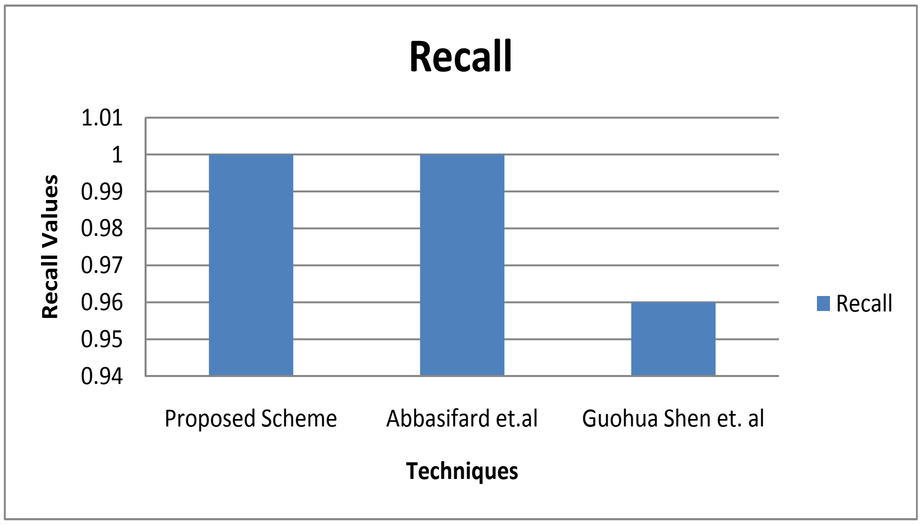


Figure 6.8 Recall Graph

The time comparison results graph is clearly presenting the efficiency of the proposed technique with the other techniques

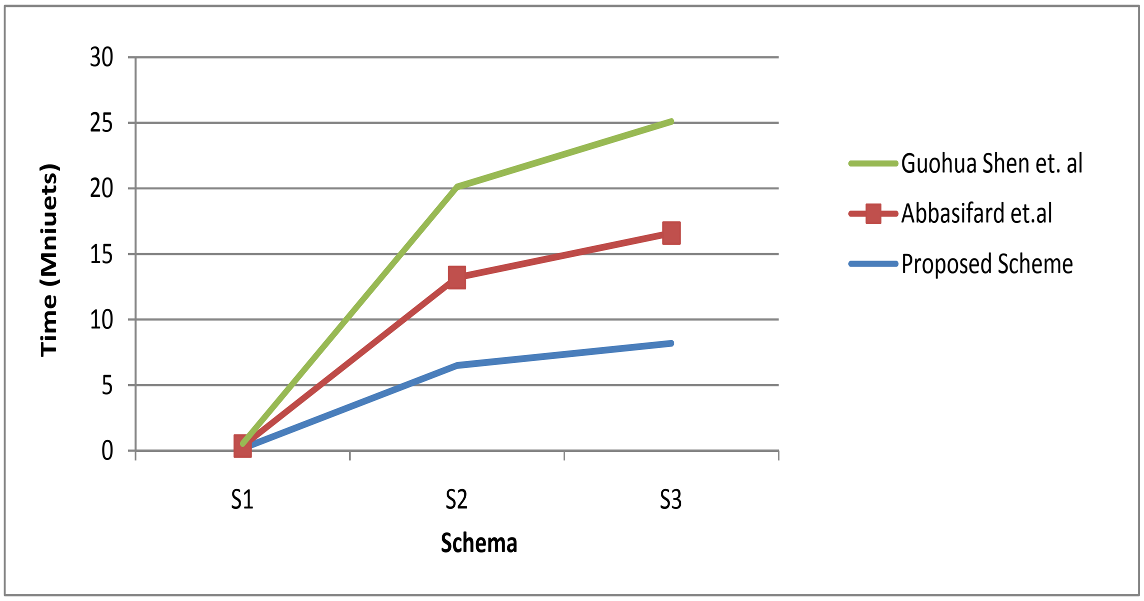


Figure 6.9 Time Comparison Chart

6.5. Precision of the algorithm

The precision of the scheme is highly dependent on the selection of threshold value. If the value is taken wrong it will result in wrong results, and vice versa. Different Graphs have been plotted in this chapter to describe it importance. The precision and recall is calculated by the following formula [26].

$$PRECISION = \frac{Tp}{Tp + Fp} \qquad \text{---------------(6.1)}$$

$$RECALL = \frac{Tp}{Tp + Fn} \qquad \text{------------------(6.2)}$$

Where Tp are true positive, Fp is false positive and Fn are false negative relationships. True positive are those relationships which the technique identifies correctly. Fp is those relationships which are identified incorrectly and false negative are relationships which exists in the schema but are not identified by the technique. The recall of the proposed technique and [26] are same i.e., 0 and 1. Table 4.9 describes the precision and recall of the system.

Methods	Total Relationships	Proposed Scheme		DataMaster	
		Precision	Recall	Precision	Recall
Without MetaData	7	1	1	0	0
With Metadata	7	1	1	1	1
Without Metadata	54	0.9	0.9	0	0
With Metadata	54	0.9	0.9	0.83	0.83
Total	61	0.95	0.95	0.91	0.91

Table 6.9 Precision and Recall

Summary

In this chapter evaluation criterion to evaluate the proposed methodology was discussed. The evaluation shows that the proposed system has optimal system performance and identification of association is accurate. Consequently, the transformation is information capacity preservation. The provision of information capacity preservation and operational goal implied that the transformation is correct. In the next chapter, conclusion the research work will be presented and it will also provide a future direction to extend research in this direction.

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1. Discussion

The semantic web is an extension of the current web. It is gaining the interests of researcher from few years. The aim of this web is to make web meaningful, understandable and machine processable. Making use of information on current web for the productiveness of future web is becoming vital. Heterogeneity of relational data coupled with present web complicates utilization of information for future web. To use the data associated with current web we need to transform it into ontology.

In the process of creation of ontologies for the database, three techniques are used: analysis of database schema, tuples or analysis of user queries [2]. From different schemes of transformation of relational schemas to ontology, the following points are observed: the database tables are mapped into the classes (i.e. ontologies, attribute of the tables are mapped as the attributes of the ontologies) and identifications of foreign keys in a database schema. However, the types of the relationship can be identified by the entries in the database.

The ontology is one of the pillars of the semantic web. Different tools for ontology creations and management are developed to implement the concept of ontology like protégé, OntoGen etc. Protégé is an ontology editor and knowledge based frame work [3] whereas OntoGen is said to be a semi automated and data driven tool that combines text mining approaches[8][9].

As future web promises for many advantages including meaningful and understandable web[7], to use the information associated with current web with future web we will need transition. This transition will require relational schema to ontology conversion step at some stage. Moreover, the relational schemas on the internet are in heterogeneous formats and it is difficult to propose and implement a single methodology for all of them. This is the major reason that there is no standard transition algorithm, and huge efforts are required for smooth transition from the existing web to semantic web.

Researchers have worked for creating tools for transformation from relational schema to ontologies (e.g. DataGenie, DataMaster etc). DataGenie [3] is a tab plug-in for

Protégé that enables Protégé to connect to database and moves portions (or all) of your database into Protégé. But, it does not support OWL ontologies or schemas. For the drawbacks in the DataGenie, the DataMaster was developed in BioSTORM [10] project which supports both OWL and frame-based ontologies. It works as a plug-in for protégé [11]. Many of these approaches merely give the results in the user required form. Moreover, the weak entities of the database are also mapped into classes (for example in datamaster, DataGenie etc). The technique presented here performs extraction of the relationships in the relational schemas when we have no enriched Meta data available.

7.2. Contribution of Project

To extract the information from a relational schema, either metadata is considered or in case it is not available we will have to derive the desired information through algorithms. But, what will be the case if some metadata is available? We have given a relational schema to DataMaster, being a metadata extractor it fails to gather the foreign key information from the MySQL as the Metadata of MySQL was not enriched. In this case if we go for the algorithm that will start deriving information from zero, this approach will increase the execution cost of the algorithm. At this point, one would definitely think to go for an efficient approach. To overcome this problem, the Algorithm-1 works with the gathered information from metadata and resolves foreign keys based on some weights assigned as parameters by the user or the default adjusted weights can be used.

7.3. Future Work

The relational schema to ontology conversion step requires the domain knowledge. The algorithm that works intelligently in a given scenario will reduce the execution cost required for conversion from relational schema to ontology. This can be done by using available metadata in a relational schema.

The algorithm’s weights and threshold required some domain knowledge. These can be auto-adjusted according to the given scenarios which will definitely add to the effectiveness of the proposed technique. Also, the types or relationships can be identified using the proposed algorithm in combination with some other scheme (i.e. making it a Hybrid system).

References

[1] Michal Laclav´ık, "RDB2Onto: Relational Database Data to Ontology Individuals Mapping", Proceeding of Ninth international Conference of Informatics, 2007.

[2] Irina Astrova1, Bela Stantic: "Reverse Engineering of Relational Databases to Ontologies: An Approach Based on an Analysis of HTML Forms". Proceedings of the 23rd IASTED International Conference on Databases and Applications (DBA), Innsbruck, Austria. 2005, 246 - 251.

[3] "Protégé Wiki", Retrieved on 1 June, 2009. Website: http://protege.cim3.net/cgi-bin/wiki.pl?datagenie

[4] "Protégé Wiki", Retrieved on 12 January, 2009.Website:http://protegewiki.stanford.edu/index.php/datamaster

[5] Csongor Nyulas, Martin O'Connor, Samson Tu: "DataMaster – a Plug-in for Importing Schemas and Data from Relational Databases into Protégé", Stanford Medical Informatics, 10th International Protege Conference, 2007.

[6] Jorge Cardoso, Premier Reference Source, "Semantic Web Services: theory tools and applications", Information Science Reference, United Kingdom, 2006

[7] T.joa et.al: "Semantic Web challenges and new requirements", Sixteenth International Workshop, Aug. 2005 , 1160 – 1163

[8]"Ontogen Semi automatic ontology editor", retrieved on 1 June, 2009. Website:http://ontogen.ijs.si/

[9] B. Fortuna, M. Grobelnik, D. Mladenic: "OntoGen: Semi-automatic Ontology Editor". HCI International 2007, Beijing,July 2007.

[10] M. Crubézy et.al: "Ontology-Centered Syndromic Surveillance for Bioterrorism". IEEE Intelligent Systems, 20(5):26-35. 2005

[11] "Protégé Wiki", Retrieved on 12 January, 2009. Website: http://protege.stanford.edu/

[13]Berners-Lee, Tim; James Hendler and Ora Lassila (May 17, 2001). "The Semantic Web". Scientific American Magazine.

[14]"W3C Semantic Web Frequently Asked Questions". W3C.

[15] http://www.w3.org/2001/sw/

[16] N.F.Noy, D.L McGuinness “Ontology Development 101: A Guide to Creating Your First Ontology”, Available at http://protege.stanford.edu/publications/ontology_development/ontology101.pdf

[17] “Semantic web”, retrieved on 8 july,2009: http://semanticweb.org/wiki/OWL

[18] I.Astrova, N.Korda, A.Kalja “Rule-Based Transformation of SQL Relational Databases to OWL Ontologies” 2nd International Conference on Metadata and Semantics Research, pages---place---October 2007.

[19] OWL Web Ontology Language Guide, available at http://www.w3.org/TR/2004/REC-owl-guide-20040210/

[20] Kiran Sonia, Sharifullah Khan, “R2O Transformation System: Relation to Ontology transformation for Scalable Data Integration”,

[21] Bizer C. D2R MAP – A DB to RDF Mapping Language. 12th International World

Wide Web Conference, Budapest. May 2003

[22] Barrasa J, Corcho O, Gómez-Pérez A. FundFinder – A case study of Database-toontology mapping. Semantic Integration Workshop, ISWC 2003. Sanibel Island,

Florida. Sept 2003

[23] Do, H.H., E. Rahm: COMA – A System for Flexible Combination of Match Algorithms. VLDB 2002

[24] A.Bucella, M.R.Penabad, F.J.Rodriguez, A.Farina, A.Cechich “From relational databases to OWL ontologies”, Digital Libraries: Advanced Methods and Technologies. Digital Collection, pages---Puschchino, Rusia, 2004.

[25] M.li, X.Du, S.Wang “Learning ontology from relational databases” Machine learning and Cybernetics, volume 6, pages 3410-3415, place--- 2005.

[26] R.Alhajj “Extracting the extended entity relationship model from a legacy relational database”, Information systems, volume 28, number 6, pages 597-618, September 2003.

APPENDIX – A

1. Association identifications (Proposed Technique)

```
 Foreignkey_indentification(Pk, R'[1,2,3…..,n])
 i=n;                (n is numbers of remaining fields )
     do
             if(Pk_NAME == R'_NAME){
                     weight=weight + w1;
             }
             if (R'_NAME conatins Pk_NAME){
                     weight=weight + w2;
             }
             if (R'_DT conatins Pk_DT){
                     weight=weight + w3;
             }
 i=i-1;
     while (i != 0)
return weight
```

2. Foreign Key Identification in DataMaster

```
class ForeignKey {

        public String name;
        public String localField;
        public String referenceField;
        public String referenceTable;


        public ForeignKey(String n, String l, String f, String t)
        {
// taking the name, localfield, referenceField and reference table as parameter
                name = n;
                localField = l;
                referenceField = f;
                referenceTable = t;
        }
        public String toString()
        {
// add all the information to the string buffer
                StringBuffer buf = new StringBuffer();
                buf.append("Name: " + name);
                buf.append("\nLocal Field: " + localField);
                buf.append("\nRef Field: " + referenceField);
                buf.append("\nRef Table: " + referenceTable);
                return buf.toString();
        }
}
```

2. Database Importer class in datamaster

```
connOptions, DataMasterImportOptions impOptions) {
                this.kb = kb;
                if (kb instanceof OWLModel) {
                        this.owlModel = (OWLModel) kb;
                }
                else {
                        this.owlModel = null;
                }

                dsURL = Global.convertJdbcUrlToOwlNamespaceUrl(
connOptions.getDataSourceURL(), Global.URL_Conversion.EliminateAllColons);
                schemaName = connOptions.getSchemaName();
                superClses = (Collection<Cls>) impOptions.getSuperClasses();

                importInCurrOntology = impOptions.importInCurrOntology();
                importInSepOntology = impOptions.importInSepOntology();
                useDiffNamespaces = impOptions.useDiffNamespaces();
                inclTableNameInColName = impOptions.inclTableNameInColName();

                columnType = impOptions.getColumnType();

                tableClassNamePrefix = impOptions.getTableClassNamePrefix();
                tableClassNameSuffix = impOptions.getTableClassNameSuffix();
                columnPropertyNamePrefix =
impOptions.getColumnPropertyNamePrefix();
                columnPropertyNameSuffix =
impOptions.getColumnPropertyNameSuffix();

        if (owlModel == null) {
                nsPrefixRelOwl = "";
        }
        else {
                        nsPrefixRelOwl = getPrefixForNamespace_RelationalOWL();
                }
        }
```

```
	public String getResourceNameForTable(String tableName){
		if (owlModel == null) {
			return getFramesNameForTable(tableName);
		}
		else {
			return getOWLNameForTable(tableName);
		}
	}

	protected String getFramesNameForTable(String tableName) {
		return tableClassNamePrefix + tableName;
	}

	protected String getOWLNameForTable(String tableName) {
		return nsPrefix + Global.replaceInvalidProtegeCharacters(tableName);
	}

	protected String getPrefixForNamespace_TableClasses() {
		if (importInCurrOntology) {
			if (useDiffNamespaces) {
				return
getPrefixForNamespace(Global.NAMESPACE_TABLE_CLASSES +
Global.NAMESPACE_BIND_DSN + dsURL + "#") + ":";
			}
			else {
				return "";
			}
		}
		else {
			//TODO see how do we deal with this case!!!!
			return "";
		}
	}

	protected String getPrefixForNamespace_TableInstances() {
		if (importInCurrOntology) {
			if (useDiffNamespaces) {
				return
getPrefixForNamespace(Global.NAMESPACE_TABLE_INSTANCES +
Global.NAMESPACE_BIND_DSN + dsURL + "#") + ":";
			}
			else {
				return "";
			}
		}
		else {
			//TODO see how do we deal with this case!!!!
			return "";
		}
	}

	protected String getPrefixForNamespace_TableClassesAndInstances() {
		if (importInCurrOntology) {
			if (useDiffNamespaces) {
				return 
getPrefixForNamespace(Global.NAMESPACE_TABLE_CLASSES_AND_INSTANC
ES + Global.NAMESPACE_BIND_DSN + dsURL + "#") + ":";
			}
			else {
				return "";
			}
		}
		else {
			//TODO see how do we deal with this case!!!!
			return "";
		}
	}

	protected String getPrefixForNamespace_RelationalOWL() {
		return 
getPrefixForNamespace(Global.NAMESPACE_RELATIONAL_OWL, "dbs") + ":";
	}

	protected String getPrefixForNamespace(String namespace) {
		return getPrefixForNamespace(namespace, "db");
	}

	protected String getPrefixForNamespace(String namespace, final String 
preferred_prefix_base) {
		final String prefix_base = preferred_prefix_base;
		NamespaceManager nsmgr = owlModel.getNamespaceManager();

		String prefix = nsmgr.getPrefix(namespace);
		if (prefix != null)
			return prefix;

		prefix = prefix_base;
		int prefix_ind = 1;
		while (nsmgr.getNamespaceForPrefix(prefix) != null) {
			prefix = prefix_base + prefix_ind++;
		}
		nsmgr.setPrefix(namespace, prefix);

		return prefix;
	}

	protected String getColumnNamePrefix(String strTableName) {
		if (inclTableNameInColName)
			return strTableName + ".";
			else
				return "";
		}

		protected OWLDatatypeProperty createDatatypePropertySafe(String
propertyName, RDFSDatatype datatype, boolean isAnnotationProperty, boolean
isFunctional) {
			OWLDatatypeProperty property	=
owlModel.getOWLDatatypeProperty(propertyName);

			if (property == null) {
				if (isAnnotationProperty)
					property =
owlModel.createAnnotationOWLDatatypeProperty(propertyName);
				else
					property =
owlModel.createOWLDatatypeProperty(propertyName, datatype);
				property.setRange(datatype);
				property.setFunctional(isFunctional);
			} else if ( ! datatype.equals(property.getRangeDatatype()) ) {
				Global.debug("WARNING! " + propertyName + " property is
already defined with the wrong type.");
			}

			return property;
		}

		protected OWLObjectProperty createObjectPropertySafe(String propertyName,
Collection allowedClasses, boolean isAnnotationProperty, boolean isFunctional) {
			OWLObjectProperty property	=
owlModel.getOWLObjectProperty(propertyName);

			if (property == null) {
				if (isAnnotationProperty)
					property =
owlModel.createAnnotationOWLObjectProperty(propertyName);
				else
					property =
owlModel.createOWLObjectProperty(propertyName, allowedClasses);
				property.setRanges(allowedClasses);
				property.setFunctional(isFunctional);
			} else if ( ! property.getRanges(false).containsAll( allowedClasses ) ) {
				Global.debug("WARNING! " + propertyName + " property is
already defined with different range specification.");
			}

			return property;
		}
```

```
	protected OWLObjectProperty createObjectPropertySafe(String propertyName,
OWLObjectProperty superProperty,
			Collection allowedClasses, boolean isAnnotationProperty,
boolean isFunctional) {

		OWLObjectProperty property		=
createObjectPropertySafe(propertyName, allowedClasses, isAnnotationProperty,
isFunctional);
		if ( ! property.isSubpropertyOf(superProperty, false)) {
			property.addSuperproperty(superProperty);
		}
		return property;
	}


	protected OWLNamedClass createOWLClassSafe(String className) {
		OWLNamedClass owlClass	=
owlModel.getOWLNamedClass(className);

		if (owlClass == null) {
			return owlModel.createOWLNamedClass(className);
		}// else if (owlClass.getRDFType() != datatype) {
		//	Global.debug("WARNING! " + className + " property is
already defined with the wrong type.");
		//}

		return owlClass;
	}

	protected RDFResource createOWLIndividualSafe(OWLClass typeClass, String
individualName) {
		RDFResource owlIndividual	=
owlModel.getRDFResource(individualName);

		if (owlIndividual == null) {
			try {
				return typeClass.createInstance(individualName);
			} catch (Exception e) {
				Global.debug("WARNING! Exception by creating
individual '" + individualName + "': Protege instance name is already in use! This may
cause further NULL POINTER EXCEPTION.");
				e.printStackTrace();
				return null;
			}
		} else {
			if ( ! owlIndividual.getRDFTypes().contains(typeClass)) {
				Global.debug("WARNING! " + individualName + "
individual is already defined with the wrong type!");
				owlIndividual.addRDFType(typeClass);
			}
		}

		return owlIndividual;
	}

	protected Instance createInstanceSafe(Cls typeClass, String instanceName) {
		Instance inst	= kb.getInstance(instanceName);

		if (inst == null) {
			try {
				return kb.createInstance(instanceName, typeClass);
			} catch (Exception e) {
				Global.debug("WARNING! Exception by creating
instance '" + instanceName + "': Protege instance name is already in use! This may
cause further NULL POINTER EXCEPTION.");
				e.printStackTrace();
				return null;
			}
		} else {
			if ( ! inst.hasType(typeClass)) {
				Global.debug("WARNING! " + instanceName + "
instance is already defined with the wrong type!");
				inst.addDirectType(typeClass);
			}
		}

		return inst;
	}

	// Returns true if String s is in collection c.
	protected boolean contains(Collection<String> c, String s)
	{
		Iterator iter = c.iterator();
		while(iter.hasNext())
		{
			if(iter.next().equals(s))
				return true;
		}

		return false;
	}


	/**
	 * This method creates a new slot in "thisCls" named "name", whose value is of
type "allowedCls"
	 *
	 * @param thisCls
	 * @param name
	 * @param allowedCls
```

```
         * @param allowMult
         * @return
         */
        protected Slot generateSlot(Cls thisCls, String name, Cls allowedCls, boolean
allowMult)
        {
                Slot newSlot = this.kb.getSlot(name);
                if(newSlot == null)
                        newSlot= this.kb.createSlot(name);
                newSlot.setValueType(ValueType.INSTANCE);
                Collection<Cls> classHolder = Collections.singletonList(allowedCls);
                newSlot.setAllowedClses(classHolder);
                newSlot.setAllowsMultipleValues(allowMult);
                thisCls.addDirectTemplateSlot(newSlot);

                return newSlot;
        }

        /**
         * This method scans a list of instances for one where the value of slotName
matches id.
         * changed because the keys may have types other than Integer (e.g. String)

         * private Instance getMatchingInstance(Collection instances, String slotName,
int id)
         *
         * @param instances
         * @param slotName
         * @param id
         * @return
         */
        protected Instance getMatchingInstance(Collection<Instance> instances, String
slotName, Object id)
        {
                Slot s = this.kb.getSlot(slotName);
                Iterator iter = instances.iterator();
                while(iter.hasNext())
                {
                        Instance inst = (Instance)iter.next();
                        Object val = inst.getOwnSlotValue(s);
                        if (val != null && val.equals(id))
                                return inst;
                }

                return null;
        }


        /**
```