A Framework and Architecture for Quality Assessment in Data Integration

Jianing Wang

March 2012

A Dissertation Submitted to
Birkbeck College, University of London
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Department of Computer Science & Information Systems
Birkbeck College
University of London

Declaration

This thesis is the result of my own work, except where explicitly acknowledged in the text.

Jianing Wang _____

June 26, 2012

To my family

Abstract

Data integration aims to combine distributed information sources conforming to different modelling methods and provide interfaces for accessing the integrated resource. Data integration processes may be complex and error-prone because of the heterogeneities of the information sources. Moreover, data integration is a collaborative task involving many people with different levels of experience, knowledge of the application domain, and expectations. It is difficult to determine and control the quality of a data integration setting due to these factors.

In this thesis, we investigate the methods of improving the quality of integrated resources with respect to the users' requirements, in an iterative integration process. We propose a quality framework that is capable of representing different quality requirements arising from stakeholders involved in the integration process. Ontology-based inferencing within this quality framework allows the data integrator to identify amendments to the integrated resource so as to satisfy users' quality expectations better. We define several quality criteria and factors specific to the context of data integration and propose a number of quality metrics for measuring these quality factors. We propose a data integration methodology that supports quality assessment of the integrated resource and an integration architecture for the realisation of this methodology. We show how the quality of an integrated resource could be improved using our quality framework, quality criteria and factors, and data integration methodology using a real-world case study.

Acknowledgements

First and foremost, I would like to thank my supervisors, Alexandra Poulovassilis and Nigel Martin, for their continuous support, their patient guidance and their faith in my work throughout these years. Without their guidance, I would have never completed this PhD.

Many thanks are due to my colleagues at Birkbeck for their input, collaboration and numerous stimulating discussions. I would particularly like to thank Lucas Zamboulis for his help in familiarisation with the AutoMed system and also for his enormous contribution to the AutoMed development.

Finally, I would like to thank my family for their support on my study throughout this period.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Historically and currently, data conforming to different formats are gathered and organised by different parties. However, users may need to access such data sources according to their own requirements. This may require redefining data into different formats, combining relevant data from different sources, and combining incomplete data sources in order to form a more complete view. Combining and transforming data from different data sources is a complex problem and is the focus of *Data Integration (DI)* research. In the DI context, data conforming to different data models can be transformed and accessed through a *global schema* using *mappings* between this schema and the data sources. A global schema is a view defined over the local schemas. In the context of this thesis, a global schema is the union of selected local schema constructs allowing users to access information from the local data sources. A typical DI setting is represented as a triple $\langle GS, LSs, M \rangle$, where GS is the global schema, LSs are the local (i.e. data source) schemas and M are the mappings between the GS and the LSs.

Many DI tools have been designed to assist data integrators in generating the global schema and mappings semi-automatically, such as tools for *schema annotation*, *schema matching*, *mapping generation* and *mapping refinement*.

However, the data integration process may be complex and error-prone even with the assistances of such tools because of the heterogeneities of the data sources. For example, it may be difficult to integrate information represented in different data models and structures using different terminologies. Moreover, data integration is a collaborative process involving many people (we use the term *users* in this thesis). The designs of the global schema and mappings relate to these users, including the data integrator's experience and understanding of the application domain and the expectations of different end-users. It is difficult to determine and control the quality of the resulting integrated resource due to these factors. In this thesis, 'integrated resource' refers to the local schemas, the data stored in the data sources, the global schema, the mappings, and additional assertions representing users' knowledge of the application domain.

This thesis investigates methods for improving the quality of integrated resources with respect to users' quality requirements. In particular, we propose a quality framework that is capable of representing different quality requirements arising from users involved in the data integration process. Ontology-based inferencing within this quality framework allows the data integrator to identify amendments to the integrated resource so as to satisfy users' quality expectations better. We define several quality criteria and factors specific to the context of data integration and we propose a number of quality metrics for measuring these quality factors. We propose an iterative data integration methodology that has embedded within it quality assessment of the integrated resource, and an integration architecture for the realisation of this methodology.

1.1 Data Integration

Data integration is a broad area and, in practice, there are many different integration scenarios. For example, data from different data sources may be combined and such data may be materialised in a repository through a 'materialised' global schema (a *data warehouse* [2]), or users may access such information through a 'virtual' global schema (this is *virtual data integration* [3]). In a virtual data integration scenario, the GS is a virtual schema, meaning that the GS does not have associated stored data but its constructs are populated by data that is derived from the data sources using the mappings when users' queries on the GS are processed. Compared with materialised data integration, there are overheads caused by the re-computation of such data in virtual DI; on the other hand, the data is guaranteed to be current, whereas this is not generally the case in a materialised approach. Other integration scenarios are when information represented in one format is transformed into another representation (this is *data exchange* [4]), or when peers containing partial information can be accessed via other peers to exchange or integrate their data (this is *peer-to-peer integration* [5]). All the above scenarios share some common tasks, such as *schema matching* and *schema mapping*.

In this thesis, we focus on virtual data integration, in which one needs to define rules for transforming and combining information from several data sources, expressed in the same or different modelling languages, and to provide a unified interface for accessing this information. This interface is termed the *global schema* (GS), the schemas provided by the data sources are termed the *local schemas* (LSs) and the rules are termed the *mappings* (M).

Schema matching is an automatic or semi-automatic process of discovering possible relationships between the constructs of two schemas, for example two local schemas LS_1 and LS_2, based on their syntax, structure

and semantics. The output of this process is a set of matches of the form (C_1, C_2, r, cs), where C_1 is a construct of schema LS_1, C_2 is a construct of schema LS_2, r is a specification of the relationship between these constructs (such as equivalence, subsumption and disjointness) and cs is the confidence score, i.e., a value in the range $[0 \dots 1]$, that specifies the level of confidence in the relationship r. Often, given that schemas may be large and complex, the schema matching process may not be able to produce complete or accurate matches. However, schema matching is valuable for reducing the search space for schema mappings, allowing the data integrator to focus on identifying more difficult matchings.

Schema mapping is a manual or (semi)-automatic process of deriving the precise mappings between two schemas. If these schemas are a local schema (LS) and a global schema (GS), the mappings can be used to transform a query posed on the GS to sub-queries posed on the LS, or vice versa.

1.2 Motivation and Research Methodology

The heterogeneity of the data sources can make the DI process complex and error-prone. Other factors also have impacts on the overall quality of the integrated resource, such as users' quality requirements, the real-world semantics, the application domain and the data integrator's experience. There is not a clear definition of what is meant by *DI quality* in the current research and commercial domains. These issues form the motivation of our research questions addressed in this thesis:

- What are the quality requirements arising from different users in the integration process? How can such requirements affect the integration process at different stages?
- How is quality defined in the context of data integration? Do different users have different quality perspectives?

- If there are inconsistencies between such quality definitions, how is the integrated resource affected?
- Can an ontology be used to formalise different users' quality perspectives? Can ontology reasoning help to detect inconsistencies between different users' quality expectations?
- How can the quality of the integrated resource be determined? What elements in the integrated resource are useful for measuring its quality?
- How can the integrated resource be improved in order to meet the users' expectations? What factors do data integrators consider to be important in improving the integrated resource?

With these research questions as a starting point, our research is organised as follows. First, we study the related research and identify the problems relating to assessing quality in a data integration setting. We also report on an interview held with several data integrators in order to capture the concerns data integrators may have in practice and to identify further questions motivating our research. Second, we propose a quality framework that can be used to represent and assess the quality of an integrated resource from different users' perspectives. This quality framework is implemented using the OWL-DL ontology language. Third, we propose a data integration methodology that incorporates quality assessment functionality within the integration process. We also propose an integration architecture as a realisation of this integration methodology. Fourth, we define a set of quality criteria, a set of corresponding quality factors, and their associated measurement methods in order to demonstrate our quality framework and integration methodology. Fifth, we use a real-world integration project in the life science domain to evaluate the methods proposed in our research.

Throughout this thesis, our work is developed with the following assumptions: We assume that information is stored in structural data sources,

such as relational databases, that are to be integrated and the data integrators have direct access to the metadata and data in the data sources (we leave consideration of the integration of semi-structured data sources as future work). The end-users of the integrated resource have some quality requirements they wish it to satisfy. The data integrators have some initial knowledge about the application domain that can be expressed with respect to an existing domain ontology.

1.3 Contribution

In this thesis, our main contributions to data integration research are our integration methodology, quality framework and quality measurement methods. In particular:

- We propose a quality framework that can represent different users' quality perspectives. We identify a set of elements in the integrated resource that are referenced by the quality measurement methods and are significant for the iterative quality improvement of the integrated resource. The richness of our quality framework allows different users' quality requirements to be expressed. Our framework is also extendable to allow more quality criteria and factors to be defined.

- We define five quality criteria and two of them have been studied intensively together with several related quality factors that are specific to the data integration context. The definitions of these quality criteria and factors are capable of forming the quality requirements from different categories of users.

- We propose quality measurement methods for calculating the level of satisfaction of the integrated resource compared with the users' requirements. The results of our quality measurement methods provide

quality observations to the data integrator and are also used to calculate the overall quality. In addition, these results are also derived to support the reasoning in our quality framework.

- We propose an integration methodology which embeds quality assessment within the integration process and allows different quality aspects of the integrated resource to be assessed and improved in an iterative fashion. In addition, because of the iterative nature of our DI methodology, data source evolution could be handled within it.

1.4 Thesis Outline

This thesis is structured as follows. Chapter 2 reviews previous work on identifying the issues causing the DI process to be difficult and previous research on quality in the context of information systems and data integration. We also review ontology representation and reasoning work that is relevant to our research. The AutoMed data integration system is also introduced.

Chapter 3 discusses the users' requirements as derived from existing research in DI supplemented by an interview with data integrators. We also present a simple case study that demonstrates some issues causing the DI process to be difficult and users' requirements relating to the quality of the integrated resource. This case study is also used for illustrating our quality framework, criteria and methodology proposed in Chapters 4, 5 and 6.

Chapter 4 presents our quality framework and gives a detailed discussion of its components. We discuss the relationships between these components and we discuss the reasoning capabilities considered in our research in order to obtain an integrated and consistent quality view of the integrated resource. The formal foundations of our approach are also laid out in this chapter.

Chapter 5 proposes five quality criteria specific to the context of data in-

tegration. We discuss in detail the *completeness* and *consistency* criteria and the quality measurement methods associated with these criteria. We also discuss the relationships between such criteria and how reasoning methods can be applied in order to obtain an integrated and consistent quality view of an integrated resource.

Chapter 6 introduces our data integration methodology which embeds quality assessment within the DI process. We also present an integration architecture for the realisation of this methodology.

Chapter 7 discusses a real-world integration project in the life sciences domain and demonstrates how our quality framework, quality criteria, measurement and reasoning methods, and integration architecture can be used to assess and improve the quality of the integrated resource derived within this project.

Chapter 8 discusses the contributions of the thesis and identifies some areas of future work.

Chapter 2

Data Integration from a Quality Perspective

Chapter 1 introduced the main data integration tasks and the importance of users' requirements in determining DI quality. It also gave the motivation and the aims of our research described in this thesis. In this chapter, we will discuss some of the important research relating to our work and indicate areas that need further work. Section 2.1 reviews a classification of the issues encountered when attempting to integrate information from different data sources. Section 2.2 then introduces different data integration tools developed in order to assist data integrators to address these issues. Section 2.3 gives a review and critical analysis of related work on identifying and measuring quality in the contexts of information systems and data integration. In Section 2.4, we give an overview of the AutoMed data integration system, which we build on for ease of development and rapid prototyping of our DI architecture. Section 2.5 summaries this chapter with more analysis of the research discussed previously and their relationships to our research.

2.1 Heterogeneity Classification

Information stored in the data sources which data integration aims to combine may differ in various aspects, such as the terminology used or the modelling methods adopted. Solving such heterogeneity issues across data sources and between the data sources has been the main concern in data integration research for many decades. Such heterogeneity issues can be categorised into three groups as discussed in [6, 7, 8, 9, 10, 11], termed 1) *Syntactic or Data Model heterogeneity*, 2) *Schema or Structural heterogeneity* and 3) *Semantic or Terminology heterogeneity*:

Syntactic or Data Model Heterogeneity A data model provides the definitions of modelling constructs that can be used to describe how data are organised under schemas [12], for example, the ER model, the Relational model or XML. Modelling constructs provided by different data models may have different semantic meanings and some of them may be particular to a model as they cannot be replicated in other models. For example, a table in the relational model is a representation of a relation which may contain multiple attributes. Such attributes have a strong relationship with their referencing table, meaning such attributes only exist when their referencing table exists. In the case of XML, different types of XML elements are supported, such as XML entities, attributes, etc. Transforming information represented in different data models may be difficult without changing the original semantics. In addition, the various datatypes defined in data models are another source of data model heterogeneity, and therefore, need to be considered carefully in data integration [10].

Schematic or Structural Heterogeneity A schema describes how data are structured using modelling constructs provided by data models. There are many types of data structures, such as various kinds of tree

representations, relationships with 1:1, 1:N or M:N cardinalities, cyclic or acyclic structures, nested or flat structures. Identifying correspondences and creating mappings to transform information between different structures may be difficult due to such heterogeneities [13, 14]. Information contained in the original data sources may be lost or manipulated unnecessarily and may result in the loss of semantics; conversely, additional information may be created which introduces new semantics during the integration process. In addition, in order to combine identical, overlapping, or disjoint information complying with different structures in the DI context, the design of the *GS* is also important in that the *GS* structure should be capable of representing all such information in compliance with the users' requirements.

Semantic or Terminology Heterogeneity refers to the variety of literal representations of information. The literal representations include the use of synonyms, homonyms, hyponyms and hypernyms at the data and metadata level. Identifying the precise relationships between such literals in different contexts may be difficult. Predefined Upper and Domain ontologies can contribute to the solution of this problem where domain specific literals and their relationships have been defined by the domain experts, such as BioTop [15] and OpenGALEN [16].

There may be other heterogeneities existing in data sources, such as the query languages designed for different data models, different query processing abilities supported by such data sources and different technologies used to access the data. These issues may also have impact on the DI processes, but they involve performance and security related issues which are beyond the scope of our research and, therefore, they are not discussed in this thesis. A further important issue which impacts on the quality of data integration is the quality of the input data itself. This has also been beyond the scope

of our research and we identify this as a future research direction in Chapter 8.

2.2 DI Research

2.2.1 DI Tools

In order to solve the heterogeneity issues discussed in the previous section, many data integration tools have been designed to assist data integrators in DI tasks such as schema matching, mapping generation, query processing and DI refinement.

Schema Matching tools such as COMA++ and Cupid, are designed to discover the correspondences between schema constructs based on their syntax, structure and semantics. Schema matching algorithms can be categorised into three types, instance level, schema level and semantic level matching. Instance level matching algorithms focus on producing matchings based on the data stored in the data sources. Schema level matching algorithms are based on the schemas' structural information. Semantic level matching algorithms produce correspondences based on the semantic information about schema constructs, such as equivalence and disjointness, and the semantics of the integration domain. Such tools normally output 1-1 or 1-n correlations between schema constructs together with confidence values indicating the certainty of the detected correlations. However, current schema matching tools are not capable of handling difficult matching cases, such as information with complex semantics. Therefore, schema matching tools are mostly used to provide matching information to the integrators in order to reduce their workload and allow them to concentrate on more difficult matchings. We discuss below two popular schema matchers

COMA++ and Cupid.

COMA++ [17] was developed as a generic matching tool for data integration and data exchange applications. It takes two schemas in various formats, such as XML and the Relational model, as its inputs and produces matchings between schema constructs together with a confidence value between 0 and 1. A range of matchers have been implemented in COMA++ in order to generate more accurate correspondences with respect to the information provided by the data sources. For example, an instance matcher analyses the similarities between the extents of schema constructs in order to discover the similarity between such schema constructs. A context matcher analyses the similarity between schema constructs based on the similarities of other schema constructs that are linked with them through relationships in the schema.

Cupid [18] is a hybrid schema matching system that also supports various data models. In contrast to COMA++, Cupid operates only at the schema level and employs linguistic, structural and constraint matchers to produce 1-1 and 1-n matches.

Mapping Generation tools take the correspondences generated by schema matching tools as their input and semi-automatically generate mappings between the *LSs* and the *GS* for transforming information stored in one schema to another schema. Different mapping approaches have been proposed, with the global-as-view (GAV) and local-as-view (LAV) approaches as the two most common ones. In GAV, each *GS* construct is defined by a conjunctive query over the *LSs* [3]. We call the mappings generated in the GAV approach *GAV mappings*. In LAV, each local schema construct is defined by a conjunctive query over the *GS* [3]. We call such mappings *LAV mappings*. There is also a more general approach called global-local-as-view (GLAV), where conjunc-

tive queries on the *LSs* are mapped to conjunctive queries on the *GS* [19]. The both-as-view (BAV) approach has also been proposed where the mappings consist of a sequence of primitive schema transformations [20]. These approaches have advantages and disadvantages when applied in different DI contexts and different kinds of mappings are also embedded with different semantics. Therefore, the integration approach needs to be chosen carefully with respect to the integration domain. We refer readers to [3] for detailed discussions. We discuss in this thesis the AutoMed and Clio integration frameworks that support the mapping generation task. In addition to the mapping generation function, a data integration framework also supports the wrapping of data sources in order to represent schema-level information from the data sources using the data modelling methods supported by the framework. AutoMed is discussed in Section 2.4 and Clio and HeP-ToX are discussed next. In the commercial domain, there exist many mapping tools that mainly depend on the manual establishment of mappings through graphical interfaces. We refer readers to [21] for a survey of such tools.

Clio [22] is a mature data integration platform which defines mappings using a logic-based declarative approach. Clio supports GAV, LAV and GLAV DI approaches. Data integrators have to create the mappings semi-automatically according to their matching correspondences. In addition to the mapping creation function, Clio also supports mapping verification. Data integrators identify unexpected data in the extension of the global schema manually and examine the mappings that generate these data against the integrity constraints of the global schema. The disadvantage of using this method is that mapping errors need to be identified manually. If the integrated resource contains a large volume of data, it is very hard to go through every row of data

and find possible errors.

HePToX [23] is a tool for integrating XML data sources in a peer-to-peer setting. The tool is able to transform a set of mapping definitions to XQuery queries that are executable at an XML data source for extracting information from it. The mapping definitions are manually defined by the integrators and the transformation process is then undertaken automatically. The authors claim that the transformation algorithm can generate efficient XQuery queries and that constraint information is not lost. However, there are no validation or quality assessment processes considered in this research.

Query Answering in DI is the problem of answering a query q over an integrated resource $\langle LSs, GS, M \rangle$. q could be expressed either on the GS or the LSs, depending on the context. For example, in peer-to-peer data integration, users may express their queries on a single peer and expect information to be extracted from other peers. In the GAV approach, query answering is mainly a query unfolding process where q is unfolded by substituting each global relation by its definition in terms of the data sources. In the LAV approach, query answering is mainly a query rewriting problem using views in order to reformulate q into an equivalent expression that refers only to the data source structures. Both approaches are followed by the query optimisation task that aims to optimise the unfolded or rewritten queries in order to reduce the cost of access to the data sources [24]. The results of query answering may be ambiguous because of the different query reformulation mechanisms and we refer readers to [25, 26] for detailed surveys.

DI Refinement aims to refine an existing DI setting that does not comply with the users' expectations. DI refinement tools can be implemented

in different ways. For example, data lineage algorithms are used to identify mappings that extract incorrect information with respect to the constraints on the *LSs* and *GS* [27]. Feedback analysing algorithms have been developed to identify mappings where the users' requirements cannot be satisfied [28]. Users' feedback may be annotations that a user provides to comment on the artifacts of a data integration system, such as relationships between query results. The design of the *GS* can also be refined by comparing the current *GS* with the stakeholders' expectations of the *GS*. We will discuss the details of such related research in Chapter 5 when comparing this work with our research.

2.2.2 Ontologies in Data Integration

Ontologies are a formal representation method and represent knowledge as a set of concepts within a domain and the relationships between those concepts. Ontology reasoning can be applied in order to make inferences about conceptual relationships between the entities within that domain. Ontologies have been used in many tasks in the data integration process, such as matching, mapping, query answering and DI refinement by using their rich expressive power.

With respect to the matching and mapping tasks, ontologies have two advantages compared with schemas. First, schemas often do not provide explicit semantics for their data, while ontologies are able to constrain the meaning of ontology definitions with a set of logical axioms. Second, ontologies provide a vocabulary of terms in some domain. Database schemas provide an application-dependent vocabulary that is not generally reusable in other applications. In contrast, ontologies support both application-dependent and independent vocabulary, such as domain ontologies and upper-level ontologies in the latter case. Ontologies are designed by

domain experts in order to formally describe the domain with a vocabulary of terms, such as BioTop for the life-science domain (`http://www.imbi.uni-freiburg.de/ontology/biotop/`) and OpenGALEN [16]. Adopting ontologies in the matching and mapping tasks in data integration, more accurate matching and mappings can be discovered. [29] gives a detailed representation method for integrated resources using Description Logic, which is the most widely used knowledge representation method used as the foundation of ontologies. We also refer to [30, 31] for detailed studies of ontology-assisted matching and mapping techniques. We survey here two well-known ontology matching/mapping tools, GLUE [32] and PROMPT [33].

GLUE [32] semi-automatically creates mappings between ontologies. GL -UE finds the most similar concepts between a pair of ontologies and calculates the joint probability distribution of the concepts using a multi-strategy learning approach for similarity measurement, including name matching, content matching and naive Bayes learning [34].

PROMPT [33] is a semi-automatic ontology merging and alignment tool. Taking as input two ontologies, PROMPT begins with name matching for the initial comparison and a set of possible matchings are generated. Users then need to verify these matchings based on their own linguistic and domain knowledge. Based on this information, PROMPT generates a merged ontology along with transformation rules from the input ontologies to this merged ontology.

Ontologies can also contribute to the query answering task using the rich semantics embedded within the ontology. The authors in [24] introduced a three-phase query answering process, comprising query expansion, query unfolding and query execution phases. They use an ontological description of the integrated resource to aid in query expansion and optimisation. Similar work has been done in [35].

The DI refinement task can also benefit from the semantics and formal

representation methods embedded within ontologies. In [36], the authors proposed a reasoning method based on the Distributed Description Logic (DDL) in order to validate inconsistencies between the semantics of the schema and the mappings. The authors in [37] proposed a sound and complete ontological reasoning method for validating mappings with respect to subsumption propagation and disjointness propagation patterns.

Currently, there are many research works that focus on transforming information from data models, such as the Relational model, into an ontological representation in order to benefit from the additional expressive and reasoning power of ontology-based techniques [38].

2.3 Data Integration Quality

In this section, we review some quality related work in the context of data integration. Some of this work was not proposed originally for the DI context, but their concepts are relevant to and can be adopted in our work.

2.3.1 Quality Oriented Research

Quality is the study of "fitness for use" [10] and is a well established research topic in both the academic and commercial domains for particular areas, such as data quality, software quality and Quality of Service (QoS). In the area of data integration, many research works in this area adopt the quality definitions proposed in information and data quality research for defining the quality of the data sources in a DI context. In [39], the authors defined a set of data quality criteria for measuring the quality of the data sources and proposed query answering methods taking into account such quality measurement. The quality criteria they considered are data accuracy, completeness, timeliness, availability, reliability and data volume.

[40] also investigated query reformulation methods with respect to the

quality indicators from the data sources. The authors considered quality indicators such as availability, price, response time and accuracy, in order to generate query plans that can return the most optimised results.

In [41], the authors adopted the definitions proposed for expressing the quality of information systems and proposed methods for measuring the quality of information retrieved from data sources. In contrast to [39], the authors in [41] focused more on the retrieval of complete information by selecting the appropriate query operation, such as left outer join or full outer join, in the query reformulation process.

There have been some research works focusing particularly on data integration quality issues. In [42], the authors discussed work on measuring the quality of the mappings, based on features calculated from instance-level information. Such features include distribution of the length of string-valued instances and frequencies of word categories in the instance sets. A full list of such features can be found in [42].

In [43], the authors defined three quality criteria explicitly for the DI context, schema completeness, datatype consistency and schema minimality. A set of measurement methods were also proposed in order to calculate the level of satisfaction of the quality definition by the schemas referenced in the DI setting. [44] took a further step based on the minimality and completeness quality criteria proposed in [43] and proposed structural comparison methods between the GS and an expert-defined schema that is assumed to be available to the data integrators.

While not motivated explicitly from a quality perspective, other techniques can be adopted for measuring the quality of integrated resources. [4] proposed the concept of mapping cores in order to generate the minimum set of mappings able to answer a set of users' queries. It provides a non-redundant instance for answering queries over the target schema. [45, 27] proposed instance checking methods which may be used to validate and

refine mappings.

Integrity constraint checking is another area that relates to our quality assessment research. In [46], the authors discussed the importance of integrity constraints and their role in query answering methods in data integration. Query answering techniques are affected heavily by the constraints on the global schema and the semantics of mappings with respect to incomplete or inconsistent information being retrieved [46, 47]. Tableaux and chase algorithms are used to test consistency between constraints in the global schemas and mappings [48, 46, 49, 50]. However, integrity constraints considered in these works are only based on the global schema, whereas the constraints on the local schemas are also important. In [51, 52, 53], the authors took the integrity constraints on the local schemas into the account in generating the DI mappings. These works focused on generating correct and complete information for the global schema, maintaining as many referential constraints on the local schemas as possible. The Clip project [54] focused on maintaining information on local schemas via mappings by introducing mapping builders, termed aggregation, transformation, and grouping. A combination of these builders constructs a Context Propagation Tree (CPT) that links different mappings, where mappings linked with different local schema constructs that are related with constraints may have effects on each other. Mappings are then generated from the CPT.

Ontology alignment introduced a new form of mappings expressed in DL [37], such as Distributed Description Logics (DDL), where mappings are inclusion assertions [55]. Such assertions are classified as onto-bridge rules and into-bridge rules. In ontology alignment, distributed ontologies are linked with DDL assertions, possibly with weights indicating their confidence values. Distributed reasoners, such as DRAGO (`http://drago.itc.it/index.html`), can also use such assertions to examine consistencies across distributed ontology sources to reduce the reasoning cost [36]. However, in the

ontology alignment setting, ontologies are mapped pairwise and mapping semantics are limited to this context, whereas in data integration, multiple data sources are available and operations can be performed across multiple data sources, such as joins and unions. In addition, having only the inclusion relationships supported by DDL mappings, it is not possible to perform data manipulation operations such as concatenations and splits.

Also, many different stakeholders have been identified in data integration [56, 57]. Each of them plays their role in the integration process. For example, data integrators are responsible for implementing the integrated resource. Administrators are responsible for maintaining the data sources in order to support the DI processes. Users are responsible for retrieving information from the integrated resource. There are also different categories of users in the DI context. For example, in [57], the authors considered scientists, legislators and policy makers as three categories of users of the integrated resource in their research. Each such stakeholder has their own requirements on the data integration process, and the authors in [56, 57] proposed methods for analysing these requirements.

The approach in [28] determines the quality of collaborative tasks, such as data integration, with respect to the users' quality requirements through users' feedback.

2.3.2 Quality Frameworks

Quality assessment comprises not only the definitions of quality criteria, an appropriate quality framework is also required in order to assess quality comprehensively. Data warehouses are used to store historical data in a centralised repository for data intensive applications such as decision support and data mining [2]. The Data Warehouse Quality (DWQ) project (`http://www.dbnet.ece.ntua.gr/~dwq/`) studied quality issues in data warehouses and quality-driven data warehouse design. The quality dimen-

sions proposed in DWQ can be categorised into design/administration quality, software implementation quality and data usage quality dimensions. Corresponding to these, the stakeholders of data warehouses are data warehouse designers/administrators, software programmers who implement the data warehouse, and decision makers who use the data warehouse. The design/administration quality dimension includes two sub-dimensions: schema quality, which is the same as the one proposed in data quality research, and metadata evolution that concerns the way the data warehouse schema evolves during the data warehouse operation. The software implementation quality dimension adopts sub-dimensions proposed in ISO9126 (`http://www.iso.org/iso/home.htm`) from a software engineering perspective, which includes quality dimensions such as functionality, reliability, usability, software efficiency, maintainability, changeability and portability. The data usage quality dimension includes two sub-dimensions, accessibility and usefulness, where the former concerns the availability of services provided by data sources or data warehouses and the latter concerns characteristics of services provided by data warehouses [2]. The DWQ project considered data warehouse implementation as a software engineering problem from a high-level viewpoint, and did not consider solutions to problems of how the data warehouse design could be modified to improve its quality.

2.4 Overview of AutoMed

For ease of development and rapid prototyping of our DI architecture we have used and extended the AutoMed data integration system (`http://www.doc.ic.ac.uk/automed/`). In this section, we give an overview of schema modelling languages in AutoMed, the AutoMed query language (IQL) and mapping language (AutoMed transformation pathways), and the AutoMed architecture, which are necessary components relating to our research. In

Chapter 6, we will use AutoMed to illustrate our implementation of the translation between relational schemas and their OWL representations, and our implementations of the quality factors and metrics of Chapter 5.

2.4.1 AutoMed's Hypergraph Data Model

The basis of AutoMed is a low-level common data modelling language called the hypergraph data model (HDM) [20, 58]. An HDM schema S comprises a triple $S = \langle N, E, C \rangle$, where N is a set of nodes, E is a set of edges and C is a set of constraints. Nodes and Edges define a labelled, directed, nested hypergraph. It is directed because edges link sequences of nodes or edges. It is nested because edges can link any number of both nodes and other edges. A query over S is an expression whose variables are members of $N \cup E$. Constraints are expressed as queries over S that return a truth value, with a constraint being valid if it evaluates to true. Given $N = \{n_1, \ldots, n_n\}$, the extent of N is denoted by $ext(N) = ext(n_1) \cup \ldots \cup ext(n_n)$, where $ext(n_i)$ is the extent of node n_i. Given $E = \{e_1, \ldots, e_m\}$, the extent of E is denoted by $ext(E) = ext(e_1) \cup \ldots \cup ext(e_m)$. Given $C = \{c_1, \ldots, c_p\}$, the extent of C is the empty set, denoted by $ext(C) = \emptyset$. The extent of a schema S is the union of the extents of all nodes and edges, $ext(S) = ext(N) \cup ext(E)$.

Given a schema S and an extent $ext(S)$, we say that S is *valid* if $ext(S)$ is evaluated to true with respect to all constraints in S.

2.4.2 Representing a Simple Relational Model in HDM

The HDM can be used to represent schemas in other data modelling languages, such as relational schemas, XML, RDF/S and OWL [59, 38]. To illustrate the usage of the HDM, we present here the HDM representation of a simple Relational model (Table 2.1) [38].

In this simple Relational model, there are four kinds of schema con-

structs. A *Table* construct is identified by a scheme $\langle\langle \mathsf{t} \rangle\rangle$, where t is the table name. The extent of the a *Table* construct $\langle\langle \mathsf{t} \rangle\rangle$ is the projection of the relation $t(ka_1, \ldots, ka_n, nka_1, \ldots, nka_m)$ onto its primary key attributes $ka_1, \ldots, ka_n$, $n \geq 1$ ($nka_1, \ldots, nka_m$ are the non-key attributes of t). An *Attribute* construct is identified by a scheme $\langle\langle \mathsf{t}, \mathsf{a} \rangle\rangle$, where a is an attribute (key or non-key) of t. The extent of each *Attribute* construct $\langle\langle \mathsf{t}, \mathsf{a} \rangle\rangle$ is the projection of t onto attributes $ka_1, ..., ka_n, a$. A primary key construct *PKey* is identified by a scheme $\langle\langle \mathsf{t\_pk}, \mathsf{t}, \langle\langle \mathsf{t}, \mathsf{ka_1} \rangle\rangle, ..., \langle\langle \mathsf{t}, \mathsf{ka_n} \rangle\rangle \rangle\rangle$, where t_pk is the name of the constraint. A foreign key construct *FKey* is identified by a scheme $\langle\langle \mathsf{t\_fk\_i}, \mathsf{t}, \langle\langle \mathsf{t}, \mathsf{a_1} \rangle\rangle, ..., \langle\langle \mathsf{t}, \mathsf{a_p} \rangle\rangle, \mathsf{s}, \langle\langle \mathsf{s}, \mathsf{b_1} \rangle\rangle, ..., \langle\langle \mathsf{s}, \mathsf{b_p} \rangle\rangle \rangle\rangle$, $p \geq 1$, where r_fk_i is the name of the constraint, $\langle\langle \mathsf{t}, \mathsf{a_1} \rangle\rangle$,...,$\langle\langle \mathsf{t}, \mathsf{a_p} \rangle\rangle$ are the referencing attributes, and $\langle\langle \mathsf{s}, \mathsf{b_1} \rangle\rangle$,...,$\langle\langle \mathsf{s}, \mathsf{b_p} \rangle\rangle$ are the referenced attributes.

Relational Construct	HDM Representation
construct: Table class: nodal scheme: $\langle\langle \mathsf{t} \rangle\rangle$	node: $\langle\langle \mathsf{t} \rangle\rangle$
construct: Attribute class: link-nodal scheme: $\langle\langle \mathsf{t}, \mathsf{a} \rangle\rangle$	node: $\langle\langle \mathsf{t} : \mathsf{a} \rangle\rangle$ edge: $\langle\langle \_, \mathsf{t}, \mathsf{t} : \mathsf{a} \rangle\rangle$
construct: PKey class: constraint scheme: $\langle\langle \mathsf{t\_pk}, \mathsf{t}, \langle\langle \mathsf{t}, \mathsf{ka_1} \rangle\rangle, \ldots \langle\langle \mathsf{t}, \mathsf{ka_n} \rangle\rangle \rangle\rangle$	constraint: count $\langle\langle \mathsf{t} \rangle\rangle$ = count $[\{\mathsf{v_1}, \ldots, \mathsf{v_n}\} \mid \{\mathsf{k}, \mathsf{v_1}\} \leftarrow \langle\langle \mathsf{t}, \mathsf{ka_1} \rangle\rangle; \ldots;$ $\{\mathsf{k}, \mathsf{v_n}\} \leftarrow \langle\langle \mathsf{t}, \mathsf{ka_n} \rangle\rangle]$
construct: FKey class: constraint scheme: use $\langle\langle \mathsf{t\_fk}, \mathsf{t}, \langle\langle \mathsf{t}, \mathsf{a_1} \rangle\rangle, \ldots \langle\langle \mathsf{t}, \mathsf{a_n} \rangle\rangle, \rangle\rangle$ s, $\langle\langle \mathsf{s}, \mathsf{b_1} \rangle\rangle$, $\ldots \langle\langle \mathsf{s}, \mathsf{b_n} \rangle\rangle$	constraint: $\mathsf{subset}[\{\mathsf{v_1}, \ldots, \mathsf{v_n}\} \mid \{\mathsf{k}, \mathsf{v_1}\} \leftarrow \langle\langle \mathsf{t}, \mathsf{a_1} \rangle\rangle; \ldots;$ $\{\mathsf{k}, \mathsf{v_n}\} \leftarrow \langle\langle \mathsf{t}, \mathsf{a_n} \rangle\rangle]$ $[\{\mathsf{v_1}, \ldots, \mathsf{v_n}\} \mid \{\mathsf{k}, \mathsf{v_1}\} \leftarrow \langle\langle \mathsf{s}, \mathsf{b_1} \rangle\rangle; \ldots;$ $\{\mathsf{k}, \mathsf{v_n}\} \leftarrow \langle\langle \mathsf{s}, \mathsf{b_n} \rangle\rangle]$

Table 2.1: Representing a Simple Relational Model in HDM

For example, a table staff(sid,name,#studentID), where #studentID denotes a reference to a foreign key attribute, would be modelled in the HDM by a *Table* construct ⟨⟨staff⟩⟩, three *Attribute* constructs ⟨⟨staff, sid⟩⟩, ⟨⟨staff, name⟩⟩ and ⟨⟨staff, studentID⟩⟩, a *PKey* construct ⟨⟨staff_pk, staff, ⟨⟨staff, sid⟩⟩⟩⟩, and a *FKey* construct ⟨⟨staff_fk_1, staff, ⟨⟨staff, studentID⟩⟩, student, ⟨⟨student, id⟩⟩⟩⟩, assuming there is another table student(id,name)) and that #studentID of staff references id of student.

2.4.3 The IQL Query Language

The AutoMed Intermediate Query Language (IQL) [60] is a typed, comprehension based functional query language. IQL subsumes query languages such as SQL-92 and OQL in expressiveness [61, 62]. Its purpose is to provide a common query language that queries written in various high level query languages (e.g. SQL, XQuery, OQL) can be translated into and out of. For example, an SQL query posed on a global schema can be automatically translated by AutoMed into an IQL query. This query is then formulated and optimized (a variety of query optimizers are implemented to optimize, annotate and evaluate IQL queries - see [63]), and then subqueries of it are translated by the data source wrappers into the query language supported by the data sources.

2.4.4 AutoMed Transformation Pathways

As discussed in Section 2.4.1, any high-level modelling language can be represented in the HDM, with each modelling constructed being defined as a combination of nodes, edges and constraints. For schemas specified in such modelling languages, AutoMed provides a set of primitive schema transformations that can be applied to schema constructs (Nodes, Edges and Con-

straints) [20]. For extensional schema constructs (Nodes and Edges), "add" and "delete" transformations can be applied for inserting and deleting, respectively, a schema construct c into and from a schema S. Associated with an "add" or "delete" transformation is a query, q, expressing how the extent of construct c can be derived from the existing or remaining constructs of S. "extend" and "contract" transformations act similarly to "add" and "delete" except that two queries are associated with them, indicating a range for the extent of c, with a lower and an upper bound, denoted by q_l and q_u respectively. The minimum extent of c is given by query q_l, which may take the constant value `Void` if no lower bound for this extent can be derived form S. The maximum extent of c is given by query q_u, which may take the constant value `Any` if no upper bound for this extent can be derived form S. These transformations are summarised in Table 2.2. For constraint schema constructs, "add" and "delete" transformations can be applied for inserting and deleting a schema constraint associated with a query q. AutoMed also provides a "rename" transformation that can be used to change the name of any schema construct of S. AutoMed primitives have been shown to be sufficient for defining the mappings in the major common schema transformation and integration scenarios [20, 58].

Transformation	**Syntax**	**Comment**
add	$addT(c, q)$	Add construct c and populate c with query q.
extend	$extendT(c, Range\ q_l\ q_u)$	Add construct c whose extent is bounded by q_l and q_u.
del	$delT(c, q)$	Delete construct c whose extent can be rederived with query q.
extract	$extractT(c, Range\ q_l\ q_u)$	Delete construct c whose extent is bounded by q_l and q_u.
rename	$renameT(c, n)$	Change the name of c to n.

Table 2.2: AutoMed Transformation Primitives

A sequence of these primitive transformations from one schema S_1 to another schema S_2 is termed a transformation pathway from S_1 to S_2. All source, intermediate and integrated schemas and pathways between them are stored in AutoMed's Schemas Transformations Repository (STR) (see the next section). AutoMed is a *both-as-view (BAV)* data integration system since add/extend transformations can be reversed by delete/extract transformations with the same arguments[1] and vice versa [64]. Each rename transformation is reversed by swapping its two arguments. As discussed in [64], BAV subsumes the GAV and LAV approaches in the sense that BAV transformations can be used to represent both GAV and LAV mappings and GAV and LAV mappings can be extracted from the BAV transformations. As discussed in [65], BAV also subsumes the GLAV approach.

In addition, AutoMed also supports the *id* transformation. The *id* transformation can be used to connect two schemas that contain the same set of schema constructs. Using this transformation, the extents of such constructs can be integrated with different semantics, such as *append*, *union*, *intersect* and *choose* [65]. In this thesis, we assume that union semantics are applied.

2.4.5 AutoMed Architecture

AutoMed has a metadata repository composed of two main components, the Model Definitions Repository (MDR) and the Schemas and Transformations Repository (STR). MDR defines how each construct of a modelling language is represented as a combination of nodes, edges and constraints in HDM. STR contains the AutoMed representation of the data source, intermediate and global schemas and the transformations between them. AutoMed Wrappers are created for transforming the schema information in the data sources into the HDM representation. There are Wrappers available for a variety of

[1]The add/extend transformations are from S_1 to S_2 and delete/contract transformations are from S_2 to S_1.

structured and semi-structured data sources. Wrappers also transform the reformulated IQL queries into the query languages supported by the data sources.

2.5 Summary

In this chapter, we discussed the heterogeneity issues that cause data integration to be complex and error-prone. We also reviewed research on the tools and tasks for data integration, the quality issues considered in the DI context, and quality frameworks for supporting quality assessment tasks.

In our analysis of related work on DI quality assessment, we have identified a number of issues that have not been addressed in the literature, which provides the motivation for our own work:

Much previous data integration research has focused on generating the integrated resource semi-automatically. Many tools have been developed, such as schema matching and mapping generation tools, and matching and mapping algorithms have been implemented in such tools [17]. Ontologies may also be used to assist such tasks, based on ontologies' richer semantic and formal representation features. However, there is still a lack of research into quality assessment in the DI context.

Some research developed in this area has adopted quality definitions from other domains, such as information and data quality [39, 40, 41]. The motivation for this research was to measure the quality of the information retrievable from an integrated resource with respect to the quality of information available from the data sources. However, such work is only applied at the query answering stage of the DI process and does not contribute to improving the quality of the integrated resource from the perspective of the users' requirements.

The research proposed in [43, 44] focuses on the quality of an integrated

resource and is close to our research. However, measurement methods only were proposed in these works, and a systematic analysis of the reason why the quality of an integrated resource may be low and how this could be improved is lacking.

The authors in [2] took a systematic view of quality issues in the data warehouse context and proposed a formalised quality framework. However, the quality criteria and definitions in this work were adopted from other domains and they do not match all the characteristics of DI. The reasoning methods for inferencing in their quality framework are simple and the functionality is limited.

Across all research works reviewed in this chapter, none has suggested how the integrated resource could be modified in order to improve its quality. Instead, most of these works focus on answering users' queries making use of knowledge of the quality of the data sources, such as information completeness or correctness, and query plans that can be executed to extract relatively complete information are then generated. Such work has a strong assumption that a complete set of mappings is created in order for the query plan generation tool to function correctly. However, this can be very difficult to achieve in real data integration projects. In addition, the quality measurement results generated from such query planning approaches are not used to modify elements of the integrated resource, such as mappings, or the global schema in order to achieve better quality. Third, handling quality requirements from different users has not been studied in detail, although there has been some work on identifying different categories of users of a DI application [2, 57]. Fourth, very few research works have contributed to the refinement of integrated resources with respect to users' quality perspectives.

In our research, the above points have been considered in designing our quality framework and architecture, and also in our definitions of the qual-

ity criteria and quality factors. In Chapter 3, we will introduce our quality assessment approach, including a data integration methodology with quality assessment functionality embedded within it, a quality framework and quality measurement methods.

Chapter 3

Requirements for a Quality Framework and Architecture for DI

As indicated in the previous chapter, a variety of users play an important role during the DI process, including the pre-integration, integration and post-integration phase. The requirements of the users at different DI stages are important to our design of a quality framework and architecture for DI, and a comprehensive review and analysis have been presented in the previous chapter. In this chapter, we first study and analyse the previous research and the outcomes of an interview with data integrators in order to develop our approach in Section 3.1. We then discuss our data integration methodology with embedded quality assessment functionality in Section 3.2. A case study is then presented in Section 3.3 that will be used in the presentation of our framework and architecture in the later chapters. Section 3.4 summaries this chapter.

3.1 Analysis of Related Work and Interview with Integrators

From the review of related research work in Chapter 2, we see that the data integration process is a complex one and the integrated resource generally needs to be defined, implemented, evaluated and refined iteratively [66, 67]. The tasks in the DI process are affected by many factors, such as the requirements from the users [56], the integrators' knowledge of the application domain [33], and the complexity of the data sources arising from the information heterogeneities between them [68]. In general, a DI process is composed of the information gathering stage, the integration stage and the evaluation stage. Refinements of the integrated resource are then made and the data integrators generally need to apply the DI process iteratively [66]. In our research, we discuss these stages using the terms: the *pre-integration* phase, the *integration* phase and the *post-integration* phase. These three DI phases are applied iteratively in order to define and refine an integrated resource that will be closer to users' expectations.

In the *pre-integration* phase, the integrators need to understand the data sources to be integrated and also the application domain. Some existing DI tools can assist data integrators in this phase, such as tools for exploring the contents of data sources (database browsers), schema visualisation tools, tools for identifying data items that violate integrity constraints applied to the data sources (data cleansing tools), and tools for linking items in data sources with their real-world meanings, e.g., through a predefined domain or upper ontology [15, 16]. Additional assertions may be created by data integrators for annotating the results of these tools [28], indicating information that can be used during the integration phase, such as characteristics of the data sources [40], the users' requirements [56], inconsistencies between information stored in the data sources, and inconsistencies between such

information and their real-world semantics. In this phase, data integrators also need to understand the users' expectations of the interface for accessing the integrated resource, i.e., the *GS*. The users' requirements are rich and cover three aspects: user-defined assertions on the integrated resource based on their domain knowledge, users' specifications of the DI quality criteria, and the quality requirements on the integrated resource from different users' perspectives [56]. Different users have their own definitions of DI quality and their own quality requirements. Such definitions and requirements may or may not be consistent with each other and need to be assessed in their totality. This has not been investigated intensively prior to our work.

During the *integration* phase, mappings need to be generated, defining the transformations of information from the data sources to the *GS*. This process is usually undertaken in two phases, the matching phase and the mapping phase. In the matching phase, possible correspondences between schema constructs are created either manually or semi-automatically, for example using schema matching tools [17, 18]. Such correspondences may be correct or incorrect, precise or imprecise, due to the quality and heterogeneity of the data sources or capability of the matching tools. Correspondences may also be defined by the users manually to resolve the deficiencies of the matching tools, or the users may need to define some additional correspondences based on their knowledge of the application domain [67]. In the mapping phase, mappings are generated defining how information from the data sources can be extracted and transformed to comply with the *GS* [3]. The quality of the mappings can be affected by many factors, for example, the quality of correspondences generated from the matching phase, the complexity of data sources, the design of the *GS*, the capability of the supporting integration framework where some information may not be able to be represented [67], and the experience of data integrators and users where inexperienced integrators may define inaccurate mappings. From the

schemas of the data sources, the *GS* and the mappings, an integrated resource can then be formed. We discovered that, in practice, integrators may find it difficult to establish suitable mappings that allows the query answering to operate correctly due to the above factors and this problem has not been investigated in detail.

During the *post-integration* phase, the integrators need to analyze the quality of the integrated resource created in the integration phase. The related research work identified several quality criteria such as the 'correctness' of the DI setting [40], the 'completeness' of the information returned from the *GS* [39, 41], the 'consistency' of such information, the query 'performance' of the integrated resource, and the 'usefulness' of the information extractable from the data sources. Some research [56, 57] experienced that different users could have their own view of these quality criteria. For example, end-users may express their own view of the completeness of the integrated resource as the amount of information extractable from the *GS*. In contrast, data integrators may consider the completeness of the integrated resource as the degree of coverage of users' requirements relating to the design of the *GS*. In addition, different users may have different quality expectations. For example, users may consider an integrated resource to be of good quality if it is complete and consistent with respect to their quality definitions. Integrators may consider an integrated resource to be of good quality if it contains accurate information, leaving the completeness criterion as an optional requirement. Such different quality expectations may or may not be consistent, meaning there may exist conflicts across different users' quality expectations. Each factor relating to users' requirements and data heterogeneity also has individual impact on the overall quality of the integrated resource and these factors need to be assessed individually and also in their totality. Very few research works have considered this in the data integration domain.

In addition to the review of related research in the previous chapter, in order to identify users' requirements and gather first-hand experience of the DI process, we organized an interview with three data integrators in 2008. These people were experienced in the practical aspects of data integration and had been involved in different stages of several different data integration applications. The details of this interview are recorded in Appendix A.

The interviewees' responses reinforced many of the points emerging from the analysis of related work. They also noted that many ad-hoc methods are used to determine the quality of an integrated resource. These include running ad-hoc queries over the integrated resource and examining the results of such queries, or using test cases to represent the data sources and examining the extents of the (virtual) constructs in the *GS*. The integrated resource may need to be assessed fully or partially using such methods depending on the users' priorities. These assessment results could assist the integrators to refine the integrated resource in various ways and this has been considered in very few research works. For example, the integrator may change or ask the users to change the assertions on the data source schemas and expect more accurate correspondences to be returned from the matching process, or the integrator may modify the mappings to include more data sources in order to obtain more complete information. In addition, the data integrators interviewed gave an example where users first identify the information that needs to be accessible via the *GS*. A first version of the *GS* could then be created, such as using one data source schema as the *GS* or using an existing *GS* defined from an earlier integration effort. This is unlikely to be suitable as the final *GS* and needs to be improved during the subsequent DI phases.

The above analysis of the related work and the practical experiences of data integrators motivate our DI approach as introduced below.

3.2 Our Data Integration Methodology

As we discussed in Section 2.5 of Chapter 2 and in Section 3.1 above, the common data integration methodology, comprising mainly schema matching, schema mapping and possible refinement tasks, is not capable of capturing and assessing users' requirements with respect to their different quality perspectives of the integrated resource. To meet this need, we propose in this thesis a DI methodology that contains a requirements gathering phase, an integration domain learning phase, an integration phase and a quality assessment phase. The integration process is then applied iteratively to refine the integrated resource with respect to the quality assessment results and in this way an integrated resource with better quality can be achieved. This DI methodology is illustrated in Figure 3.1 and is discussed in detail below.

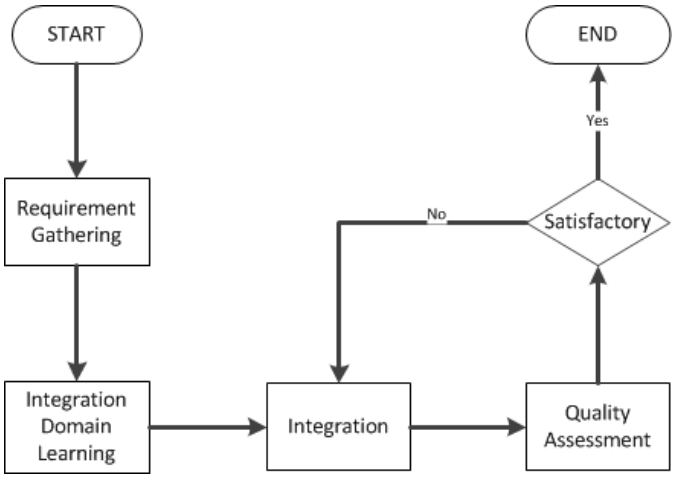


Figure 3.1: Iterative DI Methodology with Quality Assessment Functionality

Requirements Gathering Phase focuses on collecting and analysing users' requirements, and expressing these requirements systematically throughout the integration process. Such users' requirements relate to two aspects:

First, we want to collect users' requirements with respect to additional knowledge of the integration domain. Such user-defined knowledge may be more accurate than the domain knowledge extracted from the existing domain knowledge base, such as the domain ontology. However, users may not be able to define such knowledge without understanding the data sources. Therefore, this phase may need to interact with the next integration phase. In this case, the users' requirements may be expressed as text documents or assertions on schemas.

Second, we want to understand the users' requirements relating to the desired quality of the integrated resource. Such requirements may be expressed in various ways, such as logical expressions over the quality hierarchy, a prioritised list of requirements and the thresholds of the quality measurement methods. Users' requirements may relate directly to the functionalities of the measurement methods, such as supporting user-defined queries and defining assertions from users' knowledge. This phase may involve discussions with users of the integrated resource, identifying their role in the integration process and the requirements associated with their roles, and collecting knowledge of the integration domain from them.

Integration Domain Learning Phase focuses on understanding the data sources and additional domain knowledge if necessary. This phase involves understanding the integration domain, analysing the data sources that are to be integrated, deciding on the integration approach to be adopted (e.g. GAV, LAV, a combination of these), identifying correspondences between schema constructs through schema matching, and designing a global schema and a first version of the mappings between this and the local schemas, aiming towards meeting the users' quality requirements as currently expressed.

Integration Phase focuses on refining the mappings and the global schema by translating the local and global schemas into a corresponding OWL representation, defining additional users' assertions on these using extra domain knowledge, applying ontology matching to the extended OWL schemas, and refining the mappings from the local schemas to the global schema and possibly the global schema itself. The result is an integrated resource including the local schemas, the global schema, the set of mappings, and the set of users' assertions.

Quality Assessment Phase focuses on assessing the quality of the integrated resource with respect to the users' quality requirements. This phase involves extracting information from the integrated resource that could be used in the quality measurement process, applying the quality measurement methods, and comparing the measurement results with the users' quality requirements. The data integrator can use the results of the quality assessment to make changes to the integrated resource in order to achieve higher quality with respect to the users' requirements. The quality assessment results can also be referred to the end-users, by the data integrator, for their further feedback and recommendations.

The above Integration and Quality Assessment phases are applied iteratively in order to modify the integrated resource with respect to the users' feedback until an integrated resource with quality that is acceptable to the users is generated.

3.3 Case Study

To illustrate our approach to DI quality assessment and improvement, we now introduce a simple case study relating to the Higher Education (HE)

domain. This case study is used to demonstrate our quality framework, quality criteria and architecture in the following chapters. This case study relates to a simple domain and assertions can be generated based on our own knowledge of the domain. The case study is composed of three data sources, a global schema and a set of users' requirements. Database 1 is owned by the University Administration Office and it contains detailed descriptions of the degree programmes, the courses and the staff of a university. Database 2 is maintained by the Undergraduate School containing detailed information about the undergraduate studies of all the undergraduate students of the university and also of postgraduate students who are taking some advanced undergraduate courses. Database 3 contains detailed information of the postgraduate studies of postgraduate students of the university and is maintained by the Postgraduate School.

In our case study, the university Examination Office requires access to combined information from the three data sources and a suitable virtual global schema is created to support this need. We assume that a Higher Education (HE) domain ontology is available to the data integrator in OWL format. This domain ontology represents only the general structure of HE institutions. More specific information is introduced by additional users' assertions that express users' institution-specific knowledge. We next describe the three data sources, the global schema, the HE domain ontology and the additional assertions of our case study. Figure 3.2 lists the notation we use in the schema diagrams. One or more foreign keys may be assigned to the same attribute in a table as it may be linked with attributes that are represented in one or more other tables. The three data sources are managed in RDBMS. The local and global schemas and their ontology representation are wrapped and represented using the HDM data model and stored in the AutoMed STR repository. Other data models such as XML could be stored in the same way. The mappings are also stored in the same repository.

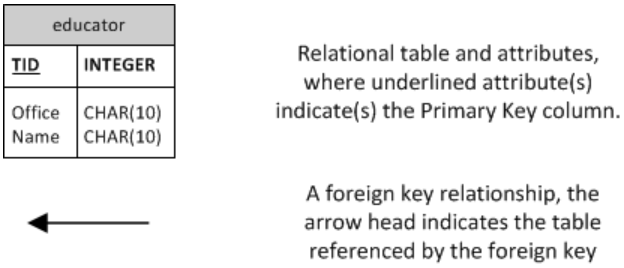


Figure 3.2: Notations used in the Case Study Schemas

3.3.1 Case Study - Data Sources

Local Schema 1 Local Schema 1 (LS1) contains detailed descriptions of the degree programmes, the courses and the staff of a university (see Figure 3.3 and 3.4).

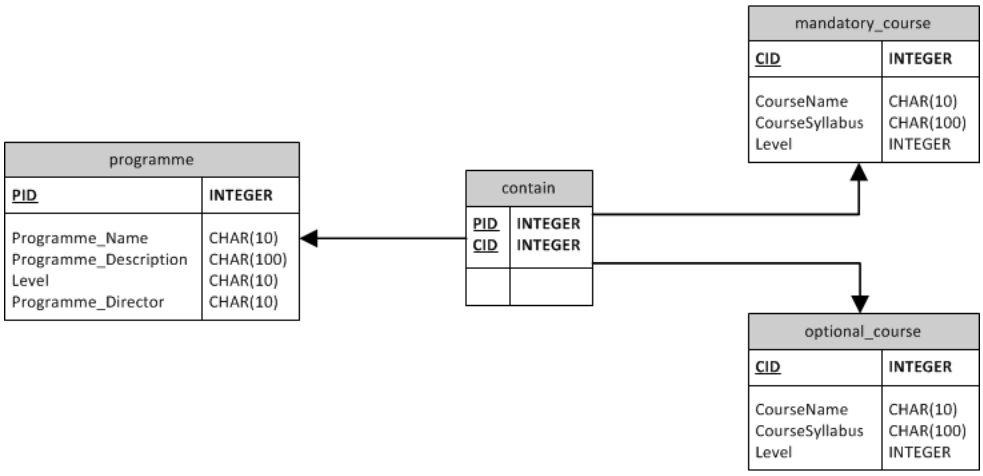


Figure 3.3: Local Schema 1 (LS1) - Programme - Course

In Figure 3.3, the *programme* table contains a unique programme ID, a programme name, a short description and a programme level indicating whether the programme is an undergraduate or postgraduate programme. A *Programme_Director* attribute is also included in the *programme* table and it has a value of null for all instances of the table. A programme contains

many courses and such courses are categorised as mandatory or optional. The number and the type of these courses being assigned to each programme depend on the specifications of the programmes. Each course has a unique course ID, a course name and a course syllabus. The course Level is an integer between 1 and 4 indicating in which year of the programme this course is taken by students on the programme. Three foreign keys are assigned to the *contain* table one referencing the *PID* attribute in the *programme* table and the others referencing the *CID* attributes in the *mandatory_course* and *optional_course* tables respectively.

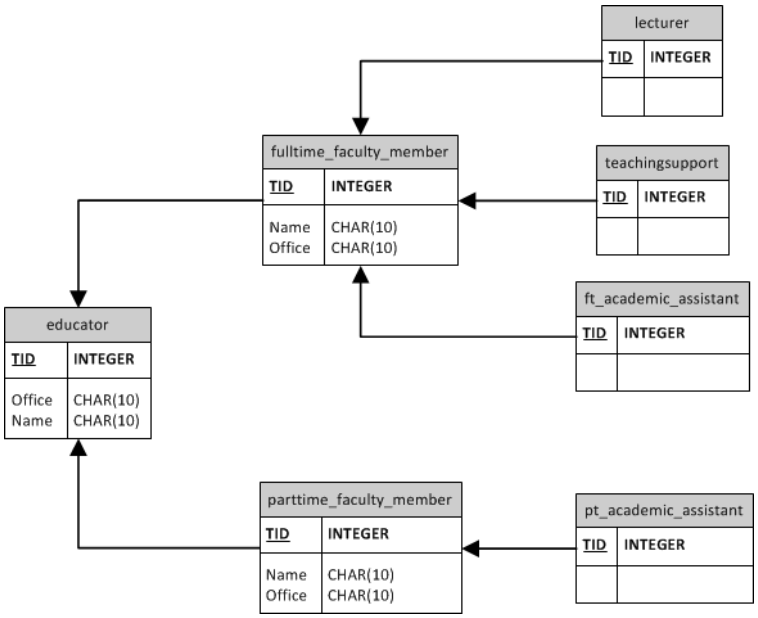


Figure 3.4: Local Schema 1 (LS1) - Staff

Figure 3.4 contains information about university staff, called educators. The *educator* table contains a unique ID, a staff name and an office location. Educators are categorized into three main groups with respect to their roles: lecturers, teaching support staff and academic assistants. Teaching support staff are university employees who do not have teaching duties but support

the running of courses. Academic assistants are students who are studying on a PhD programme and are also university employees who support the teaching side of courses. Educators are also categorised by their employment status as full-time or part-time and two foreign key are defined in the *fulltime_faculty_member* and *parttime_faculty_member* tables referencing the *TID* attribute in the *educator* table. All lecturers and teaching support staff must be employed as full-time faculty members while academic assistants can be either full-time or part-time faculty members. Four foreign keys are defined in the *lecturer*, *teachingsupport*, *ft_academic_assistant* and *pt_academic_assistant* tables, the first 3 referencing the *TID* attribute in the *fulltime_faculty_member* table and the last referencing the *TID* attribute in the *parttime_faculty_member* table. These foreign keys represent the sub-class relationships as described above.

Local Schema 2 Local Schema 2 (LS2) contains detailed information about the undergraduate studies of all the undergraduate students of the university and also of postgraduate students who are taking some advanced undergraduate courses (see Figures 3.5, 3.6 and 3.7).

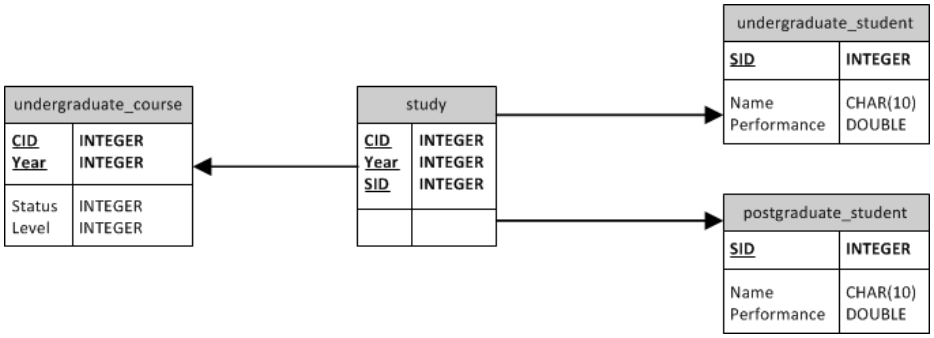


Figure 3.5: Local Schema 2 (LS2) - Student - Course

In Figure 3.5, the *undergraduate_course* table contains a unique course ID and a Year attribute, indicating the calendar year this course has been

run, e.g., 2009, 2010, 2011. The composition of course ID and year is the primary key of the *undergraduate_course* table. The *Status* attribute indicates the average of the results of all students who have registered with this course in *Year* and the *Level* attribute indicates the year this course is taken by students on a programme (i.e., an integer between 1 and 4). Undergraduate courses can be taken by undergraduate students. Undergraduate courses at level 4 can also be taken by the postgraduate students. The *undergraduate_student* and *postgraduate_student* tables contain the student ID, the student name and performance attributes. The performance attribute represents the average of the results of all courses this student has taken. The course registration information is contained in the *study* table, which is composed of the student ID, the course ID and the year this student has taken this course. Each undergraduate student can register for up to 8 undergraduate courses each year and each postgraduate student can register for up to 2 Level 4 undergraduate courses each year. Three foreign keys are defined in the *study* table, one referencing the *CID* and *Year* attributes in the *undergraduate_course* table and the others referencing the *SID* attribute in the *undergraduate_student* and *postgraduate_student* tables respectively.

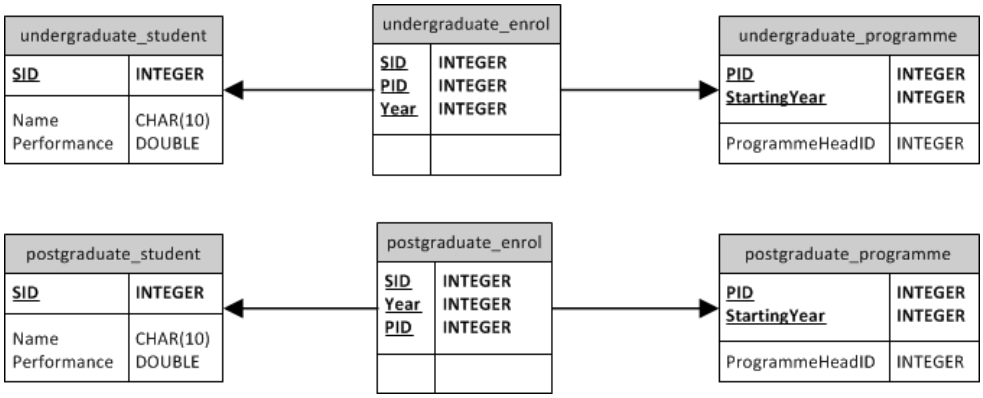


Figure 3.6: Local Schema 2 (LS2) - Student - Programme

In Figure 3.6, the *undergraduate_programme* and *postgraduate_programme*

tables each contains a programme ID and the starting year of the programme as the composite primary key. Each programme is led by one programme head, who is also a member of the university's staff and identified by their staff ID. The programme enrolment details are contained in the undergraduate enrol and postgraduate enrol tables for the undergraduate and postgraduate programmes respectively. Each undergraduate programme can be enrolled on by up to 100 undergraduate students and each postgraduate programme can be enrolled on by up to 50 postgraduate students. Each student can only enrol on one programme in each year. These constraints are represented by the foreign keys defined in the *undergraduate_enrol* and *postgraduate_enrol* tables. Two foreign keys are defined in the *undergraduate_enrol* table, one referencing the *SID* attribute in the *undergraduate_student* table and the other referencing the *PID* and *StartingYear* attributes in the *undergraduate_programme* table. Corresponding foreign keys are defined in the *postgraduate_enrol* table.

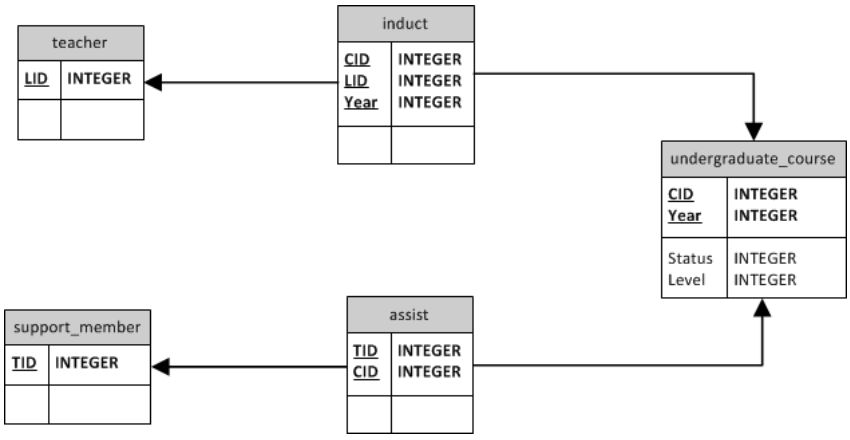


Figure 3.7: Local Schema 2 (LS2) - Staff - Course

In Figure 3.7, the *teacher* table contains the staff ID *LID* of the staff who teach undergraduate courses. Each teacher can only teach one undergraduate course and vice versa. The support members are both the teaching

support staff and the academic assistants. Each teaching support staff or academic assistant can support many undergraduate courses. Each course can be assisted by one teaching support staff and one academic assistant. Two foreign keys are defined in the *induct* table, one referencing the *LID* attribute in the *teacher* table and the other referencing the *CID* and *Year* attributes in the *undergraduate_course* table. Two other foreign keys are defined in the *assist* table, one referencing the *TID* attribute in the *support_member* table, and the other referencing the *CID* and *Year* attributes in the *undergraduate_course* table.

Local Schema 3 Local Schema 3 (LS3) contains detailed information about the postgraduate studies of postgraduate students of the university (see Figures 3.8, 3.9 and 3.10).

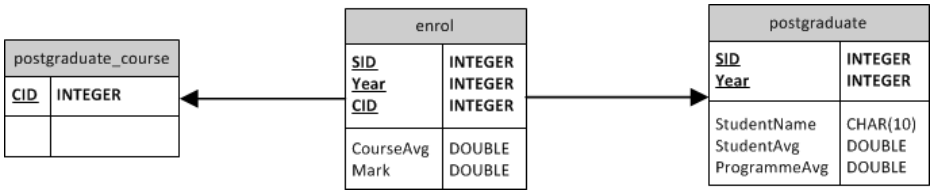


Figure 3.8: Local Schema 3 (LS3) - Student - Course

In Figure 3.8, the *postgraduate* table represents information about postgraduate students including: the student ID, the year this student is studying, the student name, the average of the results of all courses the student has taken this year and the average over all course results of all students in the same programme this year. The student ID and Year attributes comprise the composite primary key of the table. The *enrol* table contains information about course enrolment and is identified by the composite primary key of the student ID, the calendar year and the course ID. For the postgraduate study, each student can only enrol on up to 5 postgraduate courses each year. The *enrol* table also contains the course average attribute indicating

the average results of students who have registered for this course this year and the mark attribute indicating the student's own performance for this course. Two foreign keys are defined in the *enrol* table referencing the *CID* attribute in the *postgraduate_course* table and the *SID* and *Year* attributes in the *postgraduate* table respectively.

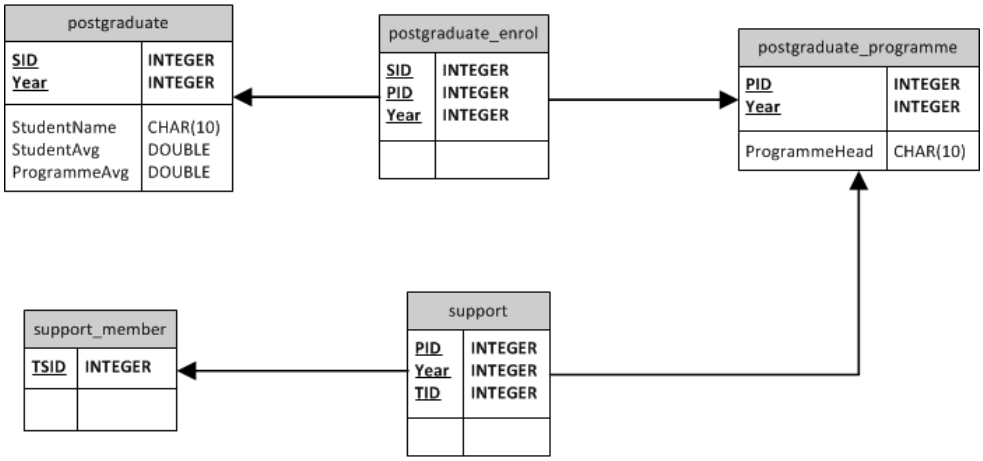


Figure 3.9: Local Schema 3 (LS3) - Student - Programme

In Figure 3.9, the *postgraduate_programme* table contains the programme ID, the starting year of the programme and programme head who leads this programme. The *ProgrammeHead* attribute contains the name of the programme head. Each programme is led by one programme head. The students' programme enrolment information is represented by the *postgraduate_enrol* table that indicates the students who have enrolled with the programme in a given year. One postgraduate programme can only be enrolled on by up to 50 postgraduate students each year. The support staff of postgraduate programmes are both the teaching support staff and the academic assistants, identified by their staff IDs and stored in the *support_member* table. Two foreign keys are defined in the *postgraduate_enrol* table referencing *SID* and *Year* attributes in the *postgraduate* table and the *PID* and *Year* attribute in the *postgraduate_programme* table respectively. Two other

foreign keys are defined in the *support* table referencing the *TSID* attribute in the *support_member* table and the *PID* and *Year* attributes in the *postgraduate_programme* table.

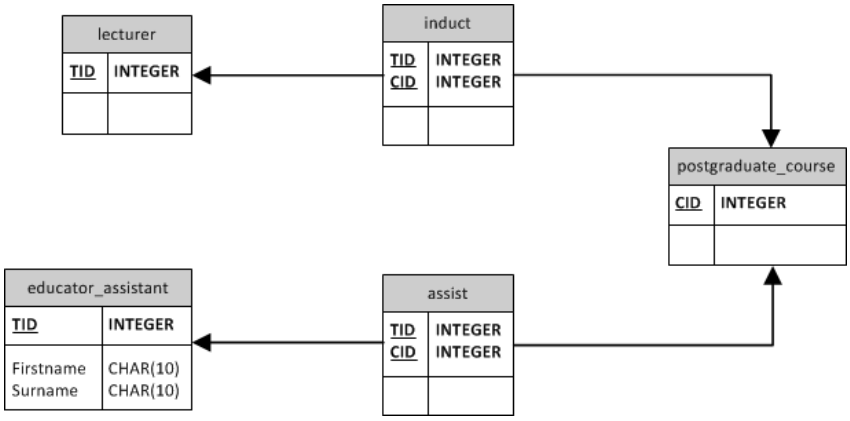


Figure 3.10: Local Schema 3 (LS3) - Staff - Course

In Figure 3.10, the lecturers who teach postgraduate courses are represented by the *lecturer* table. One course can be taught by up to 2 lecturers and each lecturer can only teach one postgraduate course. The *educator_assistant* table represents the academic assistants. Each assistant can assist with many postgraduate courses and each course can be assisted by up to 2 academic assistants. Two foreign keys are defined in the *induct* table referencing the *TID* and *CID* attributes in the *lecturer* and *postgradaute_course* tables respectively. Another two foreign keys are defined in the *assist* table referencing the *TID* attribute in the *educator_assistant* table and the *CID* attribute in the *postgraduate_course* table.

3.3.2 Case Study - Global Schema

In this case study, the global schema contains the combined information from the three local schemas (Figures 3.11 to 3.14).

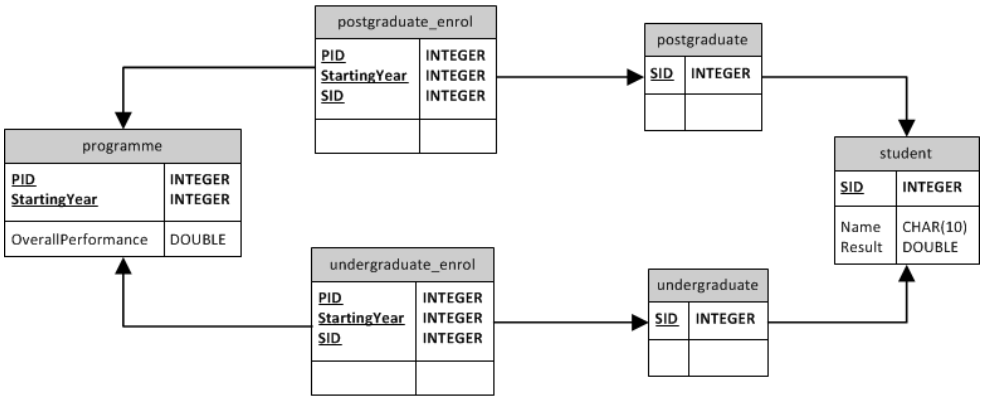


Figure 3.11: Global Schema (GS) - Programme - Student

In Figure 3.11, the *student* table contains information about each student who has registered on either an undergraduate or a postgraduate programme. The *undergraduate* and *postgraduate* tables represent the undergraduate and postgraduate students. The programme enrolment information is contained in the *undergraduate_enrol* and *postgraduate_enrol* tables. Each undergraduate programme can be registered for by up to 100 undergraduate students and each postgraduate programme can be registered for by up to 50 postgraduate students. Each student can register for only one programme. Two foreign keys are created in the *postgraduate_enrol* table referencing the *PID*, *StartingYear* attributes in the *programme* table and the *SID* attribute in the *postgraduate* table. Corresponding foreign keys are defined in the *undergraduate_enrol* table. Another foreign key is created in each of the *postgraduate* and *undergraduate* tables referencing the *SID* attribute in the *student* table.

In Figure 3.12, the *programme* table contains three attributes, the programme ID, the year the programme starts and the overall performance of all students who have registered with the programme. In each year, a programme comprises three types of courses, the undergraduate mandatory courses, the undergraduate optional courses and the postgraduate courses.

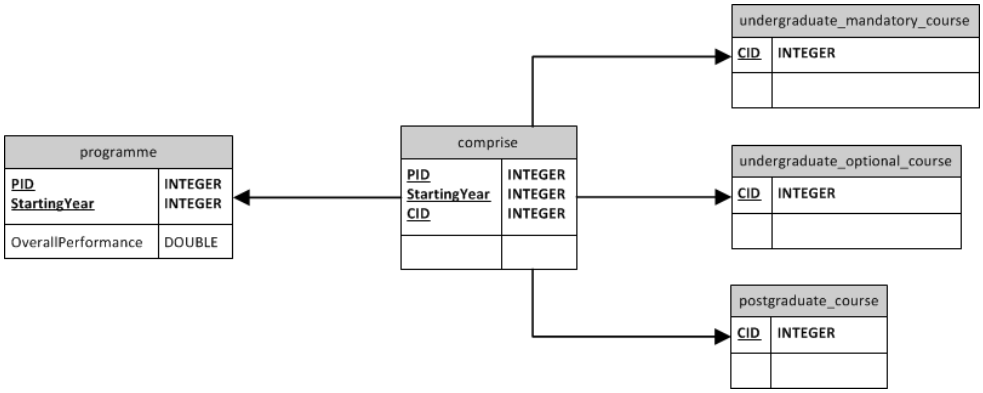


Figure 3.12: Global Schema (GS) - Programme - Course

An undergraduate programme comprises of up to 8 mandatory and optional undergraduate courses. A postgraduate programme comprises of up to 5 postgraduate courses and up to 2 undergraduate courses. Four foreign keys are defined in the *comprise* table, one referencing the *PID*, *StartingYear* attributes in the *programme* table, and three referencing the *CID* attributes in the *undergraduate_mandatory_course*, *undergraduate_optional_course* and *postgraduate_course* tables respectively.

In Figure 3.13, the *programme_head_teacher* table represents the programme heads, identified by the head teacher's staff ID *TID*. The table also contains the programme name, the programme description, the programme head's name and the programme level, which may be 'ug' or 'pg' for undergraduate and postgraduate programmes, respectively. The *programme_head_teacher* table is related to the *programme* table via the *direct* table that contains the programme head's staff ID, the programme ID and the starting year of the programme. Each programme head leads both the programme lecture members and the programme support members, as identified in the *lead* table by the programme head's staff ID, member's staff ID and the year of the programme. Two foreign keys are defined in the *direct* table, one referencing the *TID* attribute in the *programme_head_teacher*

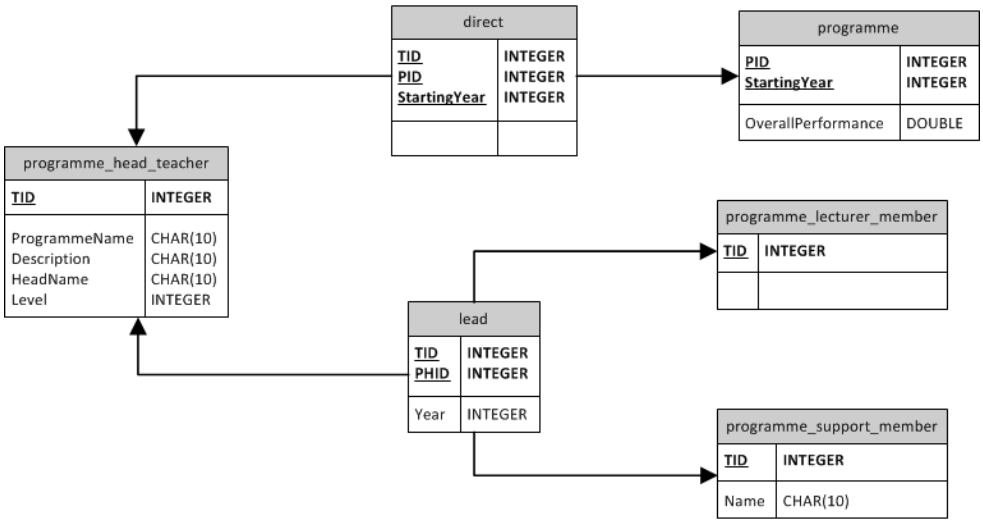


Figure 3.13: Global Schema (GS) - Programme Head - Programme

table, and the second referencing the *PID* and *StartingYear* attributes in the *programme* table. Three foreign keys are defined in the *lead* table, one referencing the *TID* attribute in the *programme_head_teacher* table, and the other two referencing the *TID* attributes in the *programme_lecturer_member* and *programme_support_member* tables respectively.

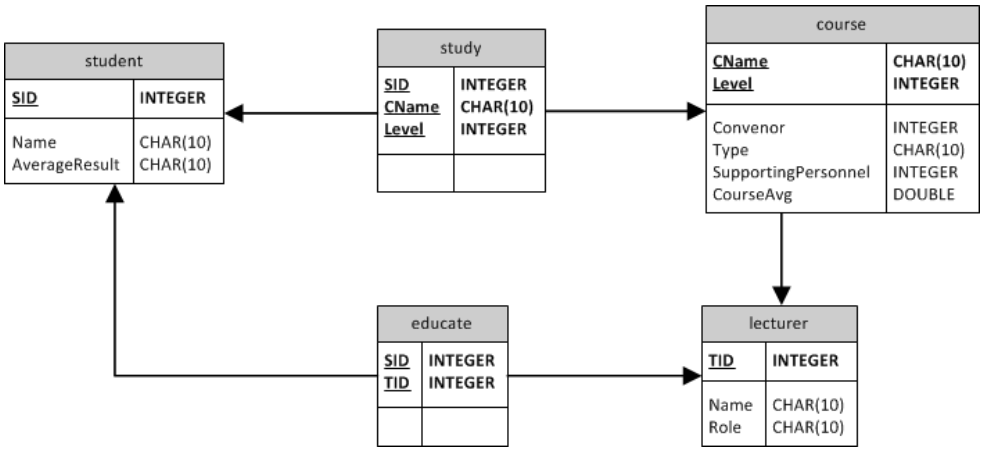


Figure 3.14: Global Schema (GS) - Staff - Course

In Figure 3.14, the *student* table contains the student ID, the student name and the average results over all courses the student has taken. The *course* table contains the course name and the level of the course as the table's primary key. The *course* table also contains the convenor attribute that represents the lecturer who inducts the course, the type attribute that represents if the course is an undergraduate or postgraduate course, the *supportingPersonnel* attribute representing the staff who support the *course* and the *courseAvg* attribute representing the results average over all students who have taken the course. The *lecturer* table contains the name and role of lecturers who teach the course. The *lecturer* table is related to the *student* table via the *educate* table representing which lecturers teach which students. Three sets of foreign keys are created. Two foreign keys are defined in the *study* table referencing the *SID* attribute in the *student* table and *CName* and *Level* attributes in the *course* table. Another two foreign keys are defined in the *educate* table referencing the *SID* attribute in the *student* table and the *TID* attribute in the *lecturer* table. The final foreign key is defined in the *course* table with the *Convenor* attribute referencing the *TID* attribute in the *lecturer* table, representing the lecturer who inducts the course.

3.3.3 Case Study - Domain Ontology

A domain ontology is used in our approach in order to represent additional real-world knowledge about the application domain with primitive constructs comprising concept, role, universal concept, bottom concept, conjunction, disjunction, existential quantification, value restriction, transitive role, inverse role, minimum cardinality and maximum cardinality. In this case study, we use a manually created HE domain ontology that provides general knowledge of a university environment (see Figure B.5 in Appendix B). This ontology has been created manually because of the lack of a suit-

able domain ontology in the HE area that is capable of being adapted to our case study. The relevant aspects to be included in this ontology are the *staff*, *student*, *course* and *programme* concepts and their sub-concepts, the properties representing the sub-class relationships between concepts, the equivalence relationships between concepts and also other more case-specific relationships as listed in Table 3.1 which is represented in Description Logic.

No.	Case-Specific Knowledge, expressed in Description Logic
B1	$postgradaute\_student \equiv \leq_1 enrol.postgraduate\_programme$
B2	$undergraduate\_student \equiv \leq_1 enrol.undergraduate\_programme$
B3	$postgraduate\_programme \equiv \leq_1 is\_directed\_by.programme\_leader$
B4	$undergraduate\_programme \equiv \leq_1 is\_directed\_by.programme\_leader$
B5	$programme\_leader \equiv \leq_1 direct.postgraduate\_programme$
B6	$programme\_leader \equiv \leq_1 direct.undergraduate\_programme$
B7	$lecturer \equiv \exists teach.postgraduate\_course$
B8	$lecturer \equiv \exists teach.undergraduate\_course$
B9	$programme\_leader \equiv \exists lead.lecturer$
B10	$undergraduate\_student \equiv \leq_8 study.undergraduate\_course$
B11	$postgraduate\_student \equiv \leq_2 study.undergraduate\_course$
B12	$postgraduate\_student \equiv \leq_5 study.postgraduate\_course$
B13	$undergraduate\_programme \equiv \leq_{200} is\_enrolled\_by.undergraduate\_student$
B14	$postgradaute\_programme \equiv \leq_{100} is\_enrolled\_by.postgraduate\_student$

Table 3.1: Case-Specific Knowledge

Description Logic (DL) is a family of formal knowledge representation languages based on the notions of *concepts* and *roles*. DL is characterised by constructors that allow complex concepts and roles to be built from atomic ones [1]. In Table 3.1, the symbol $\exists$ means that there exist some individuals associated with the concept following it; the symbol $\leq_n$ means that the maximum cardinality of individuals of the concept following it is n; the symbol $\geq_n$ means that the minimum cardinality of individuals of the concept following it is n. We refer readers to Section 4.3.1 in Chapter 4 for a full description of the DL language adopted in our research.

In this domain ontology, the properties representing the case-specific relationships that we consider are: 1) the *enrol* property indicating that an

undergraduate or postgraduate student can only enrol at most 1 undergraduate or postgraduate programme (see B1 and B2 in Table 3.1); 2) the *is_directed_by* property indicating a programme can be led by only one lecturer who is the leader of the programme (see B3 and B4 in Table 3.1); 3) the *direct* property indicating that a lecturer who plays the programme leader role can only lead 1 programme (see B5 and B6 in Table 3.1); 4) the *teach* property indicating a lecturer can teach some undergraduate or postgraduate courses (see B7 and B8 in Table 3.1); 5) the *lead* property indicating a programme leader can lead some lecturers (see B9 in Table 3.1); 6) the *study* property indicating an undergraduate student can study up to 8 undergraduate courses and a postgraduate student can study up to 2 undergraduate courses and up to 5 postgraduate courses (see B10-B12 in Table 3.1); 7) the *is_enrolled_by* property indicating an undergraduate programme can be enrolled by up to 200 undergraduate students and a postgraduate programme can be enrolled by up to 100 postgraduate students (see B13 and B14 in Table 3.1). The properties representing the sub-class and equivalence relationships are listed in Figure B.5 in Appendix B. The HE domain ontology is implemented using the OWL language that is a formal language developed with Description Logic as its logical foundation. We refer readers to Section 6.2.2 in Chapter 6 for the detailed discussion of the OWL language we adopted.

3.3.4 Case Study - Assertions

In our case study, users' assertions are expressed based on the ontology representation of the schemas. There are two reasons that we represent the schemas as ontologies. First, the data model used by the schemas may not be expressive enough with respect to the users' assertions. For example, in the Relational model, there does not exist a representation for representing the relationships between two table entities. If the users require to define

constraints between the extents of these entities, it is impossible to do so. Second, we use a domain ontology as the knowledge base of the integration domain. Representing the schemas as ontologies has the advantage of linking them semantically with this domain ontology.

In this thesis, we focus on data sources that contain information represented in the Relational model only for the purpose of fast implementation of the prototype. But in principle, data sources containing information in other data models can be integrated, knowledge expressed in such data sources can be expressed as an ontology, and the quality of the integrated resources can be measured by adopting our integration architecture since ontologies have a rich expressibility and can represent knowledge in various data models. The translation of a relational global schema into an equivalent OWL representation is undertaken using an algorithm based on [69], which describes the representation of relational databases in RDF. Similarly to [69], our translation of relational schemas into OWL can support both single-attribute and composite primary and foreign keys [38]. The purpose of this algorithm is to illustrate the transformation from relational schema to ontology so that users' assertions can be expressed on the ontology and linked with the domain ontology. The capabilities of the algorithm are sufficient to illustrate our approach. We refer readers to [70] for a survey of similar and more expressive algorithms.

We discuss this transformation method from the relational schema to an ontology and demonstrate how this ontological representation can be linked to the domain ontology as follows:

- For each relational table t in a schema S (which may be a data source schema or the global schema), if the name of t is a noun, create an ontology class c for representing t, where the name of c is in the format 'S_t' with t indicating the table's name. The extent of c is the same as the extent of t. Create a 'sub-class' relationship between c

and the concept c' in the domain ontology, if c' exists, such that c and c' can be matched semantically[1]. For example, given the table *programme_head_teacher* in *GS* and the *programme_leader* class in the domain ontology, a new class *GS_programme_head_teacher* is created as a sub-class of *programme_leader*. If there does not exist such a c' in the domain ontology, create c as a top-level class.

- For each relational table t in a schema S, if the name of t is a verb, create an ontology object property p for representing t, where the name of p is in the format 'S_t' with t indicating the table's name. The extent of p is the same as the extent of t. Create a 'sub-property' relationship between p and the ontology property p' in the domain ontology, if p' exists, such that p and p' can be matched semantically. Assert the ontology classes representing the relational tables relating to t as the domain and the range of the ontology property p. For example, given the table *undergraduate_enrol* in *GS* and the *enrol* property in the domain ontology, a new object property is created termed *GS_undergraduate_enrol* and *GS_undergraduate_enrol* is a sub-property of *enrol*. The domain and range of *GS_undergraduate_enrol* are *GS_undergraduate* and *GS_programme* respectively. If there does not exist such a p' in the domain ontology, create p as the top level object property.

- For each property p created in the above step, create the inverse property p^-, where the name of p^- is in the format '$S\_is\_p^{\sim}\_by$' with $p^{\sim}$ as the past tense of p. The domain of p^- is the range of p and the range of p^- is the domain of p. For example, given a property *GS_undergraduate_enrol*, its inverse property is *GS_is_Undergraduate*

[1]By "semantically matched" we mean here that both c and c' have the same real-world meaning.

Enrolled_by and its domain and range are *GS_programme* and *GS_under graduate* respectively.

- Ignore attributes, primary keys and foreign keys for simplicity purpose.

The extended domain ontology created by the above method allows all tables to be transformed into their corresponding ontology representations. With the sub-class relationships created between the ontology representations of the schema constructs and the classes in the original domain ontology, constraints applied in the original domain ontology can then be applied to the new sub-classes, allowing reasoning to be applied to the schema constructs for consistency checking. Assertions representing users' requirements on the integrated resource from the users' and integrators' knowledge of the application domain can subsequently be expressed on this extended domain ontology. For our case-study, these are listed in Tables 3.2, 3.3, 3.4 and 3.5 (note that there are no such assertions on *LS1*).

No.	Assertion Description and in DL
A1	A cardinality constraint is placed indicating that one undergraduate student can study up to 8 undergraduate courses. $LS_2\_undergraduate\_student \equiv \leq_8 LS_2\_study.LS_2\_undergraduate\_course$
A2	A cardinality constraint is placed indicating that one postgraduate student can study up to 2 undergraduate courses. $LS_2\_postgraduate\_student \equiv \leq_2 LS_2\_study.LS_2\_undergraduate\_course$
A3	An undergraduate programme can be enrolled on by up to 100 undergraduate students. $LS_2\_undergraduate\_programme \equiv \leq_{100}$ $LS_2\_isUndergraduateEnrolledBy.LS_2\_undergraduate\_student$
A4	A postgraduate programme can be enrolled on by up to 50 postgraduate students. $LS_2\_postgraduate\_programme \equiv \leq_{50}$ $LS_2\_isPostgraduateEnrolledBy.LS_2\_postgraduate\_student$
A5	One teacher can teach only one undergraduate course. $LS_2\_teacher \equiv \leq_1 LS_2\_induct.LS_2\_undergraduate\_course$ $LS_2\_teacher \equiv \geq_1 LS_2\_induct.LS_2\_undergraduate\_course$
A6	One undergraduate course can only be taught by only one teacher. $LS_2\_undergraduate\_course \equiv \leq_1 LS_2\_isInductedBy.LS_2\_teacher$ $LS_2\_undergraduate\_course \equiv \geq_1 LS_2\_isInductedBy.LS_2\_teacher$
A7	One support member of each kind can support many undergraduate course. $LS_2\_support\_memeber \equiv \exists LS_2\_assist.LS_2\_undergraduate\_course$

A8	One undergraduate course can only be supported by only one support member of each kind. $LS_2\_undergraduate\_course \equiv \leq_1 LS_2\_isAssistedBy.LS_2\_support\_member$ $LS_2\_undergraduate\_course \equiv \geq_1 LS_2\_isAssistedBy.LS_2\_support\_member$

Table 3.2: Users' Assertions on *LS2*

No.	**Assertion Description and in DL**
A9	A postgraduate programme can be enrolled on by up to 50 postgraduate students. $LS_3\_postgraduate\_programme \equiv \leq_{50}$ $LS_3\_isPostgraduateEnrolledBy.LS_3\_postgraduate\_student$
A10	A postgraduate student can enrol on only one postgraduate programme. $LS_3\_postgraduate \equiv \leq_1 LS_3\_postgraduate\_enrol.LS_3\_postgraduate\_programme$ $LS_3\_postgraduate \equiv \geq_1 LS_3\_postgraduate\_enrol.LS_3\_postgraduate\_programme$
A11	A postgraduate student can register for up to 5 postgraduate courses. $LS_3\_postgraduate \equiv \leq_5 LS_3\_enrol.LS_3\_postgraduate\_course$
A12	A postgraduate course can be inducted by up to 2 lecturers. $LS_3\_postgraduate\_course \equiv \leq_2 LS_3\_isInductedBy.LS_3\_lecturer$
A13	A lecturer can induct only one postgraduate course. $LS_3\_lecturer \equiv \leq_1 LS_3\_induct.LS_3\_postgraduate\_course$ $LS_3\_lecturer \equiv \geq_1 LS_3\_induct.LS_3\_postgraduate\_course$
A14	A postgraduate course can be assisted by up to 2 TAs. $LS_3\_postgraduate\_course \equiv \leq_2 LS_3\_isAssistedby.LS_3\_educator\_assistant$
A15	One TA can assist any number of postgraduate courses. $LS_3\_educator\_assistant \equiv \exists LS_3\_assist.LS_3\_postgraduate\_course$
A16	Each programme can have only one head. $LS_3\_postgraduate\_programme \equiv \leq_1 LS_3\_hasHead.LS_3\_ProgrammeHead$ $LS_3\_postgraduate\_programme \equiv \geq_1 LS_3\_hasHead.LS_3\_ProgrammeHead$

Table 3.3: User's Assertions on *LS3*

No.	Assertion Description and in DL
A17	One programme can contain up to 8 undergraduate mandatory courses. $GS\_programme \equiv \leq_8 GS\_comprise.GS\_undergraduate\_mandatory\_course$
A18	One programme can contain up to 8 undergraduate optional courses. $GS\_programme \equiv \leq_8 GS\_comprise.GS\_undergraduate\_optional\_course$
A19	One programme can contain up to 5 postgraduate courses. $GS\_programme \equiv \leq_5 GS\_comprise.GS\_postgraduate\_course$
A20	One programme head can lead many programme lecturing members $GS\_programme\_head\_teacher \equiv \exists GS\_lead.GS\_programme\_lecture\_member$
A21	One programme head can lead many programme support members $GS\_programme\_head\_teacher \equiv \exists GS\_lead.GS\_programme\_support\_member$

Table 3.4: User's Assertions on *GS*

No.	Assertion Description and in DL
A22	The programme concept in the GS contains all programmes available from the LSs. $GS\_programme \equiv\ LS_1\_programme \cup LS_2\_undergraduate\_programme$ $\cup LS_2\_postgraduate\_programme \cup LS3 : postgraduate\_programme$
A23	The postgraduate student concept in the GS consists of the postgraduate students from both LS_2 and LS_3. $GS\_postgraduate \equiv LS_2\_postgraduate\_student \cup LS_3\_postgraduate$

Table 3.5: User's Assertions across LSs and GS

3.4 Summary

In this chapter, we have discussed and analysed the requirements of data integrators with respect to the DI process, and the assessment and improvement of DI quality. We also proposed a data integration methodology with embedded quality assessment functionality. We have presented a case study that will be used to illustrate the quality criteria and metrics proposed in Chapter 5. This case study will also be used to demonstrate the functionality and usefulness of our quality framework and architecture in Chapters 4 and 6 respectively.

In Chapter 4, we will present our quality framework designed to meet data integrators' requirements as analysed in this chapter.

Chapter 4

Quality Framework for Data Integration

In the previous chapter, we have identified the importance of users' requirements throughout the different DI phases, from the analysis of the interview with data integrators and the related research surveyed in Chapter 2. The need for quality assessment methods suitable for the DI context has been raised and our research focuses in particular on assessment of the quality of integrated resources. In this chapter, we propose a *Quality Framework for Data Integration (QFDI)* that is designed specifically for the DI context. The chapter is organised as follows. In Section 4.1, we discuss the objectives and the characteristics of our quality framework. We then propose the framework specification and a discussion of the reasoning capability that needs to be supported by the framework in Sections 4.2 and 4.3 respectively. In Section 4.4, we describe formally the data resources that are assumed in the development of our quality criteria and quality factors in Chapter 5. Section 4.5 provides a summary of this chapter.

4.1 Design Objectives for the QFDI

As discussed in the previous chapter, the quality of integrated resources can have various definitions and can be measured in different ways. Different end-users may also have their own quality requirements of the integrated resources from their own perspectives. In our research, we propose a formal framework, QFDI, for representing these different views of the quality of integrated resources and we apply formal inference methods for assessing their quality. This framework is applied in the *Quality Assessment* phase within the data integration methodology proposed in Section 3.2 in the previous chapter. If the users' quality requirements can be satisfied within our quality framework, the integration process will then finish. Otherwise, the integrators may revise the integrated resource according to the feedback obtained from this framework and apply the quality assessment phase again.

The design objectives of our framework are as follows:

First, the data integration process may involve multiple users with different roles, such as administrators, integrators and end-users. Different categories of users and their quality requirements have been discussed in various research works, such as [56, 57]. Such requirements may not be consistent, meaning that the same integrated resource cannot satisfy all requirements from different users and, therefore, either the integrated resource or the users' quality requirements need to be modified. A quality framework for the data integration context therefore has to be capable of representing these varieties and providing solutions to detect and resolve the possible inconsistencies or conflicts between the users' requirements.

Second, the quality framework needs to be capable of representing a quality hierarchy comprised of *quality criteria*, *quality factors* and the relationships between them. Quality criteria represent high-level definitions of quality, such as correctness, maintainability, usability. The definitions of such criteria are open to different interpretations and more precise qual-

ity definitions are defined by their associated quality factors. We consider that the quality criteria and the quality factors for interpreting these criteria have a parent-child relationship. There may also be additional user-specified relationships between different quality criteria or factors.

Third, the tradeoffs between different quality factors should be formally expressible and examinable in such a quality framework. As discussed in previous related research in the area of data quality [71], there may exist tradeoffs between different quality factors, especially in collaborative projects with different user perspectives, such as data integration. In this case, it may not be possible to find a common solution that satisfies all quality requirements and a compromise solution needs to be considered. In our research, we approach this problem by applying weights to individual quality factors and we calculate the overall quality measurement as $w_1 \times r_1 + \ldots + w_n \times r_n$, where r_i and w_i are the measurement of a quality factor i and its user-specified weight, respectively.

Fourth, in the DI context, quality is not only difficult to define precisely, but it is also not easy to measure due to the large numbers of elements that are involved in the data integration process, including the data, the schema constructs, the users' assertions and the mappings. In addition, there may also exist correlations between such elements. For example, there may be many schema objects referenced in a mapping, each such schema object having an extent consisting of some data from the data sources; such schema objects may also be restricted by additional constraints specified in the users' assertions. A quality framework in the DI context needs to be able to establish such correlations between the elements and their referenced quality measurement methods and between the elements themselves.

Fifth, as well as the existence of correlations between different elements involved in the DI process, difficulties may also arise during the refinement process that aims to achieve better quality of the integrated resource. Such

difficulties arise from the correlations between DI elements[1] that are referenced by different measurement methods. For example, suppose we have two quality criteria, c_1 and c_2. c_1 is the schema completeness quality criterion and c_2 is the mapping consistency criterion. c_1 is associated with a quality factor f_2 proposed in Chapter 5, which defines schema completeness as the degree of coverage of local schema constructs that provide overlapping but possibly partially complete information for the same global schema constructs. c_2 is associated with quality factor f_7 proposed in Chapter 5, which defines mapping consistency as the proportion of local schema constraints that are not violated by the new constraints introduced by the mappings. Suppose there exists a GAV mapping m that derives a global schema construct c_a by referencing a local schema construct c_b and c_a is derived by m only. Suppose m is used in a the measurement of f_2, and c_a and c_b are used in the quality measurement f_7. The quality refinement process suggests that m should be deleted in order to achieve better quality for f_2 and, therefore, that c_a should also be deleted from the global schema. This action will affect the measurement of f_7 since c_a no longer exists. A quality framework should capture the wider impacts of such changes and suggest the DI elements that are possibly affected by these changes. Chapter 5 discussed the full set of quality criteria and metrics.

4.2 Development of the Quality Framework

Taking account of the design objectives discussed above, we propose a *Quality Framework for Data Integration (QFDI)* that is composed of four major parts, termed *ITEM*, *METRIC*, *QUALITY CRITERI* and *USER*, as illustrated in Figure 4.1.

[1]By 'DI elements' we mean the data, the schema constructs, the users' assertions and the mappings that together make up an integrated data resource. See the discussion in

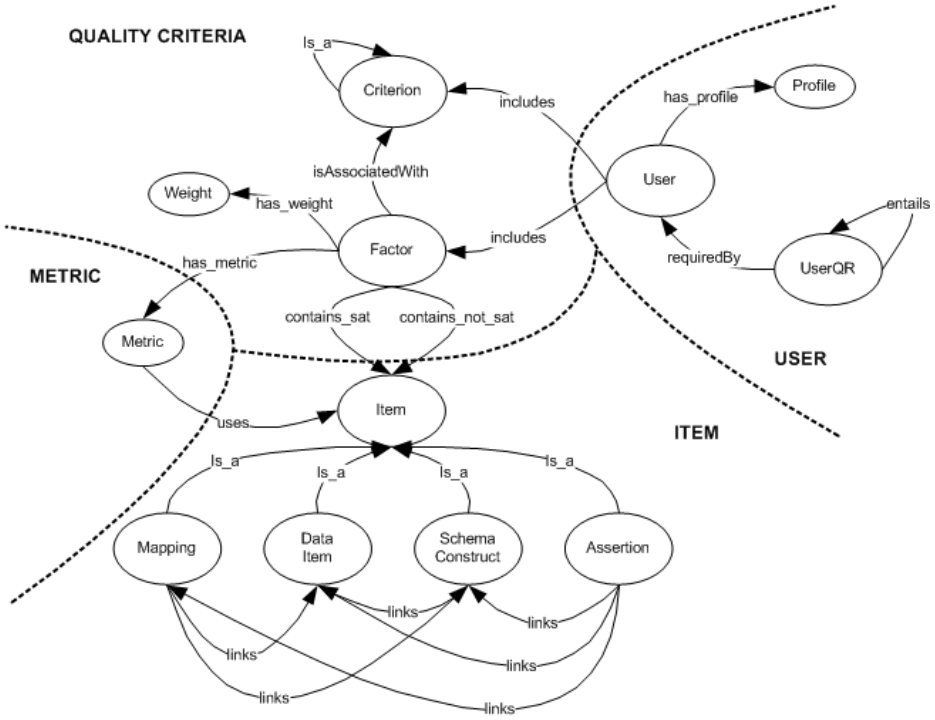


Figure 4.1: The Quality Framework for Data Integration (QFDI)

ITEM contains the representations of knowledge extractable from the elements comprising a DI setting. By 'elements' we mean the fundamental constituents of an integrated resource, including *Data Item*, *Schema Construct*, *Mapping* and *Assertion*. Assertions are defined by integrators so as to express domain-specific knowledge relating to an integrated resource. All of these are represented as sub-concepts of the *Item* concept. Links exist between these sub-concepts, represented as the *link* property, to represent how the instances of one concept relate to those of another.

In the METRIC part, different measurement methods (metrics) are represented by the *Metric* concept. Each metric is defined over instances of the Item concept in the ITEM part. The measurement results are stored as

Section 4.2.

instances of the Metric concept.

QUALITY CRITERIA contains the representation of the quality hierarchy as defined by the data integrator for a particular integrated resource. This hierarchy is built from two concepts, *Criterion* and *Factor*, and the relationships between them namely, *is_a* and *isAssociatedWith*. Each quality criterion may have several sub-criteria, linked by the *is_a* property. Each quality factor is associated with one quality criterion, using the *isAssociatedWith* property. Each quality (sub-)criterion can be associated with several quality factors. Each quality factor is associated with a quality metric in the METRIC part using the *has_metric* property. Such metrics can be defined using different methods and we discuss some of these in Chapter 5. The *Weight* concept is associated with the Factor concept and indicates the specific weight of each quality factor as defined by the users.

The *contains_sat* and *contains_not_sat* properties link the Factor concept and the Item concept. These properties represent the DI elements that satisfy and that do not satisfy a quality factor, respectively. These two properties are disjoint.

USER contains the *User* concept. Users with different roles may have different quality requirements. For example, end-users may have requirements regarding the amount and the consistency of information returned from a DI setting, whereas a data integrator may focus more on query performance requirements. Different user requirements are represented by the *UserQR* (User Quality Requirement) concept and they can be related by using the *entail* property, meaning that if the integrated resource satisfies one user requirement, the integrated resource also satisfies another user requirement that is entailed by the first one. Each User concept is composed of quality criteria, quality factors and the relationships between them. If the instances of the concepts in the QUALITY CRITERIA and METRIC parts are satisfactory for the users, then the users' quality requirements can be

regarded as being satisfied by the integrated resource. The *Profile* concept represents the overall quality of the integrated resource with respect to a specific user, calculated as $w_1 \times r_1 + \ldots + w_n \times r_n$, where r_i and w_i are the measurement of a quality factor i and its user-specified weight, respectively.

4.3 Reasoning Capability Required by the QFDI

As described previously, users can state various quality requirements on an integrated resource based on their different perspectives. In our framework, users' quality requirements are expressed as logic statements about the quality factors and relationships between them. The relationships we support are *subsumption* and *equivalence*.

4.3.1 Description Logic in Our Approach

In providing the reasoning capabilities required by the QFDI, we use a formal Description Logic language. Description Logic (DL) is a family of formal knowledge representation languages based on the notions of *concepts* and *roles*. DL is characterised by constructors that allow complex concepts and roles to be built from atomic ones.

We discuss here a basic DL language which is a subset of the $\mathcal{SI}$ DL [1] and which provides a limited but sufficient expressive power for our reasoning purposes in the QFDI. The $\mathcal{S}$ of $\mathcal{SI}$ extends the $\mathcal{ALC}$ DL [1] with the *transitive role* [72]. The $\mathcal{ALC}$ DL itself comprises constructors for concepts, universal concept, bottom concept, negation, intersection, union, restriction and existential quantification. Transitive role means that if a pair (x, y) is an instance of a role r and a pair (y, z) is an instance of r, then it is also true that the pair (x, z) is an instance of r. The $\mathcal{I}$ of $\mathcal{SI}$ extends the

$\mathcal{ALC}$ DL with the *inverse role*, where if a pair (x, y) is an instance of a role r, then the pair (y, x) is also an instance of r. In addition, in order to express cardinality information that is used in the case study in the previous chapter, we extend the DL language with a simple cardinality property $\mathcal{N}$. A cardinality property can be written as $\geq_n R.C$ (minimum cardinality) or $\leq_n R.C$ (maximum cardinality), where n ranges over the nonnegative integers.

In order to define the formal semantics of our quality hierarchy and quality requirements in this DL language, we consider an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that consists of a non-empty set $\Delta^{\mathcal{I}}$ which is the domain of the interpretation. A domain is a set of existing individuals in the world we are modelling and an interpretation function, $\cdot^{\mathcal{I}}$, which assigns sets derived from the domain $\Delta^{\mathcal{I}}$ to every statement in this language. Table 4.1 lists the full syntax of the DL language adopted in this thesis.

Syntax	**Semantics**	**Explanation**	
C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	a concept	
R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	a role	
$\top$	$\Delta^{\mathcal{I}}$	the universal concept	
$\emptyset$	$\emptyset$	the bottom concept	$\mathcal{S}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	conjunction	
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	disjunction	
$\exists R.C$	$\{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	existential quantification	
$\forall R.C$	$\{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$	value restriction	
R^+	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$	transitive role	
R^-	$\{\langle x, y\rangle \mid \langle y, x\rangle \in R^{\mathcal{I}}\}$	inverse role	$\mathcal{I}$
$\geq_n R.C$	$\{a \in \Delta^{\mathcal{I}} \mid\mid \{b \mid (a, b) \in R^{\mathcal{I}}\}\| \geq n\}$	minimum cardinality	$\mathcal{N}$
$\leq_n R.C$	$\{a \in \Delta^{\mathcal{I}} \mid\mid \{b \mid (a, b) \in R^{\mathcal{I}}\}\| \leq n\}$	maximum cardinality	

Table 4.1: Syntax of the DL we adopt

Terminology axioms state how concepts are related to each other. The terminology axioms we consider in our research are *inclusion*, $C \sqsubseteq D$, and

equality, $C \equiv D$, where C and D are concepts expressed in the syntax of Table 4.1. In our approach, such terminology axioms are used to express the users' quality requirements. The semantics of these axioms are defined as follows. An interpretation I satisfies an inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies an equality $C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. Equality can also be expressed using inclusion, since $C \equiv D$ is equivalent to $(C \sqsubseteq D) \wedge (D \sqsubseteq C)$. If $\mathcal{T}$ is a set of axioms of the above forms, then $\mathcal{I}$ *satisfies* $\mathcal{T}$ *iff* $\mathcal{I}$ satisfies each element of $\mathcal{T}$. $\mathcal{T}$ is also called the *TBox* in DL. More complex assertions can be created from these two basic ones plus negation. For example, $(C \sqcap D) \sqsubseteq E$, $(C \sqcap D) \equiv \emptyset$, $(C \sqcap D) \not\equiv \emptyset$, $(\forall R.C \sqcap D) \not\equiv \emptyset$.

In DL, there are two kinds of assertions. *Concept assertions* state that an individual a belongs to a concept C, denoted by $C(a)$. Its semantics are that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ where $a^{\mathcal{I}}$ is the interpretation of the individual a. *Role assertions* state that an individual c is a value of the role R for individual b, denoted by $R(b, c)$. Its semantics are that $(b^{\mathcal{I}}, c^{\mathcal{I}}) \in R^{\mathcal{I}}$. Such assertions are termed the *ABox* in DL, denoted as $\mathcal{A}$.

A set of individuals is denoted as $\{a_1, \ldots, a_n\}$, where $a_1, \ldots, a_n$ are the individuals' names. Such a set of individuals is interpreted as $\{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$.

In our approach, we adopt the *Unique Name Assumption* and the *Closed-World Assumption* as characteristics of our quality framework[2]. This is because in the QFDI, the quality (sub-)criteria and the factors are distinct; also, the set of DI elements that satisfy and do not satisfy a quality factor are generated from the quality metrics explicitly, and there are no other DI elements which could be classified as satisfying or not satisfying the quality factor.

Our quality framework is represented using the above DL language, where each oval in Figure 4.1 (except the USER part of the diagram) is

[2]The Unique Name Assumption [1] states that names or identifiers in $\mathcal{I}$ refer to different individuals or entities. The Closed-World Assumption [1] states that what is not currently known to be true, is false.

represented as a DL concept named by the text in the oval and each link is represented as a role named by the name of the link. The DI elements are represented as individuals of the *Data Item*, *SchemaConstruct*, *Mapping* and *Assertion* concepts and are associated with quality factors via the *contains_sat* and *contains_not_sat* roles.

Users' quality requirements are expressed as logic statements using terminology assertions as explained earlier. In Table 4.2, we list the complete set of users' quality requirements that we consider in this thesis. We only list the *satisfying* relationships in this table. The *not-satisfying* relationships can be expressed in logic statements in a similar way.

User's Quality Requirement	**Terminology Assertion**
The set of DI elements satisfying quality factor f_1 and the set of DI elements satisfying quality factor f_2 are disjoint	$(\forall contains\_sat^-.\{f_1\} \sqcap \forall contains\_sat^-.\{f_2\}) \equiv \emptyset$, $\forall contains\_sat^-.\{f_1\} \not\equiv \emptyset$, $\forall contains\_sat^-.\{f_2\} \not\equiv \emptyset$
The sets of DI elements satisfying quality factors f_1 and f_2 are overlapping	$(\forall contains\_sat^-.\{f_1\} \sqcap \forall contains\_sat^-.\{f_2\}) \not\equiv \emptyset$
The set of DI elements satisfying quality factor f_1 is a subset of the set of DI elements satisfying quality factor f_2	$\forall contains\_sat^-.\{f_1\} \sqsubseteq \forall contains\_sat^-.\{f_2\}$
The set of DI elements satisfying quality factor f_1 is the same as the set of DI elements satisfying quality factor f_2	$\forall contains\_sat^-.\{f_1\} \equiv \forall contains\_sat^-.\{f_2\}$

Table 4.2: Syntax of Users' Quality Requirements

In order to illustrate the above DL representation of our quality hierarchy and users' quality requirements, we consider the Case Study introduced in Chapter 3 and the following GAV mappings, comprising the union of a set of conjunctive queries. These mappings are expressed as DL assertions and

they derive the *programme_head_teacher* table and the attributes contained in this table in the *GS* by extracting information from various tables in three data sources. For example, in m_1, the *TID* and *ProgrammeName* attributes of the *programme_head_teacher* table are derived from the *TID* and *Name* attributes in the *educator* table in *LS1* and the *educator* table is joined with the *undergraduate_programme* table in *LS2*. The other mappings are similar.

$$m_1: \quad \forall t, n(\exists i, j, k.GS:programme\_head\_teacher(t, n, i, j, k) \leftarrow \exists o, p, y.\ LS_1:educator(t, n, o) \wedge LS_2:undergraduate\_programme(p, y, t))$$

$$m_2: \quad \forall t, n(\exists i, j, k.GS:programme\_head\_teacher(t, n, i, j, k) \leftarrow \exists o, p, y.\ LS_1:educator(t, n, o), LS_2:postgraduate\_programme(p, y, t), y > 1999)$$

$$m_3: \quad \forall t, n(\exists i, j, k.GS:programme\_head\_teacher(t, n, i, j, k) \leftarrow \exists o, p, y.\ LS_1:educator(t, n, o), LS_3:postgraduate\_programme(p, y, t))$$

Assume, in our case study, we have two quality criteria identified by c_1 and c_2. c_1 is the schema completeness quality criterion and c_2 is the mapping consistency criterion[3]. c_1 is associated with quality factor f_1, which defines schema completeness as the degree of coverage of local schema constructs that provide overlapping but possibly partially complete information for the same global schema constructs (this is Quality Factor 2 which we explain in detail in Chapter 5). c_2 is associated with quality factor f_2, which defines mapping consistency as the proportion of local schema constraints that are not violated by the new constraints introduced by the mappings (this is Quality Factor 7 which we explain in detail in Chapter 5). Table 4.3 lists the DI elements of the Case Study of Chapter 3 that satisfy and do not satisfy quality factors f_1 and f_2.

In this example, the information extracted from local schemas 1, 2 and 3 are combined to derive the *tid* and *tname* attributes of the *programme_head_teacher* table in the *GS* using the GAV mappings we listed above. In the DL representation, $\Delta^{\mathcal{I}} = \{LS_1\_educator, LS_2\_undergraduate\_programme, LS_2\_postgraduate\_programme, LS_3\_postgraduate\_programme,$

[3]We discuss these quality criteria in detail in Chapter 5, and understanding their specifics is not required in order to understand this example.

	Satisfying Elements	**not-Satisfying Elements**
f_1	$\{LS_1\_educator$, $LS_1\_tid_{educator}, LS_1\_name_{educator}$, $LS_1\_office_{educator}$, $LS_2\_undergraduate\_programme$, $LS_2\_postgraduate\_programme$, $LS_2\_pid_{undergraduate\_programme}$, $LS_2\_pid_{postgraduate\_programme}$, $LS_2\_StartingYear_{undergraduate\_programme}$, $LS_2\_StartingYear_{postgraduate\_programme}$, $LS_2\_ProgrammeHeadID_{undergraduate\_programme}$, $LS_2\_ProgrammeHeadID_{postgraduate\_programme}$, $LS_3\_postgraduate\_programme$, $LS_3\_pid_{postgraduate\_programme}$, $LS_3\_StartingYear_{postgraduate\_programme}$, $LS_3\_ProgrammeHead_{postgraduate\_programme}\}$	$\{LS_1\_fulltime\_faculty\_member$, $LS_1\_tid_{fulltime\_faculty\_member}$, $LS_1\_name_{fulltime\_faculty\_member}$, $LS_1\_office_{fulltime\_faculty\_member}$, $LS_1\_programme$, $LS_1\_pid_{programme}$, $LS_1\_level_{programme}$, $LS_1\_programme\_name_{programme}$, $LS_1\_programme\_description_{programme}$, $LS_1\_programme\_director_{programme}$, $LS_2\_teacher$, $LS_2\_lid_{teacher}$, $LS_3\_lecturer$, $LS_3\_tid_{lecturer}\}$
f_2	$\{LS_2\_undergraduate\_programme$, $LS_2\_pid_{undergraduate\_programme}$, $LS_2\_StartingYear_{undergraduate\_programme}$, $LS_2\_ProgrammeHeadID_{undergraduate\_programme}$, $LS_3\_postgraduate\_programme$, $LS_3\_pid_{postgraduate\_programme}$, $LS_3\_StartingYear_{postgraduate\_programme}$, $LS_3\_ProgrammeHead_{postgraduate\_programme}\}$	$\{LS_2\_postgraduate\_programme$, $LS_2\_pid_{postgraduate\_programme}$, $LS_2\_StartingYear_{postgraduate\_programme}$, $LS_2\_ProgrammeHeadID_{postgraduate\_programme}\}$

Table 4.3: An Example of Quality Assessments in Our Case Study

$LS_2\_tid_{undergraduate\_programme}, LS_2\_tid_{postgraduate\_programme}$,
$LS_3\_tid_{postgraduate\_programme}, LS_1\_tid_{educator}, LS_2\_name_{educator}$,
$LS_1\_office_{educator}, LS_2\_pid_{undergraduate\_programme}$,
$LS_2\_pid_{postgraduate\_programme}, LS_3\_pid_{postgraduate\_programme}$,
$LS_2\_year_{undergraduate\_programme}, LS_2\_year_{postgraduate\_programme}$,
$LS_3\_year_{postgraduate\_programme}, LS_1\_fulltime\_faculty\_member, LS_2\_teacher$,
$LS_3\_lecturer, LS_1\_programme, LS_1\_tid_{fulltime\_faculty\_member}$,
$LS_2\_tid_{teacher}, LS_3\_tid_{lecturer}, LS_1\_name_{fulltime\_faculty\_member}$,
$LS_1\_office_{fulltime\_faculty\_member}\}$.

Suppose now that there are three quality requirements issued by three users, A, B and C as listed in in Table 4.4. It is important for user A

that the integrated resource is still consistent with the constraints defined in the original data sources. In particular, user A requires that the schema constructs which satisfy the schema completeness factor should also satisfy the mapping consistency factor ($A.1$). User B does not require that schema constructs satisfy both the schema completeness and mapping consistency factors, but there should not be any which satisfy neither ($B.1$). User C, while in general requiring that constructs which satisfy the schema completeness factor should also satisfy mapping consistency, is more concerned in the case of programme director information that as much data from the data sources as possible is retained in the integrated resource, and so in that specific case schema completeness without mapping consistency is permissible. Hence, user C requires that any schema construct which satisfies schema completeness but not mapping consistency must be related to the Programme Director concept ($C.1$). We discuss next the reasoning capability of our approach and how this can be used to determine the consistency of these requirements and the DI elements that violate any of them.

4.3.2 Reasoning Capability

Given a quality hierarchy and logic statements representing different users' quality requirements, there are two validation steps in QFDI where reasoning can be applied. First, reasoning can be applied in order to validate different users' requirements, as specified from their different quality perspectives. Second, reasoning can be applied in order to validate individual quality requirements with respect to the integrated resource. In the former case, inconsistent logic statements can be identified. In the latter case, the DI elements that do not satisfy individual logic statements can be discovered. When an inconsistency is discovered, the DI elements relating to quality factors referenced in the logic statements or the logic statements themselves may need to be modified in order to resolve such inconsistencies.

No.	Requirement	Logic Statement in DL
$A.1$	Schema constructs that satisfy f_1 should also satisfy f_2.	$(Sc \sqcap \forall contains\_sat^{-}.\{f_1\})$ $\sqsubseteq$ $(Sc \sqcap \forall contains\_sat^{-}.\{f_2\})$
$B.1$	There should be no schema constructs that satisfy neither f_2 nor f_7.	$((Sc \sqcap \forall contains\_not\_sat^{-}.\{f_1\})$ $\sqcap$ $(Sc \sqcap \forall contains\_not\_sat^{-}.\{f_2\}))$ $\equiv \emptyset$
$C.1$	A schema construct which satisfies f_2 but does not satisfy f_7 must be related to the Programme Director concept.	$(Sc \sqcap \forall contains\_sat^{-}.\{f_1\}$ $\sqcap \forall contains\_not\_sat^{-}.\{f_2\})$ $\sqsubseteq (Sc \sqcap ProgrammeDirector)$

Table 4.4: Users' Requirements Example

The former case uses TBox reasoning and the latter case uses ABox reasoning. We first discuss a general Tableau algorithm for TBox reasoning [1]. Starting with a concept C_0, x is created as an individual of concept C_0 and the ABox is now $\mathcal{A} = \{C_0(x)\}$. Starting from this $\mathcal{A}$, the tableau algorithm repeatedly applies the expansion rules listed in Table 4.5 until no more rules can be applied. $\mathcal{A}'$ and $\mathcal{A}''$ are the possible alternative ABoxes after applying each expansion rule. If the final ABox, $\mathcal{A}^F$, does not contain a contradiction with both $C(x)$ and $\neg C(x)$ existing in $\mathcal{A}^F$, then the original ABox A is consistent, otherwise it is inconsistent. In Tableau algorithms, the satisfaction of inclusions is reduced to unsatisfiability, i.e., $C \sqsubseteq D$ is satisfied *iff* $C \sqcap \neg D$ is not satisfied.

In ABox reasoning, an Abox $\mathcal{A}$ is consistent with respect to a TBox $\mathcal{T}$, that is the set of terminology axioms, if there is an interpretation that is a model of both $\mathcal{A}$ and $\mathcal{T}$. ABox reasoning also consists of an expansion process with expansion rules as in a Tableau algorithm. However, instead

$\sqcap$ - rule:	If $\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but it does not contain both $C_1(x)$ and $C_2(x)$. Then $\mathcal{A}' = \mathcal{A} \cup \{C_1(x), C_2(x)\}$.
$\sqcup$ - rule:	If $\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$, but it contains neither $C_1(x)$ nor $C_2(x)$. Then $\mathcal{A}' = \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}'' = \mathcal{A} \cup \{C_2(x)\}$.
$\exists$ - rule:	$\mathcal{A}$ contains $(\exists R.C)(x)$, but there is no individual name z such that $C(z)$ and $R(x, z)$ are in A. Then $\mathcal{A}' = \mathcal{A} \cup \{C(y), R(x, y)\}$, where y is an individual name not occurring in $\mathcal{A}$
$\forall$ - rule:	$\mathcal{A}$ contains $(\forall R.C)(x)$ and $R(x, y)$, but it does not contain $C(y)$. Then $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$

Table 4.5: Tableau Expansion Rules (from [1])

of starting with an arbitrary individual x, ABox reasoning begins with an existing individual a of a concept C. If the final ABox, $\mathcal{A}^F$, does not contain a contradiction, then the original ABox A is consistent with respect to $\mathcal{T}$, otherwise it is inconsistent.

In our example (see Tables 4.3 and 4.4), inferring from the users' logic statements first, without involving the DI elements (i.e., undertaking TBox reasoning), we can discover that $A.1$ and $C.1$ are not consistent since $A.1$ implies that the set of DI elements satisfying f_1 should be a subset of the DI elements satisfying f_2, whereas $C.1$ implies that there could exist some programme leadership-related DI elements that do satisfy f_1 and do not satisfy f_2. Therefore, either $A.1$ or $C.1$ has to be modified. Suppose the data integrator removes $A.1$ because $A.1$ is a general and overall requirement and loosening it enables the users to define more detailed and flexible quality requirements such as $C.1$. We then repeat the inference process. There is now no conflict between the remaining logic statements ($B.1$ and $C.1$).

Next, for each remaining user requirement, we undertake reasoning again, this time including the DI elements (i.e., undertaking ABox reasoning). We can discover that while $B.1$ is satisfied by the DI elements listed in Table 4.3, $C.1$ is not since there are schema constructs not related to the Programme Director concept which satisfy f_1 but do not satisfy f_2, for exam-

ple $LS_2\_pid_{postgraduate\_programme}$ and $LS_2\_StartingYear_{postgraduate\_programme}$. Hence, the reasoner throws an exception indicating there is a problem with the extent of $contains\_not\_sat^{-}.\{f_2\}$.

Once the data integrator has validated the users' requirements and identified the DI elements which are indicated by the reasoner as having a problem, the mappings referencing those DI elements can be re-examined. In our example, a problem has been identified with the attributes of $LS_2\_postgraduate\_programme$ which do not satisfy f_2. These DI elements are referenced in mapping m_2. Suppose the data integrator modifies mapping m_2 by removing the constraint $y > 1999$ (a new mapping m_2' is generated). This will expand the set $contains\_sat^{-}.\{f_2\}$ and make $contains\_not\_sat^{-}.\{f_2\}$ an empty set. Both B.1 and C.1 can then be satisfied.

$$m_2': \quad \forall t, n(\exists i, j, k.GS : programme\_head\_teacher(t, n, i, j, k) \leftarrow \\ \exists o, p, y.\ LS_1 : educator(t, n, o), LS_2 : postgraduate\_programme(p, y, t))$$

4.4 Formal Foundations of Our DI Setting

In this section, we provide the formal foundations for the data modelling languages encompassed by our DI quality framework, and the integrated resources themselves.

4.4.1 Data Model Description

In general, the constructs of any data modelling language may be classified as either extensional constructs or constraint constructs. *Extensional constructs* of a schema S, denoted by $extensional(S)$, are the constructs that are populated with data values from some value domain. There are three types of extensional constructs proposed in [59], *nodal*, *linking* and *nodal-linking* and we adopt these definitions in our research:

Nodal constructs may be present in a schema independent of any other

constructs. For example, in the Entity Relationship (ER) model, entities are nodal constructs since they may exist independently of other constructs. The set of nodal constructs in a schema S is denoted by $extensional(S)^{nodal}$.

Link constructs associate other extensional constructs and can only exist when these other constructs exist. The extent of a link construct is a subset of the Cartesian product of the extents of the constructs which this link construct associates. For example, in an ER model, relationships are link constructs as they link and depend on one or more ER entities. The set of linking constructs in a schema S is denoted by $extensional(S)^{linking}$.

Link-Nodal constructs are nodal constructs that can only exist when certain other nodal constructs exist, and that are linked to these constructs. For example, in the ER model, attributes of entities and relationships are link-nodal constructs. They have an extent, but each value of the extent must be associated with a value in the extent of the entity or relationship the attributes link with. The set of nodal-linking constructs in a schema S, is denoted by $extensional(S)^{nodal-linking}$.

Constraint constructs of a schema S, denoted by $constraints(S)$, represent restrictions on the extents of extensional constructs and are not populated with data. For example, in the ER model, cardinalities are constraint constructs since they have no extent but restrict the extents of entities, relationships and attributes that they apply to.

4.4.2 Integrated Resource Description

In our research, we consider *an integrated resource* to be a triple $\langle LSs, GS, M\rangle$, comprising a set of local schemas LSs, a global schema GS and a set of

mappings M. The set of local schemas is also sometimes represented as $\{LS_1, \ldots, LS_n\}$ indicating n local schemas. Mappings may be either *local-as-view* (LAV) or *global-as-view* (GAV) [3] and both of them are supported by our quality framework discussed earlier[4]. Therefore, $M = M_{GAV} \cup M_{LAV}$, where M_{GAV} is a set of GAV mappings and M_{LAV} is a set of LAV mappings.

A GAV mapping, $o \leftarrow q_{LSs}$, derives a GS construct o using a conjunctive query over the LSs, denoted by q_{LSs}. In general, there may be a set of such mappings for a GS construct o. We denote by $sources(LSs, o)$ the set of local schema constructs that appear in the right-hand-sides (RHS) of o's mappings. We denote by $sources^{corr}(LSs, o)$ the subset of $sources(LSs, o)$ whose extents overlap with the extent of o. This is also called the set of *corresponding local schema constructs for o*.

A LAV mapping, $o \leftarrow q_{GS}$, derives a local schema construct o using a conjunctive query over the GS, denoted by q_{GS}. In general, there is only one such mapping for a local schema construct o. We denote by $sources(GS, o)$ the set of global schema constructs that appear in the RHS of this mapping. We denote by $sources^{corr}(GS, o)$ the subset of $sources(GS, o)$ whose extents overlap with the extent of o. This is also called the set of *corresponding global schema constructs for o*.

4.5 Summary

We have introduced in this chapter a quality framework, QFDI, that is designed for the DI context. Our quality framework aims to meet the requirements identified from our interviews with data integrators and from related research in data quality. With respect to the first objective in Section 4.1, the QFDI allows multiple users to express their quality requirements over the quality hierarchy and inconsistencies between such requirements can be

[4]We leave consideration of GLAV mappings [19] as future work.

discovered by applying reasoning over the framework. Inconsistent users' quality requirements can be discovered by the reasoning process. With respect to the second objective in Section 4.1, the quality framework comprises the concepts of quality criterion, quality factor, quality metric and the relationships between them. This enables different interpretations of quality to be defined and measured. With respect to the third objective in Section 4.1, the Weight and Metric concepts in the framework allow tradeoffs to be calculated with respect to different quality perspectives. With respect to the fourth objective in Section 4.1, our quality framework contains the elements of the integrated resource that are needed in the quality measurement. Such elements can be correlated in order to represent the relationships between them. With respect to the fifth objective in Section 4.1, reasoning over the quality hierarchy allows the DI elements that do not satisfy individual quality requirements to be discovered.

We have proposed a DL representation of our quality framework and discussed the reasoning requirements that need to be met in order to discover (i) inconsistencies between users' quality requirements, and (ii) DI elements that conflict with users' requirements. We have also introduced the formal foundations of data modelling languages and integrated resources that will be referred to in Chapter 5.

In Chapter 5, we will define a set of quality criteria that are appropriate in the context of data integration and a set of associated quality factors. We will also specify quality metrics for some of these quality factors and we will describe how the results from measuring these metrics can be used to improve the overall quality of an integrated resource using our QFDI framework.

Chapter 5

Quality Criteria and Metrics for Data Integration

In the previous chapter, we introduced our quality framework, QFDI, and proposed a quality hierarchy comprised of quality criteria, quality factors and metrics for measuring these. We also discussed the reasoning capabilities required to generate a quality view of the integrated resource with respect to different users' quality requirements as a whole and also individually.

In this chapter, we discuss in more detail the quality criteria and quality factors that are relevant in the context of data integration, and possible quality metrics associated with each quality factor. The chapter is organised as follows. In Section 5.1, we identify our quality criteria and sub-criteria in the DI context. A set of quality factors associated with the completeness and consistency quality criteria are then proposed and discussed in detail in Section 5.2 and 5.3 respectively. Quality metrics for measuring these quality factors are also proposed in these sections.

The quality factors and measurement methods that we propose in this chapter are not exhaustive. They are indicators of what is possible within our quality framework and quality hierarchy, and they may be refined and

extended in the future, following validation with real-world case studies and users. We will also discuss other quality criteria briefly in Section 5.4, including accuracy, minimality and performance, and we leave detailed research of these as future work. A comparison of our work with other research in the area of information and data quality is discussed in Section 5.5. Section 5.6 provides a summary of the contributions of this chapter.

5.1 Overview of DI Quality Criteria

Motivated by existing research into information and data quality discussed in Chapter 2 and the analysis of requirements in Chapter 3, we propose in this chapter a set of quality criteria defined specifically for the DI context. Of these, we consider that the *completeness* and *consistency* quality criteria are important in the context of data integration because these quality criteria affect the quality of the information extractable from an integrated resource by its users. We also consider that *accuracy*, *minimality* and *performance* are important quality criteria in the DI context because they relate to the precision of the information extractable from the integrated resource, the size and the effectiveness of the integrated resource from the users' perspectives.

The *completeness* quality criterion considers the degree of *coverage of information* of an integrated resource. We consider here that information is represented by the DI elements: data, schema constructs, mappings and assertions. This quality criterion is categorised into three sub-criteria: *schema completeness*, *mapping completeness* and *query completeness*. The notion of *information coverage* can be interpreted in different ways with respect to each sub-criterion and the quality factors associated with such sub-criteria. *Schema completeness* represents the degree of coverage of information by the *GS* or the *LSs* within an integrated resource. *Mapping completeness* represents the degree of coverage of information by the schema constructs

that are referenced in the mappings. *Query completeness* represents the degree of coverage of information represented by the integrated resource that is extractable by a set of user-defined queries. The completeness quality criterion is discussed in detail in Section 5.2.

The *consistency* quality criterion is categorized into three sub-criteria: *schema consistency*, *mapping consistency* and *query consistency*. *Schema consistency* captures the conformance of the local and global schemas with respect to the modeling methods used, such as model-related data types and constraints, and to other required semantics such as domain-related constraints in or between schemas. In this thesis, we consider that a constraint on a schema S is a query over the extensional constructs in S that needs to evaluate to true for all instances of S. *Mapping consistency* captures whether the information represented by the mappings conforms to the semantics of information represented by the schemas of the integrated resource and also to the semantics of any additional relationships between schema constructs derived from the integration domain and the users' requirements. Such user-defined relationships may be *inclusion dependencies*, *conditional inclusion dependencies*, *functional dependencies*, *conditional functional dependencies* and *implications*. *Query consistency* captures the degree of satisfaction of results returned from user-defined queries with respect to the user-defined relationships between such result sets. The consistency quality criterion is discussed in detail in Section 5.3.

	Completeness	**Consistency**
Schema	F_1 GS,LSs	F_5 GS,LSs
	F_2 LSs,Ontology	F_6 GS,LSs,Ontology
Mapping	F_3 GS,LSs	F_7 LSs
Query	F_4 LSs,Ontology	F_8 Data,Ontology

Table 5.1: Summary of Our Quality Factors (F_i)

In the next two sections, we discuss in detail the completeness and con-

sistency quality criteria, and the quality factors associated with them. In the sections that follow, we also introduce briefly the accuracy, minimality and performance criteria, leaving a detailed consideration of these for further work.

Before introducing the quality factors, we present here the general measurement method that will be applied to all quality factors. Each quality factor F_i is associated with a user-defined threshold μ_i in the range $(0, 1]$ indicating the minimum user-desired level for the value of F_i. If the measurement method returns a value λ_i that is less than μ_i, then we consider that the integrated resource does not satisfy this quality factor with respect to the users' minimum desired level.

Table 5.1 provides a summary of the quality factors we propose in this chapter. Each quality factor F_i is associated with a combination of *GS*, *LSs*, Ontology and Data, indicating from which parts of the integrated resource information is used for measuring F_i.

5.2 The Completeness Criterion

In this section, we propose quality factors for the interpretations of *information coverage* we identified in Section 5.1, and we also introduce one or more quality metrics for measuring each quality factor.

5.2.1 The Schema Completeness Criterion

In this subsection, we present two quality factors relating to the schema completeness sub-criterion, and measurement methods for each.

FACTOR 1: Schema completeness is measured as the proportion of information coverage of the global schema *GS* via the mappings *M* with respect to the information represented by all local schemas *LSs*.

In this context, we consider that the information represented by the GS is given by the extensional schema constructs in the GS (ie. we ignore the integrity constraints). We first define this quality factor separately for the GAV and LAV mapping approaches, and then consider a combined GAV and LAV mapping approach (as supported by the AutoMed DI system, for example [64]).

If the GAV approach is used, this quality factor can be defined as the number of extensional local schema constructs that have been used for deriving the GS via the mappings M compared with the total number of extensional local schema constructs in the integrated resource. This can be calculated in the following way:

1. For each extensional GS construct $o \in extensional(GS)$, let $sources(LSs, M_{GAV}, o)$ denote the set of extensional local schema constructs from which information is extracted for deriving the extent of o from the set of GAV mappings in M, M_{GAV}. This is the set of the local schema constructs appearing in the right-hand-side (RHS) of the GAV mappings whose left-hand-side (LHS) is o.

2. Form the union of the sets $sources(LSs, M_{GAV}, o)$ over all constructs $o \in extensional(GS)$. This will return a set of distinct extensional local schema constructs from which information is extracted for deriving the GS, which we denote by $sources(LSs, M_{GAV}, GS)$.

3. The degree of completeness of this DI setting is then calculated as

$$\lambda_{1,GAV} = \frac{|sources(LSs, M_{GAV}, GS)|}{\sum_{i=1}^{n} |extensional(LS_i)|}$$

   where $\sum_{i=1}^{n} |extensional(LS_i)|$ is the sum of the number of extensional local schema constructs over all local schemas.

If the LAV approach is used, this quality factor can be defined as the number of extensional local schema constructs that are derived from the GS by the LAV mappings compared with the total number of extensional local schema constructs. This can be calculated in the following way:

1. Let $LAVdefined(LSs, M_{LAV})$ denote the subset of local schema constructs such that for each $o \in LAVdefined(LSs, M_{LAV})$, there is a LAV mapping whose LHS is o. M_{LAV} is the set of LAV mappings in M.

2. The degree of completeness of this DI setting is then calculated as

$$\lambda_{1,LAV} = \frac{|LAVdefined(LSs, M_{LAV})|}{\sum_{i=1}^{n} |extensional(LS_i)|}$$

If both GAV and LAV mapping approaches have been used in the DI setting, this quality factor can be measured using Formula 5.1 which generalises the previous two formulae[1]:

$$\lambda_1 = \frac{|sources(LSs, M_{GAV}, GS) \cup LAVdefined(LSs, M_{LAV})|}{\sum_{i=1}^{n} |extensional(LS_i)|} \tag{5.1}$$

Example: In our Case Study in Chapter 3, we have 39 schema constructs in Local Schema 1 in Figures 3.3 and 3.4, 44 schema constructs in Local Schema 2 in Figures 3.5, 3.6 and 3.7, and 40 schema constructs in Local Schema 3 in Figures 3.8, 3.9 and 3.10. The total number of extensional schema constructs represented in the three local schemas is $39 + 44 + 40 = 123$. The number of extensional local schema constructs referenced in the GAV mappings is 45 as listed in Table B.3 and the number of local schema constructs derived by the LAV mappings is 4. There is one construct, $Name$ in the *educator* table, that appears in both the GAV and the LAV mappings. Therefore, the total number of local schema construct

[1]Note that duplicate schema constructs are eliminated by the union operation.

that have been referenced in GAV or LAV mappings is $45 + 4 - 1 = 48$ and the above formula returns a quality value of $48/123 = 0.39$ for this quality factor.

FACTOR 2: Schema completeness is measured as the average level of coverage of extensional local schema constructs that provide overlapping but possibly partially complete information for deriving the same global schema constructs.

The extents of the local schema constructs from different data sources may provide *overlapping information*, i.e. may represent the same concept in the domain ontology. The information in the data sources may be partially complete, and one of the purposes of data integration is to reduce incompleteness by combining information from different data sources. This quality factor can be used to verify that the information extracted by the mappings deriving the GS constructs covers sufficient breadth over the data sources.

In this context, we consider that information represented by the GS is given by the extensional schema constructs in the GS. We first define this quality factor separately for the GAV and LAV mapping approaches, and then consider a combined GAV and LAV approach.

We denote by $concepts(S, O)$ the set of real-world concepts corresponding to concepts in the domain ontology O that are represented by extensional constructs of a schema S. We denote by $reduce(C, O)$ the set of unique real-world concepts in a set of concepts, C, obtained by removing concepts that are equivalent to or subsumed by other concepts in the ontology O.

If the GAV approach is used, this quality factor can be defined as the average level of coverage of information represented by extensional local schema constructs that relate to the same real-world concept. The measurement of this quality factor is illustrated in Formula 5.2, where $extensional(LSs, c)$ is the set of extensional local schema constructs representing the real-world

concept c, and $sources(LSs, M_{GAV}, c)$ is the set of extensional local schema constructs representing the real-world concept c that also appear in the RHSs of the set of GAV mappings M_{GAV}:

$$\lambda_{2,GAV} = \frac{\sum_{c \in \cup_{j=1}^{n} reduce(concepts(LS_j, O), O)} \frac{|sources(LSs, M_{GAV}, c)|}{|extensional(LSs, c)|}}{|\bigcup_{i=1}^{n} reduce(concepts(LS_i, O), O)|} \tag{5.2}$$

If the LAV approach is used, this quality factor is again defined as the average level of coverage of information represented by extensional local schema constructs that relate to the same real-world concept. This can be calculated using Formula 5.3:

$$\lambda_{2,LAV} = \frac{\sum_{c \in \cup_{j=1}^{n} reduce(concepts(LS_j, O), O)} \frac{|LAVdefined(LSs, M_{LAV}, c)|}{|extensional(LSs, c)|}}{|\bigcup_{i=1}^{n} reduce(concepts(LS_i, O), O)|} \tag{5.3}$$

If both GAV and LAV approaches may be used in the DI setting, this quality factor can be measured using Formula 5.4 which generalises the previous two formulae:

$$\lambda_{2} = \frac{\sum_{c \in \cup_{j=1}^{n} reduce(concepts(LS_j, O), O)} \frac{|sources(LSs, M_{GAV}, c) \cup LAVdefined(LSs, M_{LAV}, c)|}{|extensional(LSs, c)|}}{|\bigcup_{i=1}^{n} reduce(concepts(LS_i, O), O)|} \tag{5.4}$$

Example: In our Case Study in Chapter 3, we have identified 14 unique concepts that are represented by the extensional local schema constructs referenced in the mappings in Figures B.3. These are illustrated in Table 5.2, where **S** indicates the number of local schema constructs that satisfy this quality factor and **NS** indicates the number of local schema constructs that do not satisfy this quality factor; **Coverage** indicates, for each concept, the proportion of local schema constructs representing this concept. This quality factor is measured by using Formula 5.4 and returns $(0.40 + 0.33 + 1.00 + 0.40 + 0.50 + 0.55 + 0.43 + 0.33 + 1.00 + 0.29 + 0.67 + 1.00 + 0.33 + 0.60)/14 = 0.56$.

5.2.2 The Mapping Completeness Criterion

In this subsection, we present a quality factor relating to the mapping completeness sub-criterion, and a measurement method for it.

FACTOR 3: Mapping completeness is measured as the proportion of local schema constraints removed by the mappings without information loss compared with the total number of local schema constraints removed.

Schema constraints play a crucial role in data modelling as they restrict the extents of the extensional schema constructs. Such constraints can be associated with the extensional schema constructs in two ways: *construct definition* and *extent restriction*. In the former category, we consider constraints such as data types and value ranges. In the latter category, we consider constraints over the extents of extensional schema constructs. In this quality factor, we focus on the latter category of constraints and we adopt the categories of constraints on the extents of extensional schema constructs proposed in [73]. These are: inclusion, exclusion, union, mandatory, unique and reflexive constraints which are shown in [73] to be sufficiently expressive to represent the constraints supported in the major data modelling languages. An inclusion constraint between two extensional constructs s_1 and s_2 states that $ext(s_1)$ is always a subset of $ext(s_2)$. An exclusion constraint between two extensional constructs s_1 and s_2 states that the intersection of

	S	**NS**	**Coverage**		**S**	**NS**	**Coverage**
CID	4	6	0.40	Programme	1	2	0.33
Description	1	2	0.33	register	3	0	1.00
Head	4	0	1.00	SID	2	5	0.29
induct	2	3	0.40	Student	2	1	0.67
Level	2	2	0.50	study	2	0	1.00
Name	6	5	0.55	Teacher	3	6	0.33
PID	3	4	0.43	TID	9	6	0.60

Table 5.2: Results of Factor 2

$ext(s_1)$ and $ext(s_2)$ is always an empty set. A union constraint between an extensional construct s and a set of extensional constructs $s_1, ..., s_n$ states that $ext(s)$ is equivalent to $ext(s_1) \cup ... \cup ext(s_n)$. A mandatory constraint between a set of nodal constructs $n_1, ..., n_n$ and an edge construct e states that every combination of the values in the extents of $n_1, ..., n_n$ must appear at least once in $ext(e)$. A unique constraint between a set of nodal constructs $n_1, ..., n_n$ and an edge construct e states that every combination of the values in extents of $n_1, ..., n_n$ must appear at most once in $ext(e)$. A reflexive constraint states that if an extent of an extensional construct s appears in edge e, then a member of the extent of e must be an identity tuple. Together, the inclusion, mandatory and unique constraints can be used to represent primary and foreign key constraints [73].

In the DI context, the information represented by such constraints in the local schemas may be lost via the mappings if the local schema constraints are removed by the mappings but no corresponding constraints are generated on the *GS*. Assume there exists an extensional schema construct o and a constraint c restricting the extent of o in schema S, and another extensional construct o' in another schema S' linked by the mappings to schema S. We say o' is a *corresponding construct* of o if o' is derived from o by the mappings and $ext(o') \subseteq ext(o)$. We denote by $source^{corr}(S, M, o)$ the set of corresponding constructs of o in S'. We say c' is a *corresponding constraint* of c if $c \in reformulate(c', M)$. Taking an input of a query representing constraint c' on the GS and a set of mappings M, the $reformulate$ function returns a set of queries that are executable on the data sources. We refer the reader to Section 2.2 for a discussion of query reformation. We denote by $source^{corr}(S, M, c)$ the set of corresponding constraints of c in S'.

For both the GAV and LAV approaches, this quality factor is defined as the proportion of constraints on the local schemas that have been removed by the mappings but whose corresponding constraints on the *GS* are also

generated in the mappings compared with the total number of local schema constraints that have been removed by the mappings. This can be calculated in the following way:

1. Determine the set of constraints in the local schemas, denoted as $constraints(LSs)$.

2. For each constraint $c \in constraints(LSs)$, detect if c is removed in the GAV mappings M_{GAV}. Removing a constraint in the mappings can be achieved either implicitly or explicitly. In the former case, the removal of an extensional schema construct results in the removal of all constraints associated with this schema construct. In the latter case, the removal of constraints is achieved by the mapping primitives, such as the $deleteConstraint()$ operation in AutoMed (see detailed discussion on AutoMed in Chapter 2). The set of such constraints is then denoted by $removed(constraints(LSs), M_{GAV})$.

3. We then need to determine if there exists a corresponding constraint c' in the GS for each $c \in removed(constraints(LSs), M_{GAV})$. This may be determined by reformulating the query associated with c', denoted by $q_{c'}$, via the mappings M_{GAV}. We denote the set of such reformulated queries as $reformulate(q_{c'}, M)$. If q_c is contained by $reformulate(q_{c'}, M)$, we say that c' is a corresponding constraint of c and we assign 1 to $corr(q_c, reformulate(q_{c'}, M))$[2]. Otherwise, we assign 0 to $corr(c, reformulate(q_{c'}, M))$.

4. The degree of completeness of this integrated resource is then calculated as in Formula 5.5, where $c' \in constraints(GS)$.

[2]We say that q_c is contained by $reformulate(q_{c'}, M)$ if all schema constructs O referenced in q_c are also referenced in $q \in reformulate(q_{c'}, M)$ and there does not exist any tuple $t \in ext(O)$ that satisfies q but does not satisfy q_c.

$$\lambda_3 = \frac{\sum_{c \in removal(constraints(LSs)), c' \in constraints(GS)} corr(q_c, reformulate(q_{c'}, M))}{|removal(constraints(LSs))|} \quad (5.5)$$

Example: In our Case Study in Chapter 3, we removed 58 local schema constraints (we consider the primary key and foreign key constraints here) in the mappings and 50 of them have corresponding constraints on the *GS* created. Therefore, using Formula 5.5, this quality factor returns a figure of $50/58 = 0.86$.

5.2.3 The Query Completeness Criterion

In this subsection, we present a quality factor relating to the query completeness sub-criterion, and a measurement method for it.

FACTOR 4: Query completeness is measured as the average level of coverage of each real-world concept represented by the local schema constructs that are referenced in the queries arising from reformulating the set of users' queries on the *GS*.

As discussed in quality factor 2, schema constructs may provide overlapping but possibly partially complete information. Different from quality factor 2, we consider here the coverage of real-world concepts as the proportion of the local schema constructs that are referenced in the reformulated users' queries on the global schema compared with the total number of local schema constructs that represent the same concepts. For this quality factor, we extend the DI setting to a quadruple $\langle LSs, M, GS, Q^{GS} \rangle$, where Q^{GS} is a set of queries on the global schema the users want the integrated resource to answer.

For both the GAV and LAV approaches, this quality factor is calculated using Formula 5.6 below, where:
$rccr(q, M, O) = reduce(concept(construct(reformulate(q, M)), O), O)$ is the set of unique real-world concepts represented by local schema constructs ref-

erenced in a reformulated query $q \in Q^{GS}$ with respect to the domain ontology O; $construct(reformulate(q, M), c)$ is the set of local schema constructs representing $c \in rccr(q, M, O)$ that are also referenced in $reformulate(q, M)$; and $construct(LSs, c)$ is the total set of local schema constructs representing c.

$$\lambda_4 = \sum_{q \in Q^{GS}} \sum_{c \in rccr(q,M,O)} \frac{|construct(reformulate(q, M), c)|}{|construct(LSs, c)| \times |rccr(q, M, O)| \times |Q^{GS}|} \tag{5.6}$$

Example: In our Case Study in Chapter 3, we have one test query for retrieving information about the programme heads and all members of the programme the programme heads lead. This query covers the *programme_head_teacher*, *lead* and *lecturer* tables. The reformulated query covers 6 unique concepts in the domain ontology (illustrated in Table 5.3) and this quality factor returns a figure of $(1.00 + 0.40 + 0.09 + 1.00 + 0.33 + 1.00)/6 = 0.64$ by using Formula 5.6

Concept	Referenced	Unreferenced	Result
Head	4	0	1.00
induct	2	3	0.40
Name	1	10	0.09
register	3	0	1.00
Student	1	2	0.33
study	2	0	1.00

Table 5.3: Results of Factor 4

5.3 The Consistency Criterion

We categorize the consistency criterion into three sub-criteria: schema consistency, mapping consistency and query consistency and discuss them in detail below.

5.3.1 The Schema Consistency Criterion

In this subsection, we present two quality factors relating to the schema consistency sub-criterion, and measurement methods for each.

FACTOR 5: Schema consistency is measured as the number of *GS* constructs whose definitions and associated constraints can be applied to their corresponding local schema constructs, if they exist, compared with the total number of *GS* constructs.

In a DI setting, extensional construct definitions in the local schemas define the format of the extents of a construct: its data type and any constraints on the extents of the construct such as value ranges. In different local schemas, different definitions may be given for schema constructs representing the same real-world concept. Such definitions may not be consistent, in the sense that the extent of one schema construct cannot be transformed into the extent of another construct without loss of equivalence.

There may be constraints in the *GS* restricting the extents of the nodal-linking and linking constructs that connect schema constructs from different local schemas. Such constraints may not be satisfiable, in the sense that there cannot exist extents extracted from the local schemas that satisfy these constraints. In our research, we consider subsumption, functional dependencies and cardinality constraints because they can be validated using techniques such as chase [74].

Given a DI setting $\langle LSs, GS, M \rangle$, the schema consistency criterion can be measured as the proportion of *GS* constructs whose definitions and associated constraints can also be applied to the corresponding local schema constructs without causing errors. This quality factor can be calculated as follows:

For measuring the consistency of the *construct definitions* in *GS*:

1. We first need to determine the set of *GS* constructs that have been derived by a GAV mapping, $extensional(GS, M_{GAV})$.

2. We then need to compute the set of local schema constructs where, for each $o \in extensional(GS, M_{GAV})$, the extent of such local schema constructs derive o directly, denoted by $sources^{corr}(LSs, M_{GAV}, o)$.

3. For each construct $o \in extensional(GS, M_{GAV})$, we need to determine if the definition of o subsumes the definition of the local schema constructs in $sources^{corr}(LSs, M_{GAV}, o)$. By subsumption of a data type T_A over a data type T_B, we mean that the extent of a schema construct defined using T_B can be cast to be the extent of a schema construct defined using T_A without *information loss*. By information loss, we mean here loss of precision associated with the source data type in the casting process. By subsumption of a value range V_A over a value range V_B, we mean that the extent of a schema construct defined with value range V_B can be transformed to be the extent of a schema construct defined with value range V_A without conflicts.

4. This quality factor can then be calculated using Formula 5.7 for the construct definition aspect:

$$\lambda_{5,E} = \sum_{o \in extensional(GS, M_{GAV})} \frac{|consistent(sources^{corr}(LSs, M_{GAV}, o), o)|}{|sources^{corr}(LSs, M_{GAV}, o)| \times |extensional(GS, M_{GS})|} \quad (5.7)$$

For measuring the consistency of the *constraint definitions* in GS:

1. We need to first determine the set of constraints on the GS, $constraints(GS)$.

2. For each constraint $o \in constraints(GS)$, we reformulate the query comprising o, q_o, and obtain a set of queries on the local schemas, $reformulate(q_o, M_{GAV})$. For each query $q \in reformulate(q_o, M_{GAV})$ on a local schema, we need to detect if q is satisfied by the local schema with respect to the current schema instances.

3. This quality factor can then be calculated using Formula 5.8 for the constraint construct aspect, where $consistent(q, LSs)$ is assigned 1 if q is satisfied. Otherwise, $consistent(q, LSs)$ is assigned 0.

$$\lambda_{5,C} = \sum_{o \in constraints(GS)} \sum_{q \in reformulate(q_o, M_{GAV})} \frac{consistent(q, LSs)}{|reformulate(q_o, M_{GAV})| \times |constraints(GS)|} \tag{5.8}$$

The overall quality measurement, λ_5, is given by weighting $\lambda_{5,E}$ and $\lambda_{5,C}$ according to user-specified weights, denoted by w_E and w_C, respectively.

$$\lambda_5 = w_E \times \lambda_{5,E} + w_C \times \lambda_{5,C}$$

Example: In our Case Study, for simplicity we test the consistency of the construct definitions only. $\lambda_{5,E}$ is calculated as 1 since the definition of each *GS* construct subsumes the definition of the corresponding local schema constructs, and $\lambda_{5,C}$ is set to 0 since it is not considered in our case study. We give equal weight to both consistency categories for this quality factor, with $w_E = w_C = 0.5$. This quality factor then results in $1 \times 0.5 + 0 \times 0.5 = 0.5$.

FACTOR 6: Schema consistency can be measured as the proportion of local schema constructs that satisfy their real-world semantics and whose corresponding *GS* constructs also satisfy the same real-world semantics.

The information represented by the *LSs* may or may not be consistent with the intended real-world semantics. In data integration, the local schema constructs that provide such consistent information are important and data integrators may want to maintain such consistencies in the *GS* if these local schema constructs are also represented in the *GS*. This quality factor is defined regardless of which integration approach is used and can be measured as the number of extensional local schema constructs that satisfy their real-world semantics and whose corresponding *GS* constructs also satisfy the same real-world semantics compared with the total number of local schema

constructs. Currently, we assume this is a manual process, but it could be undertaken semi-automatically by linking and comparing the ontological representation of the data source schemas with the domain ontology. By *corresponding* GS *constructs*, we mean here the constructs that represent the same real-world concept.

This quality factor can be measured using Formula 5.9, where: $consistent(extensional(LSs), O)$ is the set of local schema constructs that are consistent with the definitions of their corresponding real-world concepts with respect to a domain ontology O; $corr(GS, consistent(extensional(LSs), O), M)$ is the set of global schema constructs corresponding to $consistent(extensional(LSs), O)$; and $consistent(corr(GS, consistent(extensional(LSs), O), M), O)$ is the subset of $consistent(extensional(LSs), O)$, whose corresponding GS constructs are also consistent with the definition of the same real-world concepts:

$$\lambda_6 = \frac{|consistent(corr(GS, consistent(extensional(LSs), O), M), O)|}{|consistent(extensional(LSs), O)|} \tag{5.9}$$

By 'definition of the real-world concepts', we mean here the data type and the value range of the corresponding ontology concepts and the constraints associated with these concepts. This information can be captured from the domain ontology. We also need to capture the corresponding information from the schemas: the data type and value ranges of the nodal and nodal-linking constructs, and the constraints over the nodal-linking and linking constructs. In the former case, this is specified by the definitions of these constructs. In the latter case, it is detected by analyzing the constraints associated with such schema constructs.

Example: In our Case Study, the real-world semantics are defined as users' assertions in Tables 3.2 and 3.3. We can detect that there exist 8 local schema constructs that comply with 15 such assertions in the three local schemas (A1-A15 in Tables 3.2 and 3.3). In the integrated resources, 11 of these users' assertions complied with by 6 local schema constructs

are also complied with by the corresponding *GS* constructs (A1-A4, A7-A11, A14, A15 in Tables 3.2 and 3.3). This quality factor then results in $6/8 = 0.75$ using Formula 5.9.

5.3.2 The Mapping Consistency Criterion

In this subsection, we present a quality factor relating to the mapping consistency sub-criterion, and a measurement method for it.

FACTOR 7: Mapping consistency can be measured as the proportion of local schema constraints that are not violated by new constraints introduced by the mappings *M*.

Constraints on the local schemas contain important information as they form restrictions on the extents of the extensional schema constructs. When such information is transformed for deriving the *GS*, there is a risk that the extents extracted from the *LSs* no longer comply with the local schema constraints. Local integrity and entity constraints may be violated by the way that mappings are specified. For example, suppose that a 1-to-many relationship exists between a pair of attributes in two tables in a local schema. The integrator may make a mistake that establishes a many-to-many relationship between this attribute pair. Therefore, the local integrity constraint is violated in the integrated resource. New constraints may be added via the mappings *M* explicitly or implicitly as discussed in Quality Factor 3 earlier. In the former case, new constraints can be added to schemas using schema transformation primitives supported by the integration system, such as the addConstraint primitive in AutoMed [59]. In the latter case, new constraints can be expressed in the mapping queries, restricting the extents that are extracted from the data sources. Constraints on the local schemas may also be modified or deleted. Again, we adopt the constraint categories proposed in [73] for this quality factor.

If the GAV mapping approach is used, this quality factor can be measured

as the proportion of local schema constructs that satisfy both the queries representing the constraints on the *LSs* and also the queries introducing new constraints in the mappings. This can be calculated using Formula 5.10, where $constraints(LSs)$ is the set of local schema constraints, q_o is the query corresponding to the local schema constraint o, and $q_{s,o}$ is the set of queries introducing new constraints in the mappings relating to schema constructs referenced in q_o. $evaluate(q_o, q_{s,o})$ is assigned 1 if both q_o and each member of $q_{s,o}$ evaluate to true, in the sense that all extents which satisfy q_o also satisfy all members of $q_{s,o}$. Otherwise, $evaluate(q_o, q_{s,o})$ is assigned 0.

$$\lambda_{7,GAV} = \sum_{o \in constraints(LSs)} \frac{evaluate(q_o, q_{s,o})}{|constraints(LSs)|} \tag{5.10}$$

If the LAV mapping approach is used, this quality factor can be measured as the proportion of queries representing constraints on the *LSs* that can be reformulated as queries on the *GS* and still evaluate to true, compared with the number of constraints on the *LSs*. This can be calculated using Formula 5.11, where $evaluate(q_o, reformulate(q_o, M_{LAV}))$ is assigned 1 if both q_o and $reformulate(q_o, M_{LAV})$ evaluate to true. Otherwise, $evaluate(reformulate(q_o, M_{LAV}))$ is assigned 0.

$$\lambda_{7,LAV} = \sum_{o \in constraints(LSs)} \frac{evaluate(q_o, reformulate(q_o, M_{LAV}))}{|constraints(LSs)|} \tag{5.11}$$

If both the GAV and LAV approaches may be used in the DI setting, this quality factor can be measured using Formula 5.12, which generalises the previous two formulae:

$$\lambda_7 = \sum_{o \in constraints(LSs)} \frac{min(evaluate(q_o, qs), evaluate(q_o, reformulate(q_o, M_{LAV})))}{|constraints(LSs)|} \tag{5.12}$$

Example: In our Case Study in Chapter 3, the new constraints introduced in the mappings are the primary key and foreign key constraints on the *GS*.

Therefore, we evaluate queries associated with each of these constraints and all of them can be evaluated to be true in the integrated resource. Therefore, this quality factor is calculated as 1 using Formula 5.12.

5.3.3 The Query Consistency Criterion

In this subsection, we present a quality factor relating to the query consistency sub-criterion, and a measurement method for it.

FACTOR 8: The degree of query consistency is measured as the level of satisfaction of users' requirements relating to the relationships between the information retrieved by pairs of users' queries in a DI setting.

As indicated in our analysis of requirements in Chapter 3, queries have an important role in assessing the quality of an integrated resource as they allow users to specify what data are expected to be returned from the integrated resource without having to fully enumerate the data: enumerating such data may not be easy, especially from data sources containing large volumes of data [75]. Some researchers have investigated detecting inconsistencies in an integrated resource by evaluating queries on the GS and examining the possible results returned from these queries with respect to the constraints in the GS [28]. We adopt a similar approach, but our purpose is to define and measure the consistency of a DI setting by investigating the results returned from a set of users' queries.

For this quality factor, we assume a DI setting is a quadruple $\langle LSs, GS, M, Req\rangle$ where Req is a set of requirements defining the expected relationships between the results retrieved by pairs of users' queries over the LSs and GS. Each requirement is a triple $Q_i^{LSs,GS} = \langle q_{LSs}, q_{GS}, relationship\rangle$, where q_{LSs} is a user-defined query on a single local schema or across several local schemas (on their union schema), q_{GS} is a user-defined query on the GS, and $relationship$ is the users' expected relationship between the results

returned from q_{LSs} and from q_{GS}. For this quality criterion, we consider $relationship \in \{=, \subseteq, \supseteq\}$.

For both the GAV and LAV approaches, this quality factor is defined as the average level of satisfaction of Req. For each $Q_i^{LSs,GS} \in Req$, the level of satisfaction of $Q_i^{LSs,GS}$ is defined by calculating the level of satisfaction of $relationship$, denoted by $satisfy(Q_i^{LSs,GS})$, with respect to q_{LSs} and $reformulate(q_{GS}, M)$, where $reformulate(q_{GS}, M)$ is the query on the local schemas created by reformulating q_{GS} using the mappings M. For both the GAV and LAV approach, the measurement method for this quality factor is the same, except that the reformulation process in each case is different. For the relationships $=$, $\subseteq$ and $\supseteq$, $satisfy(Q_i^{LSs,GS})$, where $0 \leq satisfy(Q_i^{LSs,GS}) \leq 1$, can be calculated by using Formula 5.13, Formula 5.14 and Formula 5.15 respectively. In these formulae, $ext(q)$ is the result of evaluating q over the integrated resource. This quality factor is then defined as the average of $satisfy(Q_i^{LSs,GS})$ for all $Q_i^{LSs,GS} \in Req$.

$$satisfy(\langle q_{LSs}, q_{GS}, \text{“}=\text{”} \rangle) = \frac{|ext(q_{LSs}) \cap ext(reformulate(q_{GS}, M))|}{|ext(q_{LSs}) \cup ext(reformulate(q_{GS}, M))|} \tag{5.13}$$

$$satisfy(\langle q_{LSs}, q_{GS}, \text{“}\subseteq\text{”} \rangle) = \frac{|ext(q_{LSs}) \cap ext(reformulate(q_{GS}, M))|}{|ext(q_{LSs})|} \tag{5.14}$$

$$satisfy(\langle q_{LSs}, q_{GS}, \text{“}\supseteq\text{”} \rangle) = \frac{|ext(q_{LSs}) \cap ext(reformulate(q_{GS}, M))|}{|ext(reformulate(q_{GS}, M))|} \tag{5.15}$$

Example: In our Case Study in Chapter 3, we assume that the users provide a requirement $Q^{LSs,GS}$ comprising a single triple: q_{LSs} is a query on local schema 3 that retrieves information about all lecturers teaching postgraduate courses and q_{GS} is a query on the GS that retrieves information about all educators. The relationship between q_{LSs} and q_{GS} is specified by the users as $q_{LSs} \subseteq q_{GS}$. This quality factor is therefore calculated using Formula 5.14 and results in $4/4 = 1$ as 4 lecturers teach postgraduate courses

(*TID* 505 - 508) and 8 lecturers can be retrieved from the *GS* including these 4 lecturers teaching postgraduate courses (*TID* 505 - 508) and 4 more lecturers teaching undergraduate courses (*TID* 501 - 504).

5.4 Other Quality Criteria

Other quality criteria that can be considered as being relevant to a DI setting are *accuracy*, *minimality* and *redundancy*. We briefly discuss these quality criteria here, but leave a detailed development of metrics as future work.

The *accuracy* criterion considers the degree of precision of information represented in an integrated resource. By *precision* we mean the scope of information represented by a schema or a mapping, the constraints preserved in it, and the granularity of information representation it supports. This quality criterion is categorized into two sub-criteria: *schema accuracy* and *mapping accuracy*. *Schema accuracy* captures to what degree the information provided by the local schemas is represented by the *GS* without information loss. Information loss could arise in many ways, such as inaccurate data types used, missing constraints in different schemas, and imprecise terminologies used for describing the same information. *Mapping accuracy* captures to what degree the information represented by the local schemas is transformed in the mappings to the global schema without losing information. Such information loss can occur when some existing semantic information is ignored or reduced by the mappings.

The *minimality* criterion considers the degree of redundancy existing in an integrated resource. By *redundancy* we mean the schemas and mappings from which the same information may be extracted. We also use *redundancy* to describe unnecessary transformations occurring in mappings. Such redundancies may have impact on performance since overlapping information is processed unnecessarily. It may also increase the risk of schema and query

inconsistencies if redundant schema constructs exist in the DI setting but do not have an identical extent. The minimality quality criterion may be categorised into *schema minimality* and *mapping minimality* sub-criteria. The schema minimality sub-criterion captures the redundancies of schema constructs and information extractable from schema constructs. The mapping minimality sub-criterion captures unnecessary transformations occurring within mappings.

The *performance* criterion considers the cost of query processing in an integrated resource. The cost of query processing is affected by the capability of the DI tool's global query processor and the data sources' local query processors. The capability of the global query processor determines how user queries are reformulated, optimised and evaluated. The capabilities of the data sources' query processors determine how individual reformulated sub-queries are optimised and evaluated by the local query processors. To measure the performance criterion, cost models need to be available or specified for both global and local query processing, so that the performance of a set of users' queries on the global schema, reformulated via the mappings, can be measured.

5.5 Comparison with Related Work

In this section, we summarise and compare related work with the quality factors and associated quality metrics described above. We also discuss how techniques from related work can be used within our quality framework. Table 5.4 presents a summary of related work, and its relationship to our quality framework.

	Completeness	**Consistency**	**Accuracy**	**Minimality**	**Performance**
Schema	[43] LSs,GS,Onto [76] Data [27, 45] Data	[43] LSs,GS,Onto [68, 77] LSs,GS	[45, 27] Data	[43] LSs,GS,Onto	
Mapping	[42] Data [28] GS,Data [45, 27] Data	[42] Data [28] GS,Data [68, 77] LSs,GS [78] LSs,GS,M	[42] Data [76] Data	[42] Data [4] Data [65] M	[65] M
Query	[76] Data				

Table 5.4: Summary of Techniques

The work closes to ours is [43]. The authors propose several quality measurement methods for the DI context relating to the schema completeness, schema consistency and schema minimality quality criteria from our research perspective. Quality metrics are proposed based on information extracted from the schema metadata, which is one of the four types of DI elements that we consider in our approach. Their measurement method proposed for measuring schema completeness is defined as the proportion of concepts in the application domain that are represented by a schema. In order to use this measurement method in our approach, this method can be developed further to discover the sets of schema constructs that satisfy and do not satisfy this quality definition. Their measurement method proposed for measuring schema consistency is defined as the proportion of schema constructs representing the same concepts that are also defined with the same data type. This definition can be categorised as a schema consistency sub-criterion in our approach and their measurement method generates the set of schema constructs that are defined with consistent data types and the set of schema constructs that are not. Their measurement method proposed for measuring schema minimality is defined as the proportion of schema constructs where there do not exist other schema constructs in the same schema representing the same concept. This definition can be categorised as a schema minimality sub-criterion in our approach. The redundant schema constructs discovered

by using this quality metric are considered as the ones that do not satisfy this quality definition.

In [42], the authors discuss an approach for mapping selection based on the comparison of pairs of instances of the source schema and the target schema extracted via different mappings in a Data Exchange context. The principle of this approach is that the more similar the data extractable from the two schemas via the mappings, the better the mappings are. In order to discover the similarities of such data, several sampling and analysis functions are used. This work can be applied within our quality framework with a user-defined threshold on the similarity comparison indicating its desired level from the users' perspective. Mappings whose results are below this threshold are considered as not satisfying the quality criterion.

[28] discusses the quality of collaborative tasks, such as data integration, with respect to users' quality requirements expressed in the form of user feedback, in contrast to being expressed as logical statements over the quality hierarchy in our approach. Although their work is primarily focused on user feedback handling and clustering, their validation methods of the user feedback with respect to integrated resources can still be applied in our approach in order to identify the DI elements that are not consistent with the user feedback.

There also exists other work relating to our research whose original motivation is not from a quality perspective. However, such work can also be adapted for measuring quality factors in our approach. Instance-checking methods have been used in many works in order to validate and refine mappings for example [45, 27]. In this work, instances of the *GS* specified by users can be 'traced back' to the data sources via the mappings and users can discover if such source data are the users' desired ones. This can be developed further in our approach into the schema completeness, schema accuracy and mapping completeness sub-criteria. Constraint validation is

another area relating closely to our research. There is a rich breadth of such work, for example [68, 77]. This work allows the identification of schema constructs that may cause schema inconsistencies and can be embedded into our approach to measure the schema consistency and mapping consistency quality sub-criteria.

There is also relevant work on checking the consistency of mappings with respect to schema constraints using Distributed Description Logics and ontology reasoning techniques [78]. The work in [4] discusses the core mapping concept so as to generate the minimum set of mappings with respect to the users' queries. The work in [65] discusses the mapping minimality problem focusing on redundant mapping operations within the same mappings. Both [78] and [65] can be considered as supporting the mapping minimality sub-criterion in our approach as they could be applied to remove redundant mappings and optimise mappings in the integrated resource.

All the works discussed above are relevant to our research, but they need to developed further from the quality perspective in order to be applied within our quality framework.

5.6 Summary

We have defined in this chapter five quality criteria and their sub-criteria in the context of data integration: completeness, consistency, accuracy, minimality and performance. A set of quality factors associated with the completeness and consistency quality criteria have been proposed and discussed in detail, together with quality metrics for measuring such factors using information extracted from the DI elements.

The measurement methods that we have proposed in this chapter are not exhaustive. They are indicators of what is possible, and they may be refined and extended in the future, following validation with real-world case studies

and users. We briefly introduced the accuracy, minimality and performance quality criteria and leave detailed research into these as future work. A comparison of our research with other relevant work was also discussed and we showed how techniques from this work could be applied within our quality framework.

In Chapter 6, we will describe our data integration methodology that embeds quality assessment within the DI process. We will also present an integration architecture for the realisation of this methodology. The implementations of the quality factors and metrics introduced in this chapter will also be discussed in Chapter 6.

Chapter 6

Data Integration Methodology and Architecture

In the previous two chapters, we described our quality assessment approach, including the quality framework that focuses on representing the users' quality requirements and the set of quality criteria, factors and associated quality metrics for defining and determining the quality of integrated resources. In this chapter, we propose a data integration methodology that has embedded within it the quality assessment functionality, in contrast to the traditional data integration methodology which comprises mainly of schema matching and mapping tasks. We also propose in this chapter a DI architecture as a realisation of our methodology and we discuss the implementations of key components of this architecture.

This chapter is organised as follows. In Section 6.1, we discuss a DI architecture that realises our DI methodology proposed in Section 3.2 in Chapter 3 and we also illustrate the DI workflow using this architecture. The implementations of key components of this DI architecture and of the quality factors proposed in Chapter 5 are discussed in Section 6.2 including: the ontology representation of our QFDI, the transformation algorithm from

a relational schema to an ontology and the implementation of the quality factors defined in Chapter 5. Section 6.3 provides a summary of this chapter.

6.1 Data Integration Architecture and Workflow

In Section 3.2 of Chapter 3, we propose a DI methodology that contains a requirements gathering phase, an integration domain learning phase, an integration phase and a quality assessment phase. The integration process is then applied iteratively to refine the integrated resource with respect to the quality assessment results and in this way an integrated resource with better quality can be achieved. In this section, we propose a DI architecture as a realisation of our DI methodology.

6.1.1 DI Architecture with Quality Assessment Functionality

Our architecture (see Figure 6.1) is composed of four main components and covers three of the integration phases: Integration Domain Learning Phase, Integration Phase, Quality Assessment Phase. The four components are a *pre-existing schema matching tool*, *data integration tool* and *ontology matching tool*, and the new *Quality Measurement tool* developed in our research:

1) The schema matching tool, such as COMA++ (`http://dbs.uni-leipzig.de/Research/coma.html`), can discover correlations between schemas automatically. This information helps generate matches between schema constructs manually or (semi-)automatically.

2) The data integration tool provides functionalities for creating and storing the integrated resource and for global query processing; we are using the AutoMed data integration toolkit (`http://www.doc.ic.ac.uk/automed/`)

and we refer to this in our following discussion. In AutoMed, mappings between the *GS* and the *LSs* are composed of transformation pathways. Each transformation results in an intermediate schema. Modelling language specifications, schemas and transformation pathways are stored in the AutoMed repository. We will describe in more detail the AutoMed integration toolkit in the next section.

3) The ontology matching tool, as also supported by COMA++, discovers more precise correlations between ontology representations using semantic information, as surveyed in [79].

4) The Quality Measurement tool provides functionalities for the users to express their DI quality requirements as quality criteria, factors and relationships between them, measures each quality factor based on knowledge extracted from the integrated resource and also applies reasoning functionalities in order to have an integrated and consistent quality view of the integrated resource with respect to various users' quality requirements as discussed in Chapter 5.

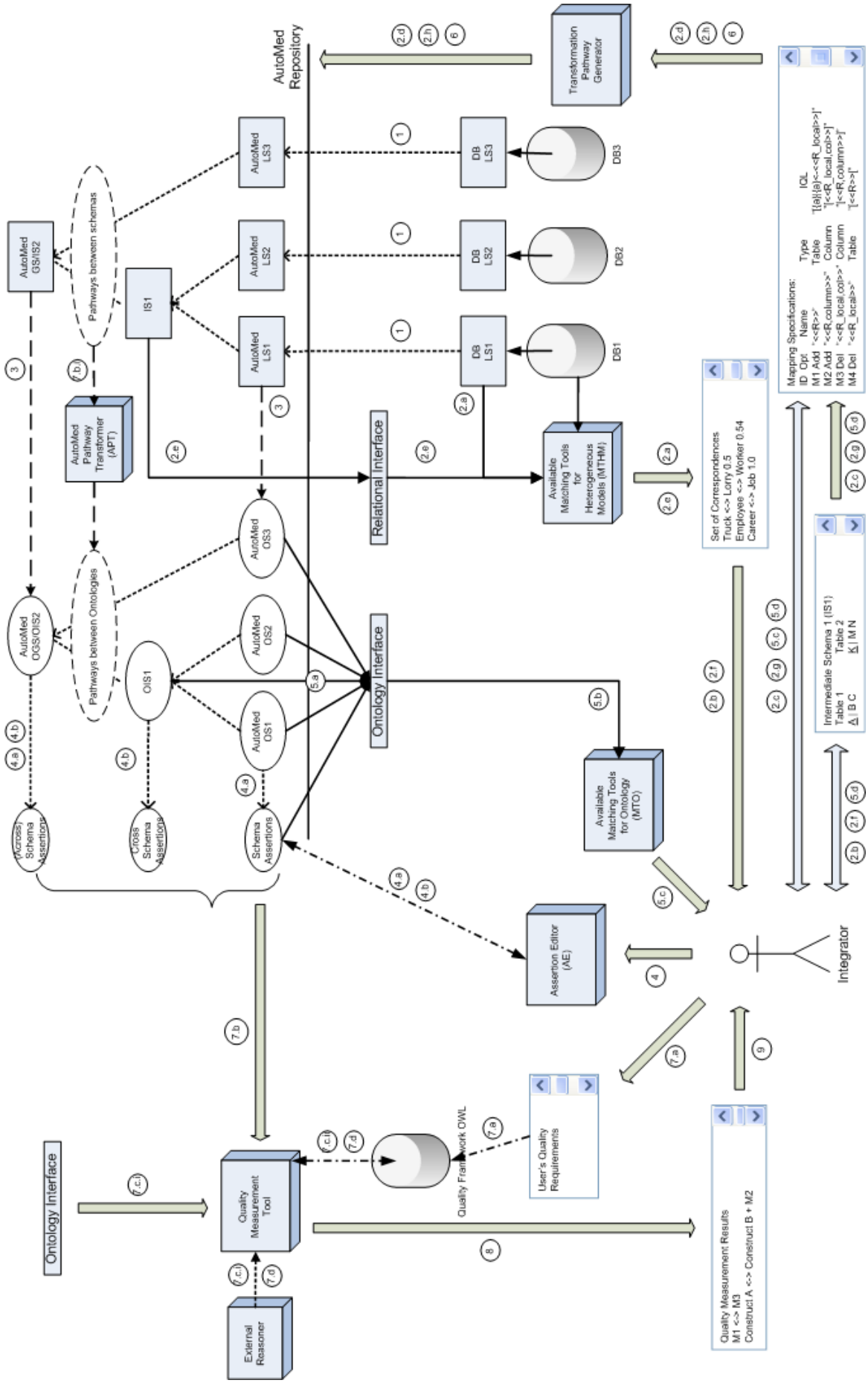


Figure 6.1: Integration Architecture with Quality Assessment

6.1.2 Data Integration Workflow

At the outset of a DI process, the data integrator needs to determine the available data sources and the correspondences between them in order to begin to design the mappings and the global schema. Additional information, such as domain specific knowledge, can then be added by the integrator, resulting in an iterative improvement of the integrated resource. The three integration processes supported are *Integration Domain Learning Phase*, *Integration Phase* and *Quality Assessment Phase*. The *Requirements Gathering Phase* is not covered explicitly in our architecture, but the tasks involved in this phase are required for supporting these three phases.

In the *Integration Domain Learning Phase* (steps 1 and 2 in Figure 6.1), the integrator first wraps the data sources (shown as DB LS1, DB LS2, DB LS3 in the figure) using the corresponding AutoMed wrappers. This results in metadata that describes the data sources being stored within the AutoMed STR repository. The integrator then defines the initial mappings by analysing the results returned after applying the COMA++ matching tool. For example, a "ladder" integration strategy [66] can be used in the architecture as follows. DB LS1 and DB LS2 are first matched, and the integrator examines the correspondences returned and designs the first intermediate schema IS1 (steps 2.a-2.b). IS1 will contain a set of constructs derived from LS1 and LS2. The integrator then defines mappings between IS1 and LS1, and IS1 and LS2. The mappings specified are automatically translated into AutoMed transformation pathways and stored in the AutoMed repository (steps 2.c-2.d). The matching tool is then run again on IS1 and DB LS3 in order to discover correspondences between these (step 2.e). The matching tool will be able to access the data underlying the virtual schema IS1 (derived from LS1 and LS2) via the global query processing capabilities of AutoMed. The integrator analyses the returned correspondences and designs the second intermediate schema IS2 (step 2.f). IS2 is also the same as

the final global schema *GS* in this example, but more generally this "ladder" integration strategy is repeated as many times as necessary. The integrator then defines the mappings between IS2 and IS1, and IS2 and LS3, and these mappings will also be stored in the AutoMed repository (steps 2.g-2.h).

According to [80], there is no perfect schema matching tool. Therefore, our process involves human effort in defining mappings from the schema matchings returned by the schema matching tool.

In the *Integration Phase* (steps 3-6 in Figure 6.1), all the local, intermediate and global schemas (LS1, LS2, LS3, IS1, GS/IS2) are first translated automatically into their corresponding OWL representation (OS1, OS2, OS3, OIS1, OGS/OIS2, respectively, in the figure), expressing the translation logic in the form of AutoMed transformation pathways (step 3). The transformation pathways between pairs of OWL schemas are automatically generated by using AutoMed to compose the transformation pathways between the corresponding non-OWL versions of the schemas with the transformation pathways expressing the translation logic. The integrator can then define additional assertions on OS1, OS2, OS3, OIS1, OGS/OIS2 capturing additional domain knowledge. These assertions can be defined within the same OWL schema, or across OWL schemas (step 4). In the former case, the assertions can be added to the local/global OWL schemas (OS1, OS2, OS3, OGS). In the latter case, assertions can be added to the intermediate OWL schemas (OIS1, OGS/OIS2). The integrator can then refine the existing mappings by examining the results returned from an ontology matching tool (in the current prototype, we use COMA++ again) (steps 5.a-5.b). The ontology matching tool accesses both the OWL schemas and their underlying data via AutoMed, and matches data source schemas OS1 and OS2. The integrator examines the matching results and defines mappings between OIS1 and OS1, and OIS1 and OS2 (step 5.c). The new mapping specification needs to be compared with the previous one defined in step 2.c (step 5.d): if a new

mapping has no corresponding mapping in the previous specification, then it is retained; if both a new and an old mapping involve the same schema constructs, then the integrator needs to refine the old mapping, using the information encompassed within the new mapping. The set of new mappings then needs to be translated into the corresponding AutoMed transformation pathways and stored in the AutoMed metadata repository (the integrator should first delete all existing mappings in the AutoMed metadata repository that have been subsumed by new versions) (step 6). Steps 4, 5 and 6 are repeated until the integrator is satisfied with the integration setting.

In the *Quality Assessment Phase* (steps 7-9 in Figure 6.1), the integrator first sets up the quality hierarchy comprising the appropriate quality criteria and quality factors, and then expresses the users' quality requirements as DL terminology axioms (step 7.a). The OWL representation of the DI elements that is input to the Quality Measurement tool (step 7.b) has been generated in the Integration Phase. For each quality factor, the integrator uses the Quality Measurement tool to measure the quality factor using its associated metric and to populate the quality framework with the results (step 7.c). After all the quality factors have been measured, inconsistencies between the various quality criteria/factors may be discovered by applying an ontology reasoner (step 7.d). For example, we can discover the DI elements that may cause inconsistencies across different users' quality requirements. A quality report presenting the quality measurement results will be generated. The integrator then examines the quality report and makes changes to the integrated resource, e.g. modifying the schema and mapping specifications, modifying the assertions, or changing the quality requirements in consultation with end-users (steps 8 and 9).

The whole process is iterative, and steps 4 onwards may be applied multiple times during the integration process.

6.2 Implementation of our DI Architecture

In our prototype implementation of the data integration architecture described in Section 6.1, we adopt the existing tools COMA++ and AutoMed for the schema matching, data integration and ontology matching tasks. The schema matching tool COMA++ is used in two tasks in our architecture, the identification of correspondences between schema constructs from data sources (step 2.a in Figure 6.1) and the identification of correspondences between ontology constructs that are transformed from the schemas from the data sources (step 5.b in Figure 6.1). COMA++ could be replaced or supplemented by other schema matching tools, such as Porsche [81] and Similarity Flooding [82].

In our prototype implementation, AutoMed is used by the integrators for three tasks. First, AutoMed is used to wrap the local schemas and the global schema into the representation supported by AutoMed (HDM) (step 1 in Figure 6.1). Data integrators also use AutoMed to create and store mappings (transformation pathways in AutoMed) (step 3 in Figure 6.1). AutoMed also supports query reformulation on the integrated resource. This function is invoked in query-related metrics, such as quality factors 4 and 8 in Chapter 5. Again, other data integration frameworks can be adopted in our architecture, such as Clio [22], so long as they support all functions described above. Our Quality Measurement tool is mainly invoked in step 7.c and we have implemented the major functions in this tool including: expressing formally the definitions of the users' quality requirements, measuring the quality factors involved in the users' requirements, and invoking reasoning on QFDI in order to detect the DI elements that cause inconsistencies between users' requirements and the integrated resources.

In this section, we first discuss the implementation of the schema to ontology transformation algorithm (Section 6.2.1). Then we discuss the implementation of the Quality Measurement tool, including the reasoning

component of our architecture that is based on the OWL ontology language and adopts the FaCT++ ontology reasoner for inference purposes (Section 6.2.2). We then discuss the implementation of the quality factors proposed in Chapter 5 (Section 6.2.3).

6.2.1 Schema to Ontology Representation

The translation of a relational global schema into an equivalent OWL representation in Section 6.1 the Integration Domain Learning Phase is undertaken using an algorithm based on [69], which describes the representation of relational databases in RDF. Similarly to [69], our translation of relational schemas into OWL can support both single-attribute and composite primary and foreign keys [38]. In this thesis, we focus on data sources that contain information represented in the Relational model only for the purpose of fast implementation of our DI architecture. But in principle, data sources containing information in other data models can be integrated, knowledge expressed in such data sources can be expressed in OWL, and the quality of the integrated resource can be measured by adopting our integration architecture.

The algorithm, listed in Panel 4 in Appendix C, takes an AutoMed relational schema S_{Rel} as input and outputs an AutoMed OWL schema S_{Ont}. The algorithm has three parts. The first part (lines 55–61), translates the relations of schema S_{Rel}. In particular, a relation $\langle\langle \mathsf{R} \rangle\rangle$ translates to a Class C in S_{Ont}, each of its attributes $\langle\langle \mathsf{R}, \mathsf{a} \rangle\rangle$ translates to a Property $\langle\langle \mathsf{a}, \mathsf{C}, \mathsf{rdfs:Literal} \rangle\rangle$, while the primary key of $\langle\langle \mathsf{R} \rangle\rangle$ translates into another Class $\langle\langle \mathsf{C_{pk}} \rangle\rangle$ and a Property $\langle\langle \mathsf{pk}, \mathsf{C}, \mathsf{C_{pk}} \rangle\rangle$. The second part (lines 62–70), translates the foreign key constraints of schema S_{Rel}. In particular, the algorithm creates two Class constructs, $\langle\langle \mathsf{C_{R_{fk}}} \rangle\rangle$ and $\langle\langle \mathsf{C_{S_{fk}}} \rangle\rangle$, representing the set of attributes of relation R and the set of attributes of relation S that reference the former. The algorithm also creates Property constructs $\langle\langle \mathsf{fk}, \mathsf{C_R}, \mathsf{C_{R_{fk}}} \rangle\rangle$,

$\langle\langle \mathsf{fk}, \mathsf{C_S}, \mathsf{C_{S_{fk}}} \rangle\rangle$ and $\langle\langle \mathsf{fk}, \mathsf{C_{S_{fk}}}, \mathsf{C_{R_{fk}}} \rangle\rangle$ that link the newly added Class constructs together with each other and with the Class constructs that represent relations R and S. The third part (line 71), which removes the relational schema constructs from schema S_{Ont} is straightforward and omitted [38].

For example, taking the example of a relational table `staff(sid,name,#studentID)` discussed in Section 2.4.2, its HDM representations are a *Table* construct $\langle\langle \mathsf{staff} \rangle\rangle$, three *Attribute* constructs $\langle\langle \mathsf{staff}, \mathsf{sid} \rangle\rangle$, $\langle\langle \mathsf{staff}, \mathsf{name} \rangle\rangle$ and $\langle\langle \mathsf{staff}, \mathsf{studentID} \rangle\rangle$, a *PKey* construct $\langle\langle \mathsf{staff\_pk}, \mathsf{staff}, \langle\langle \mathsf{staff}, \mathsf{sid} \rangle\rangle \rangle\rangle$, and a *FKey* construct $\langle\langle \mathsf{staff\_fk\_1}, \mathsf{staff}, \langle\langle \mathsf{staff}, \mathsf{studentID} \rangle\rangle, \mathsf{student}, \langle\langle \mathsf{student}, \mathsf{id} \rangle\rangle \rangle\rangle$, assuming there is another table `student(id,name))` and that `#studentID` of `staff` references `id` of `student`. According to our algorithm, the following set of corresponding ontology constructs are created: $\langle\langle \mathsf{staff} \rangle\rangle$ for the *Table* construct; $\langle\langle \mathsf{sid}, \mathsf{staff}, \mathsf{rdfs:Literal} \rangle\rangle$, $\langle\langle \mathsf{name}, \mathsf{staff}, \mathsf{rdfs:Literal} \rangle\rangle$ and $\langle\langle \mathsf{studentID}, \mathsf{staff}, \mathsf{rdfs:Literal} \rangle\rangle$ for the *Attribute* constructs; $\langle\langle \mathsf{pk}, \mathsf{staff}, \mathsf{sid} \rangle\rangle$ for the *PKey* construct; and $\langle\langle \mathsf{fk}, \mathsf{staff}, \mathsf{studentID} \rangle\rangle$, $\langle\langle \mathsf{fk}, \mathsf{student}, \mathsf{id} \rangle\rangle$, $\langle\langle \mathsf{fk}, \mathsf{id}, \mathsf{studentID} \rangle\rangle$ for the *FKey* construct referencing to the `student` table.

The advantage of using this method is that it is able to present different kinds of keys, such as single and composite primary keys and foreign keys. It is also able to support alternative candidate keys. The disadvantage is that it contains redundant extents for the ontology, since the extents of ontology objects overlap significantly. However, at the time this algorithm was developed, other similar algorithms close to our purpose had similar problems [70] and the disadvantage of this algorithm does not affect the main functionalities in our approach. It is an area of possible further work to reduce the redundancies contained in the ontology and also its extents. This algorithm has been used in the iSpider [83] and ASSIST [84] projects.

6.2.2 Implementation of an OWL Representation of QFDI

In our work, we implement our quality framework (QFDI) using a subset of the Ontology Web Language (OWL), called OWL-DL. We discuss and demonstrate here how OWL reasoning using an off-the-shelf OWL reasoner can be applied in order to discover DI elements that cause inconsistencies between the users' quality requirements as a whole and also individually.

OWL Language

Ontology Web Language (OWL) is a formal language developed with Description Logic as its logical foundation. There are three OWL languages with increasing expressive abilities, termed OWL-Lite, OWL-DL and OWL-Full. The OWL syntax and the corresponding DL expressions are listed in Table 6.1. We use OWL-DL in our research since it is expressive enough for representing our quality framework and also the reasoning capability of OWL-DL can support the reasoning requirements discussed in Chapter 4. There are also many off-the-shelf OWL reasoners that can be used in the implementation of our framework, such as FaCT++ (`http://owl.man.ac.uk/factplusplus/`) and Pellet (`http://clarkparsia.com/pellet/`).

DL Syntax	OWL-DL Syntax	Explanation
C	$Class$	a concept
R	$Property$	a general role
	$ObjectProperty$	an object role
	$DatatypeProperty$	a data role
$\top$	$Thing$	the universal concept
$\emptyset$	$Nothing$	the bottom concept
$C \sqcap D$	$intersectionOf$	conjunction
$C \sqcup D$	$unionOf$	disjunction
$\exists R.C$	$someValueFrom$	existential quantification
$\forall R.C$	$allValueFrom$	value restriction
R^+	$transitiveProperty$	transitive role
R^-	$inverseOf$	inverse role
$\geq_n R.C$	$minCardinality$	minimum cardinality of n
$\leq_n R.C$	$maxCardinality$	maximum cardinality of n
$C \sqsubseteq D$	$subClassOf$	subsumption axiom
$C \equiv D$	$equivalentClass$	equivalent axiom

Table 6.1: OWL-DL Syntax

An OWL Representation of QFDI

Figure D.1 in Appendix D illustrates the OWL implementation of our quality framework (see Section 4.1 in Chapter 4) in Protege. The *criterion* concept is represented as an OWL class termed $Criterion$ and the extent of this class is the identifiers representing different quality criteria, such as criteria $c1$ and $c2$ in Chapter 4, Section 4.3.2. The *factor* concept is also represented as an OWL class termed $Factor$ and the extent of this class is the identifiers representing different quality factors, appended with strings s or ns representing the aspects of a quality factor f that can or cannot be satisfied by the integrated resource, respectively, e.g., f_1s and f_1ns represent the part of the quality factor f_1 that can and cannot be satisfied by the integrated resources. We use this technique in order to define

the closure of properties discussed later. The *Item* concept is represented as an OWL class termed *Item* and the extent of this class is the identifiers representing different DI elements, such as the table $LS1\_educator$ and the attribute $LS1\_educator\_tid$. Four OWL object properties are created, *contains_sat*, *contains_not_sat*, *is_contained_sat* and *is_contained_not_sat*, where *is_contained_sat* and *is_contained_not_sat* are the inverse properties of *contains_sat* and *contains_not_sat*, respectively. The domain of *contains_sat* and *contains_not_sat* is the OWL *Factor* class and the range is the *Item* OWL class. We also set all individuals to be distinct, meaning each individual is uniquely identified by their name. The individuals of the *Factor* class are linked to the individuals of the *Item* class with these properties as listed in Table 4.3 in Chapter 4.

The *User* class represents the set of users' requirements and each user is represented as a subclass of the *User* class, such as users *A*, *B*, *C* in Chapter 4. In our implementation of the *UserQR* concept, we decompose the logical statements for expressing users' quality requirements into smaller segments. Taking the example in Chapter 4, requirement *A*.1 is defined as $(Sc \sqcap \forall contains\_sat^{-}.\{f_1\}) \sqsubseteq (Sc \sqcap \forall contains\_sat^{-}.\{f_2\})$. In our implementation, we define two additional classes *A*.1.1 and *A*.1.2, where $A.1.1 = (Sc \sqcap \forall contains\_sat^{-}.\{f_1\})$ and $A.1.2 = (Sc \sqcap \forall contains\_sat^{-}.\{f_2\})$. We then create a general axiom indicating that *A*.1.2 is a subclass of *A*.1.1. The same process applies to *B*.1 and *C*.1. We then have a set of segments: $A.1.1 = (Sc \sqcap \forall contains\_sat^{-}.\{f_1 s\})$, $A.1.2 = (Sc \sqcap \forall contains\_sat^{-}.\{f_2 s\})$, $B.1.1 = ((Sc \sqcap \forall contains\_not\_sat^{-}.\{f_1 ns\}) \sqcap (Sc \sqcap \forall contains\_not\_sat^{-}.\{f_2 ns\}))$, $B.1.2 = \emptyset$, $C.1.1 = (Sc \sqcap \forall contains\_sat^{-}.\{f_1 s\} \sqcap \forall contains\_not\_sat^{-}.\{f_2 ns\})$, $C.1.2 = (Sc \sqcap ProgrammeDirector)$, $A.1.1 \subseteq A.1.2$, $B.1.1 \equiv B.1.2$ and $C.1.1 \subseteq C.1.2$.

We also adopt a closed-world assumption in the reasoning over our quality framework, but most off-the-shelf reasoners support reasoning under the

open-world assumption. In order to implement this, we use value restriction syntax that explicitly indicates the extent relating to a role. This is also termed the closure of a role [1]. Figure 6.2 lists the four role closures relating to the example in Section 4.1.

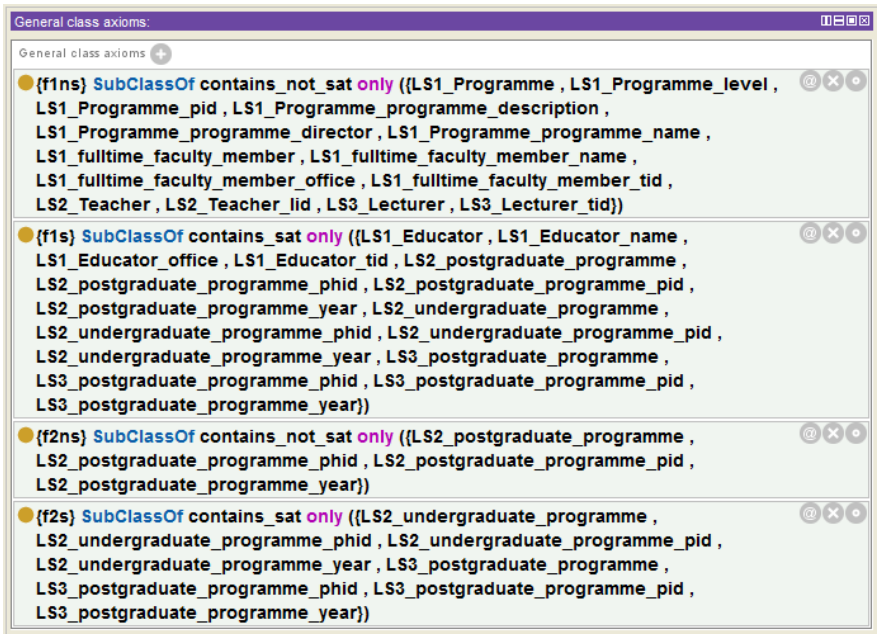


Figure 6.2: Implementation of the Closed-World Assumption in OWL-DL

Reasoning in QFDI using FaCT++

In our current implementation, we use the FaCT++ reasoner[1] to undertake inferencing over our quality framework because FaCT++ is supported in the Protege ontology editor and it has sufficient reasoning power to support our quality framework. Other reasoners that have equivalent inference power could also be used, such as Pellet[2]. As discussed in Chapter 4 the reasoning on the QFDI could discover inconsistencies between different users' quality

[1] `http://owl.man.ac.uk/factplusplus/`
[2] `http://clarkparsia.com/pellet/`

requirements and also inconsistencies of each individual requirement with respect to the DI elements in the integrated resource. In the example we discussed in Section 4.1, we expect that requirements $A.1$ and $C.1$ are not consistent. This can be discovered by the FaCT++ reasoner and suggestions can be given by the reasoner about the concepts causing the inconsistency, as illustrated in Figure 6.3.

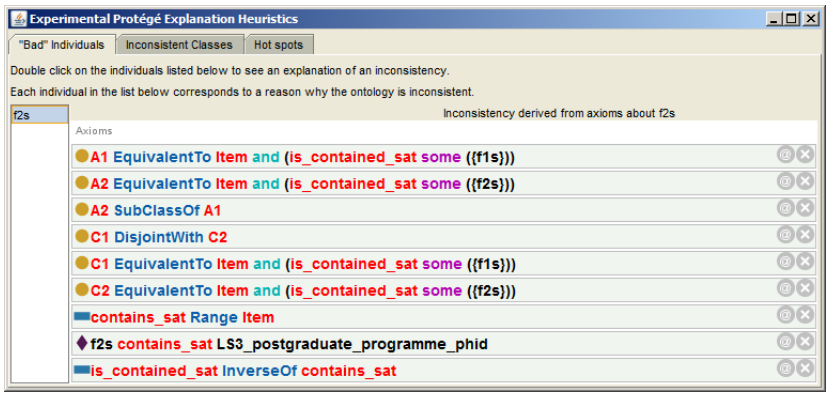


Figure 6.3: Inconsistency between User Requirements $A.1$ and $C.1$

Suppose we remove requirement A.1 completely from consideration as discussed in Chapter 4 and we run the FaCT++ reasoner again. This now shows that no inconsistencies have been discovered between the users' quality requirements.

Next, we run the FaCT++ reasoner to discover possible inconsistencies of individual quality requirements with respect to the DI elements in the integrated resource. The reasoner detects that requirement $C.1$ is not consistent (see Figure 6.4 for the error messages). The reasoner has also indicated that requirement $B.1$ is consistent with respect to the integrated resource.

We can then make modifications on the DI elements that do not satisfy quality factor $f2$ referred to in $C.1$ as discussed in Chapter 4 and run the reasoning again. The reasoner will now show that no inconsistencies are discovered.

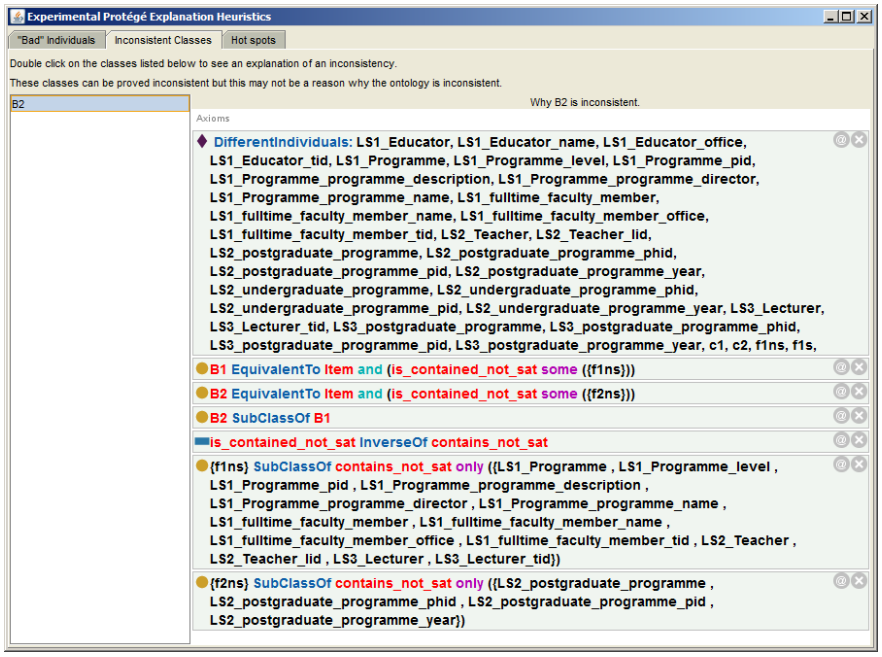


Figure 6.4: Inconsistency between $B.1$ and the Instance Level Information

6.2.3 Implementations of Quality Factors

In this section, we discuss how the quality factors proposed in Chapter 5 are implemented using AutoMed.

FACTOR 1: Schema completeness is measured as the proportion of information coverage of the global schema *GS* via the mappings *M* with respect to the information represented by all local schemas *LSs*.

In AutoMed, this quality factor can be implemented by using the $sc()$ function [65]. Given an IQL query q posed on schema S, the $sc()$ function takes q as its input and returns the set of schema constructs referenced by q. Each of the schema constructs in this set is then processed further by the $sc()$ function and returns the set of AutoMed schemes upon which the definition of this schema construct depends.

Measuring this quality factor with GAV mappings, we undertake the following steps, denoting all transformations in the AutoMed repository as T.

1. For each extensional GS construct $o \in extensional(GS)$, we search through T and find the "add" or "extend" transformations that derive o. If there are no such steps, then o appears in one or more of the LSs, and we add o to the set $sources(LSs, o)$.

2. For each discovered "add" or "extend" transformation, obtain the IQL query associated with this transformation, denoted by q_{GAV}.

3. Apply the $sc()$ function to q_{GAV} and obtain the set of AutoMed schemes that are referenced in this query.

4. For each such scheme, if it is derived from other schemes, repeat the first 3 steps for it. Once the LS schemes are detected, we add these local schema constructs to $sources(LSs, o)$. Finally, we will obtain a set of extensional local schema constructs from which o is derived, $sources(LSs, o)$.

5. Repeat the above four steps for all extensional GS constructs. Create a set containing all unique local schema constructs in $sources(LSs, o)$ for each $o \in extensional(GS)$, denoted by $sources(LSs, GS)$.

6. Count the total number of extensional LS constructs in all LSs as $\sum_{i=0}^{n} |extensional(LS_i)|$, and compute $\frac{|sources(LSs,GS)|}{\sum_{i=0}^{n} |extensional_(LS_i)|}$ as the measurement of this quality factor for the GAV mappings.

Panel 1: Pseudocode for traceGAV

Input: IQL query q

Output: A set of local schema constructs C

```
1  C = ∅;
2  schemes = sc(q);
3  for e ∈ schemes do
4      Obtain the "add" or "extend" transformation T'' for deriving e;
5      if T'' = ∅ then
6          Obtain the local schema construct in e as C;
7      else
8          for t ∈ T'' do
9              Obtain the IQL query q_t in t;
10             C = C ∪ traceGAV(q_t);
11     return C;
```

Panel 2: Pseudocode for Factor 1

Input: AutoMed Local Schemas *LSs*, the Global Schema *GS*, AutoMed Transformations T, GAV/LAV mapping type indicator gl

Output: Schema Completeness λ_1

```
sources(LSs, GS) = ∅;
LAVdefined(LSs) = ∅;
if gl = GAV then
    for o ∈ extensional(GS) do
        Obtain T' that are the "add" or "extend" transformations deriving
        o where T' ⊆ T;
        if T' = ∅ then
            sources(LSs, GS) = sources(LSs, GS) ∪ o;
        else
            for t ∈ T' do
                Obtain the IQL query q_t in t;
                sources(LSs, o) = traceGAV(q_t);
                sources(LSs, GS) = sources(LSs, GS) ∪ sources(LSs, o);
else if gl = LAV then
    for o ∈ extensional(LSs) do
        Obtain T' that are the "delete" or "contract" transformations
        deriving o where T' ⊆ T;
        if T' = ∅ then
            LAVdefined(LSs) = LAVdefined(LSs) ∪ o;
        else
            for t ∈ T' do
                Obtain the IQL query q_t in t;
                u = traceLAV(q_t);
                if u = 1 then
                    LAVdefined(LSs) = LAVdefined(LSs) ∪ o;
λ1 = |sources(LSs,GS) ∪ LAVdefined(LSs)| / |extensional(LSs)|;
```

Measuring this quality factor with LAV mappings, we undertake the following steps:

1. For each extensional LS construct $o \in extensional(GS)$, we search through T and find the "delete" or "contract" transformations that derive o. If there are no such steps, then o appears in the GS, and we add o to the set $LAVdefined(LSs)$.

2. For each discovered "delete" or "contract" transformation, obtain the IQL query associated with this transformation, denoted as q_{LAV}.

3. Apply the $sc()$ function to q_{LAV} and obtain the set of AutoMed schemes that are referenced in this query.

4. For each such scheme, if it is derived from other schemes, repeat the first 3 steps for it. If at least one GS scheme is detected, we will add o to $LAVdefined(LSs)$.

5. Count the total number of available extensional LS constructs in all LSs as $\sum_{i=0}^{n} |extensional(LS_i)$, and compute $\frac{|LAVdefined(LSs)|}{\sum_{i=0}^{n} |extensional(LS_i)|}$ as the measurement of this quality factor for the LAV mappings.

Panel 3: Pseudocode for traceLAV

```
Input: IQL query q
Output: Indicator u
39 u = 0;
40 schemes = sc(q);
41 for e ∈ schemes do
42     Obtain the "delete" or "extract" transformation T'' for deriving e;
43     if T'' = ∅ then
44         Obtain the schema construct referenced in e, denoted by c;
45         if c ∈ extensional(GS) then
46             u = 1;
47         else
48             u = 0;
49     else
50         for t ∈ T'' do
51             Obtain the IQL query q_t in t;
52             u = traceLAV(q_t);
53     return u;
```

FACTOR 2: Schema completeness is measured as the average level of coverage of the extensional local schema constructs that provide overlapping but possibly partially complete information for deriving the same global schema constructs.

In AutoMed, for both the GAV and LAV approaches, we need to detect first all distinct real-world concepts represented by the extensional schema constructs in the *LSs*, $reduce(\bigcup_{i=1}^{n} concepts(LS_i))$. We use a domain ontology that relates to the integration domain and assume that each local schema construct has a corresponding ontology concept. This quality factor can be implemented as follows:

1. Identify a domain ontology relating to the integration domain and establish 1-1 correspondences between the LS constructs and this domain ontology using the names suggested by the construct labels and ontology concepts. This is equivalent to the $\bigcup_{i=1}^{n} concept(LS_i, O)$ function.

2. Identify the set of unique ontology concepts represented by the local schema constructs, denoted by $reduce(\bigcup_{i=1}^{n} concept(LS_i, O), O)$. By unique ontology concepts, we mean the ontology concepts that do not hold any equivalent or subsumption relationships with other concepts in this set. For equivalent concepts, we choose any of these as the unique concept and all other equivalent concepts are identified by this unique one. For subsumption relationships, we remove concepts that are subsumed by another concept.

3. Determine the set of schema constructs that have the same ontology representations, denoted by $extensional(LSs, c)$, where c is one of the unique concepts. $sources(LSs, M_{GAV}, c)$ is the set of local schema constructs that are referenced in a GAV mapping for c and $LAVdefined(LSs, M_{LAV}, c)$ is the set of local schema constructs that are derived by LAV mappings for c.

This quality factor can then be calculated using Formula 5.4.

FACTOR 3: Mapping completeness is measured as the proportion of the local schema constraints removed by the mappings without information loss compared with the total number of local schema constraints removed.

In AutoMed, schema constraints are expressed as constraint constructs associated with a query that needs to evaluate to true. Schema constraints can be added and deleted in the transformation pathways. In the former case, constraints can only be added using the "addConstraint" primitive. In

the latter case, constraints can be deleted either explicitly or implicitly. In AutoMed, a constraint is deleted explicitly by using the "deleteConstraint" primitive. A constraint can also be deleted automatically, if its associated extensional schema construct is deleted from the AutoMed repository.

If the GAV approach is used, this quality factor is implemented as follows:

1. Determine the set of constraint constructs on the local schemas, denoted by $Constraint(LSs)$ and the set of extensional schema constructs with which such constraints are associated $Construct(LSs, Constraint(LSs))$.

2. Determine the set of constraint constructs that have been deleted either explicitly or implicitly, denoted by $removed(constraints(LSs), M_{GAV})$.

3. We then need to determine if there exists a corresponding constraint c' for $c \in removed(constraints(LSs), M_{GAV})$. In AutoMed, this can be achieved by comparing both the constraint categories of c and c', and the extents of the schema constructs referenced in c and c'. If c and c' are defined in the same constraint category, such as "mandatory" or "unique" proposed in [73] and $Construct(GS, c')$ is the corresponding construct of $Construct(LSs, c)$, where $ext(Construct(GS, c')) \subseteq ext(Construct(LSs, c))$, assign 1 to $corr(q_c, reformulate(q_{c'}, M))$. Otherwise, assign 0 to it.

The degree of completeness of this integrated resource is then calculated as in Formula 5.5. If the LAV approach is used, this quality factor can be implemented following similar steps as the GAV approach, but we need to determine for each constraint removed from the GS whether the LAV mappings introduce a corresponding constraint to the local schemas.

FACTOR 4: Query completeness is measured as the average level of coverage of each real-world concept represented by the

local schema constructs referenced in the queries arising from reformulating the set of users' queries on the *GS*.

In AutoMed, for both the GAV and LAV approaches, we first need to calculate the set of queries on the local schemas that are reformulated from the user-defined queries on the GS via GAV or LAV query reformulation, denoted by $reformulat(q, M)$. We then determine, for all concepts represented by the extensional local schema constructs referenced in $reformulat(q, M)$, the average degree of the coverage of these concepts. This can be accomplished in the following steps:

1. Identify the set of local schema constructs that have been referenced in $reformulate(q, M)$, denoted by $construct(reformulate(q, M))$.

2. Determine the set of unique concepts represented by these constructs, denoted by $rccr(q, M, O) = reduce(concept(construct(reformulate(q, M)), O), O)$.

3. For each concept $c \in rccr(q, M, O)$, we determine the set of extensional local schema constructs that represent c or whose ontology representations are equivalent to or subsumed by c, denoted by $extensional(LSs, c)$. We also determine the set of local schema constructs that represent c or whose ontology representations are equivalent to or subsumed by c, denoted by $construct(reformulate(q_{GS}, M), c)$. This quality factor can then be calculated by Formula 5.6.

FACTOR 5: Schema consistency is measured as the number of *GS* constructs whose definitions and associated constraints can be applied to their corresponding local schema constructs, if they exist, compared with the total number of *GS* constructs.

This quality factor comprises of two constraints, the extensional construct definitions and the constraints restricting the extents of the relationship constructs. In the former case, this quality factor can be implemented

in AutoMed by simply checking the data type and value range of the two corresponding local and global schema constructs. In the latter case, this quality factor can be implemented as follows:

1. We need to determine the set of constraints on the GS, denoted by $constraints(GS)$.

2. For each constraint $o \in constraints(GS)$, we reformulate the query comprising o, q_o, and obtain a set of queries on the local schemas, $reformulate(q_o, M_{GAV})$ using the AutoMed query reformulator. For each query $q \in reformulate(q_o, M_{GAV})$ on a local schema, we need to detect if q is satisfied by the local schema with respect to the current schema instance.

3. This quality factor can then be calculated using Formula 5.8 for the constraint construct aspect, where $consistent(q, LSs)$ is assigned 1 if q can be evaluated to true. Otherwise, $consistent(q, LSs)$ is assigned 0.

FACTOR 6: Schema consistency can be measured as the proportion of local schema constructs that satisfy their real-world semantics and whose corresponding *GS* constructs also satisfy the same real-world semantics.

In order to implement this quality factor in AutoMed, we need to first determine the set of local schema constructs whose semantics are consistent with their real-world semantics. This is implemented in the following way:

1. Discover the mappings between the extensional constructs in the local schemas LSs and the domain ontology.

2. Discover the mappings between the extensional construct in the global schema GS and the domain ontology.

3. For detecting the consistency between the schema constructs and their corresponding domain ontology representation with respect to the data type and value range definition, we can simply compare these definitions and determine if the extent of the schema construct can be transformed to be defined as the extent of the ontology construct.

4. For detecting the consistency between the schema constructs and their corresponding domain ontology representation with respect to the sch -ema constraints, we need to determine the set of constraint constructs on the local schemas, denoted by $Constraint(LSs)$ and the set of extensional schema constructs with which such constraints are associated $Construct(LSs, Constraint(LSs)) \subseteq Construct(LSs)$.

5. Determine if the semantics of these constraints in $Constraint(LSs)$ is consistent with the semantics represented in the ontology between the ontology representations of the schema constructs in $Construct(LSs$, $Constraint(LSs))$.

6. If they are consistent in both cases, we denote such local schema constructs as the set $consistent(extensional(LSs), O)$. We then need to discover their corresponding global schema constructs $corr(GS, consis\ tent(extensional(LSs), O), M)$ and apply the same process again. In this case, we obtain a set of local schema constructs that satisfy this quality factor, denoted by $consistent(corr(GS, consistent(extensional\ (LSs), O), M), O)$.

This quality factor can then be calculated using Formula 5.9.

FACTOR 7: Mapping consistency can be measured as the proportion of local schema constraints that are not violated by new constraints introduced by the mappings M.

AutoMed supports new constraints to be added or deleted by using the AutoMed transformation primitives, *addConstraint(con,q)* and *deleteCon-*

straint(con,q), where *con* is the constraint identifier and q is the query representing this constraint.. This quality factor can be calculated in the following steps for GAV mappings:

1. Determine the set queries embedded in *addConstraint()* transformations in the AutoMed repository, denoted by Q_c and determine the set of the constraints on the local schemas, denoted by $constraints(LSs)$.

2. For each local schema constraint $o \in constraints(LSs)$, disable all other constraints and evaluate each query in the set Q_c. If all queries in Q_c evaluate to true, denoted by $evalute(q_o, Q_c, M)$, we say that o is not violated by any additional constraints introduced in the mappings and we assign $evalute(q_o, Q_c, M)$ to be 1. Otherwise, assign $evalute(q_o, Q_c, M)$ to be 0.

Calculate this quality factor using Formula 5.10.

If LAV mappings have been used, this quality factor can be calculated in the following steps:

1. Determine the set of constraints on the local schemas, denoted by $constraints(LSs)$.

2. For each local schema constraint $o \in constraints(LSs)$, reformulate the query associated with o using the LAV mappings and determine a set of reformulated queries on the *GS*, denoted by $reformulate(q_o, M_{LAV})$.

3. Determine if both q_o and $reformulate(q_o, M_{LAV})$ evaluate to true on the local schemas and the *GS*, respectively. If so, assign $evaluate(q_o, reformulate(q_o, M_{LAV}))$ to be 1. Otherwise, assign it to be 0.

Calculate this quality factor using Formula 5.11. If both GAV and LAV mappings are created, this quality factor can be calculated using Formula 5.12.

FACTOR 8: The degree of query consistency is measured as the level of satisfaction of users' requirements relating to the relationships between the information retrieved by pairs of users' queries in a DI setting.

For both the GAV and LAV approaches, this quality factor is defined as the average level of satisfaction of Req. For each $Q_i^{LS,GS} \in Req$, the level of satisfaction of $Q_i^{LS,GS}$ is defined by calculating the level of satisfaction of *relationship*, denoted by $satisfy(Q_i^{LS,GS})$ with respect to q_{LSs} and $reformulate(q_{GS})$. In our approach, we calculate the level of satisfaction of *relationship* by comparing the results of evaluating q_{LSs} and $reformulate(q_{GS})$ over the integrated resource. This can be achieved by using AutoMed's *join* and *count* operators, assuming the schema constructs referenced in q_{LSs} and $reformulate(q_{GS})$ are the same. For the $=$ relationship (Formula 5.13), the extent of $ext(q_{LSs}) \cap ext(reformulate(q_{GS}))$ can be calculated by a natural join of the result set returned by q_{LSs} and $reformulate(q_{GS})$, while the extent of $ext(q_{LSs}) \cup ext(reformulate(q_{GS}))$ can be calculated by taking a left outer join of the result set returned by q_{LSs} and $reformulate(q_{GS})$ and a right outer join of the result set returned by q_{LSs} and $reformulate(q_{GS})$, then performing a union of both result sets. A *count* operation in AutoMed can then be applied in order to calculate the number of tuples in these result sets. For the $\subseteq$ and $\supseteq$ relationships (Formulas 5.13, 5.14), the same method can be applied.

6.3 Summary

We have proposed in this chapter a data integration methodology that has quality assessment functionality embedded within it. We have also proposed a DI architecture as a realisation of this methodology and we have discussed the implementations of the major components of this architecture. We have

described an implementation of the QFDI in OWL-DL and how the reasoning can be performed by using off-the-shelf ontology reasoners in order to detect inconsistent users' quality requirements and DI elements that violate such requirements in the integrated resource. We have also discussed the implementation of the quality factors proposed in Chapter 5 using AutoMed.

Our main contribution in this chapter is the integration methodology incorporating quality assessment, the integration architecture and the implementation of the quality factors. Our methodology includes the requirements gathering phase, the integration domain learning phase, the integration phase and the quality assessment phase. In contrast to other integration methodologies, a key distinguishing characteristic of our methodology is the quality assessment phase embedded within successive iterations of the integration process.

In the next chapter, we will consider a real-world integration project in the life sciences domain and will demonstrate how our quality framework, quality criteria, measuring and reasoning methods, and integration architecture can be used to assess and improve the quality of the integrated resource.

Chapter 7

Evaluation

In the previous chapters, we discussed in detail our approach for assessing the quality of integrated resources. Our approach comprises of the quality framework (QFDI), a set of quality criteria, factors and metrics, an integration methodology with quality assessment functionality embedded within it and an integration architecture as a realisation of our integration methodology. In this chapter, we aim to demonstrate the usefulness of our quality assessment approach and evaluate our approach with respect to a real integration project in the life sciences domain called iSpider (see `http://www.ispider.manchester.ac.uk/`).

The evaluation approach we undertake in this chapter includes three steps: we first generate an initial integrated resource by adopting one of the local schemas as the global schema as was done in the original iSpider project. Second, we measure the quality of this integrated resource with respect to a subset of the quality factors proposed in Chapter 5 (we consider in this chapter quality factors 1, 4 and 7) and calculate the overall quality of this integrated resource. Third, we validate a set of users' quality requirements and identify possible amendments that should be made in order to fulfil these requirements. Fourth, we iteratively apply these amendments

to our initial integrated resource, make the quality measurements and compare the overall quality of the updated integrated resource with the overall quality of the initial integrated resource.

In this chapter, we apply Quality Factors 1, 4 and 7 that have been proposed in Chapter 5. In contrast to the case study introduced the in Chapter 3, the iSpider project has two advantages for our evaluation purpose in this chapter. First, this case study is developed based on a real-world data integration project with more realistic users' requirements. Second, the integrated resource is more complex than the case study in Chapter 4 and it is therefore more suitable for evaluating our whole approach.

This chapter is organised as follows. Section 7.1 introduces the initial integrated resource designed for the iSpider project including the three data sources and the domain ontology. In Section 7.2, we describe the information collected from the original iSpider project users including a set of users' queries, users' quality requirements and users' assertions. We then demonstrate the integration of these data sources and several quality improvements to the integrated resource in three iterations in Section 7.3. Section 7.4 provides a summary of this chapter.

7.1 The Evaluation Domain

In order to evaluate our quality assessment approach, we use an existing integration project, called In Silico Proteome Integrated Data Environment Resource (iSpider). This project aims to develop an integrated platform of proteome-related resources, using existing standards from proteomics, bioinformatics and e-Science, for answering users' requests expressed as a set of queries over this platform. In our integration architecture, the integration process comprises of the Requirements Gathering Phase, Integration Domain Learning Phase, Integration Phase and Quality Assessment Phase.

We reuse the schemas and mappings initially created by the iSpider team to demonstrate how an initial integrated resource can be created and how its quality can be iteratively improved by applying our quality assessment methods. The iSpider project is developed based on AutoMed and, therefore, the following discussions are presented using the AutoMed syntax for the mappings and the queries.

7.1.1 Data Sources

In the iSpider project, three autonomous proteomics resources are integrated and all of them contain overlapping and distinct information on protein/peptide identification. All data sources are represented using the Relational model. The original schemas of the data sources are huge and we illustrate here only the core part of such schemas to illustrate our approach. We refer the readers to the websites of these data sources for full details of them.

The Proteome Experimental Data RepOsitory (PEDRo `http://pedro.man.ac.uk/`) stores a collection of descriptions of experimental data sets in proteomics such as details of the experimenter, the sample source, the methods and equipment employed and results from these experimentations. In our evaluation, we consider the PEDRo schema shown in Figure 7.1.

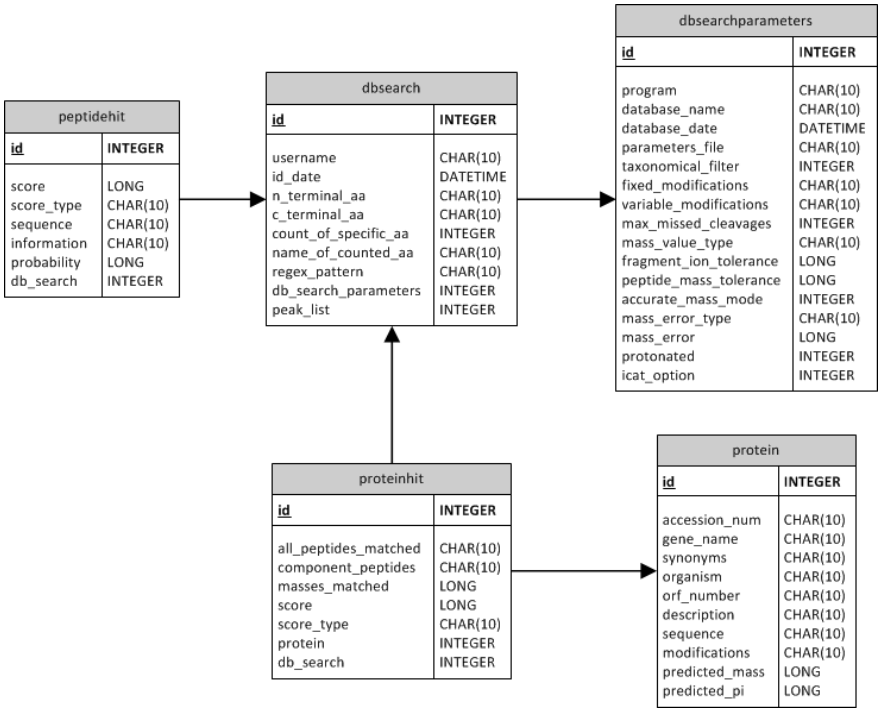


Figure 7.1: Data Source PEDRo

Two foreign keys are defined in the *proteinhit* table, one with the *db_search* attribute referencing the *id* attribute in the *dbsearch* table, and the other with the *protein* attribute referencing the *id* attribute in the *protein* table. Another foreign key is defined in the *dbsearch* table with the *db_search_parameters* attribute referencing the *id* attribute in the *dbsearchparameters* table. Similarly, another foreign key is defined in the *peptidehit* table with the *db_search* attribute referencing the *id* attribute in the *dbsearch* table.

The Global Proteome Machine Database (gpmDB `http://www.thegpm.org/`) is a publicly available database with tens of thousands of data contributed by researchers around the world in order to assist in the validation of peptide MS/MS spectra and protein coverage patterns. We consider the gpmDB schema shown in Figure 7.2 in our evaluation process.

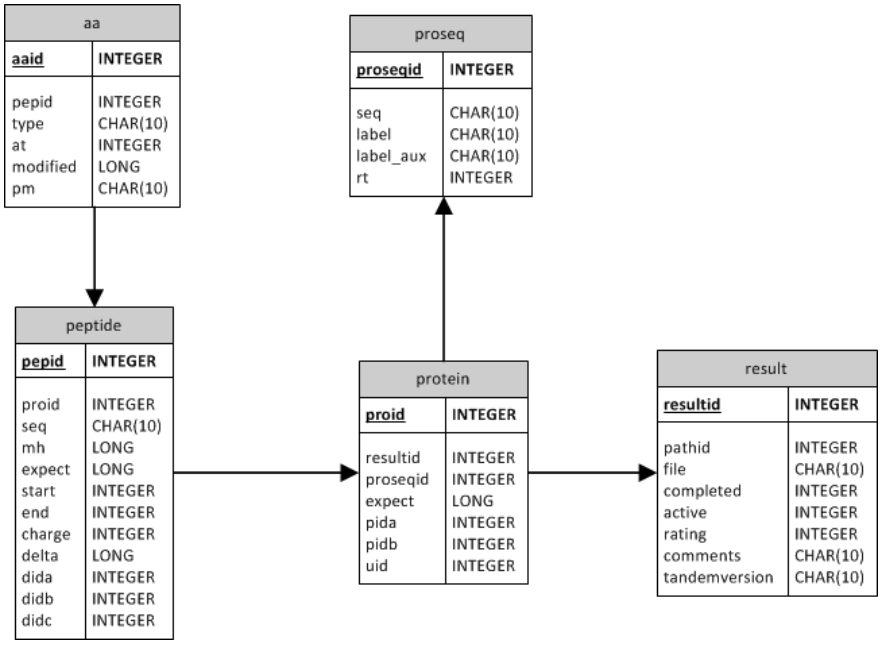


Figure 7.2: Data Source gpmDB

A foreign key is defined in the *aa* table with the *pepid* attribute referencing the *peptid* attribute in the *peptide* table. Another foreign key is defined in the *peptide* table with the *proid* attribute referencing the *proid* attribute in the *protein* table. Two foreign keys are defined in the *protein* table referencing the *proseqid* attribute in the *proseq* table and the *resultid* attribute in the *result* table respectively.

PepSeeker (`http://nwsr.smith.man.ac.uk/pepseeker`) is a database capturing the peptide identification and ion information from proteome experiments. The database currently contains 185000+ peptides and associated database search information. Users of this database can retrieve peptide, protein and spectral information based on protein or peptide information, such as amino acid sequences. We consider the PepSeeker schema in Figure 7.3 in our evaluation process.

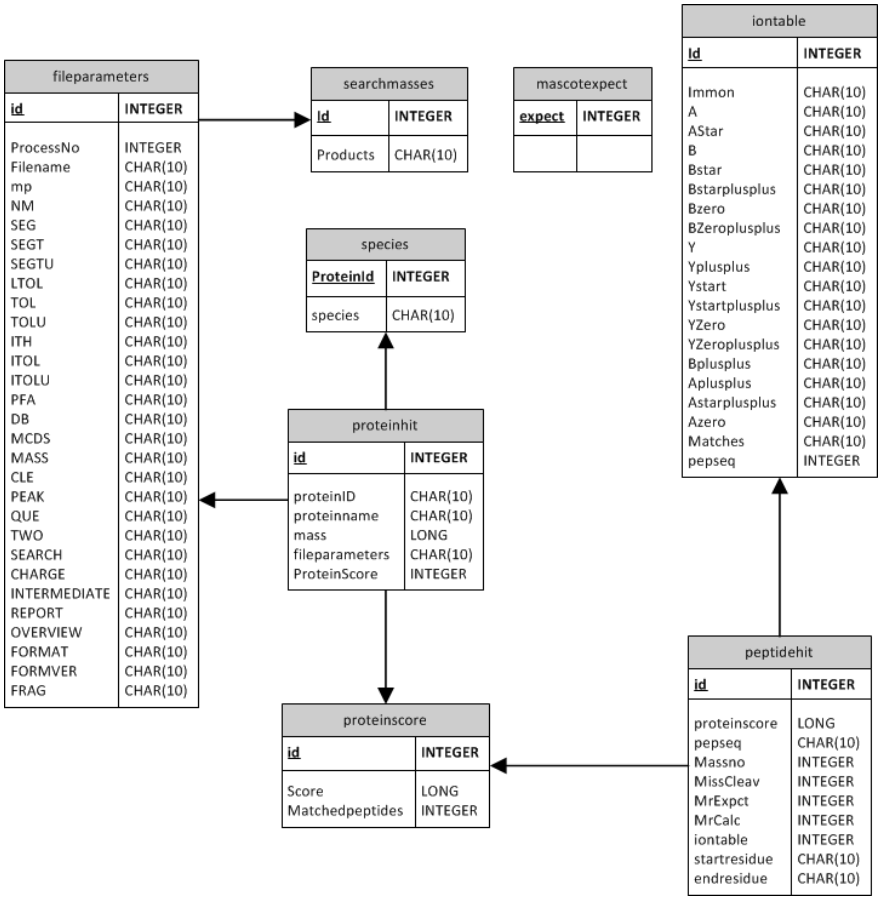


Figure 7.3: Data Source PepSeeker

Three foreign keys are defined in the *proteinhit* table with the *ProteinScore* attribute referencing the *id* attributes in the *proteinscore* table, the

fileparameters attribute referencing the *id* attribute in the *fileparameters* table, and the *proteinID* attribute referencing the *ProteinId* attribute in the *species* table. Two foreign keys are defined in the *peptidehit* table with the *iontable* attribute referencing the *id* attributes in the *iontable* table and with the *proteinscore* attribute referencing the *id* attribute in the *proteinscore* table. Another foreign key is defined in the *fileparameters* table with the *MASS* attribute referencing the *id* attribute in the *searchmasses* table.

7.1.2 Domain Ontology

In the life sciences domain, there exist many ontologies for representing information about different parts of this very large domain. However, there is no existing domain ontology that represents the same proteomics concepts as appearing in the schemas in the iSpider project. Therefore, in this case study, we consulted the information provided by the original iSpider project domain experts and identified manually the schema constructs that represent the same real-world concepts. Figures E.4, E.5 and E.6 in Appendix E.3 list all such concepts and their corresponding schema constructs. Since no real-world ontology has been discovered for this project, we do not transform the information represented in the Relational model into an ontological representation, but we work directly with the correspondences of the data source schema constructs to the identified real-world concepts. This also serves to illustrate that the quality measurements aspects of our approach can be used even if there is not available a suitable initial domain ontology. However, other techniques could also be used in order to generate the conceptual representations (e.g., domain ontologies), such as database reverse engineering [85].

7.2 Users' Requirements

In this section, we list two types of users' requirements: the set of queries users want to be answered by the integrated resource and the set of users' quality requirements. We also list here a users' assertion stating users' additional knowledge of this domain. These requirements are derived from the iSpider project.

7.2.1 User's Queries

Q^1 Retrieve all protein identifications for a given protein accession number

Q^2 Retrieve all protein identifications for a given group of proteins

Q^3 Retrieve all protein identifications for a given organism

Q^4 Retrieve all protein identifications given a certain peptide and their related amino acid information

Q^5 Retrieve all identifications of a given protein given a certain peptide

Q^6 Retrieve all peptide-related information for a given protein identification

Q^7 Retrieve all ion related information

Q^1: [{an, lsid}|{lsid, an} ← ⟨⟨UProteinHit, accession_number⟩⟩; an = 'ENSP00000339074']
Q^2: [{an, lsid}|{lsid, an} ← ⟨⟨UProteinHit, accession_number⟩⟩; member [lsid|{lsid, d} ← ⟨⟨UProteinHit, description⟩⟩; like d '%Actin%'] lsid]
Q^3: [{an, lsid}|{lsid, an} ← ⟨⟨UProteinHit, accession_number⟩⟩; member [lsid|{lsid, o} ← ⟨⟨UProteinHit, organism⟩⟩; like o '%sapiens%'] lsid]
Q^4: [{pr, sc}|{lsid1, pr} ← ⟨⟨UProteinHit, protein⟩⟩; {lsid2, seq} ← ⟨⟨UPeptideHit, sequence⟩⟩; seq = 'ATLTSDK'; {pepID, protID} ← ⟨⟨UPeptideHitToProteinHit_mm⟩⟩; lsid2 = pepID; lsid1 = protID; {lsid2, sc} ← ⟨⟨UPeptideHit, score⟩⟩; {aid, pid} ← ⟨⟨GS1_aa, GS1_pepid⟩⟩; pid = pepID]
Q^5: [{an, lsid1, sc}|{lsid2, seq} ← ⟨⟨UPeptideHit, sequence⟩⟩; seq = 'LVNELTEFAK'; {lsid1, an} ← ⟨⟨UProteinHit, accession_number⟩⟩; an = 'gi—229552'; {pepID, protID} ← ⟨⟨UPeptideHitToProteinHit_mm⟩⟩; lsid2 = pepID; lsid1 = protID; {lsid2, sc} ← ⟨⟨UPeptideHit, score⟩⟩]
Q^6 : [{an, seq, sc, pr, dbs}|{lsid1, an} ← ⟨⟨UProteinHit, accession_number⟩⟩; lsid1 = {'URN:LSID:ispider.man.ac.uk:pedro', 1069}; {pepID, protID} ← ⟨⟨UPeptideHitToProteinHit_mm⟩⟩; lsid1 = protID; {lsid2, seq} ← ⟨⟨UPeptideHit, sequence⟩⟩; lsid2 = pepID; {lsid2, sc} ← ⟨⟨UPeptideHit, score⟩⟩; {lsid2, pr} ← ⟨⟨UPeptideHit, probability⟩⟩; {lsid1, dbs} ← ⟨⟨UProteinHit, dbsearch⟩⟩]
Q^7 : [{ionid, mat, imm}|{ionid} ← ⟨⟨GS1_iontable⟩⟩, {ionid, mat} ← ⟨⟨GS1_iontable, GS1_Matches⟩⟩, {ionid, imm} ← ⟨⟨GS1_iontable, GS1_Immon⟩⟩]

Table 7.1: Users' Queries in IQL

7.2.2 Users' Quality Requirements and Validation

The users' quality requirements can be expressed using various relationships between quality factors 1, 4 and 7 as below:

R1 Schema constructs relating to the same domain concept that satisfy

f_4 should also satisfy f_7 (where *Concept* denotes the set of domain concepts and Sc denotes the set of schema constructs):

$(Sc \sqcap \forall Concept \sqcap \forall contains\_sat^{-}.\{f_4\}) \sqsubseteq$
$(Sc \sqcap \forall Concept \sqcap \forall contains\_sat^{-}.\{f_7\})$[1]

This is decomposed into two segments:
$R1.1 = (Sc \sqcap \forall Concept \sqcap \forall contains\_sat^{-}.\{f_4\})$
$R1.2 = (Sc \sqcap \forall Concept \sqcap \forall contains\_sat^{-}.\{f_7\})$

R2 There should be at least one schema construct relating to the *Peptide* domain concept that satisfies f_4:

$(Sc \sqcap Peptide \sqcap \forall contains\_sat^{-}.\{f_4\}) \not\equiv \emptyset$

We have one segment:
$R2.1 = (Sc \sqcap Peptide \sqcap \forall contains\_sat^{-}.\{f_4\})$

R3 Any schema construct that satisfies f_4 but does not satisfy f_7 must be related to the *Peptide* concept:

$(Sc \sqcap \forall contains\_sat^{-}.\{f_4\} \sqcap \forall contains\_not\_sat^{-}.\{f_7\}) \sqsubseteq (Sc \sqcap Peptide)$

This is decomposed into two segments:
$R3.1 = (Sc \sqcap \forall contains\_sat^{-}\{f_4\} \sqcap \forall contains\_not\_sat^{-}.\{f_7\})$
$R3.2 = (Sc \sqcap Peptide)$

R4 Schema constructs that relate to the *Mass* concept should satisfy f_4:

$Mass \sqcap \forall contains\_not\_sat^{-}.\{f_4\} \equiv \emptyset$

We have one segment:
$R4.1 = (Mass \sqcap \forall contains\_not\_sat^{-}.\{f_4\})$

[1] We note that $\forall Concept$ is not defined in the DL syntax presented in Table 6.1. Its usage within requirement *R1* is a shorthand, and there are actually a *set* of requirements $R1_{c_1}$, ..., $R1_{c_n}$, one for each concept c_i in the domain ontology. For example, in our case study we define $R1_{Peptide}$ as $(Sc \sqcap Peptide \sqcap \forall contains\_sat^{-}.\{f_4\}) \sqsubseteq (Sc \sqcap Peptide \sqcap \forall contains\_sat^{-}.\{f_7\})$ for the *Peptide* concept; and similarly for other domain concepts.

R5 Information relating to the amino acid (AA) concept and information relating to the ion concept should be retrievable via the users' queries:

$(AA \sqcap \forall contains\_sat^{-}.\{f_4\}) \not\equiv \emptyset$
$(Ion \sqcap \forall contains\_sat^{-}.\{f_4\}) \not\equiv \emptyset$

These comprise two segments:
$R5.1 = (AA \sqcap \forall contains\_sat^{-}.\{f_4\})$
$R5.2 = (Ion \sqcap \forall contains\_sat^{-}.\{f_4\})$

7.2.3 Users' Assertion

An additional users' assertion states that the *proteinID* and *id* attributes in the *proteinhit* table of the PepSeeker data source are disjoint.

A1 $(pepseeker\_proteinhit\_proteinID \sqcap pepseeker\_proteinhit\_id) \equiv \emptyset$

7.3 Data Integration Using our Approach

In this section, we undertake our data integration process in three iterations. We assess the quality of the integrated resource with respect to quality factors 1, 4 and 7 (see Sections 5.2 and 5.3 in Chapter 5) after each iteration and derive the amendments to the integrated resource for the subsequent iteration. In the first iteration, we use the PEDRo schema as Global Schema 1 (*GS1*) and establish mappings between the three data sources and *GS1*. In the second iteration, we improve the global schema by incorporating the schema constructs from the gpmDB data source, obtaining Global Schema 2 (*GS2*) and establish mappings for deriving the additional schema constructs of this schema. In the third iteration, we update *GS2* by involving the schema constructs from the PepSeeker data source, obtaining Global Schema 3 (*GS3*) and establish the corresponding mappings.

7.3.1 The First DI Iteration

Definition of Global Schema 1

In the first iteration, we use the PEDRo schema as Global Schema 1 (*GS1*) due to the much richer contents of this schema compared with the gpmDB and PepSeeker schemas. AutoMed mappings are then established between the three data sources and *GS1* for deriving the schema constructs in *GS1*. All *GS1* schema constructs have a derivation from the PEDRo data source. Information from the gpmDB and PepSeeker data sources is used for deriving *GS1* schema constructs that relate to the *GS1* tables: *dbsearch*, *dbsearch-parameters*, *peak*, *peptidehit*, *protein* and *proteinhit*. Table E.1 in Appendix E lists all these mappings. We list there only the mappings associating *GS1* constructs with non-empty extents and we omit all other *GS1* constructs which are populated with *Void* in the mappings.

Quality Measurement for Iteration 1

The quality of this integrated resource is then measured with respect to quality factors 1, 4 and 7. Regarding quality factor 1, there are 674 extensional schema constructs (tables and attributes) available from the three data sources and 473 of them are referenced in the mappings. Therefore, quality factor 1 is calculated as $473/674 = 0.702$ using Formula 5.1. Regarding quality factor 4, the set of users' queries that need to be supported by the integrated resource are listed in Section 7.2.2. The current integrated resource represents 11 domain concepts as listed in Figure E.4 in Appendix E and 7 of them relate to the reformulated queries corresponding to users' queries. These domain concepts are *Mass*, *Peptide*, *Protein*, *Result*, *Peptide Sequence*, *Protein Sequence* and *Score*. Therefore quality factor 4 is calculated as $(0.18 + 0.29 + 0.60 + 0.20 + 0.25 + 0.40)/8 = 0.24$ using Formula 5.6.

Regarding quality factor 7, there are 18 local constraint constructs (primary key and foreign key constraints) and a users' assertion indicating that the *proteinID* and *id* attributes in the *proteinhit* table in Pepseeker are disjoint (see Section 7.2.3). However, we discovered in undertaking this Case Study that the transformation defined by the iSpider project team for deriving the foreign key constraint on *GS1* between the *protein* and *proteinHit* tables references the *proteinID* and *id* attributes that are sourced from the *proteinhit* table in PepSeeker, and is therefore not consistent with the users' assertion. We list here the relevant transformations, with the definition of the foreign key constraint being the last one:

add $\langle\langle$GS1_proteinhit$\rangle\rangle$ /* adding the proteinhit table in GS1 */
$[\{'\text{proteinhit}' + \text{d}\} | \text{d} \leftarrow \langle\langle\text{pepseeker\_proteinhit}\rangle\rangle]$;
add $\langle\langle$GS1_proteinhit, GS1_protein$\rangle\rangle$ /* adding its protein attribute */
$[\{\{'\text{proteinhit}', \text{d}\}, \{'\text{proteinhit}', \text{x}\}\} | \{\text{d}, \text{x}\} \leftarrow \langle\langle\text{pepseeker\_proteinhit, pepseeker\_ProteinID}\rangle\rangle]$;
add $\langle\langle$GS1_protein$\rangle\rangle$ /* adding the protein table */
$[\{'\text{proteinhit}', \text{d}\} | \text{d} \leftarrow \langle\langle\text{pepseeker\_proteinhit}\rangle\rangle]$;
add $\langle\langle$GS1_protein, GS1_id$\rangle\rangle$ /* adding its id attribute */
$[\{\{'\text{proteinhit}', \text{d}\}, \{'\text{proteinhit}', \text{d}\}\} | \text{d} \leftarrow \langle\langle\text{pepseeker\_proteinhit}\rangle\rangle]$;
add /* adding a foreign key constraint */
$\langle\langle$GS1_fk_proteinhit_protein1, GS1_proteinhit, $\langle\langle$GS1_proteinhit, GS1_protein$\rangle\rangle$,
GS1_protein, $\langle\langle$GS1_protein, GS1_id$\rangle\rangle\rangle\rangle$;

Therefore, quality factor 7 is calculated as $18/19 = 0.95$ using Formula 5.12.

Assuming these three factors are assigned equal weight, the overall quality of the integrated resource after iteration 1 is calculated as $\frac{0.702+0.24+0.95}{3} = 0.63$. The set of items that do and do not satisfy each quality factor is summarised in Tables E.4,E.5,E.6, E.7 and E.8 in Appendix E. We next validate the users' quality requirements with respect to the measurement results, as described below.

Reasoning in QFDI for Iteration 1

We extended the OWL-DL implementation of our QFDI with users' quality requirements and DI items that do and do not satisfy each quality factor (see Figure 7.4). Two kinds of reasonings are applied, TBox reasoning, where only the classes and axioms are involved in the process, and ABox reasoning, where instances are also considered in the inference process. As the result of TBox reasoning, the reasoner detected that the quality requirements of Section 7.2.2 are not consistent with each other since the reasoner cannot generate a model that can satisfy all of them. The error messages from the reasoner are illustrated in Figure 7.5.

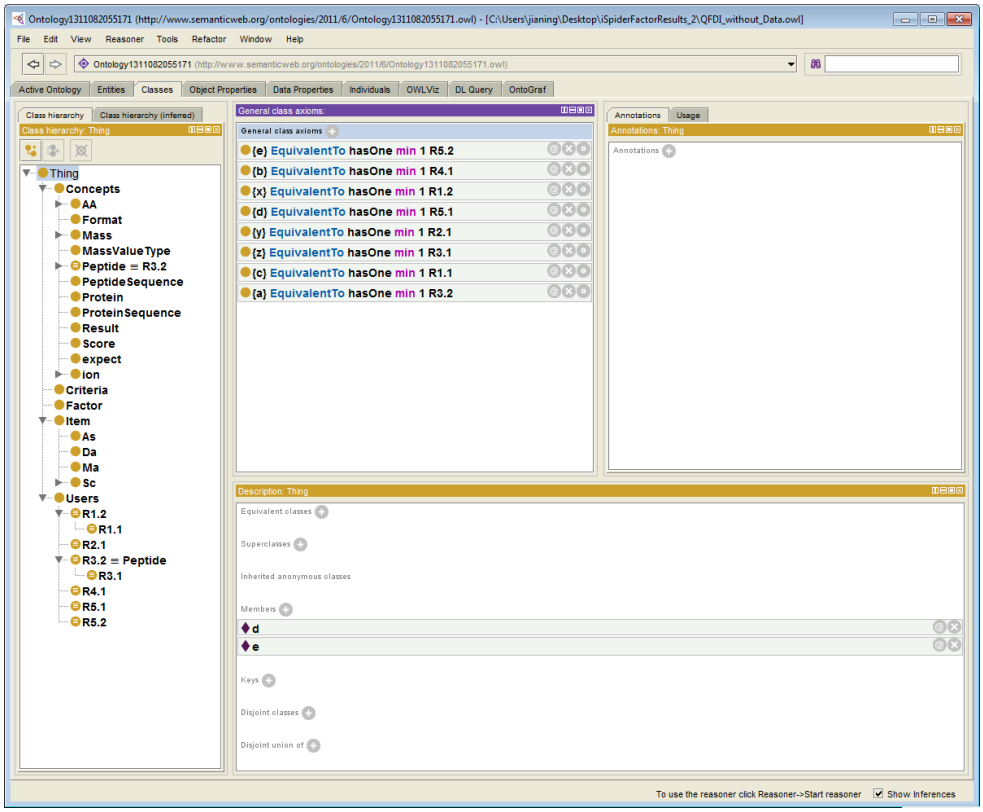
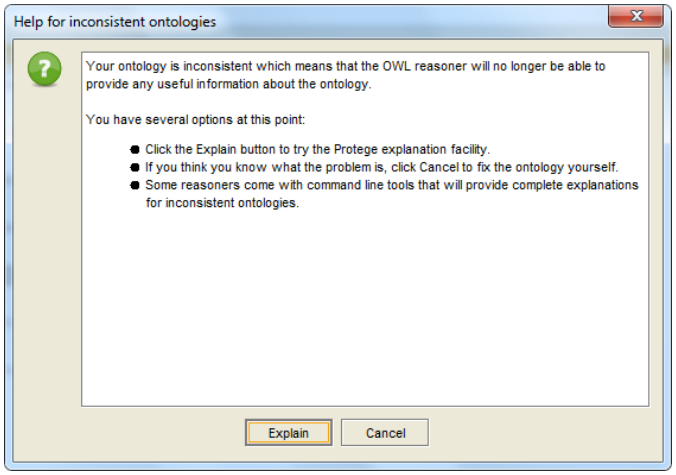


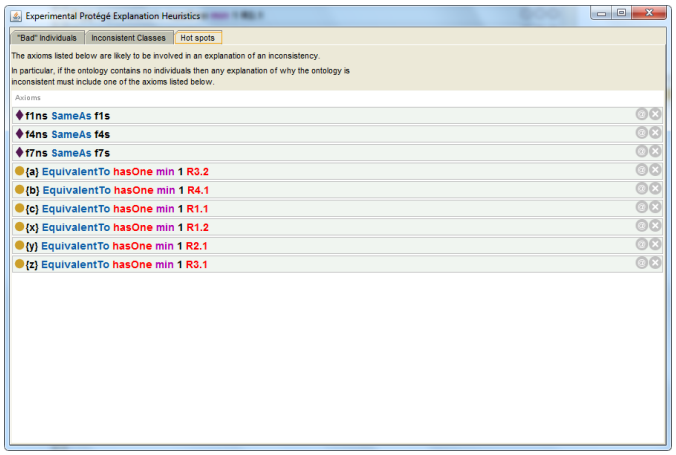
Figure 7.4: QFDI in OWL DL - This figure shows the domain concepts, QFDI concepts including the set of users' quality requirements, and a set of general axioms that are used for TBox reasoning

In order to resolve these inconsistency issues, we apply TBox reasoning again, after relaxing the general axioms stating that there must exist an individual satisfying each user quality requirement. Generally, we remove in turn each axiom as suggested in the reasoning output in Figure 7.5 and observe the new reasoning result. In particular, if we remove the general axiom as relating to requirement *R3.1* we discover, using the reasoner, that *R3.1* must be associated with no individual (identified by the *Nothing* concept being inferred for *R3.1* — see Figure 7.6). Alternatively, if we remove the subclass relationship between *R1.1* and *R1.2*, there will not be any in-

consistency exceptions generated by the reasoner. Therefore, we conclude that *R1* and *R3* cannot be satisfied at the same time. In our case study, we chose to remove *R1* from the set of users' quality requirements to resolve this inconsistency, resulting in an updated set of users' quality requirements comprising *R2*, *R3*, *R4* and *R5*.

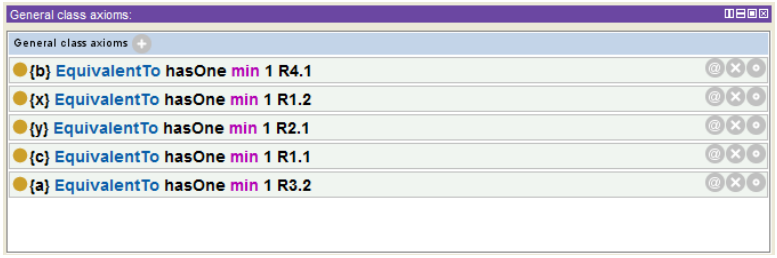


(a)

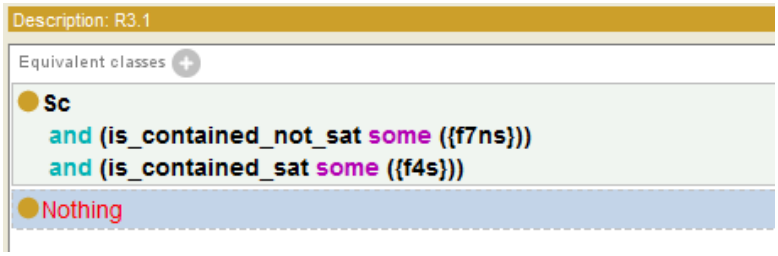


(b)

Figure 7.5: Output of TBox Reasoning after Iteration 1. Figure (a) shows inconsistencies have been discovered and Figure (b) shows the set of axioms that may cause these inconsistencies



(a)



(b)

Figure 7.6: Solution 1 to Resolve Inconsistency - Remove Axiom for *R3*. Figure (a) shows that the axiom stating *R3.1* has an individual is removed and Figure (b) shows that an axiom stating *R3.1* is associated with *Nothing* is generated by the reasoner

For applying ABox reasoning, we specified assertions indicating the items that do and do not satisfy each quality factor in our framework by generating the OWL-DL individuals and corresponding axioms for each quality factor. Such items are automatically collected in the assessment processes for quality factors discussed in Chapter 5 and stored in the file system. We then developed a program to generate OWL-DL assertions with respect to these items.

After we applied the ABox reasoning process, each quality requirement has associated with it a set of items arising from the inference (see Figures 7.7, 7.8, 7.9 and 7.10 for quality requirements *R2*, *R3*, *R4* and *R5* respectively). The original *R2* indicated that there should be at least one item to satisfy *R2* and it has been confirmed by the reasoning results (Figure 7.7).

For *R3*, we then apply the subclass relationship between *R3.1* and *R3.2* (see Figure 7.8) and we discover that this relationship cannot be satisfied with respect to the current integrated resource. For *R4*, there should not be any item inferred as satisfying it, but the current integrated resource cannot satisfy this requirement (see Figure 7.9). For *R5*, the original *R5* indicated that there should be at least one item satisfying it, but the reasoner inferred that *R5* is associated with no item. Therefore, the current integrated resource needs to be modified in order to satisfy *R3*, *R4* and *R5*.

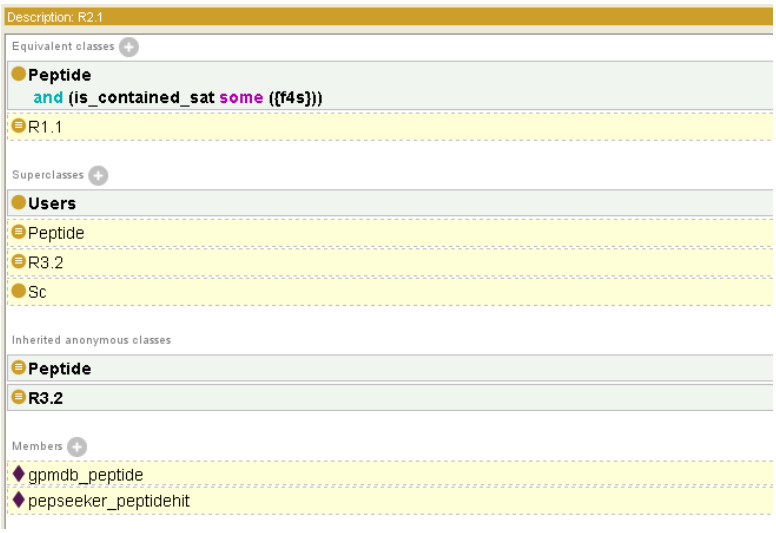
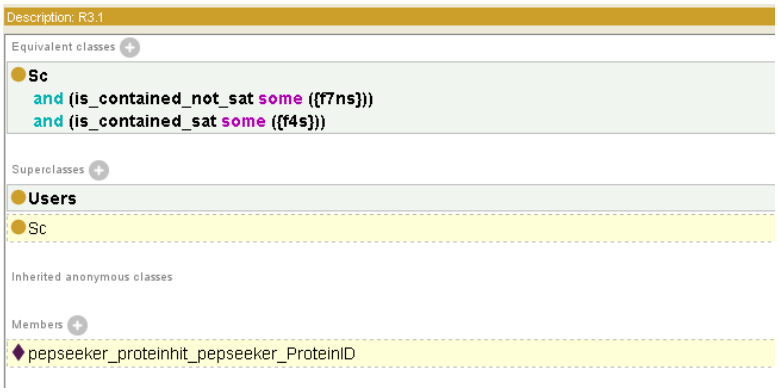
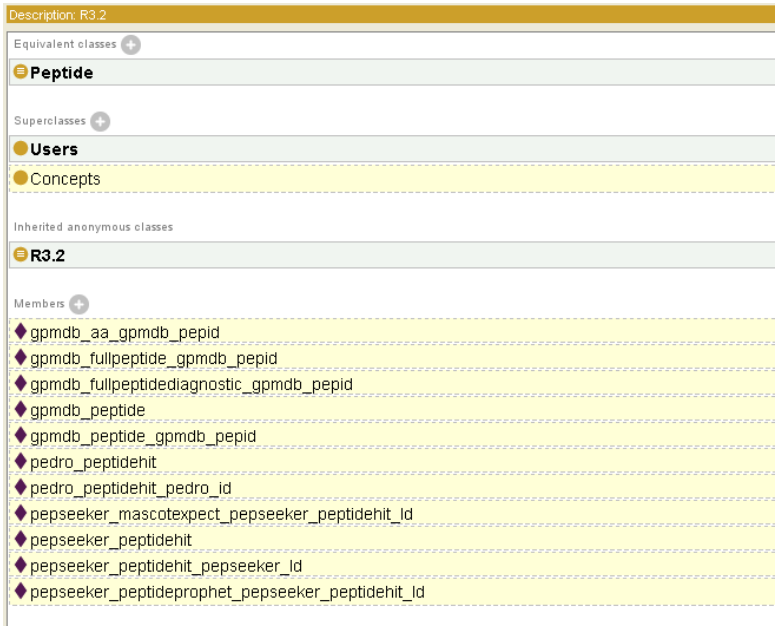


Figure 7.7: ABox Reasoning for *R2* after Iteration 1, showing *R2.1* has two individuals *gpmdb_peptde* and *pepseeker_peptidehit* associated with it inferred by the reasoner



(a)



(b)

Figure 7.8: ABox Reasoning for *R3* after Iteration 1, showing *R3.1* has one individual *pepseeker_proteinhit_pepseeker_ProteinID* associated with it inferred by the reasoner and *R3.2* now has 11 individuals

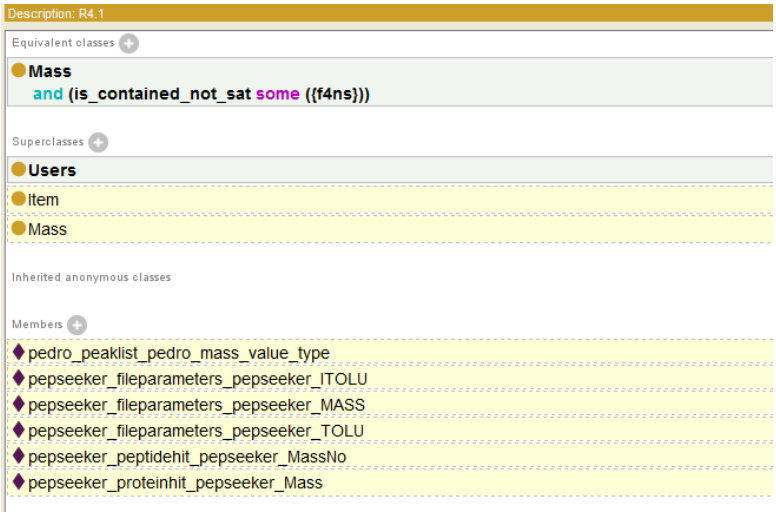


Figure 7.9: ABox Reasoning for *R4* after Iteration 1, showing *R4.1* has 6 individuals associated with it inferred by the reasoner

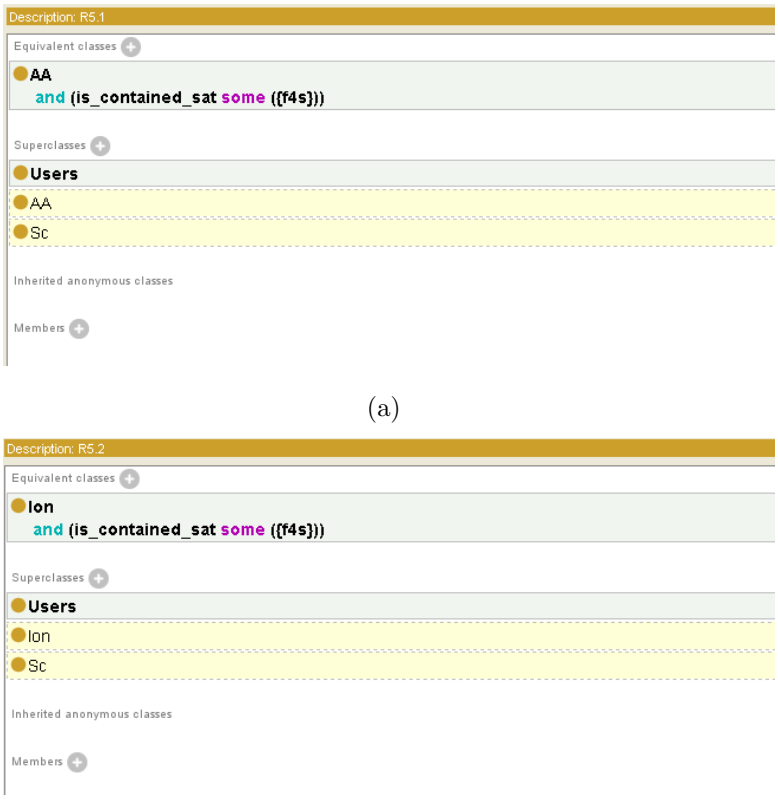


Figure 7.10: ABox Reasoning for *R5* after Iteration 1, showing both *R5.1* and *R5.2* have no individuals associated with them inferred by the reasoner

7.3.2 The Second DI Iteration

Definition of Global Schema 2

As discussed in the previous section, we have identified that quality requirements 3, 4 and 5 cannot be satisfied. We then analysed the reasoning results and the previous integrated resource, and provided the following amendments in the second iteration.

In the second iteration, we improved the global schema by adding to it the schema constructs from the gpmDB data source, resulting in global schema

GS2. This enriches the query answerability relating to quality requirement 5. Regarding quality requirement 3, we took the solution of removing the items that do satisfy quality factor 4 and do not satisfy quality factor 7 by modifying the PEDRo schema. In the updated PEDRo schema, we modified the foreign key between the *protein* and *proteinHit* tables by linking the *id* attributes in each table instead of linking the *id* attribute in the *protein* table and the *ProteinID* attribute in the *proteinHit* table. We also modified the transformations for deriving this foreign key. We list here the relevant transformations, with the updated foreign key definition being the last of these:

```
add  ⟨⟨GS1_proteinhit⟩⟩    /* adding the proteinhit table in GS1 */
   [{'proteinhit' + d}|d ← ⟨⟨pepseeker_proteinhit⟩⟩];
add  ⟨⟨GS1_proteinhit, GS1_id⟩⟩    /* adding its id attribute */
   [{{'proteinhit', d}, {'proteinhit', d}}|d ← ⟨⟨pepseeker_proteinhit⟩⟩];
add  ⟨⟨GS1_protein⟩⟩    /* adding the protein table */
   [{'proteinhit', d}|d ← ⟨⟨pepseeker_proteinhit⟩⟩];
add  ⟨⟨GS1_protein, GS1_id⟩⟩    /* adding its id attribute */
   [{{'proteinhit', d}, {'proteinhit', d}}|d ← ⟨⟨pepseeker_proteinhit⟩⟩];
add  /* adding a foreign key constraint */
   ⟨⟨GS1_fk_proteinhit_protein1, GS1_proteinhit, ⟨⟨GS1_proteinhit, GS1_id⟩⟩,
      GS1_protein, ⟨⟨GS1_protein, GS1_id⟩⟩⟩⟩;
```

In addition, we also modify the transformations that are used for deriving the *mass_value_type* and *mass_error_type* constructs on *GS1* by referencing also the *MASS* and *TOLU* schema constructs from the PepSeeker data source, which were missing from the old iSpider mappings. This also improved the concept coverage in quality factor 4. Table E.2 in Appendix E lists the set of mappings that were amended in the second iteration.

The quality of this integrated resource is again measured with respect to quality factors 1, 4 and 7. Regarding quality factor 1, there are 674 extensional schema constructs (tables and attributes) available from the three

data sources and 569 of them are now referenced in the mappings. Therefore, quality factor 1 is calculated as $569/674 = 0.844$ using Formula 5.1 (higher than the value of 0.702 in iteration 1).

Regarding the same set of users' queries that need to be supported by the integrated resource in quality factor 4, the modified integrated resource represents 11 domain concepts as listed in Table E.5 in Appendix E and 8 of them relate to the reformulated queries of users' queries. These domain concepts are *Mass*, *Peptide*, *Protein*, *Result*, *Peptide Sequence*, *Protein Sequence*, *Score* and AA, and quality factor 4 is calculated as $(0.27 + 0.36 + 0.25 + 0.60 + 0.20 + 0.40 + 0.50)/8 = 0.32$ using Formula 5.6.

Regarding quality factor 7, there are 18 local constraint constructs and 1 users' assertion. After the modification of the foreign key between the *protein* and *proteinHit* tables and the associated transformations resolves the inconsistency with the users' assertion $A1$, all constraints are now valid with respect to the current set of transformations. Therefore, quality factor 7 is calculated as $19/19 = 1.00$ using Formula 5.12.

Assuming these three factors are assigned equal weight, the overall quality of the integrated resource after iteration 2 is calculated as $\frac{0.844+0.32+1.00}{3} = 0.72$. We note that this is higher than the value of 0.63 in iteration 1. The set of items that do and do not satisfy each quality factor is summarised in Tables E.9, E.10, E.11 and E.12 in Appendix E. We next validated users' quality requirements with respect to the measurement results as follows.

Reasoning in QFDI for Iteration 2

We extended the OWL-DL implementation of our QFDI with the updated DI items that do and do not satisfy each quality factor in this iteration. TBox and ABox reasonings are then applied. Regarding the TBox reasoning, since there are no changes to the users' quality requirements from the first iteration, the reasoning process results in no inconsistency exceptions.

Regarding the ABox reasoning, the reasoner is applied to the updated individuals for each quality factor. The outputs for quality requirement *R2*, *R3*, *R4* and *R5* change as a result (see Figures 7.11, 7.12, 7.13 and 7.14 respectively). *R2* and *R3* are now satisfied. *R4* has been improved since the number of individuals that do not satisfy this quality requirement has reduced to 4 from the previous value of 6. *R5* has also been improved because an individual *gpmdb_aa_gpmdb_aaid* is now associated with *R5.1*. We discovered that quality requirement *R5.2* is still not satisfied since the updated integrated resource still lacks the *Ion* information and we designed the third iteration in order to satisfy quality requirements *R4* and *R5.2*.

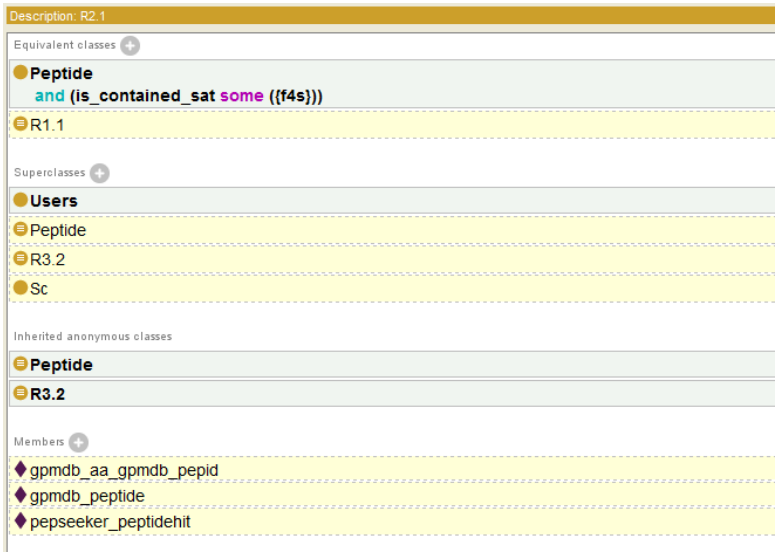


Figure 7.11: ABox Reasoning for *R2* after Iteration 2, showing *R2.1* has three individuals *gpmdb_aa_gpmdb_pepid*, *gpmdb_peptde* and *pepseeker_peptidehit* associated with it inferred by the reasoner

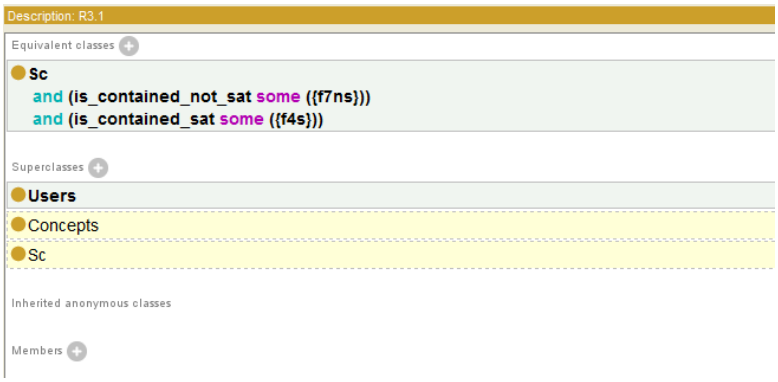


Figure 7.12: ABox Reasoning for *R3* after Iteration 2, showing there is no individual associated with *R3.1* inferred by the reasoner

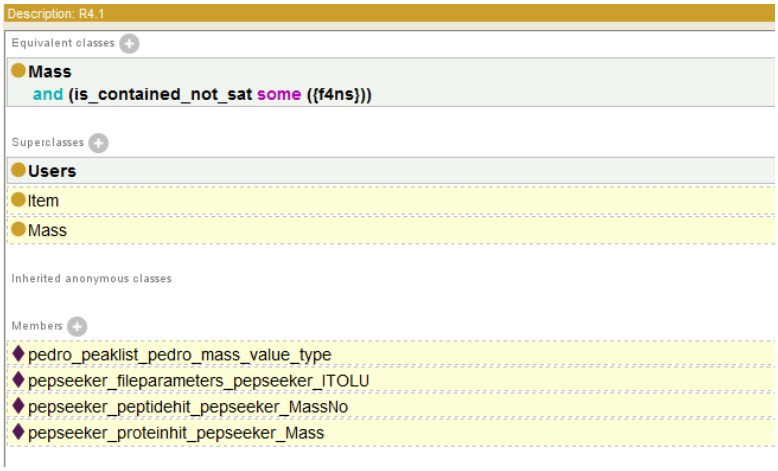


Figure 7.13: ABox Reasoning for *R4* after Iteration 2, showing *R4.1* has four individuals associated with it inferred by the reasoner

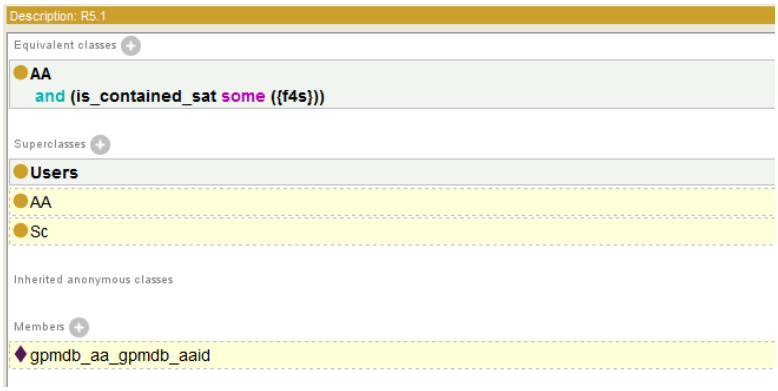


Figure 7.14: ABox Reasoning for *R5* after Iteration 2, showing *R5.1* has individual *gpmdb_aa_gpmdb_aaid* associated with it inferred by the reasoner

7.3.3 The Third DI Iteration

Definition of Global Schema 3

In the second iteration, we have improved the overall quality from 0.654 to 0.741, but the integrated resource still lacks support for quality requirements *R4* and *R5.2*. Therefore, we applied a third integration iteration involving information available only from the PepSeeker data source in the integration process. We updated the global schema by adding the schema constructs from the PepSeeker data source, giving new global schema *GS3*. This enriches the query answerability relating to quality requirement *R5*. In addition, we also modified the mappings that were used for deriving the *charge* constructs on *GS2* by referencing also the *CHARGE* schema construct from the PepSeeker data source, which were missing from the old iSpider mappings. This also improved the concept coverage in quality factor 4. Table E.3 in Appendix E lists the mappings that were amended in the third iteration.

The quality of this integrated resource is again measured with respect to quality factors 1, 4 and 7. Regarding quality factor 1, there are 674 ex-

tensional schema constructs (tables and attributes) available from the three data sources and all of them are now referenced in the mappings. Therefore, quality factor 1 is calculated as $674/674 = 1$ using Formula 5.1.

Regarding the users' queries that need to be supported by the integrated resource in quality factor 4, the modified integrated resource represents 11 domain concepts as listed in Table E.6 in Appendix E and 9 of them relate to the reformulated queries of the users' queries. These domain concepts are *Mass*, *Peptide*, *Protein*, *Result*, *Peptide Sequence*, *Protein Sequence*, *Score*, *AA* and *Ion*, and quality factor 4 is calculated as $(0.27 + 0.36 + 0.25 + 0.33 + 0.60 + 0.20 + 0.40 + 0.50)/8 = 0.36$ using Formula 5.6. Regarding quality factor 7, there are 18 local constraint constructs and 1 users' assertion. All such constraints are valid with respect to the current set of mappings. Therefore, quality factor 7 is calculated as $19/19 = 1.00$ using Formula 5.12. Assuming these three factors are assigned qual weight, the overall quality of the integrated resource after iteration 3 is calculated as $\frac{1.00+0.36+1.00}{3} = 0.79$. The set of items that do and do not satisfy each quality factor is summarised in Tables E.13, E.14 and E.15 in Appendix E. We next validated users' quality requirements with respect to the measurement results as follows.

Reasoning in QFDI for Iteration 3

We extended the OWL-DL implementation of our QFDI with the updated DI items that do and do not satisfy each quality factor in this iteration TBox and ABox reasonings are then applied. Regarding the TBox reasoning, since there are no changes to the users' quality requirements from the first iteration, the reasoning process results in no inconsistency exceptions. Regarding the ABox reasoning, the reasoner is applied to the updated individuals for each quality factor. Quality requirement *R5.2* is now associated with an individual *pepseeker_iontable* inferred by the reasoner (see Figure 7.15). However, quality requirement *R4* is still not fully satisfied by the

current integrated resource, but there are fewer individuals are associated with this requirement than in the first iteration.

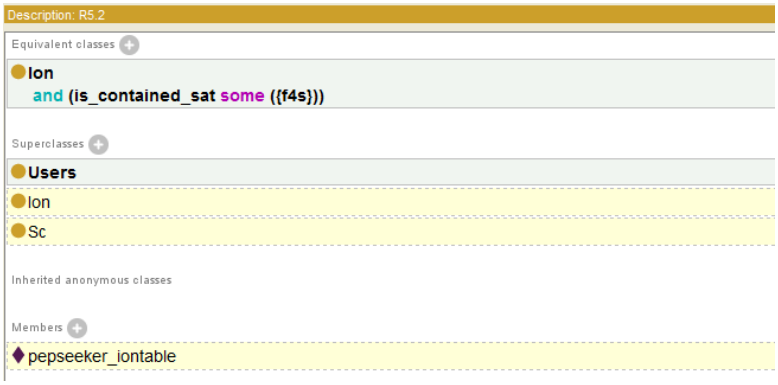


Figure 7.15: ABox Reasoning for *R5* after Iteration 3, showing *5.2* is associated with an individual *pepseeker_iontable* inferred by the reasoner

7.3.4 Quality Improvement over Three Iterations

We have demonstrated above how our QFDI can be used for increasing the quality of the integrated iSPIDER resource with respect to the users' quality requirements. Figure 7.16 illustrates the quality increase of each quality factor in the three integration iterations. We can see that each quality factor and the overall quality have been improved incrementally. All users' quality requirements have been satisfied by the final integrated resource except *R4*. However, there are also improvements with respect to *R4* compared with the initial global schema *GS1*, because the number of items that do not satisfy *R4* has been reduced from 6 to 4.

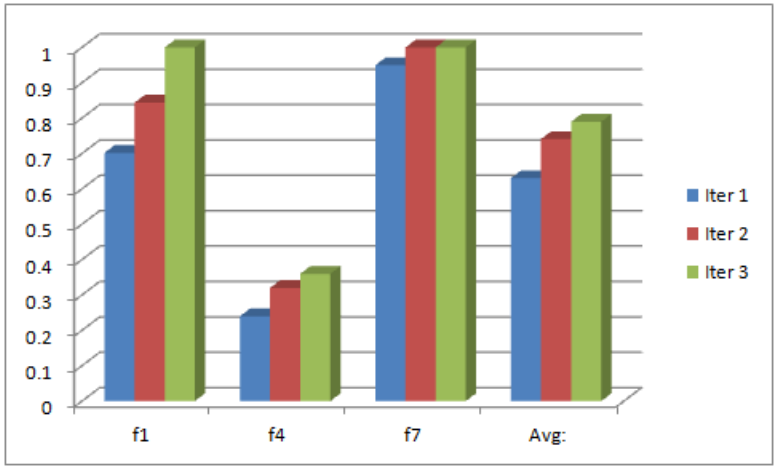


Figure 7.16: Increase of Quality Factor in 3 Iterations

7.4 Conclusion

We have presented in this chapter an evaluation of our approach to assessing and improving the quality of integrated resources, using the iSpider life sciences data integration project. Our evaluation comprised of three steps, the creation of the initial integrated resource, the measurement and quality assessment step, and the quality improvement step. This was repeated over three iterations. Through this simple evaluation, we have demonstrated that the quality of the iSpider integrated resource can be improved by using our quality assessment and improvement approach.

Future work will focus on a larger evaluation of our quality assessment approach based on the iSpider project. In our current evaluation process, we began by adopting the PEDRo schema as the original integrated resource, as was done by the original iSpider integrators. In our future work, we will create a number of alternative initial integrated resources and compare their quality. Since there is no existing domain ontology in the life-science domain that is suitable to represent the information stored in the data sources,

in our evaluation process we did not consider the ontology-related quality factors. Therefore, in our future work, we will construct an ontology that uses terminology similar to that of the iSpider data sources in order to be able to link the data sources to the domain ontology and to evaluate our ontology-related quality factors.

Chapter 8

Conclusions and Future Work

In this thesis, we have presented a quality assessment approach in the context of data integration which comprises a quality framework, a set of quality criteria and factors, and a novel data integration methodology incorporating quality assessment functionality. In this chapter, we first provide an overview of the thesis and then discuss its contributions and the areas of future work.

In Chapter 2, we reviewed the major quality related issues in data integration and information systems, with particular focus on the difficulties and characteristics of data quality definition and measurement methods in the DI context. We also reviewed related work on quality frameworks proposed in the areas of data integration and e-science. The AutoMed heterogeneous data integration system, which was used as the basis of implementing our approach, is also introduced in this chapter.

In Chapter 3, we proposed our research approach derived by analysing the research literature, supplemented by the practical experiences of a group of data integrators identified through an interview with them conducted in 2008. We identified the users' requirements that we consider are crucial in our research. In this chapter, we also proposed our DI methodology incorporating quality assessment functionality. We then introduced a case

study that was used in our discussions in later chapters.

In Chapter 4, we introduced our Quality Framework for Data Integration (QFDI) and proposed a quality hierarchy comprised of quality criteria, quality factors and metrics for measuring these. We also discussed the reasoning capabilities required to generate a consistent and integrated quality view of the integrated resource with respect to different users' quality requirements as a whole and also individually.

In Chapter 5, we defined five quality criteria and their sub-criteria in the DI context: completeness, consistency, accuracy, minimality and performance. A set of quality factors relating to the completeness and consistency quality criteria were proposed and discussed in detail, together with quality metrics for measuring these factors using knowledge extracted from the DI elements.

In Chapter 6, we introduced a DI architecture that realises our DI methodology proposed in Chapter 3. We also illustrated a typical data integration workflow using this architecture. We discussed in detail the implementation of key components of our DI architecture, including the OWL representation of our QFDI, the transformation algorithm from a relational schema to an ontology representation and the implementation of the quality factors proposed in Chapter 5.

In Chapter 7, we evaluated our quality assessment approach, demonstrating its usefulness with respect to a real-world life sciences data integration project, iSpider. We demonstrated quality measurement of the iSpider integrated resource and quality improvement adopting our approach.

Our research presented in this thesis makes several contributions:

- We have proposed a quality framework that is able to represent different users' quality perspectives. We have identified a set of elements in the integrated resource that can be referenced by quality measurement methods and that are significant for the iterative quality improvement

of the integrated resource.

- We have defined five quality criteria and a set of quality factors for interpreting two of these criteria that are specific to the data integration context.

- We have proposed a set of quality measurement methods for calculating the level of satisfaction of the integrated resource with respect to the quality factors and the users' requirements.

- We have proposed a novel data integration methodology. This integration methodology embeds quality assessment functionalities within the integration process and allows the quality assessment and iterative improvement of the integrated resource. We have demonstrated the use of our quality assessment approach in a real-world life sciences data integration project and have illustrated how our approach can be used to improve the quality of the integrated resource.

- We have proposed an ontological representation of our quality framework and have identified the ontological expressiveness and reasoning abilities necessary for this framework in order to generate a consistent and integrated quality view. This enables off-the-shelf ontology reasoners to be adopted in detecting inconsistencies within and between the users' quality requirements and the integrated resource.

This thesis has also presented an implementation of our quality assessment approach that uses the AutoMed data integration system in order to provide the mappings and query processing capabilities. However, our approach could be implemented using any other data integration system, so long as this supports GAV and LAV mappings and sufficiently expressive query reformulation capability. In our implementation, we also used

COMA++ and FaCT++ for the matching and inference aspects, respectively. Other schema matching tools and ontology reasoning tools can also be adopted in our approach if they provide sufficient schema matching and reasoning capabilities as described in Chapter 4. We also illustrated in Chapter 7 that the quality measurement and improvement aspects of our approach can be used even if a suitable initial domain ontology is not available.

The results of our thesis work have been published as follows. A first version of the transformation algorithm between the relational schema and its ontology representation was described in [38]. An early version of our DI methodology and its realisation was proposed in [86] and our quality framework was first introduced in [87]. The application of our approach introduced in this thesis was described in [88].

Throughout our research, we identified and addressed several challenges. First, data integration is a complex process and there are various stages within it where quality can be assessed. We resolved this problem by introducing our DI methodology that enables the different quality aspects of the integrated resource to be assessed iteratively. Second, it is difficult to identify the many elements that have impact on the quality aspect of data integration. We resolved this by focussing on the crucial DI elements: schema constructs, mappings, assertions and data, in developing our quality factors. Third, the quality of the integrated resources can be interpreted from various perspectives. We resolved this problem by proposing our quality framework with a range of quality criteria and associated quality factors, each of which can be extended. We also proposed quality measurement methods for measuring these quality factors. Such measurement methods can also be extended in order to provide more precise quality measurements and provide more precise inputs to our quality framework. Fourth, the richness of the quality framework made its formalisation a challenge and we also required that a certain level of reasoning can be applied in order to

identify the DI elements that may cause inconsistencies between the users' quality requirements and the integrated resource. We resolved this problem by adopting the description logic and OWL-DL language for formalisation of our framework. These provide sufficient expressiveness and inferencing power for representing our quality framework in this thesis. However, the reasoning mechanisms of such techniques are based on weak assumptions, such as the Open-World assumption, and reasoning with more restricted assumptions would be more appropriate. In addition, the possible sources of inconsistency resulting from the current reasoning approach can be extensive and it is difficult to identify the precise elements that give rise to inconsistency.

The strengths of our approach are the rich expressiveness of our quality framework, which is also extendable to allow more quality criteria and factors to be defined. Our DI methodology allows different quality aspects of the integrated resource to be assessed iteratively. In addition, because of the iterative nature of our DI methodology, data source evolution could be handled within it. The current limitations of our approach are the complexity that arises from the richness of our quality framework. The feedback from the reasoners can also be extensive. Therefore, future work is needed in these areas also.

There are several other directions of future work building on the results of this thesis:

- Develop more quality factors for the completeness and consistency quality criteria proposed in Chapter 5.

  So far, we have proposed one quality factor for each quality criterion and sub-criterion. In the future, more quality factors can be developed in order to interpret quality criteria from different perspectives. Another task is to develop quality metrics that can measure more accurately the quality of an integrated resource and return the DI

elements that are more critical to its quality improvement. We will also consider the characteristics of semi-structured data sources when considering assessment and improvement of an integrated resource.

- Extend our work for the accuracy, minimality and performance quality criteria.

  In addition to the completeness and consistency quality criteria discussed in this thesis, in the future we will also investigate the accuracy, minimality and performance criteria and their quality factors interpreting them with respect to our quality framework. These quality criteria focus on the quality of the integrated resource from different aspects and they could provide users with more options in expressing their quality requirements.

- Provide graphical user interfaces for users to express their quality requirements.

  Throughout our work, we found that users may find it difficult to express their quality requirements in the formal language we adopted in our research if users are not familiar with this language and our approach. Therefore, in the future, we will take this into account by researching and developing appropriate graphical interfaces in order to guide users in expressing their quality requirements, and automatically transforming these into their formal representations.

- Extend our work in order to detect more precisely the DI elements that cause inconsistency.

  In this thesis, we have proposed a reasoning approach that can detect inconsistencies between the users' quality requirements and the integrated resource. However, current off-the-shelf reasoners return a rather low-level set of suggestions on what the inconsistencies are and

it is difficult to discover the useful information from the reasoning feedback. Therefore, we need to develop reasoning tools that can provide more meaningful feedback with respect to the quality improvement task in the DI context. In addition, the reasoning should take into account the relationships between DI elements, which is not supported by our current reasoning implementation.

- Extend our quality framework by adopting more expressive logic languages.

  In this thesis, we have proposed our quality framework based on the logical foundation of description logics and implemented it using the OWL language. Such languages are sufficient to formalise our quality framework and off-the-shelf reasoners are available in order to support inferencing within the framework. However, our quality framework can be extended to support more complex users' quality requirements as discussed previously. Therefore, in order to formalise such additional requirements and extend the inferencing capability accordingly, other more expressive logic languages could be considered, such as fuzzy or temporal logics.

- Extend the framework to capture the quality of the data sources and to extract semantic information related to the data sources.

  An important issue which impacts on the quality of data integration is the intrinsic quality of the input data itself. This has been beyond the scope of this thesis, but extending our framework to incorporate this aspect would be a significant enhancement. Our framework could also be extended with a database reverse engineering phase for extracting conceptual schemas from the data sources as a first step towards the construction of an ontology in cases where a domain ontology is missing.

- We will continue the experimentation with other integration projects with a wider range of data sources.

  In this thesis, we have adopted the iSpider project for demonstrating the usefulness of our quality assessment approach. Other non-academic data integration projects that may be more complex than iSpider should also be investigated in the future work. This, while beyond the scope of this thesis work, would be valuable since it would demonstrate how our quality assessment approach can handle more complex data integration scenarios.

In conclusion, in this thesis we have presented our quality assessment approach in the DI context. Our work provides solutions for several issues that are not addressed or not researched in detail by state-of-the-art approaches found in the literature. Data integration is a complex and error-prone process. There have been many research works in this domain and most of them focus on the creation of the integrated resource, including schema matching, mapping generation, and processing of users' requirements and feedback. However, since data integration is a collaborative process involving a variety of users, it is important that users are involved throughout the DI process. Our work has demonstrated that data integration quality can be formally modelled, assessed and improved with respect to the users' quality requirements.

Assessing and improving the quality of an integrated resource is crucial as it relates directly to the users' expectations with respect to a data integration application. This thesis has proposed a new approach in data integration research that could assist data integrators to create integrated resources that are closer to users' requirements. This is a topic which can be expected to be of growing practical importance as the degree to which information systems meet users' requirements increases in significance.

Appendix A

Report on the Interview With Data Integrators

This appendix gives details of the interview held with data integrators in order to identify users' requirements and gather first-hand experience of the DI process. This interview took place at London Knowledge Lab in 2008 and the integrators who contributed in the interview were Alexandra Poulovassilis (AP), Nigel Martin (NM) and Lucas Zamboulis (LZ). There were three main objectives of this interview: 1) We aimed to identify the procedures that data integrators undertake during the DI processes in practice and understand the difficulties that integrators face from their experience in order to identify the key factors to be considered in our research. 2) We aimed to identify the users' requirements relating to the integration process and how data integrators aim to meet such requirements. 3) We aimed to study possible approaches to addressing the DI difficulties that integrators may adopt in their experience. We aimed to identify from such solutions the type of information our methods should provide to data integrators in order to help in improvement of an integrated resource. The questions posed, and the answers of the data integrators, were as follows:

What features do you expect an automatic data integration tool to provide if there exists one? What are the inputs and outputs?

Answer - Manual data integration process is complex, time-consuming, error-prone and boring. Computer scientists are searching for the methods that could assist integrators in the data integration process either automatically or semi-automatically. In fact, data integration involves many different information domains and it is not feasible for a computer program to have such comprehensive information. Therefore, it is not feasible to achieve full integration process automatically at this time. Therefore, LZ noted that, more likely, integration tools are semi-automatic and provide some or all of the features categorised as follows: *Schema matching tool* aims to identify correspondences across data sources, and between data sources and the *GS*; *Mapping creation tool* aims to create mappings between data sources and the *GS* specified in the mapping language supported by the tool. This is also considered as part of a schema matching tool in some research works; *Debugging tool* aims to detect errors in the mappings that are generated automatically or manually; *GS creation tool* aims to create the *GS* according to the users' requirements; *Schema annotation tool* aims to automatically annotate the *GS* using possible annotations from data sources.

Apart from the above features, functions are also required to examine the data sources. NM discussed this problem from data warehouse designers' perspective. He also indicated that a data transformation tool should be considered as one of the data integration tools, which is to convert data from one database into another one. The integrator has to decide what datatype needs to be used, whether to convert to the target datatype before the integration or use native datatypes directly. That also raises another problem when local data sources are not formatted correctly in terms of the *GS*. As an instance, LZ gave an example in the ASSIST project, where no primary keys were given for relational data sources. This needs lots of

human efforts to prune the data sources.

AP also added that it could be an interactive tool beyond automatic tools, such as the Clio project, to visualise the mappings and the GS constructing processes. It would be very helpful especially when there exist a number of constraints on data sources, such as accessing permission constraints, that need to be handled. An integrator may not know them comprehensively. This tool will express these constraints on the actual schema merging process. Also, there could also be tools to evaluate alternative mapping solutions.

How much information do you know about data sources before the integration process?

Answer - Because data integration projects may involve various application domains, it is highly likely that the integrators do not have much knowledge about such domains. Therefore, the integration resources may be correct with respect to the application domain, but may be wrong or inaccurate from the users' perspectives. Apart from the domain information, integrators also need to be aware of information contained in the data sources. NM gave two examples: the data sources may not be well structured, such as no primary key defined in the relational data sources, or the semantics of schemas may be separated from the data itself, especially when middleware are used. Such information may or may not be documented and delivered with the data sources, such as schema diagrams. However, AP said that these documents may not be enough or may be wrong, out of date, incomplete or too abstract. Therefore, the data integrators may need to interview the data providers, the users, and also examine the data sources if necessary. Data profiling techniques can be used in this process. Useful information can be discovered in the profiling process, such as NULL values, duplications at data or schema levels. NM also said that, sometimes, data examination may

also not be enough, if data is not easy to be understood, or if the related data are widely distributed over many schemas. Therefore, interviews with data source providers may be necessary.

Is there any regulation you can follow for data integration?

Answer - In data integration, regulations can be expressed in two aspects, the data models used and the DI process standards the integrators have to follow, said by AP. In the former aspect, the integrators have to consider the data models adopted by the data sources, the mapping languages used by the integration tool, the query language supported. In the latter aspect, NM indicated that there exist some standards in building data warehouses that the warehouse programmers need to follow, such as architectural standards and quality standards. However, these standards are not highly adopted.

Which integration methodologies do you use, GAV, LAV, GLAV or BAV? What are the difficulties in each approach?

Answer - The choice of different integration methodology depends on the integration scenarios, said by AP. For example, GAV is more suitable in the scenario when the details of the data sources are known, but the GS is not. LAV is suitable when there exists one or many well-adopted or well-known global schemas, such as the domain ontology. LZ gave an example of the ASSIST project where integrations are required to integrate data sources in order to comply with an medical domain ontology as the global schema. AP also said that different information is contained in the mapping when different methodologies are use.

How do you create the global schema?

Answer - The global schema may be given by the users or the integrators have to create one with respect to the data sources. In the latter case,

LZ said that one solution is to use one schema from the data sources as the GS as a starting point. During the integration process, the integrators may find that this temporary GS is not capable of representing information available from all data sources. In this case, the integrators need to modify this GS to meet the requirements. Another way of creating the GS is a two step process. Firstly, the data integrators need to construct a GS that can represent all constructs from all data sources. Secondly, the integrator needs to refine this schema with respect to the users' requirements.

Regarding the performance issue in data integration, LZ said that the primary goal of data integration is to have a working version that meets all users' requirements. Performance will be improved after the primary goal has been achieved. This is because it is hard to identify the places where the performance drawbacks are. This is also agreed by AP and NM.

Another GS design concern from LZ is that the GS should be as simple as possible. LZ said that two aspects need to be considered during the GS creation process, the users' requirements and the information contained in data sources. Users may require that certain information must or must not appear in the GS. Also, users may change their perspectives during the integration process. Therefore, the integration process has to be incremental and mappings can be improved in each iteration.

Regarding the aspect of the complexity of the global schema, AP said that integrators do not want to have the same information appear in multiple places in the GS, which will increase its complexity. If the data sources contain duplicated information, it is better to inherit them in the GS, then drop these duplications in the following step, if necessary. She also said that it is not necessary to achieve normal form in the GS since that could increase the complexity of the mappings.

Do you validate the mappings after the integration process?

Answer - LZ said that he normally runs queries on the GS and checks the query results in order to decide if the mappings are correct. However, he also indicated a few problems when this approach is used. For example, the integrators can only detect if the integrated resources are working and return results, such as no exceptions being returned. However, the data integrators do not know that if the returned values are correct with respect to the users' requirements. In this case, AP said one solution is to develop the test cases. In a complete test case, the integrators need to give the inputs and expected outputs. This solution is similar to the acceptance test in software engineering domain. LZ said that, in the iSpider project, this approach was adopted with only testing queries, not the expected results. Real data are preferred in such test cases. In some cases, sample data also need to be generated. In the future, programs should be created to capture the characteristics of the data sources and generate sample data automatically.

For the question of at which stage, the mappings should be validated, LZ said he would run queries for each GS construct when it is created and discover if it returns correct results. Then the integrators need to decide how they will examine the mappings with respect to such results. NM gave an example of examining mappings relating to particular areas of interest. The integrators may be interested in all GS constructs representing relationships between sequences and structure of proteins, and completely leave aside the functional analysis of what these proteins do. Therefore, he concluded that, in this step, integrators need to consider particular groups of applications which connect with a subset of the overall GS and examine this subset of constructs one by one. This is also agreed by AP. LZ added an example from the iSpider project that protein queries and peptide queries were tested separately in the testing stage.

How do you find the conflicts in the mappings? What kinds of conflicts have you discovered from your past work?

Answer - In terms of conflicts existing in the mappings, LZ presented the examples of four possible conflicts, data types, structure, synonym and antonym names used in the corresponding schema constructs. Information could be lost due to these conflicts. NM and LZ gave an example in the iSpider project where the structures used in the data sources contain some features which the GS does not represent. In this case, the information represented in the data sources are lost after the creation of mappings between the data sources and the GS. Such features could be that the GS constructs have 1-1 relationships, whereas the data sources support M-M relationships between the corresponding constructs. Such information loss is not avoidable and in the worst case, the integrators have to admit this loss and be careful in writing the queries over the GS. The only solution to this problem is to change the local sources or create a more comprehensive GS.

NM also addressed that in addition to the information completeness issue, the correctness of query is also important. The integrator has to guarantee that he is not writing a query to extract uncertain information. AP said, in this case, integrators also have to design queries to investigate the answer returned from users' queries. Correctness of queries is not a simple concept, she concluded. It requires to have the precise users' requirements to which kinds of correctness criteria users require. Therefore, it depends on the users' requirements and the users' queries to decide whether to provide complete, sound or exact query results.

How can you tell that your mappings include all related constructs in the data sources?

Answer - Manually, said by LZ. There is no other way to examine this issue.

If functions, such as aggregation and type transformation functions, are used in the mappings, is it possible to track back to the original sources where information are extracted?

Answer - AP said this is the research of data lineage which can be found in many research works. She also suggested the work done by Fan Hao in the AutoMed project.

Do you consider reusing your mappings? How do you reuse the mappings? Is it often?

Answer - LZ said he would reuse his mappings depending on the integration scenarios. If the data source evolves, he may use the original mappings as the starting point and modify them to meet the changes.

Appendix B

Integration Setting for the Case Study

This appendix contains the case study relating to the HE domain we used to discuss the quality framework, quality factors and quality architecture in Chapters 4, 5 and 6. Section B.1 presents three local schemas we integrate in the case study and Section B.2 presents the GS in this case study. Section B.3 presents the HE domain ontology we created manually. Section B.4 lists the matchings and mappings generated for this case study and Section B.5 lists the corresponding quality measurement results discussed in Chapter 6.

B.1 Local Schemas for the Case Study

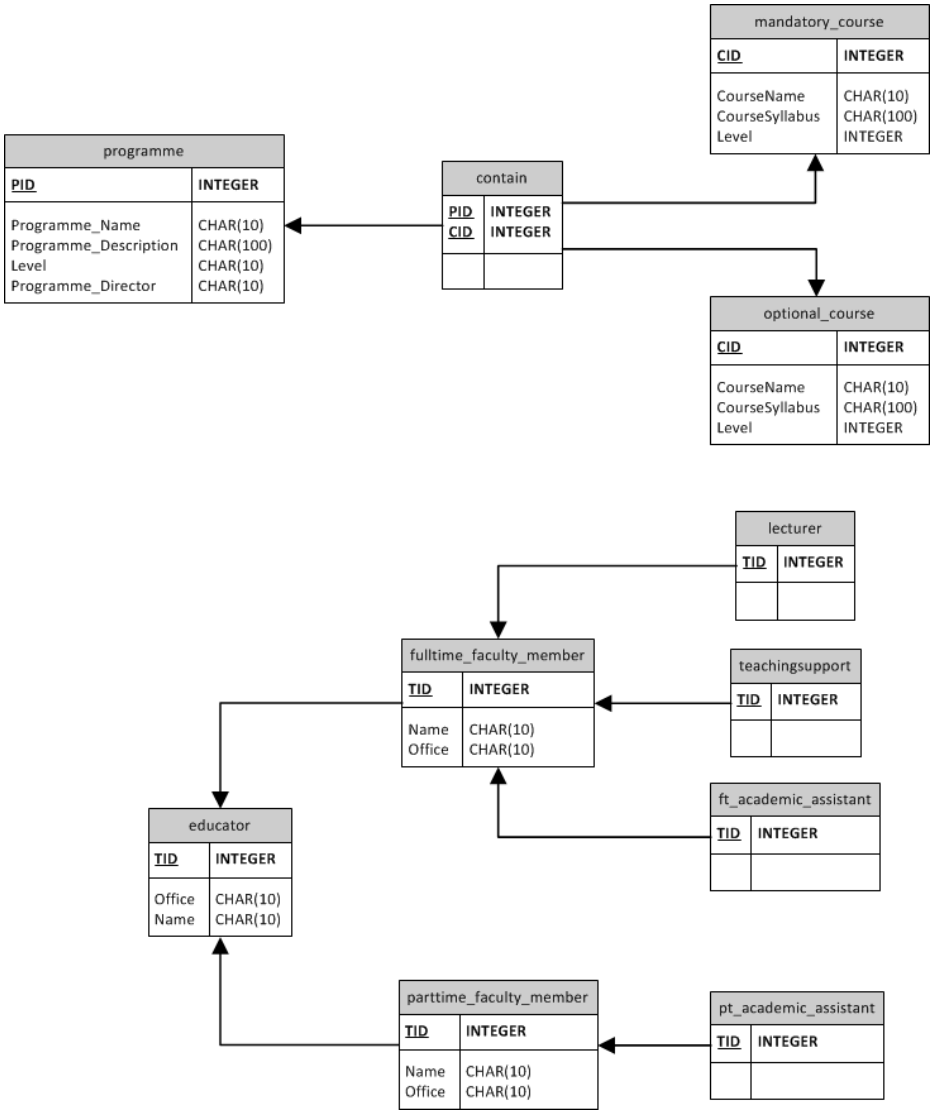


Figure B.1: Local Schema 1 (LS1)

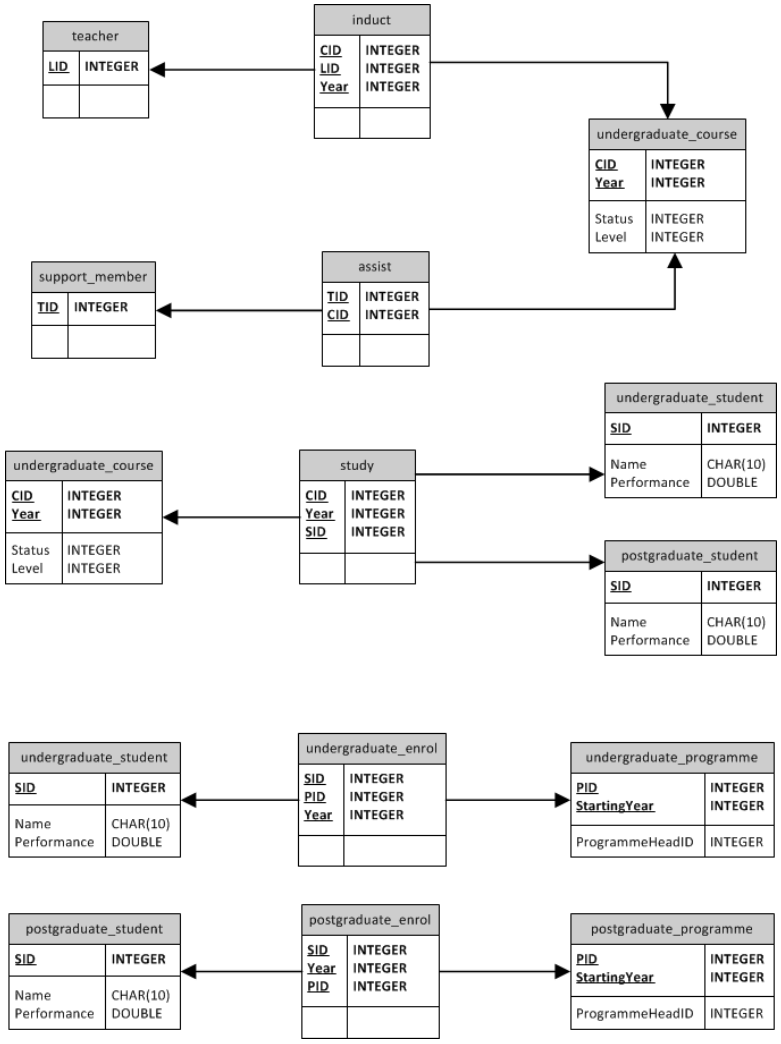


Figure B.2: Local Schema 2 (LS2)

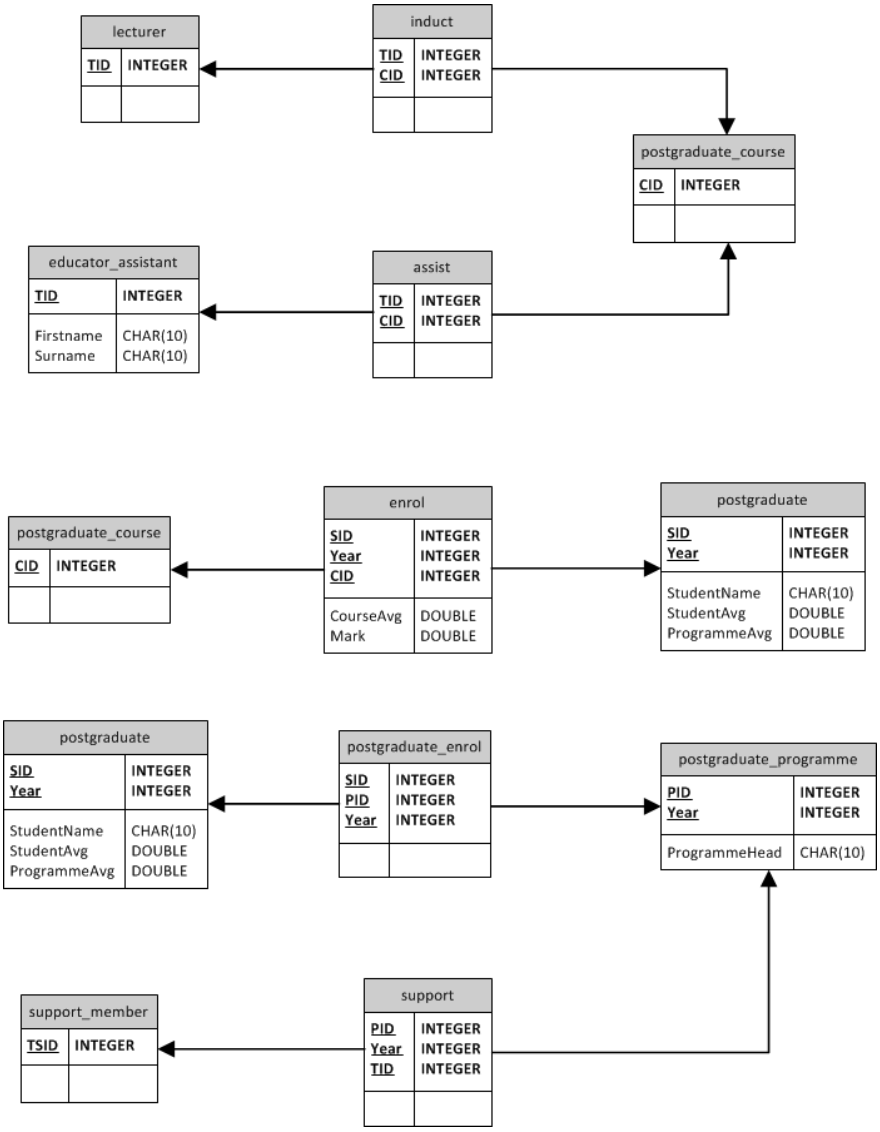


Figure B.3: Local Schema 3 (LS3)

B.2 GS for the Case Study

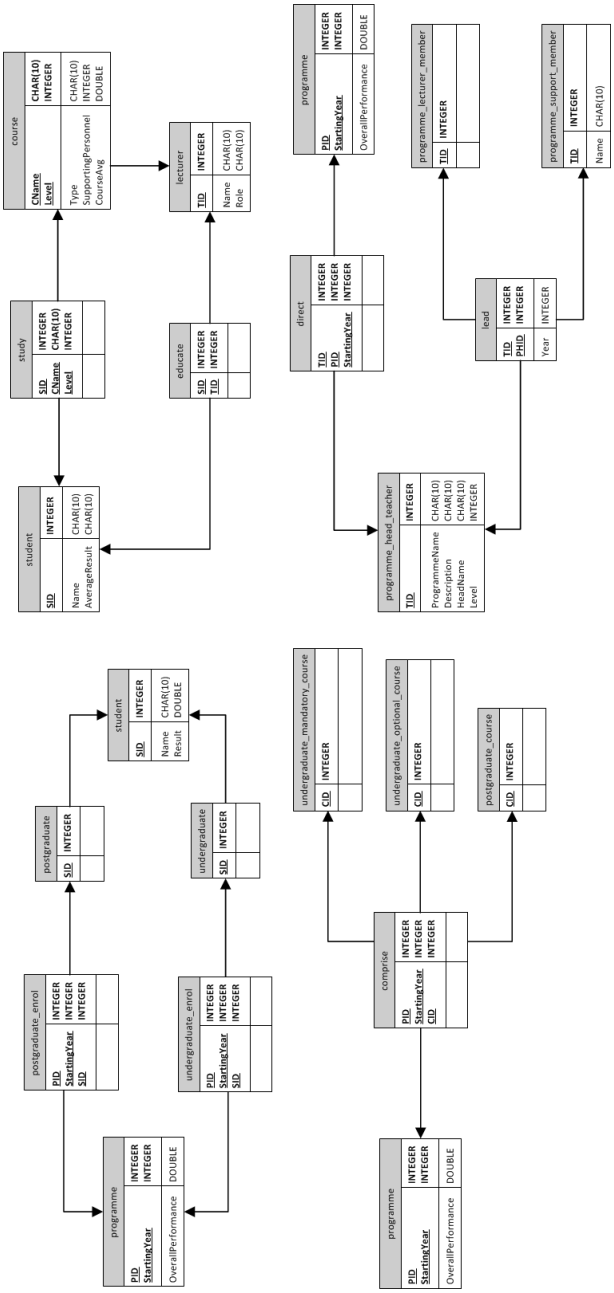


Figure B.4: Global Schema (GS)

B.3 University Domain Ontology

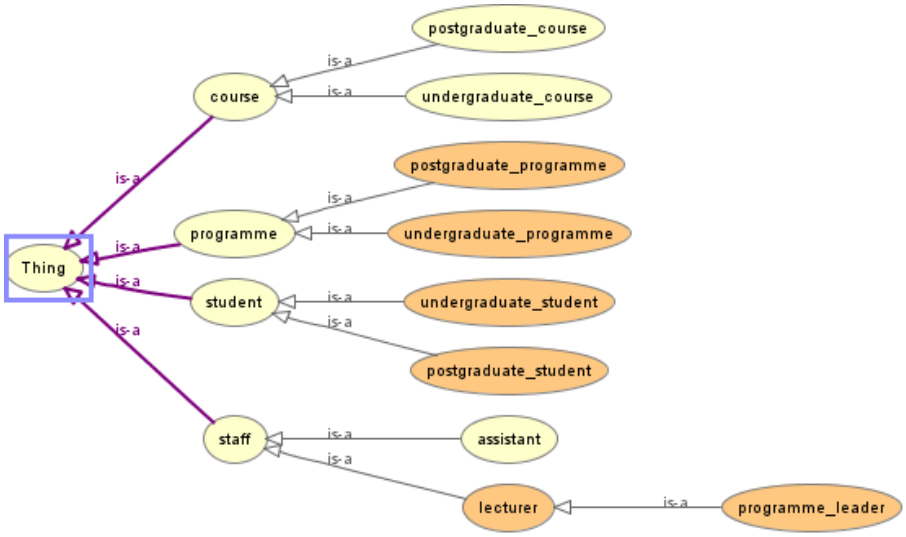


Figure B.5: The University Domain Ontology

No.	Case-Specific Knowledge, expressed in OWL-DL
B1	$postgradaute\_student \equiv enrol\ max\ 1 postgraduate\_programme$
B2	$undergraduate\_student \equiv enrol\ max\ 1 undergraduate\_programme$
B3	$postgraduate\_programme \equiv is\_directed\_by\ max\ 1 programme\_leader$
B4	$undergraduate\_programme \equiv is\_directed\_by\ max\ 1 programme\_leader$
B5	$programme\_leader \equiv direct\ max\ 1 postgraduate\_programme$
B6	$programme\_leader \equiv direct\ max\ 1 undergraduate\_programme$
B7	$lecturer \equiv teachsome postgraduate\_course$
B8	$lecturer \equiv teachsome undergraduate\_course$
B9	$programme\_leader \equiv leadsome lecturer$
B10	$undergraduate\_student \equiv study\ max\ 8 undergraduate\_course$
B11	$postgraduate\_student \equiv study\ max\ 2 undergraduate\_course$
B12	$postgraduate\_student \equiv study\ max\ 5 postgraduate\_course$
B13	$undergraduate\_programme \equiv is\_enrolled\_by\ max\ 200 undergraduate\_student$
B14	$postgradaute\_programme \equiv is\_enrolled\_by\ max\ 100 postgraduate\_student$

Table B.1: Case-Specific Knowledge

B.4 Matchings and Mappings for the Case Study

B.4.1 Matching Results

No.	Correspondence
1	$GS : ProgrammeHead - LS_1 : ProgrammeHead$
2	$GS : ProgrammeHead - LS_2 : ProgrammeHead$
3	$GS : ProgrammeHead - LS_3 : ProgrammeHead$
4	$GS : ProgrammeStaff - LS_1 : Staff$
5	$GS : ProgrammeStaff - LS_1 : Lecturers$
6	$GS : ProgrammeStaff - LS_2 : Lecturer$
7	$GS : ProgrammeStaff - LS_3 : Teacher$
8	$GS : Course - LS_1 : MandatoryCourse$
9	$GS : Course - LS_1 : OptionalCourse$
10	$GS : Course - LS_2 : ugCourse$
11	$GS : Course - LS_3 : pgCourse$
12	$GS : pgCourse - LS_3 : Module$
13	$GS : Student - LS_2 : ugStudents$
14	$GS : Student - LS_2 : pgStudents$
15	$GS : undergraduate - LS_2 : ugStudents$
16	$GS : postgraduate - LS_2 : pgStudents$
17	$GS : postgraduate - LS_3 : postgraduate$
18	$GS : AcademicStaff - LS_1 : Staff$
19	$GS : AcademicStaff - LS_1 : Lecturers$
20	$GS : AcademicStaff - LS_2 : Lecturer$
21	$GS : AcademicStaff - LS_3 : Teacher$
22	$GS : assist - LS_3 : assists$
23	$GS : teaches - LS_2 : lecture$
24	$GS : teaches - LS_3 : teach$
25	$GS : uenrol - LS_2 : uenrol$
26	$GS : penrol - LS_2 : penrol$
27	$GS : penrol - LS_3 : penrol$
28	$GS : studies - ls_2 : studies$

Table B.2: Matching Results

B.4.2 Mappings

No.	GS Construct : – GAV mappings

1	⟨⟨programme_head_teacher⟩⟩ :
	[{tid}|{pid, phname}←⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Director⟩⟩; {tid, sname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩; phname = sname] ++ [{tid}|{{pid, year}, tid}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩] ++ [{tid}|{{pid, year}, tid}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩; year > 1999] ++ [{tid}|{{pid, year}, phname}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩; {tid, tname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩; phname = tname]
2	⟨⟨programme_head_teacher, TID⟩⟩ :
	[{tid, tid}|{pid, phname}←⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Director⟩⟩; {tid, sname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩; phname = sname] ++ [{tid, tid}|{{pid, year}, tid}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩] ++ [{tid, tid}|{{pid, year}, tid}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩] ++ [{tid, tid}|{{pid, year}, phname}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩; {tid, tname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩; phname = tname]
3	⟨⟨programme_head_teacher, ProgrammeName⟩⟩ :
	[{phid, pname}|{pid, pname}←⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Name⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩] ++ [{phid, pname}|{pid, pname}←⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Name⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩] ++ [{phid, pname}|{pid, pname}←⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Name⟩⟩; {{year, pid}, phname}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩; {phid, phname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩]
4	⟨⟨programme_head_teacher, Description⟩⟩ :
	[{phid, pdes}|{pid, pdes}←⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Description⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩] ++ [{phid, pdes}|{pid, pdes}←⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Description⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩] ++ [{phid, pdes}|{pid, pdes}←⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Description⟩⟩; {{year, pid}, phname}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩; {phid, phname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩]
5	⟨⟨programme_head_teacher, HeadName⟩⟩ :
	[{phid, tname}|{{pid, year}, phid}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩; {tsid, tid}←⟨⟨ls1_LS1_lecturer, ls1_LS1_TID⟩⟩; phid = tid; {tid, tname}←⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩; tid = tsid] ++ [{phid, tname}|{{pid, year}, phid}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩; {tid, tname}←⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩; phid = tid] ++ [{tid, tname}|{{pid, year}, phname}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩; {tid, tname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩; phname = tname]
6	⟨⟨programme_head_teacher, Level⟩⟩ :
	[{phid, “ug” }|{{pid, year}, phid}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩] ++ [{phid, “pg” }|{{pid, year}, phid}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩] ++ [{phid, “pg” }|{{pid, year}, phname}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩; {phid, phname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩]
7	⟨⟨lead⟩⟩ :
	[{phid, lid}|{cid, lid, year}←⟨⟨ls2_LS2_induct⟩⟩; {year, cid}←⟨⟨ls2_LS2_undergraduate_course⟩⟩;

	{sid, cid, year}←⟨⟨ls2_LS2_study⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩; {sid, pid}←⟨⟨ls2_LS2_undergraduate_enrol⟩⟩; {sid}←⟨⟨ls2_LS2_undergraduate_student⟩⟩] ++ [{phid, lid}|{cid, lid, year}←⟨⟨ls2_LS2_induct⟩⟩; {year, cid}←⟨⟨ls2_LS2_undergraduate_course⟩⟩; {sid, cid, year}←⟨⟨ls2_LS2_study⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩; {sid, pid}←⟨⟨ls2_LS2_postgraduate_enrol⟩⟩; {sid}←⟨⟨ls2_LS2_postgraduate_student⟩⟩] ++ [{phid, tid}|{tid, cid}←⟨⟨ls3_LS3_induct⟩⟩; {sid, cid}←⟨⟨ls3_LS3_enrol⟩⟩; {year, sid, pid}←⟨⟨ls3_LS3_postgraduate_enrol⟩⟩; {phid, phname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩; {{year, pid}, phname}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩]
8	⟨⟨lead, TID⟩⟩ :
	[{{phid, lid}, lid}|{cid, lid, year}←⟨⟨ls2_LS2_induct⟩⟩; {year, cid}←⟨⟨ls2_LS2_undergraduate_course⟩⟩; {sid, cid, year}←⟨⟨ls2_LS2_study⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩; {sid, pid}←⟨⟨ls2_LS2_undergraduate_enrol⟩⟩; {sid}←⟨⟨ls2_LS2_undergraduate_student⟩⟩] ++ [{{phid, lid}, lid}|{cid, lid, year}←⟨⟨ls2_LS2_induct⟩⟩; {year, cid}←⟨⟨ls2_LS2_undergraduate_course⟩⟩; {sid, cid, year}←⟨⟨ls2_LS2_study⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩; {sid}←⟨⟨ls2_LS2_postgraduate_student⟩⟩; {sid, pid}←⟨⟨ls2_LS2_postgraduate_enrol⟩⟩; ++ [{{phid, tid}, tid}|{tid, cid}←⟨⟨ls3_LS3_induct⟩⟩; {year, sid, pid}←⟨⟨ls3_LS3_postgraduate_enrol⟩⟩; {sid, cid}←⟨⟨ls3_LS3_enrol⟩⟩; {phid, phname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩; {{year, pid}, phname}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩;
9	⟨⟨lead, PHID⟩⟩ :
	[{{phid, lid}, phid}|{cid, lid, year}←⟨⟨ls2_LS2_induct⟩⟩; {sid, cid, year}←⟨⟨ls2_LS2_study⟩⟩; {year, cid}←⟨⟨ls2_LS2_undergraduate_course⟩⟩; {sid}←⟨⟨ls2_LS2_undergraduate_student⟩⟩; {sid, pid}←⟨⟨ls2_LS2_undergraduate_enrol⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩] ++ [{{phid, lid}, phid}|{cid, lid, year}←⟨⟨ls2_LS2_induct⟩⟩; {year, cid}←⟨⟨ls2_LS2_undergraduate_course⟩⟩; {sid, cid, year}←⟨⟨ls2_LS2_study⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩; {sid}←⟨⟨ls2_LS2_postgraduate_student⟩⟩; {sid, pid}←⟨⟨ls2_LS2_postgraduate_enrol⟩⟩] ++ [{{phid, tid}, phid}|{tid, cid}←⟨⟨ls3_LS3_induct⟩⟩; {year, sid, pid}←⟨⟨ls3_LS3_postgraduate_enrol⟩⟩; {sid, cid}←⟨⟨ls3_LS3_enrol⟩⟩; {{year, pid}, phname}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩; {phid, phname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩]
10	⟨⟨lead, Year⟩⟩ :
	[{{phid, lid}, year}|{cid, lid, year}←⟨⟨ls2_LS2_induct⟩⟩; {year, cid}←⟨⟨ls2_LS2_undergraduate_course⟩⟩; {sid, cid, year}←⟨⟨ls2_LS2_study⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩; {sid}←⟨⟨ls2_LS2_undergraduate_student⟩⟩; {sid, pid}←⟨⟨ls2_LS2_undergraduate_enrol⟩⟩] ++ [{{phid, lid}, year}|{cid, lid, year}←⟨⟨ls2_LS2_induct⟩⟩; {year, cid}←⟨⟨ls2_LS2_undergraduate_course⟩⟩; {sid, cid, year}←⟨⟨ls2_LS2_study⟩⟩; {{year, pid}, phid}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩; {sid}←⟨⟨ls2_LS2_postgraduate_student⟩⟩; {sid, pid}←⟨⟨ls2_LS2_postgraduate_enrol⟩⟩; ++ [{{phid, tid}, year}|{tid, cid}←⟨⟨ls3_LS3_induct⟩⟩; {sid, cid}←⟨⟨ls3_LS3_enrol⟩⟩; {{year, pid}, phname}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩; {year, sid, pid}←⟨⟨ls3_LS3_postgraduate_enrol⟩⟩; {phid, phname}←⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩]
11	⟨⟨programme_lecturer_member⟩⟩ :
	[{lid}|lid←⟨⟨ls2_LS2_teacher⟩⟩] ++ [{tid}|tid←⟨⟨ls3_LS3_lecturer⟩⟩]
12	⟨⟨programme_lecturer_member, TID⟩⟩ :
	[{lid, lid}|lid←⟨⟨ls2_LS2_teacher⟩⟩] ++ [{tid, tid}|tid←⟨⟨ls3_LS3_lecturer⟩⟩]
13	⟨⟨programme_support_member⟩⟩ :
	[{tid}|{tid, ttid}←⟨⟨ls1_LS1_teachingsupport, ls1_LS1_TID⟩⟩] ++ [{ftid}|{ftid, ltid}←⟨⟨ls1_LS1_ft_academic_assistant, ls1_LS1_TID⟩⟩; {ftid, ltid}←⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩] ++

	[{ptid}|{ptid, ltid}←⟨⟨ls1_LS1_pt_academic_assistant, ls1_LS1_TID⟩⟩; {ptid, ltid}←⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩] ++ [{ftid}|{ftid, ltid}←⟨⟨ls1_LS1_ft_academic_assistant, ls1_LS1_TID⟩⟩; {ftid, ltid}←⟨⟨ls3_LS3_educator_assistant, ls3_LS3_TID⟩⟩] ++ [{ptid}|{ptid, ltid}←⟨⟨ls1_LS1_pt_academic_assistant, ls1_LS1_TID⟩⟩; {ptid, ltid}←⟨⟨ls3_LS3_support_member, ls3_LS3_TSID⟩⟩]
14	⟨⟨programme_support_member, TID⟩⟩ :
	[{tsid, tsid}|{tsid, tid}←⟨⟨ls1_LS1_teachingsupport, ls1_LS1_TID⟩⟩] ++ [{ftid, ftid}|{ftid, lid}←⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩] ++ [{taid, taid}|{taid, lid}←⟨⟨ls3_LS3_educator_assistant, ls3_LS3_TID⟩⟩] ++ [{tsid, tsid}|{tsid, lid}←⟨⟨ls3_LS3_support_member, ls3_LS3_TSID⟩⟩]
15	⟨⟨programme_support_member, Name⟩⟩ :
	[{ftid, name}|{ftid, name}←⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩; {ftid, lid}←⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩] ++ [{ptid, name}|{ptid, name}←⟨⟨ls1_LS1_parttime_faculty_member, ls1_LS1_Name⟩⟩; {ptid, lid}←⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩] ++ [{ftid, name}|{ftid, name}←⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩; {taid, lid}←⟨⟨ls3_LS3_educator_assistant, ls3_LS3_TID⟩⟩; ftid = taid] ++ [{ptid, name}|{ptid, name}←⟨⟨ls1_LS1_parttime_faculty_member, ls1_LS1_Name⟩⟩; {ptid, lid}←⟨⟨ls3_LS3_support_member, ls3_LS3_TSID⟩⟩]
16	⟨⟨course⟩⟩ :
	[{cname, level}|{cid, cname}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩] ++ [{cname, level}|{cid, cname}←⟨⟨ls1_LS1_optional_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_optional_course, ls1_LS1_Level⟩⟩]
17	⟨⟨course, Cname⟩⟩ :
	[{{cname, level}, cname}|{cid, cname}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩] ++ [{{cname, level}, cname}|{cid, cname}←⟨⟨ls1_LS1_optional_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_optional_course, ls1_LS1_Level⟩⟩]
18	⟨⟨course, Level⟩⟩ :
	[{{cname, level}, level}|{cid, cname}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩] ++ [{{cname, level}, level}|{cid, cname}←⟨⟨ls1_LS1_optional_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_optional_course, ls1_LS1_Level⟩⟩]
19	⟨⟨course, Convenor⟩⟩ :
	[{{cname, level}, name}|{{year, tid, cid}, ccid}←⟨⟨ls2_LS2_induct, ls2_LS2_CID⟩⟩; {tid, ttid}←⟨⟨ls1_LS1_lecturer, ls1_LS1_TID⟩⟩; {tid, name}←⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩; {cid, cname}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩] ++ [{{cname, level}, name}|{{tid, cid}, ccid}←⟨⟨ls3_LS3_induct, ls3_LS3_CID⟩⟩; {tid, ttid}←⟨⟨ls1_LS1_lecturer, ls1_LS1_TID⟩⟩; {tid, name}←⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩; {cid, cname}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩]
20	⟨⟨course, Type⟩⟩ :
	[{{cname, level}, "ug"}|{cid, ccid}←⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_CID⟩⟩; {cid, cname}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩] ++ [{{cname, level}, "pg"}|{cid, ccid}←⟨⟨ls3_LS3_postgraduate_course, ls3_LS3_CID⟩⟩; {cid, cname}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩]
21	⟨⟨course, SupportingPersonnel⟩⟩ :
	[{{cname, level}, tid}|{cid}←⟨⟨ls2_LS2_undergraduate_course⟩⟩; {{tid, cid}, ttid}←⟨⟨ls2_LS2_assist, ls2_LS2_TID⟩⟩; {cid, cname}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩]

	++ [{{cname, level}, tid}|{cid, ccid}←⟨⟨ls3_LS3_postgraduate_course, ls3_LS3_CID⟩⟩; {{tid, cid}, ttid}←⟨⟨ls3_LS3_assist, ls3_LS3_TID⟩⟩; {cid, cname}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩; {cid, level}←⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩]
22	⟨⟨lecturer⟩⟩ :
	[{tid}|{tid, name}←⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩; {tid, ttid}←⟨⟨ls1_LS1_lecturer, ls1_LS1_TID⟩⟩]
23	⟨⟨lecturer, TID⟩⟩ :
	[{tid, tid}|{tid, name}←⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩; {tid, ttid}←⟨⟨ls1_LS1_lecturer, ls1_LS1_TID⟩⟩]
24	⟨⟨educate⟩⟩ :
	[{lid, sid}|{{year, lid, cid}, ccid}←⟨⟨ls2_LS2_induct, ls2_LS2_CID⟩⟩; {{year, sid, cid}, ssid}←⟨⟨ls2_LS2_study, ls2_LS2_SID⟩⟩] ++ [{tid, sid}|{{tid, cid}, ttid}←⟨⟨ls3_LS3_induct, ls3_LS3_TID⟩⟩; {{year, sid, cid}, ssid}←⟨⟨ls3_LS3_enrol, ls3_LS3_SID⟩⟩]
25	⟨⟨programme⟩⟩ :
	[{pid, year}|pid←⟨⟨ls1_LS1_programme⟩⟩; {{year, pid}, a}←⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_PID⟩⟩] ++ [{pid, year}|pid←⟨⟨ls1_LS1_programme⟩⟩; {{year, pid}, a}←⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_PID⟩⟩; year > 1999] ++ [{pid, year}|{{year, pid}, a}←⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_PID⟩⟩]

Table B.3: GAV and LAV Mappings

B.5 Case Study Results

Quality Factor 1

Satisfying Elements	
⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩	⟨⟨ls2_LS2_study, ls2_LS2_SID⟩⟩
⟨⟨ls1_LS1_ft_academic_assistant, ls1_LS1_TID⟩⟩	⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩
⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩	⟨⟨ls2_LS2_teacher⟩⟩
⟨⟨ls1_LS1_lecturer, ls1_LS1_TID⟩⟩	⟨⟨ls2_LS2_undergraduate_course⟩⟩
⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩	⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_CID⟩⟩
⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩	⟨⟨ls2_LS2_undergraduate_enrol⟩⟩
⟨⟨ls1_LS1_optional_course, ls1_LS1_CourseName⟩⟩	⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_PID⟩⟩
⟨⟨ls1_LS1_optional_course, ls1_LS1_Level⟩⟩	⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩
⟨⟨ls1_LS1_parttime_faculty_member, ls1_LS1_Name⟩⟩	⟨⟨ls2_LS2_undergraduate_student⟩⟩
⟨⟨ls1_LS1_programme⟩⟩	⟨⟨ls3_LS3_assist, ls3_LS3_TID⟩⟩
⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Description⟩⟩	⟨⟨ls3_LS3_educator_assistant, ls3_LS3_TID⟩⟩
⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Director⟩⟩	⟨⟨ls3_LS3_induct⟩⟩
⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Name⟩⟩	⟨⟨ls3_LS3_induct, ls3_LS3_CID⟩⟩
⟨⟨ls1_LS1_pt_academic_assistant, ls1_LS1_TID⟩⟩	⟨⟨ls3_LS3_induct, ls3_LS3_TID⟩⟩
⟨⟨ls1_LS1_teachingsupport, ls1_LS1_TID⟩⟩	⟨⟨ls3_LS3_lecturer⟩⟩
⟨⟨ls2_LS2_assist, ls2_LS2_TID⟩⟩	⟨⟨ls3_LS3_postgraduate_course, ls3_LS3_CID⟩⟩
⟨⟨ls2_LS2_induct⟩⟩	⟨⟨ls3_LS3_postgraduate_enrol⟩⟩
⟨⟨ls2_LS2_induct, ls2_LS2_CID⟩⟩	⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_PID⟩⟩
⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_PID⟩⟩	⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩
⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩	⟨⟨ls3_LS3_study⟩⟩
⟨⟨ls2_LS2_postgraduate_enrol⟩⟩	⟨⟨ls3_LS3_study, ls3_LS3_SID⟩⟩
⟨⟨ls2_LS2_postgraduate_student⟩⟩	⟨⟨ls3_LS3_support_member, ls3_LS3_TSID⟩⟩
⟨⟨ls2_LS2_study⟩⟩	

Table B.4: Satisfying Elements for Quality Factor 1

not-Satisfying Elements	
⟨⟨ls1_LS1_contain⟩⟩	⟨⟨ls2_LS2_teacher, ls2_LS2_LID⟩⟩
⟨⟨ls1_LS1_contain, ls1_LS1_CID⟩⟩	⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_Level⟩⟩

⟨⟨ls1_LS1_contain, ls1_LS1_PID⟩⟩	⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_Status⟩⟩
⟨⟨ls1_LS1_educator⟩⟩	⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_Year⟩⟩
⟨⟨ls1_LS1_educator, ls1_LS1_Office⟩⟩	⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_PID⟩⟩
⟨⟨ls1_LS1_educator, ls1_LS1_TID⟩⟩	⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_SID⟩⟩
⟨⟨ls1_LS1_ft_academic_assistant⟩⟩	⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_Year⟩⟩
⟨⟨ls1_LS1_fulltime_faculty_member⟩⟩	⟨⟨ls2_LS2_undergraduate_programme⟩⟩
⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Office⟩⟩	⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_StartingYear⟩⟩
⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_TID⟩⟩	⟨⟨ls2_LS2_undergraduate_student, ls2_LS2_Name⟩⟩
⟨⟨ls1_LS1_lecturer⟩⟩	⟨⟨ls2_LS2_undergraduate_student, ls2_LS2_Performance⟩⟩
⟨⟨ls1_LS1_mandatory_course⟩⟩	⟨⟨ls2_LS2_undergraduate_student, ls2_LS2_SID⟩⟩
⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CID⟩⟩	⟨⟨ls3_LS3_assist⟩⟩
⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseSyllabus⟩⟩	⟨⟨ls3_LS3_assist, ls3_LS3_CID⟩⟩
⟨⟨ls1_LS1_optional_course⟩⟩	⟨⟨ls3_LS3_educator_assistant⟩⟩
⟨⟨ls1_LS1_optional_course, ls1_LS1_CID⟩⟩	⟨⟨ls3_LS3_educator_assistant, ls3_LS3_Firstname⟩⟩
⟨⟨ls1_LS1_optional_course, ls1_LS1_CourseSyllabus⟩⟩	⟨⟨ls3_LS3_educator_assistant, ls3_LS3_Surname⟩⟩
⟨⟨ls1_LS1_parttime_faculty_member⟩⟩	⟨⟨ls3_LS3_lecturer, ls3_LS3_TID⟩⟩
⟨⟨ls1_LS1_parttime_faculty_member, ls1_LS1_Office⟩⟩	⟨⟨ls3_LS3_postgraduate⟩⟩
⟨⟨ls1_LS1_parttime_faculty_member, ls1_LS1_TID⟩⟩	⟨⟨ls3_LS3_postgraduate, ls3_LS3_ProgrammeAvg⟩⟩
⟨⟨ls1_LS1_programme, ls1_LS1_Level⟩⟩	⟨⟨ls3_LS3_postgraduate, ls3_LS3_SID⟩⟩
⟨⟨ls1_LS1_programme, ls1_LS1_PID⟩⟩	⟨⟨ls3_LS3_postgraduate, ls3_LS3_StudentAvg⟩⟩
⟨⟨ls1_LS1_pt_academic_assistant⟩⟩	⟨⟨ls3_LS3_postgraduate, ls3_LS3_StudentName⟩⟩
⟨⟨ls1_LS1_teachingsupport⟩⟩	⟨⟨ls3_LS3_postgraduate, ls3_LS3_Year⟩⟩
⟨⟨ls2_LS2_assist⟩⟩	⟨⟨ls3_LS3_postgraduate_course⟩⟩
⟨⟨ls2_LS2_assist, ls2_LS2_CID⟩⟩	⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_PID⟩⟩
⟨⟨ls2_LS2_induct, ls2_LS2_LID⟩⟩	⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_SID⟩⟩
⟨⟨ls2_LS2_induct, ls2_LS2_Year⟩⟩	⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_Year⟩⟩
⟨⟨ls2_LS2_postgraduate_programme⟩⟩	⟨⟨ls3_LS3_postgraduate_programme⟩⟩
⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_StartingYear⟩⟩	⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_Year⟩⟩
⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_PID⟩⟩	⟨⟨ls3_LS3_study, ls3_LS3_CID⟩⟩
⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_SID⟩⟩	⟨⟨ls3_LS3_study, ls3_LS3_CourseAvg⟩⟩
⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_Year⟩⟩	⟨⟨ls3_LS3_study, ls3_LS3_Mark⟩⟩
⟨⟨ls2_LS2_postgraduate_student, ls2_LS2_Name⟩⟩	⟨⟨ls3_LS3_study, ls3_LS3_Year⟩⟩
⟨⟨ls2_LS2_postgraduate_student, ls2_LS2_Performance⟩⟩	⟨⟨ls3_LS3_support⟩⟩
⟨⟨ls2_LS2_postgraduate_student, ls2_LS2_SID⟩⟩	⟨⟨ls3_LS3_support, ls3_LS3_PID⟩⟩
⟨⟨ls2_LS2_study, ls2_LS2_CID⟩⟩	⟨⟨ls3_LS3_support, ls3_LS3_TID⟩⟩
⟨⟨ls2_LS2_study, ls2_LS2_Year⟩⟩	⟨⟨ls3_LS3_support_member⟩⟩
⟨⟨ls2_LS2_support_member⟩⟩	

Table B.5: not-Satisfying Elements for Quality Factor 1

Quality Factor 2

Concept	Referenced	Unreferenced	Result
CID	4	7	0.40
Description	1	2	0.33
Head	4	0	1.00
induct	2	3	0.40
Level	2	2	0.50
Name	6	5	0.55
PID	3	4	0.43
Programme	1	2	0.33
register	3	0	1.00
SID	2	6	0.29
Student	2	1	0.67
study	2	0	1.00
Teacher	3	6	0.33
TID	9	6	0.60

Table B.6: Coverage of Concepts for Quality Factor 2

Concepts	Satisfying Elements	non-Satisfying Elements
CID	⟨⟨ls2_LS2_induct, ls2_LS2_CID⟩⟩ ⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_CID⟩⟩ ⟨⟨ls3_LS3_induct, ls3_LS3_CID⟩⟩ ⟨⟨ls3_LS3_postgraduate_course, ls3_LS3_CID⟩⟩	⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CID⟩⟩ ⟨⟨ls1_LS1_optional_course, ls1_LS1_CID⟩⟩ ⟨⟨ls2_LS2_assist, ls2_LS2_CID⟩⟩ ⟨⟨ls2_LS2_study, ls2_LS2_CID⟩⟩ ⟨⟨ls3_LS3_assist, ls3_LS3_CID⟩⟩ ⟨⟨ls3_LS3_study, ls3_LS3_CID⟩⟩
Description	⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Description⟩⟩	⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseSyllabus⟩⟩ ⟨⟨ls1_LS1_optional_course, ls1_LS1_CourseSyllabus⟩⟩
Head	⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Director⟩⟩ ⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩ ⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩ ⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩	
induct	⟨⟨ls2_LS2_induct⟩⟩ ⟨⟨ls3_LS3_induct⟩⟩	⟨⟨ls2_LS2_assist⟩⟩ ⟨⟨ls3_LS3_assist⟩⟩ ⟨⟨ls3_LS3_support⟩⟩
Level	⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩ ⟨⟨ls1_LS1_optional_course, ls1_LS1_Level⟩⟩	⟨⟨ls1_LS1_programme, ls1_LS1_Level⟩⟩ ⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_Level⟩⟩
Name	⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩ ⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩ ⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩ ⟨⟨ls1_LS1_optional_course, ls1_LS1_CourseName⟩⟩ ⟨⟨ls1_LS1_parttime_faculty_member, ls1_LS1_Name⟩⟩ ⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Name⟩⟩	⟨⟨ls2_LS2_postgraduate_student, ls2_LS2_Name⟩⟩ ⟨⟨ls2_LS2_undergraduate_student, ls2_LS2_Name⟩⟩ ⟨⟨ls3_LS3_educator_assistant, ls3_LS3_Firstname⟩⟩ ⟨⟨ls3_LS3_educator_assistant, ls3_LS3_Surname⟩⟩ ⟨⟨ls3_LS3_postgraduate, ls3_LS3_StudentName⟩⟩
PID	⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_PID⟩⟩ ⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_PID⟩⟩ ⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_PID⟩⟩	⟨⟨ls1_LS1_programme, ls1_LS1_PID⟩⟩ ⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_PID⟩⟩ ⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_PID⟩⟩ ⟨⟨ls3_LS3_support, ls3_LS3_PID⟩⟩
Programme	⟨⟨ls1_LS1_programme⟩⟩	⟨⟨ls2_LS2_postgraduate_programme⟩⟩ ⟨⟨ls3_LS3_postgraduate_programme⟩⟩
register	⟨⟨ls2_LS2_postgraduate_enrol⟩⟩ ⟨⟨ls2_LS2_undergraduate_enrol⟩⟩ ⟨⟨ls3_LS3_postgraduate_enrol⟩⟩	
SID	⟨⟨ls2_LS2_study, ls2_LS2_SID⟩⟩ ⟨⟨ls3_LS3_study, ls3_LS3_SID⟩⟩	⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_SID⟩⟩ ⟨⟨ls2_LS2_postgraduate_student, ls2_LS2_SID⟩⟩ ⟨⟨ls2_LS2_undergraduate_student, ls2_LS2_SID⟩⟩ ⟨⟨ls3_LS3_postgraduate, ls3_LS3_SID⟩⟩ ⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_SID⟩⟩
Student	⟨⟨ls2_LS2_postgraduate_student⟩⟩ ⟨⟨ls2_LS2_undergraduate_student⟩⟩	⟨⟨ls3_LS3_postgraduate⟩⟩
study	⟨⟨ls2_LS2_study⟩⟩ ⟨⟨ls3_LS3_study⟩⟩	
Teacher	⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩ ⟨⟨ls2_LS2_teacher⟩⟩ ⟨⟨ls3_LS3_lecturer⟩⟩	⟨⟨ls1_LS1_fulltime_faculty_member⟩⟩ ⟨⟨ls1_LS1_lecturer⟩⟩ ⟨⟨ls1_LS1_pt_academic_assistant⟩⟩ ⟨⟨ls1_LS1_teachingsupport⟩⟩ ⟨⟨ls3_LS3_educator_assistant⟩⟩ ⟨⟨ls3_LS3_support_member⟩⟩
TID	⟨⟨ls1_LS1_ft_academic_assistant, ls1_LS1_TID⟩⟩ ⟨⟨ls1_LS1_lecturer, ls1_LS1_TID⟩⟩ ⟨⟨ls1_LS1_pt_academic_assistant, ls1_LS1_TID⟩⟩ ⟨⟨ls1_LS1_teachingsupport, ls1_LS1_TID⟩⟩ ⟨⟨ls2_LS2_assist, ls2_LS2_TID⟩⟩ ⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩ ⟨⟨ls3_LS3_assist, ls3_LS3_TID⟩⟩ ⟨⟨ls3_LS3_induct, ls3_LS3_TID⟩⟩ ⟨⟨ls3_LS3_support_member, ls3_LS3_TSID⟩⟩	⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_TID⟩⟩ ⟨⟨ls1_LS1_parttime_faculty_member, ls1_LS1_TID⟩⟩ ⟨⟨ls2_LS2_induct, ls2_LS2_LID⟩⟩ ⟨⟨ls2_LS2_teacher, ls2_LS2_LID⟩⟩ ⟨⟨ls3_LS3_lecturer, ls3_LS3_TID⟩⟩ ⟨⟨ls3_LS3_support, ls3_LS3_TID⟩⟩

Table B.7: Satisfying and non-Satisfying Elements for Quality Factor 2

Quality Factor 3

Satisfying Elements
⟨⟨ls1_LS1_educator_pkey, ls1_LS1_educator, ⟨⟨ls1_LS1_educator, ls1_LS1_TID⟩⟩⟩
⟨⟨ls1_LS1_ft_academic_assistant_pkey, ls1_LS1_ft_academic_assistant, ⟨⟨ls1_LS1_ft_academic_assistant, ls1_LS1_TID⟩⟩⟩
⟨⟨ls1_LS1_fulltime_faculty_member_pkey, ls1_LS1_fulltime_faculty_member, ⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_TID⟩⟩⟩
⟨⟨ls1_LS1_lecturer_pkey, ls1_LS1_lecturer, ⟨⟨ls1_LS1_lecturer, ls1_LS1_TID⟩⟩⟩
⟨⟨ls1_LS1_mandatory_course_pkey, ls1_LS1_mandatory_course, ⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CID⟩⟩⟩
⟨⟨ls1_LS1_optional_course_pkey, ls1_LS1_optional_course, ⟨⟨ls1_LS1_optional_course, ls1_LS1_CID⟩⟩⟩
⟨⟨ls1_LS1_parttime_faculty_member_pkey, ls1_LS1_parttime_faculty_member, ⟨⟨ls1_LS1_parttime_faculty_member, ls1_LS1_TID⟩⟩⟩
⟨⟨ls1_LS1_programme_pkey, ls1_LS1_programme, ⟨⟨ls1_LS1_programme, ls1_LS1_PID⟩⟩⟩
⟨⟨ls1_LS1_pt_academic_assistant_pkey, ls1_LS1_pt_academic_assistant, ⟨⟨ls1_LS1_pt_academic_assistant, ls1_LS1_TID⟩⟩⟩
⟨⟨ls1_LS1_teachingsupport_pkey, ls1_LS1_teachingsupport, ⟨⟨ls1_LS1_teachingsupport, ls1_LS1_TID⟩⟩⟩
⟨⟨ls2_LS2_assist_pkey, ls2_LS2_assist, ⟨⟨ls2_LS2_assist, ls2_LS2_TID⟩⟩⟩
⟨⟨ls2_LS2_assist_pkey, ls2_LS2_assist, ⟨⟨ls2_LS2_assist, ls2_LS2_CID⟩⟩⟩
⟨⟨ls2_LS2_induct_pkey, ls2_LS2_induct, ⟨⟨ls2_LS2_induct, ls2_LS2_CID⟩⟩⟩
⟨⟨ls2_LS2_induct_pkey, ls2_LS2_induct, ⟨⟨ls2_LS2_induct, ls2_LS2_LID⟩⟩⟩
⟨⟨ls2_LS2_induct_pkey, ls2_LS2_induct, ⟨⟨ls2_LS2_induct, ls2_LS2_Year⟩⟩⟩
⟨⟨ls2_LS2_postgraduate_programme_pkey, ls2_LS2_postgraduate_programme, ⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_PID⟩⟩⟩
⟨⟨ls2_LS2_postgraduate_programme_pkey, ls2_LS2_postgraduate_programme, ⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_StartingYear⟩⟩⟩
⟨⟨ls2_LS2_postgraduate_enrol_pkey, ls2_LS2_postgraduate_enrol, ⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_SID⟩⟩⟩
⟨⟨ls2_LS2_postgraduate_enrol_pkey, ls2_LS2_postgraduate_enrol, ⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_Year⟩⟩⟩
⟨⟨ls2_LS2_postgraduate_enrol_pkey, ls2_LS2_postgraduate_enrol, ⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_PID⟩⟩⟩
⟨⟨ls2_LS2_postgraduate_student_pkey, ls2_LS2_postgraduate_student, ⟨⟨ls2_LS2_postgraduate_student, ls2_LS2_SID⟩⟩⟩
⟨⟨ls2_LS2_study_pkey, ls2_LS2_study, ⟨⟨ls2_LS2_study, ls2_LS2_SID⟩⟩⟩
⟨⟨ls2_LS2_study_pkey, ls2_LS2_study, ⟨⟨ls2_LS2_study, ls2_LS2_CID⟩⟩⟩
⟨⟨ls2_LS2_study_pkey, ls2_LS2_study, ⟨⟨ls2_LS2_study, ls2_LS2_Year⟩⟩⟩
⟨⟨ls2_LS2_support_member_pkey, ls2_LS2_support_member, ⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩⟩
⟨⟨ls2_LS2_teacher_pkey, ls2_LS2_teacher, ⟨⟨ls2_LS2_teacher, ls2_LS2_LID⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_course_pkey, ls2_LS2_undergraduate_course, ⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_CID⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_course_pkey, ls2_LS2_undergraduate_course, ⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_Year⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_enrol_pkey, ls2_LS2_undergraduate_enrol, ⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_SID⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_enrol_pkey, ls2_LS2_undergraduate_enrol, ⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_PID⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_enrol_pkey, ls2_LS2_undergraduate_enrol, ⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_Year⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_programme_pkey, ls2_LS2_undergraduate_programme, ⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_PID⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_programme_pkey, ls2_LS2_undergraduate_programme, ⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_StartingYear⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_student_pkey, ls2_LS2_undergraduate_student, ⟨⟨ls2_LS2_undergraduate_student, ls2_LS2_SID⟩⟩⟩
⟨⟨ls3_LS3_assist_pkey, ls3_LS3_assist, ⟨⟨ls3_LS3_assist, ls3_LS3_TID⟩⟩⟩
⟨⟨ls3_LS3_assist_pkey, ls3_LS3_assist, ⟨⟨ls3_LS3_assist, ls3_LS3_CID⟩⟩⟩
⟨⟨ls3_LS3_educator_assistant_pkey, ls3_LS3_educator_assistant, ⟨⟨ls3_LS3_educator_assistant, ls3_LS3_TID⟩⟩⟩
⟨⟨ls3_LS3_induct_pkey, ls3_LS3_induct, ⟨⟨ls3_LS3_induct, ls3_LS3_TID⟩⟩⟩
⟨⟨ls3_LS3_induct_pkey, ls3_LS3_induct, ⟨⟨ls3_LS3_induct, ls3_LS3_CID⟩⟩⟩
⟨⟨ls3_LS3_lecturer_pkey, ls3_LS3_lecturer, ⟨⟨ls3_LS3_lecturer, ls3_LS3_TID⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_course_pkey, ls3_LS3_postgraduate_course, ⟨⟨ls3_LS3_postgraduate_course, ls3_LS3_CID⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_enrol_pkey, ls3_LS3_postgraduate_enrol, ⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_SID⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_enrol_pkey, ls3_LS3_postgraduate_enrol, ⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_PID⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_enrol_pkey, ls3_LS3_postgraduate_enrol, ⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_Year⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_programme_pkey, ls3_LS3_postgraduate_programme, ⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_PID⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_programme_pkey, ls3_LS3_postgraduate_programme, ⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_Year⟩⟩⟩
⟨⟨ls3_LS3_study_pkey, ls3_LS3_study, ⟨⟨ls3_LS3_study, ls3_LS3_SID⟩⟩⟩
⟨⟨ls3_LS3_study_pkey, ls3_LS3_study, ⟨⟨ls3_LS3_study, ls3_LS3_CID⟩⟩⟩
⟨⟨ls3_LS3_study_pkey, ls3_LS3_study, ⟨⟨ls3_LS3_study, ls3_LS3_Year⟩⟩⟩
⟨⟨ls3_LS3_support_member_pkey, ls3_LS3_support_member, ⟨⟨ls3_LS3_support_member, ls3_LS3_TSID⟩⟩⟩

Table B.8: Satisfying Elements for Quality Factor 3

not-Satisfying Elements
⟨⟨ls1_LS1_contain_pkey, ls1_LS1_contain, ⟨⟨ls1_LS1_contain, ls1_LS1_PID⟩⟩⟩
⟨⟨ls1_LS1_contain_pkey, ls1_LS1_contain, ⟨⟨ls1_LS1_contain, ls1_LS1_CID⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_pkey, ls3_LS3_postgraduate, ⟨⟨ls3_LS3_postgraduate, ls3_LS3_SID⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_pkey, ls3_LS3_postgraduate, ⟨⟨ls3_LS3_postgraduate, ls3_LS3_Year⟩⟩⟩
⟨⟨ls3_LS3_support_pkey, ls3_LS3_support, ⟨⟨ls3_LS3_support, ls3_LS3_TID⟩⟩⟩
⟨⟨ls3_LS3_support_pkey, ls3_LS3_support, ⟨⟨ls3_LS3_support, ls3_LS3_PID⟩⟩⟩

Table B.9: Not-Satisfying Elements for Quality Factor 3

Quality Factor 4

Concept	Referenced	Unreferenced	Result
Head	4	0	1.00
induct	2	3	0.40
Name	1	10	0.09
register	3	0	1.00
Student	1	2	0.33
study	2	0	1.00

Table B.10: Coverage of Concepts for Quality Factor 4

Concept	Referenced	Unreferenced
Head	⟨⟨ls1_LS1_programmels1_LS1_Programme_Director⟩⟩ ⟨⟨ls2_LS2_postgraduate_programmels2_LS2_ProgrammeHeadID⟩⟩ ⟨⟨ls2_LS2_undergraduate_programmels2_LS2_ProgrammeHeadID⟩⟩ ⟨⟨ls3_LS3_postgraduate_programmels3_LS3_ProgrammeHead⟩⟩	
induct	⟨⟨ls2_LS2_induct⟩⟩ ⟨⟨ls3_LS3_induct⟩⟩	⟨⟨ls2_LS2_assist⟩⟩ ⟨⟨ls3_LS3_assist⟩⟩ ⟨⟨ls3_LS3_support⟩⟩
Name	⟨⟨ls1_LS1_educatorls1_LS1_Name⟩⟩	⟨⟨ls2_LS2_postgraduate_studentls2_LS2_Name⟩⟩ ⟨⟨ls2_LS2_undergraduate_studentls2_LS2_Name⟩⟩ ⟨⟨ls3_LS3_educator_assistantls3_LS3_Firstname⟩⟩ ⟨⟨ls3_LS3_educator_assistantls3_LS3_Surname⟩⟩ ⟨⟨ls3_LS3_postgraduatels3_LS3_StudentName⟩⟩ ⟨⟨ls1_LS1_fulltime_faculty_memberls1_LS1_Name⟩⟩ ⟨⟨ls1_LS1_mandatory_coursels1_LS1_CourseName⟩⟩ ⟨⟨ls1_LS1_optional_coursels1_LS1_CourseName⟩⟩ ⟨⟨ls1_LS1_parttime_faculty_memberls1_LS1_Name⟩⟩ ⟨⟨ls1_LS1_programmels1_LS1_Programme_Name⟩⟩
register	⟨⟨ls2_LS2_postgraduate_enrol⟩⟩ ⟨⟨ls2_LS2_undergraduate_enrol⟩⟩ ⟨⟨ls3_LS3_postgraduate_enrol⟩⟩	
Student	⟨⟨ls2_LS2_undergraduate_student⟩⟩	⟨⟨ls3_LS3_postgraduate⟩⟩ ⟨⟨ls2_LS2_postgraduate_student⟩⟩
study	⟨⟨ls2_LS2_study⟩⟩ ⟨⟨ls3_LS3_study⟩⟩	

Table B.11: Satisfying and not-Satisfying Elements for Quality Factor 4

Quality Factor 5

Satisfying Elements	
⟨⟨ls1_LS1_educator, ls1_LS1_Name⟩⟩	⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩
⟨⟨ls1_LS1_ft_academic_assistant, ls1_LS1_TID⟩⟩	⟨⟨ls2_LS2_study, ls2_LS2_SID⟩⟩
⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_Name⟩⟩	⟨⟨ls2_LS2_study⟩⟩
⟨⟨ls1_LS1_lecturer, ls1_LS1_TID⟩⟩	⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩
⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CourseName⟩⟩	⟨⟨ls2_LS2_teacher⟩⟩
⟨⟨ls1_LS1_mandatory_course, ls1_LS1_Level⟩⟩	⟨⟨ls2_LS2_undergraduate_course⟩⟩
⟨⟨ls1_LS1_optional_course, ls1_LS1_CourseName⟩⟩	⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_PID⟩⟩
⟨⟨ls1_LS1_optional_course, ls1_LS1_Level⟩⟩	⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_ProgrammeHeadID⟩⟩
⟨⟨ls1_LS1_parttime_faculty_member, ls1_LS1_Name⟩⟩	⟨⟨ls3_LS3_assist, ls3_LS3_TID⟩⟩
⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Description⟩⟩	⟨⟨ls3_LS3_educator_assistant, ls3_LS3_TID⟩⟩
⟨⟨ls1_LS1_programme, ls1_LS1_Programme_Name⟩⟩	⟨⟨ls3_LS3_induct, ls3_LS3_TID⟩⟩
⟨⟨ls1_LS1_programme⟩⟩	⟨⟨ls3_LS3_induct⟩⟩
⟨⟨ls1_LS1_pt_academic_assistant, ls1_LS1_TID⟩⟩	⟨⟨ls3_LS3_lecturer⟩⟩
⟨⟨ls1_LS1_teachingsupport, ls1_LS1_TID⟩⟩	⟨⟨ls3_LS3_postgraduate_enrol⟩⟩
⟨⟨ls2_LS2_assist, ls2_LS2_TID⟩⟩	⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_PID⟩⟩
⟨⟨ls2_LS2_induct, ls2_LS2_CID⟩⟩	⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_ProgrammeHead⟩⟩
⟨⟨ls2_LS2_induct⟩⟩	⟨⟨ls3_LS3_study, ls3_LS3_SID⟩⟩
⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_PID⟩⟩	⟨⟨ls3_LS3_support_member, ls3_LS3_TSID⟩⟩

Table B.12: Satisfying Elements for Quality Factor 5

Quality Factor 6

Satisfying Elements	
⟨⟨ls1_LS1_contain, ls1_LS1_CID⟩⟩	⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_Year⟩⟩
⟨⟨ls1_LS1_contain, ls1_LS1_PID⟩⟩	⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_PID⟩⟩
⟨⟨ls1_LS1_educator, ls1_LS1_TID⟩⟩	⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_SID⟩⟩
⟨⟨ls1_LS1_ft_academic_assistant, ls1_LS1_TID⟩⟩	⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_Year⟩⟩
⟨⟨ls1_LS1_fulltime_faculty_member, ls1_LS1_TID⟩⟩	⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_PID⟩⟩
⟨⟨ls1_LS1_lecturer, ls1_LS1_TID⟩⟩	⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_StartingYear⟩⟩
⟨⟨ls1_LS1_mandatory_course, ls1_LS1_CID⟩⟩	⟨⟨ls2_LS2_undergraduate_student, ls2_LS2_SID⟩⟩
⟨⟨ls1_LS1_optional_course, ls1_LS1_CID⟩⟩	⟨⟨ls3_LS3_assist, ls3_LS3_CID⟩⟩
⟨⟨ls1_LS1_parttime_faculty_member, ls1_LS1_TID⟩⟩	⟨⟨ls3_LS3_assist, ls3_LS3_TID⟩⟩
⟨⟨ls1_LS1_programme, ls1_LS1_PID⟩⟩	⟨⟨ls3_LS3_educator_assistant, ls3_LS3_TID⟩⟩
⟨⟨ls1_LS1_pt_academic_assistant, ls1_LS1_TID⟩⟩	⟨⟨ls3_LS3_induct, ls3_LS3_CID⟩⟩
⟨⟨ls1_LS1_teachingsupport, ls1_LS1_TID⟩⟩	⟨⟨ls3_LS3_induct, ls3_LS3_TID⟩⟩
⟨⟨ls2_LS2_assist, ls2_LS2_CID⟩⟩	⟨⟨ls3_LS3_lecturer, ls3_LS3_TID⟩⟩
⟨⟨ls2_LS2_assist, ls2_LS2_TID⟩⟩	⟨⟨ls3_LS3_postgraduate⟩⟩
⟨⟨ls2_LS2_induct, ls2_LS2_CID⟩⟩	⟨⟨ls3_LS3_postgraduate, ls3_LS3_SID⟩⟩
⟨⟨ls2_LS2_induct, ls2_LS2_LID⟩⟩	⟨⟨ls3_LS3_postgraduate, ls3_LS3_Year⟩⟩
⟨⟨ls2_LS2_induct, ls2_LS2_Year⟩⟩	⟨⟨ls3_LS3_postgraduate_course, ls3_LS3_CID⟩⟩
⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_PID⟩⟩	⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_PID⟩⟩
⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_SID⟩⟩	⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_SID⟩⟩
⟨⟨ls2_LS2_postgraduate_enrol, ls2_LS2_Year⟩⟩	⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_Year⟩⟩
⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_PID⟩⟩	⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_PID⟩⟩
⟨⟨ls2_LS2_postgraduate_programme, ls2_LS2_StartingYear⟩⟩	⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_Year⟩⟩
⟨⟨ls2_LS2_postgraduate_student, ls2_LS2_SID⟩⟩	⟨⟨ls3_LS3_study, ls3_LS3_CID⟩⟩
⟨⟨ls2_LS2_study, ls2_LS2_CID⟩⟩	⟨⟨ls3_LS3_study, ls3_LS3_SID⟩⟩
⟨⟨ls2_LS2_study, ls2_LS2_SID⟩⟩	⟨⟨ls3_LS3_study, ls3_LS3_Year⟩⟩
⟨⟨ls2_LS2_study, ls2_LS2_Year⟩⟩	⟨⟨ls3_LS3_support, ls3_LS3_PID⟩⟩
⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩	⟨⟨ls3_LS3_support, ls3_LS3_TID⟩⟩
⟨⟨ls2_LS2_teacher, ls2_LS2_LID⟩⟩	⟨⟨ls3_LS3_support_member, ls3_LS3_TSID⟩⟩
⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_CID⟩⟩	

Table B.13: Satisfying Elements for Quality Factor 6

Quality Factor 7

Satisfying Elements
⟨⟨ls2_LS2_support_member_pkey, ls2_LS2_support_member, ⟨⟨ls2_LS2_support_member, ls2_LS2_TID⟩⟩⟩⟩
⟨⟨ls2_LS2_teacher_pkey, ls2_LS2_teacher, ⟨⟨ls2_LS2_teacher, ls2_LS2_LID⟩⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_course_pkey, ls2_LS2_undergraduate_course, ⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_CID⟩⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_course_pkey, ls2_LS2_undergraduate_course, ⟨⟨ls2_LS2_undergraduate_course, ls2_LS2_Year⟩⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_enrol_pkey, ls2_LS2_undergraduate_enrol, ⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_SID⟩⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_enrol_pkey, ls2_LS2_undergraduate_enrol, ⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_PID⟩⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_enrol_pkey, ls2_LS2_undergraduate_enrol, ⟨⟨ls2_LS2_undergraduate_enrol, ls2_LS2_Year⟩⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_programme_pkey, ls2_LS2_undergraduate_programme, ⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_PID⟩⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_programme_pkey, ls2_LS2_undergraduate_programme, ⟨⟨ls2_LS2_undergraduate_programme, ls2_LS2_StartingYear⟩⟩⟩⟩
⟨⟨ls2_LS2_undergraduate_student_pkey, ls2_LS2_undergraduate_student, ⟨⟨ls2_LS2_undergraduate_student, ls2_LS2_SID⟩⟩⟩⟩
⟨⟨ls3_LS3_assist_pkey, ls3_LS3_assist, ⟨⟨ls3_LS3_assist, ls3_LS3_TID⟩⟩⟩⟩
⟨⟨ls3_LS3_assist_pkey, ls3_LS3_assist, ⟨⟨ls3_LS3_assist, ls3_LS3_CID⟩⟩⟩⟩
⟨⟨ls3_LS3_educator_assistant_pkey, ls3_LS3_educator_assistant, ⟨⟨ls3_LS3_educator_assistant, ls3_LS3_TID⟩⟩⟩⟩
⟨⟨ls3_LS3_induct_pkey, ls3_LS3_induct, ⟨⟨ls3_LS3_induct, ls3_LS3_TID⟩⟩⟩⟩
⟨⟨ls3_LS3_induct_pkey, ls3_LS3_induct, ⟨⟨ls3_LS3_induct, ls3_LS3_CID⟩⟩⟩⟩
⟨⟨ls3_LS3_lecturer_pkey, ls3_LS3_lecturer, ⟨⟨ls3_LS3_lecturer, ls3_LS3_TID⟩⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_course_pkey, ls3_LS3_postgraduate_course, ⟨⟨ls3_LS3_postgraduate_course, ls3_LS3_CID⟩⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_enrol_pkey, ls3_LS3_postgraduate_enrol, ⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_SID⟩⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_enrol_pkey, ls3_LS3_postgraduate_enrol, ⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_PID⟩⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_enrol_pkey, ls3_LS3_postgraduate_enrol, ⟨⟨ls3_LS3_postgraduate_enrol, ls3_LS3_Year⟩⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_programme_pkey, ls3_LS3_postgraduate_programme, ⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_PID⟩⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_programme_pkey, ls3_LS3_postgraduate_programme, ⟨⟨ls3_LS3_postgraduate_programme, ls3_LS3_Year⟩⟩⟩⟩
⟨⟨ls3_LS3_study_pkey, ls3_LS3_study, ⟨⟨ls3_LS3_study, ls3_LS3_SID⟩⟩⟩⟩
⟨⟨ls3_LS3_study_pkey, ls3_LS3_study, ⟨⟨ls3_LS3_study, ls3_LS3_CID⟩⟩⟩⟩
⟨⟨ls3_LS3_study_pkey, ls3_LS3_study, ⟨⟨ls3_LS3_study, ls3_LS3_Year⟩⟩⟩⟩
⟨⟨ls3_LS3_support_member_pkey, ls3_LS3_support_member, ⟨⟨ls3_LS3_support_member, ls3_LS3_TSID⟩⟩⟩⟩
⟨⟨ls1_LS1_contain_pkey, ls1_LS1_contain, ⟨⟨ls1_LS1_contain, ls1_LS1_PID⟩⟩⟩⟩
⟨⟨ls1_LS1_contain_pkey, ls1_LS1_contain, ⟨⟨ls1_LS1_contain, ls1_LS1_CID⟩⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_pkey, ls3_LS3_postgraduate, ⟨⟨ls3_LS3_postgraduate, ls3_LS3_SID⟩⟩⟩⟩
⟨⟨ls3_LS3_postgraduate_pkey, ls3_LS3_postgraduate, ⟨⟨ls3_LS3_postgraduate, ls3_LS3_Year⟩⟩⟩⟩
⟨⟨ls3_LS3_support_pkey, ls3_LS3_support, ⟨⟨ls3_LS3_support, ls3_LS3_TID⟩⟩⟩⟩
⟨⟨ls3_LS3_support_pkey, ls3_LS3_support, ⟨⟨ls3_LS3_support, ls3_LS3_PID⟩⟩⟩⟩

Table B.14: Satisfying Elements for Quality Factor 7

Quality Factor 8

Satisfied Data Item
505,506,507,508

Table B.15: Satisfying Elements for Quality Factor 8

Appendix C

Schema to Ontology Representation Transformation Algorithm

This appendix lists the algorithm that transforms a relational schema to its corresponding ontological representation. This algorithm takes as input a set of relational schema constructs in the HDM representation and outputs the corresponding ontology constructs also in their HDM representation. This algorithm has three parts. The first part (lines 55–61), translates the relations of schema S_{Rel}. In particular, a relation $\langle\langle \mathsf{R} \rangle\rangle$ translates to a Class C in the output ontology, S_{Ont}, each of its attributes $\langle\langle \mathsf{R}, \mathsf{a} \rangle\rangle$ translates to a Property $\langle\langle \mathsf{a}, \mathsf{C}, \mathsf{rdfs:Literal} \rangle\rangle$, while the primary key of $\langle\langle \mathsf{R} \rangle\rangle$ translates into another Class $\langle\langle \mathsf{C_{pk}} \rangle\rangle$ and a Property $\langle\langle \mathsf{pk}, \mathsf{C}, \mathsf{C_{pk}} \rangle\rangle$. The second part (lines 62–70), translates the foreign key constraints of schema S_{Rel}. In particular, the algorithm creates two Class constructs, $\langle\langle \mathsf{C_{R_{fk}}} \rangle\rangle$ and $\langle\langle \mathsf{C_{S_{fk}}} \rangle\rangle$, representing the set of attributes of relation R and the set of attributes of relation S that reference the former. The algorithm also creates Property constructs $\langle\langle \mathsf{fk}, \mathsf{C_R}, \mathsf{C_{R_{fk}}} \rangle\rangle$, $\langle\langle \mathsf{fk}, \mathsf{C_S}, \mathsf{C_{S_{fk}}} \rangle\rangle$ and $\langle\langle \mathsf{fk}, \mathsf{C_{S_{fk}}}, \mathsf{C_{R_{fk}}} \rangle\rangle$ that link the newly added

Class constructs together with each other and with the Class constructs that represent relations R and S. The third part (line 71) removes the relational schema constructs from schema S_{Ont}.

Panel 4: Relational-to-OWL Translation

Input: AutoMed Relational Schema S_{Rel}

Output: AutoMed OWL Schema S_{Ont}

54 Copy S_{Rel} to S_{Ont}

55 Add class $\langle\langle \mathsf{rdfs:Literal} \rangle\rangle$ to S_{Ont}

56 **for** each relation R in S_{Rel} **do**

57 Add class $\langle\langle \mathsf{C} \rangle\rangle$ to S_{Ont} and populate its extent using query $[\mathsf{getLSID}\ \langle\langle \mathsf{R} \rangle\rangle\ \mathsf{r} | \mathsf{r} \leftarrow \langle\langle \mathsf{R} \rangle\rangle]$

58 **for** each attribute a of R **do**

59 Add property $\langle\langle \mathsf{a}, \mathsf{C}, \mathsf{rdfs:Literal} \rangle\rangle$ to S_{Ont} and populate its extent using query $[\{(\mathsf{getLSID}\ \langle\langle \mathsf{R} \rangle\rangle\ \mathsf{r}), \mathsf{a}\} | \{\mathsf{r}, \mathsf{a}\} \leftarrow \langle\langle \mathsf{R}, \mathsf{a} \rangle\rangle]$

60 Add class $\langle\langle \mathsf{C_{pk}} \rangle\rangle$ to S_{Ont} and populate its extent using query $[\mathsf{getLSID}\ \langle\langle \mathsf{R} \rangle\rangle\ \mathsf{r} | \mathsf{r} \leftarrow \langle\langle \mathsf{R} \rangle\rangle]$

61 Add property $\langle\langle \mathsf{pk}, \mathsf{C}, \mathsf{C_{pk}} \rangle\rangle$ and populate its extent using query $[\{(\mathsf{getLSID}\ \langle\langle \mathsf{R} \rangle\rangle\ \mathsf{r}), (\mathsf{getLSID}\ \langle\langle \mathsf{R} \rangle\rangle\ \mathsf{r})\} | \mathsf{r} \leftarrow \langle\langle \mathsf{R} \rangle\rangle]$

62 **for** each relation R in S_{Rel} **do**

63 **for** each foreign key with label fk identifying attributes a_i of R being referenced by attributes b_i of S $(1 \leq i \leq n)$ **do**

64 Let Q_1 be $[\{\mathsf{r}, \{\mathsf{a_1}, \ldots, \mathsf{a_i}, \ldots, \mathsf{a_n}\}\} | \mathsf{r} \leftarrow \langle\langle \mathsf{R} \rangle\rangle; \{\mathsf{r}, \mathsf{a_1}\} \leftarrow \langle\langle \mathsf{R}, \mathsf{a_1} \rangle\rangle; \ldots; \{\mathsf{r}, \mathsf{a_i}\} \leftarrow \langle\langle \mathsf{R}, \mathsf{a_i} \rangle\rangle \ldots; \{\mathsf{r}, \mathsf{a_n}\} \leftarrow \langle\langle \mathsf{R}, \mathsf{a_n} \rangle\rangle]$

65 Let Q_2 be $[\{\mathsf{s}, \{\mathsf{b_1}, \ldots, \mathsf{b_i}, \ldots, \mathsf{b_n}\}\} | \mathsf{s} \leftarrow \langle\langle \mathsf{S} \rangle\rangle; \{\mathsf{s}, \mathsf{b_1}\} \leftarrow \langle\langle \mathsf{S}, \mathsf{b_1} \rangle\rangle; \ldots; \{\mathsf{s}, \mathsf{b_i}\} \leftarrow \langle\langle \mathsf{S}, \mathsf{b_i} \rangle\rangle \ldots; \{\mathsf{s}, \mathsf{b_n}\} \leftarrow \langle\langle \mathsf{S}, \mathsf{b_n} \rangle\rangle]$

66 Add class $\langle\langle \mathsf{C_{R_{fk}}} \rangle\rangle$ to S_{Ont} and populate its extent using query $[(\mathsf{getLSID}\ \langle\langle \mathsf{R} \rangle\rangle\ \mathsf{cr}) | \{\mathsf{r}, \mathsf{cr}\} \leftarrow \mathsf{Q_1}]$

67 Add class $\langle\langle \mathsf{C_{S_{fk}}} \rangle\rangle$ to S_{Ont} and populate its extent using query $[(\mathsf{getLSID}\ \langle\langle \mathsf{S} \rangle\rangle\ \mathsf{cs}) | \{\mathsf{s}, \mathsf{cs}\} \leftarrow \mathsf{Q_2}]$

68 Add property $\langle\langle \mathsf{fk}, \mathsf{C_R}, \mathsf{C_{R_{fk}}} \rangle\rangle$ to S_{Ont} and populate its extent using query Q_1

69 Add property $\langle\langle \mathsf{fk}, \mathsf{C_S}, \mathsf{C_{S_{fk}}} \rangle\rangle$ to S_{Ont} and populate its extent using query Q_2

70 Add property $\langle\langle \mathsf{fk}, \mathsf{C_{S_{fk}}}, \mathsf{C_{R_{fk}}} \rangle\rangle$ to S_{Ont} and populate its extent using query $[\{(\mathsf{getLSID}\ \langle\langle \mathsf{R} \rangle\rangle\ \mathsf{cs}), (\mathsf{getLSID}\ \langle\langle \mathsf{S} \rangle\rangle\ \mathsf{cs})\} | \{\mathsf{s}, \mathsf{cs}\} \leftarrow \mathsf{Q_2}]$

71 deleteRelationalConstructs(S_{Ont})

Appendix D

QFDI in OWL-DL

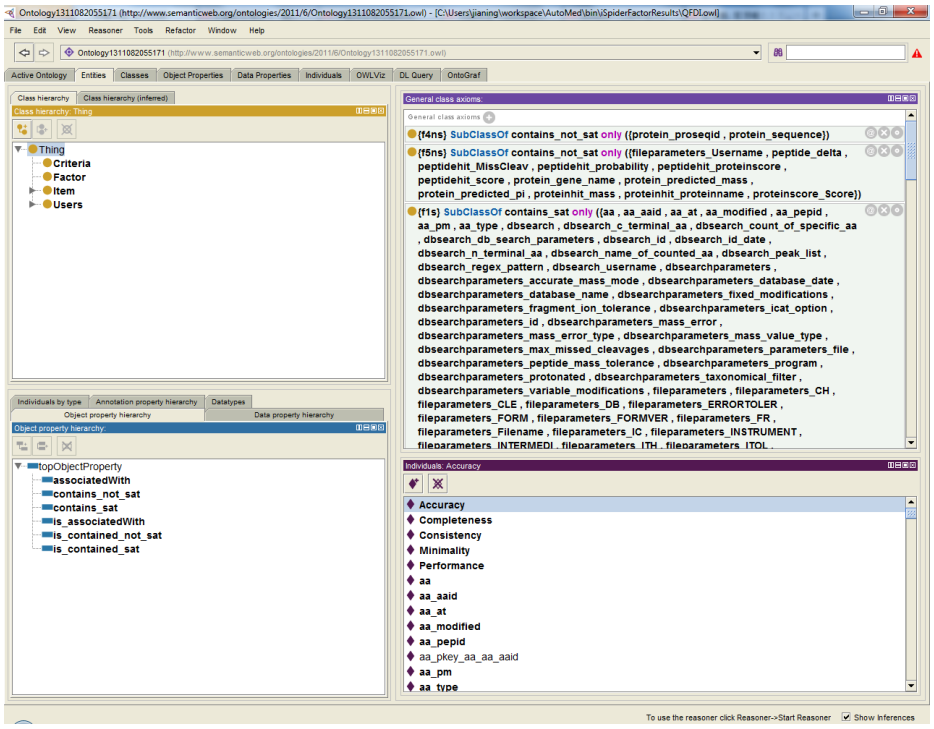


Figure D.1: QFDI in OWL-DL

Appendix E

iSpider Experimentation and Evaluation

This appendix contains data relating to the iSpider project that is used for evaluating our approach in Chapter 7. Section E.1 illustrates the schemas of three data sources: PEDRo, gpmDB and PepSeeker. Section E.2 lists all the mappings generated in three DI iterations. Section E.3 illustrates the concept coverage of the integrated resource in three DI iterations. Section E.4 lists the DI elements that do and do not satisfy Quality Factor 1, 4 and 7 for each DI iteration.

E.1 iSpider Schemas

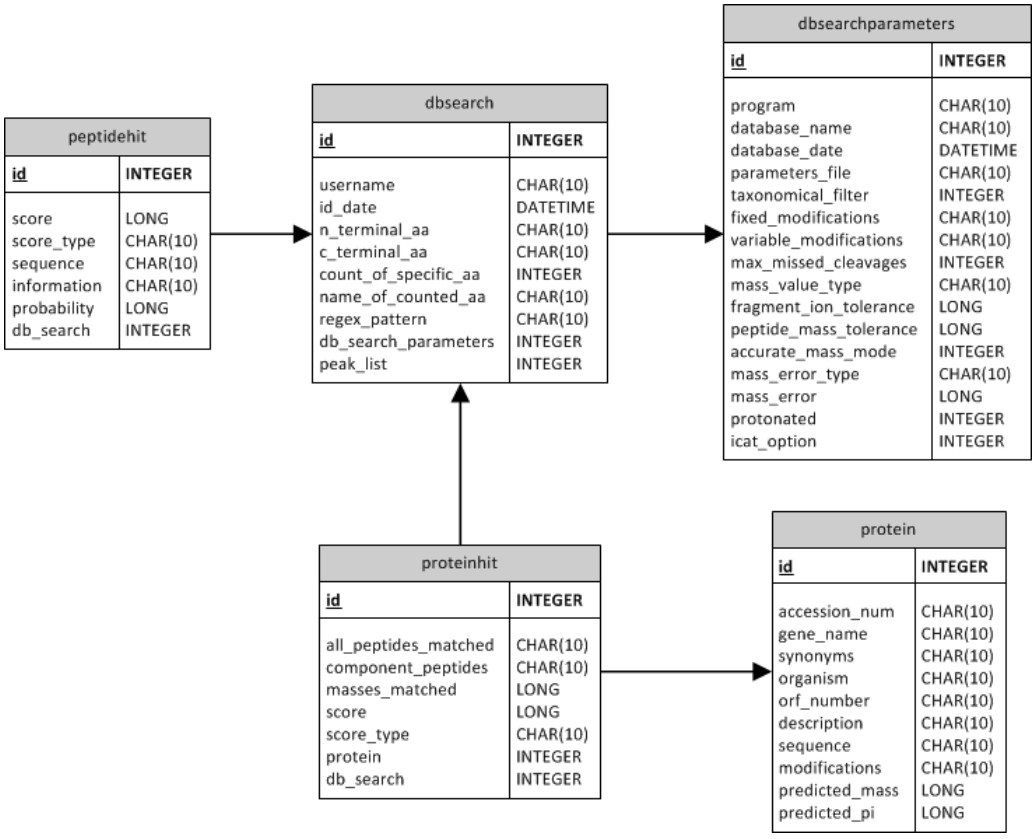


Figure E.1: Data Source PEDRo

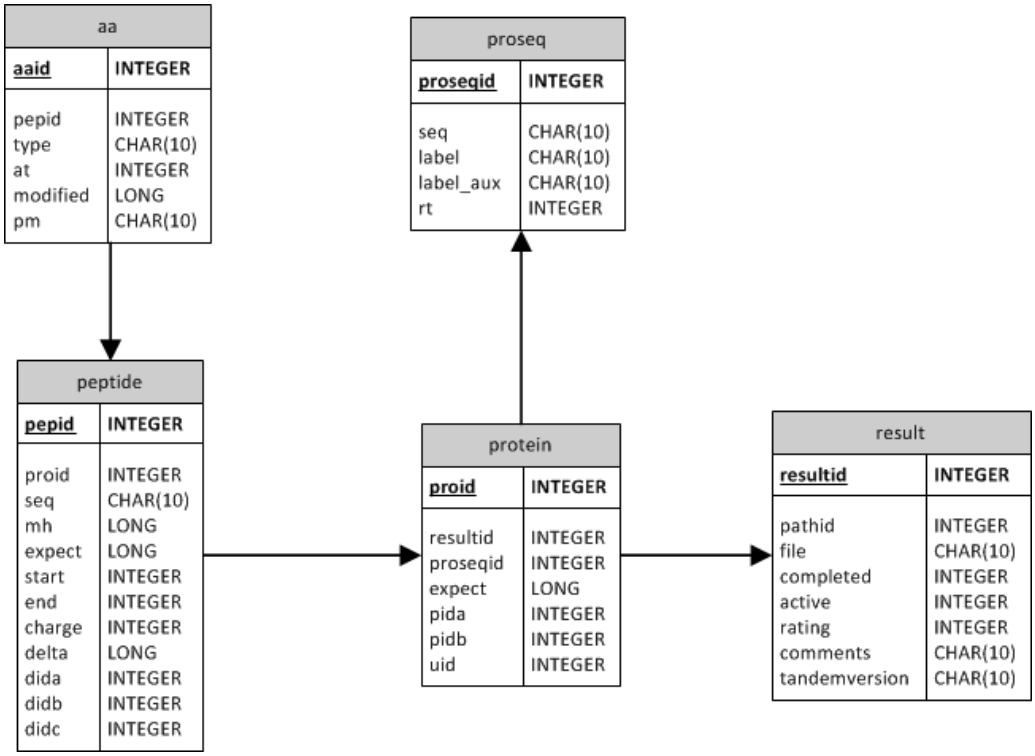


Figure E.2: Data Source gpmDB

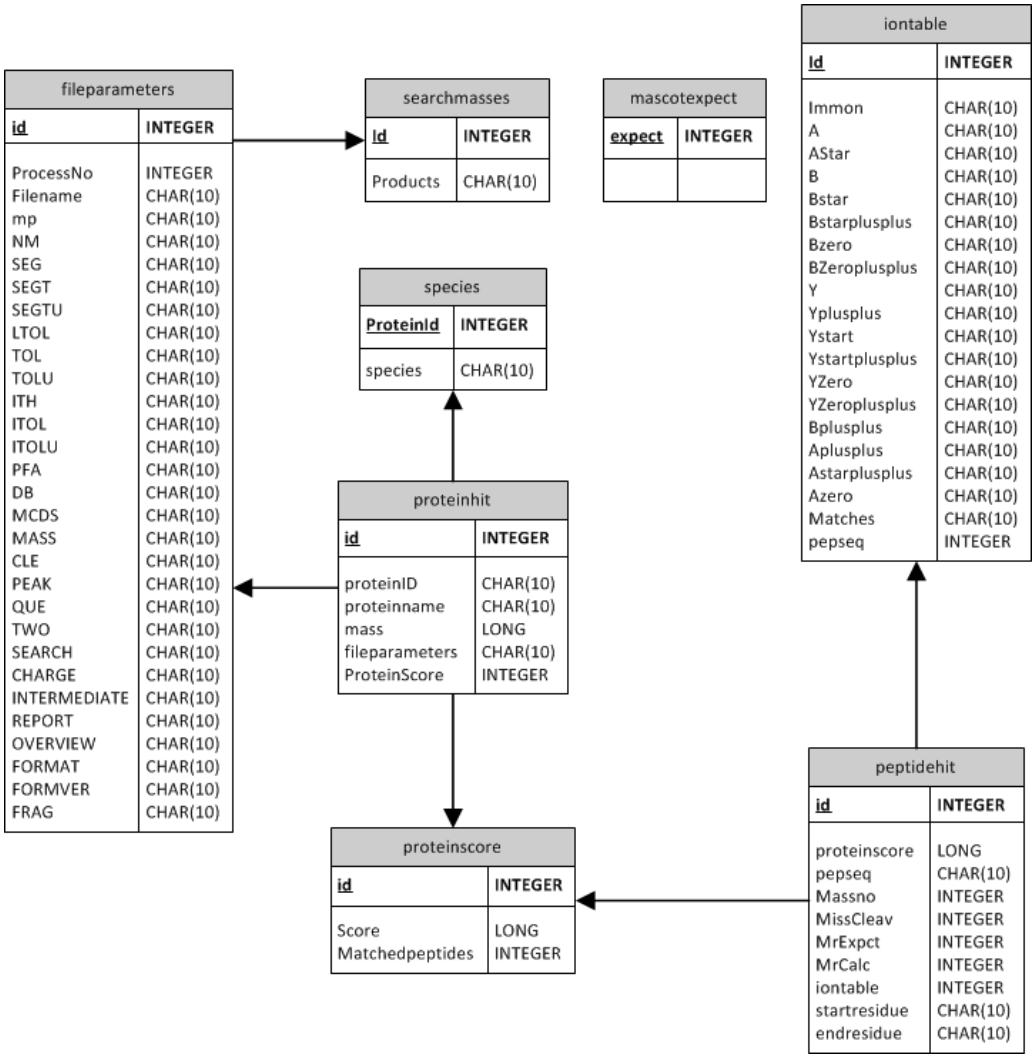


Figure E.3: Data Source PepSeeker

E.2 Mappings

AutoMed transformation pathways for the three iterations are listed in Tables E.1, E.2 and E.3 below.

gpmDB → *GS1*

extend	⟨⟨GS1_dbsearch⟩⟩	[{'URN : LSID : ispider.man.ac.uk : gpmdb : result', d}|d ← ⟨⟨gpmdb_result⟩⟩]
extend	⟨⟨GS1_dbsearch, GS1_db_search_parameters⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : result', d}, {'URN : LSID : ispider.man.ac.uk : gpmdb : result', d}}|d ← ⟨⟨gpmdb_result⟩⟩]
extend	⟨⟨GS1_dbsearch, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : result', d}, {'URN : LSID : ispider.man.ac.uk : gpmdb : result', d}}|d ← ⟨⟨gpmdb_result⟩⟩]
extend	⟨⟨GS1_dbsearchparameters⟩⟩	[{'URN : LSID : ispider.man.ac.uk : gpmdb : result', d}|d ← ⟨⟨gpmdb_result⟩⟩]
extend	⟨⟨GS1_dbsearchparameters, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : result', d}, {'URN : LSID : ispider.man.ac.uk : gpmdb : result', d}}|d ← ⟨⟨gpmdb_result⟩⟩]
extend	⟨⟨GS1_dbsearchparameters, GS1_program⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : result', d}, tv}|{d, tv} ← ⟨⟨gpmdb_result, gpmdb_tandemversion⟩⟩]
add	⟨⟨GS1_peptidehit⟩⟩	[{'URN : LSID : ispider.man.ac.uk : gpmdb : peptide', d}|d ← ⟨⟨gpmdb_peptide⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : peptide', d}, {'URN : LSID : ispider.man.ac.uk : gpmdb : peptide', d}}|d ← ⟨⟨gpmdb_peptide⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_probability⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : peptide', d}, x}|{d, x} ← ⟨⟨gpmdb_peptide, gpmdb_expect⟩⟩]
extend	⟨⟨GS1_peptidehit, GS1_score⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : peptide', d}, 'Null'}|d ← ⟨⟨gpmdb_peptide⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_sequence⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : peptide', d}, x}|{d, x} ← ⟨⟨gpmdb_peptide, gpmdb_seq⟩⟩]
add	⟨⟨GS1_protein⟩⟩	[{'URN : LSID : ispider.man.ac.uk : gpmdb : protein', d}|d ← ⟨⟨gpmdb_protein⟩⟩]
add	⟨⟨GS1_protein, GS1_accession_num⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : protein', pid}, x}|{pid, proseqid} ← ⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩; {proseqid, x} ← ⟨⟨gpmdb_proseq, gpmdb_label⟩⟩]
add	⟨⟨GS1_protein, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : protein', d}, {'URN : LSID : ispider.man.ac.uk : gpmdb : protein', d}}|d ← ⟨⟨gpmdb_protein⟩⟩]
add	⟨⟨GS1_protein, GS1_sequence⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : protein', pid}, x}|{pid, proseqid} ← ⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩; {proseqid, x} ← ⟨⟨gpmdb_proseq, gpmdb_seq⟩⟩]
add	⟨⟨GS1_proteinhit⟩⟩	[{'URN : LSID : ispider.man.ac.uk : gpmdb : protein', d}|d ← ⟨⟨gpmdb_protein⟩⟩]
extend	⟨⟨GS1_proteinhit, GS1_db_search⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : protein', pid}, {'URN : LSID : ispider.man.ac.uk : gpmdb : result', rid}}| {pid, rid} ← ⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : protein', d}, {'URN : LSID : ispider.man.ac.uk : gpmdb : protein', d}}|d ← ⟨⟨gpmdb_protein⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_protein⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : protein', d}, {'URN : LSID : ispider.man.ac.uk : gpmdb : proseqid', x}}| {d, x} ← ⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩]

PEDRo → *GS1*

add	⟨⟨GS1_dbsearch⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}|d ← ⟨⟨pedro_dbsearch⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_c_terminal_aa⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}, x}|{d, x} ← ⟨⟨pedro_dbsearch, pedro_c_terminal_aa⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_count_of_specific_aa⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}, x}|{d, x} ← ⟨⟨pedro_dbsearch, pedro_count_of_specific_aa⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_db_search_parameters⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}, x}|{d, x} ← ⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_id_date⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}, x}|{d, x} ← ⟨⟨pedro_dbsearch, pedro_id_date⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}, x}|{d, x} ← ⟨⟨pedro_dbsearch, pedro_id⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_n_terminal_aa⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}, x}|{d, x} ← ⟨⟨pedro_dbsearch, pedro_n_terminal_aa⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_name_of_counted_aa⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}, x}|{d, x} ← ⟨⟨pedro_dbsearch, pedro_name_of_counted_aa⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_peak_list⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}, x}|{d, x} ← ⟨⟨pedro_dbsearch, pedro_peak_list⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_regex_pattern⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}, x}|{d, x} ← ⟨⟨pedro_dbsearch, pedro_regex_pattern⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_username⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearch', d}, x}|{d, x} ← ⟨⟨pedro_dbsearch, pedro_username⟩⟩]
add	⟨⟨GS1_dbsearchparameters⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}|d ← ⟨⟨pedro_dbsearchparameters⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_accurate_mass_mode⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_accurate_mass_mode⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_database_date⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_database_date⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_database_name⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_database_name⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_fixed_modifications⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_fixed_modifications⟩⟩]

add	⟨⟨GS1_dbsearchparameters, GS1_fragment_ion_tolerance⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_fragment_ion_tolerance⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_icat_option⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_icat_option⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_mass_error_type⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_mass_error_type⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_mass_error⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_mass_error⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_mass_value_type⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_mass_value_type⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_max_missed_cleavages⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_max_missed_cleavages⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_parameters_file⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_parameters_file⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_peptide_mass_tolerance⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_peptide_mass_tolerance⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_program⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_program⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_protonated⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_protonated⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_taxonomical_filter⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_taxonomical_filter⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_variable_modifications⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : dbsearchparameters', d}, x}|{d, x} ← ⟨⟨pedro_dbsearchparameters, pedro_variable_modifications⟩⟩]
add	⟨⟨GS1_peak⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pedro : peak', d}|d ← ⟨⟨pedro_peak⟩⟩]
add	⟨⟨GS1_peak, GS1_abundance⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peak', d}, x}|{d, x} ← ⟨⟨pedro_peak, pedro_abundance⟩⟩]
add	⟨⟨GS1_peak, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peak', d}, x}|{d, x} ← ⟨⟨pedro_peak, pedro_id⟩⟩]
add	⟨⟨GS1_peak, GS1_m_to_z⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peak', d}, x}|{d, x} ← ⟨⟨pedro_peak, pedro_m_to_z⟩⟩]
add	⟨⟨GS1_peak, GS1_multiplicity⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peak', d}, x}|{d, x} ← ⟨⟨pedro_peak, pedro_multiplicity⟩⟩]
add	⟨⟨GS1_peak, GS1_peak_list⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peak', d}, x}|{d, x} ← ⟨⟨pedro_peak, pedro_peak_list⟩⟩]
add	⟨⟨GS1_peptidehit⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pedro : peptidehit', d}|d ← ⟨⟨pedro_peptidehit⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_db_search⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peptidehit', d}, x}|{d, x} ← ⟨⟨pedro_peptidehit, pedro_db_search⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peptidehit', d}, x}|{d, x} ← ⟨⟨pedro_peptidehit, pedro_id⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_information⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peptidehit', d}, x}|{d, x} ← ⟨⟨pedro_peptidehit, pedro_information⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_probability⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peptidehit', d}, x}|{d, x} ← ⟨⟨pedro_peptidehit, pedro_probability⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_score_type⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peptidehit', d}, x}|{d, x} ← ⟨⟨pedro_peptidehit, pedro_score_type⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_score⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peptidehit', d}, x}|{d, x} ← ⟨⟨pedro_peptidehit, pedro_score⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_sequence⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : peptidehit', d}, x}|{d, x} ← ⟨⟨pedro_peptidehit, pedro_sequence⟩⟩]
add	⟨⟨GS1_protein⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}|d ← ⟨⟨pedro_protein⟩⟩]
add	⟨⟨GS1_protein, GS1_accession_num⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_accession_num⟩⟩]
add	⟨⟨GS1_protein, GS1_description⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_description⟩⟩]
add	⟨⟨GS1_protein, GS1_gene_name⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_gene_name⟩⟩]
add	⟨⟨GS1_protein, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_id⟩⟩]
add	⟨⟨GS1_protein, GS1_modifications⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_modifications⟩⟩]
add	⟨⟨GS1_protein, GS1_orf_number⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_orf_number⟩⟩]
add	⟨⟨GS1_protein, GS1_organism⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_organism⟩⟩]
add	⟨⟨GS1_protein, GS1_predicted_mass⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_predicted_mass⟩⟩]
add	⟨⟨GS1_protein, GS1_predicted_pi⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_predicted_pi⟩⟩]
add	⟨⟨GS1_protein, GS1_sequence⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_sequence⟩⟩]
add	⟨⟨GS1_protein, GS1_synonyms⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : protein', d}, x}|{d, x} ← ⟨⟨pedro_protein, pedro_synonyms⟩⟩]
add	⟨⟨GS1_proteinhit⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pedro : proteinhit', d}|d ← ⟨⟨pedro_proteinhit⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_all_peptides_matched⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : proteinhit', d}, x}|{d, x} ← ⟨⟨pedro_proteinhit, pedro_all_peptides_matched⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_component_peptides⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : proteinhit', d}, x}|{d, x} ← ⟨⟨pedro_proteinhit, pedro_component_peptides⟩⟩]

add	⟨⟨GS1_proteinhit, GS1_db_search⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : proteinhit', d}, x}|{d, x} ← ⟨⟨pedro_proteinhit, pedro_db_search⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : proteinhit', d}, x}|{d, x} ← ⟨⟨pedro_proteinhit, pedro_id⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_masses_matched⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : proteinhit', d}, x}|{d, x} ← ⟨⟨pedro_proteinhit, pedro_masses_matched⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_protein⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : proteinhit', d}, x}|{d, x} ← ⟨⟨pedro_proteinhit, pedro_protein⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_score_type⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : proteinhit', d}, x}|{d, x} ← ⟨⟨pedro_proteinhit, pedro_score_type⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_score⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pedro : proteinhit', d}, x}|{d, x} ← ⟨⟨pedro_proteinhit, pedro_score⟩⟩]
	Pepseeker → *GS1*	
add	⟨⟨GS1_dbsearch⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}|d ← ⟨⟨pepseeker_fileparameters⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_db_search_parameters⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, {'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', x}}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_Id⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, {'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}}|d ← ⟨⟨pepseeker_fileparameters⟩⟩]
add	⟨⟨GS1_dbsearch, GS1_username⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_Username⟩⟩]
add	⟨⟨GS1_dbsearchparameters⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}|d ← ⟨⟨pepseeker_fileparameters⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_accurate_mass_mode⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_SEG⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_fixed_modifications⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_MODS⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_fragment_ion_tolerance⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_ITOL⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, {'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}}|d ← ⟨⟨pepseeker_fileparameters⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_max_missed_cleavages⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_PFA⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_parameters_file⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_Filename⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_peptide_mass_tolerance⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_TOL⟩⟩]
extend	⟨⟨GS1_dbsearchparameters, GS1_program⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, 'Mascot'}|d ← ⟨⟨pepseeker_fileparameters⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_protonated⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_CHARGE⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_taxonomical_filter⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_TAXONOMY⟩⟩]
add	⟨⟨GS1_dbsearchparameters, GS1_variable_modifications⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_IT_MODS⟩⟩]
add	⟨⟨GS1_peptidehit⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit', d}|d ← ⟨⟨pepseeker_peptidehit⟩⟩]
extend	⟨⟨GS1_peptidehit, GS1_db_search⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit', d}, 'Null'}|d ← ⟨⟨pepseeker_peptidehit⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker', k}, {'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit', k}}|k ← ⟨⟨pepseeker_peptidehit⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_probability⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit', d}, x}|{d, x} ← ⟨⟨pepseeker_mascotexpect, pepseeker_expect⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_score⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit', d}, x}|{d, x} ← ⟨⟨pepseeker_peptidehit, pepseeker_proteinscore⟩⟩]
add	⟨⟨GS1_peptidehit, GS1_sequence⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit', d}, x}|{d, x} ← ⟨⟨pepseeker_peptidehit, pepseeker_pepseq⟩⟩]
add	⟨⟨GS1_protein⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', d}|d ← ⟨⟨pepseeker_proteinhit⟩⟩]
add	⟨⟨GS1_protein, GS1_accession_num⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', pid}, x}|{pid, x} ← ⟨⟨pepseeker_proteinhit⟩⟩]
add	⟨⟨GS1_protein, GS1_gene_name⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', pid}, x}|{pid, x} ← ⟨⟨pepseeker_proteinhit, pepseeker_Proteinname⟩⟩]
add	⟨⟨GS1_protein, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', d}, {'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', d}}|d ← ⟨⟨pepseeker_proteinhit⟩⟩]
add	⟨⟨GS1_protein, GS1_organism⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', ph}, oname}|{s, ph} ← ⟨⟨pepseeker_species, pepseeker_ProteinId⟩⟩; {s, oname} ← ⟨⟨pepseeker_species, pepseeker_species⟩⟩]
add	⟨⟨GS1_protein, GS1_predicted_mass⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', d}, x}|{d, x} ← ⟨⟨pepseeker_proteinhit, pepseeker_Mass⟩⟩]

add	⟨⟨GS1_proteinhit⟩⟩	[{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', d}|d ← ⟨⟨pepseeker_proteinhit⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_db_search⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', d}, {'URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters', x}}|{d, x} ← ⟨⟨pepseeker_proteinhit, pepseeker_fileparameters⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', d}, {'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', d}}|d ← ⟨⟨pepseeker_proteinhit⟩⟩]
add	⟨⟨GS1_proteinhit, GS1_protein⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', d}, {'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', x}}|{d, x} ← ⟨⟨pepseeker_proteinhit, pepseeker_ProteinID⟩⟩]
extend	⟨⟨GS1_proteinhit, GS1_score⟩⟩	distinct [{{'URN : LSID : ispider.man.ac.uk : pepseeker : proteinhit', k2}, x}|{k2, x} ← ⟨⟨pepseeker_proteinscore, pepseeker_Score⟩⟩]

Table E.1: Mappings for Iteration 1

	gpmDB → *GS2*	
add	⟨⟨GS2_aa⟩⟩	[{'URN : LSID : ispider.man.ac.uk : gpmdb : aa′, d}|d ← ⟨⟨gpmdb_aa⟩⟩]
add	⟨⟨GS2_aa, GS2_aaid⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : aa′, d}, x}|{d, x} ← ⟨⟨gpmdb_aa, gpmdb_aaid⟩⟩]
add	⟨⟨GS2_aa, GS2_pepid⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : aa′, d}, x}|{d, x} ← ⟨⟨gpmdb_aa, gpmdb_pepid⟩⟩]
add	⟨⟨GS2_aa, GS2_type⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : aa′, d}, x}|{d, x} ← ⟨⟨gpmdb_aa, gpmdb_type⟩⟩]
add	⟨⟨GS2_aa, GS2_at⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : aa′, d}, x}|{d, x} ← ⟨⟨gpmdb_aa, gpmdb_at⟩⟩]
add	⟨⟨GS2_aa, GS2_modified⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : aa′, d}, x}|{d, x} ← ⟨⟨gpmdb_aa, gpmdb_modified⟩⟩]
add	⟨⟨GS2_aa, GS2_pm⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : aa′, d}, x}|{d, x} ← ⟨⟨gpmdb_aa, gpmdb_pm⟩⟩]
add	⟨⟨GS2_peptide, GS2_mh⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : peptide′, d}, x}|{d, x} ← ⟨⟨gpmdb_peptide, gpmdb_mh⟩⟩]
add	⟨⟨GS2_peptide, GS2_start⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : peptide′, d}, x}|{d, x} ← ⟨⟨gpmdb_peptide, gpmdb_start⟩⟩]
add	⟨⟨GS2_peptide, GS2_end⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : peptide′, d}, x}|{d, x} ← ⟨⟨gpmdb_peptide, gpmdb_end⟩⟩]
add	⟨⟨GS2_peptide, GS2_charge⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : peptide′, d}, x}|{d, x} ← ⟨⟨gpmdb_peptide, gpmdb_charge⟩⟩]
add	⟨⟨GS2_peptide, GS2_delta⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : peptide′, d}, x}|{d, x} ← ⟨⟨gpmdb_peptide, gpmdb_delta⟩⟩]
add	⟨⟨GS2_proseq, GS2_rf⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : proseq′, d}, x}|{d, x} ← ⟨⟨gpmdb_proseq, gpmdb_rf⟩⟩]
add	⟨⟨GS2_protein, GS2_expect⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : gpmdb : protein′, d}, x}|{d, x} ← ⟨⟨gpmdb_protein, gpmdb_expect⟩⟩]
add	⟨⟨GS2_aa_pkey, GS2_aa, ⟨⟨GS2_aa, GS2_aaid⟩⟩⟩⟩	null
extend	⟨⟨GS1_fk_proteinhit_protein1, GS1_proteinhit, ⟨⟨GS1_proteinhit, GS1_id⟩⟩, GS1_protein, ⟨⟨GS1_protein, GS1_id⟩⟩⟩⟩	null
	PEDRo → *GS2*	
add	⟨⟨GS2_aa⟩⟩	Void
add	⟨⟨GS2_aa, GS2_aaid⟩⟩	Void
add	⟨⟨GS2_aa, GS2_pepid⟩⟩	Void
add	⟨⟨GS2_aa, GS2_type⟩⟩	Void
add	⟨⟨GS2_aa, GS2_at⟩⟩	Void
add	⟨⟨GS2_aa, GS2_modified⟩⟩	Void
add	⟨⟨GS2_aa, GS2_pm⟩⟩	Void
add	⟨⟨GS2_peptide, GS2_mh⟩⟩	Void
add	⟨⟨GS2_peptide, GS2_start⟩⟩	Void
add	⟨⟨GS2_peptide, GS2_end⟩⟩	Void
add	⟨⟨GS2_peptide, GS2_charge⟩⟩	Void
add	⟨⟨GS2_peptide, GS2_delta⟩⟩	Void
add	⟨⟨GS2_proseq, GS2_rf⟩⟩	Void
add	⟨⟨GS2_protein, GS2_expect⟩⟩	Void
add	⟨⟨GS2_aa_pkey, GS2_aa, ⟨⟨GS2_aa, GS2_aaid⟩⟩⟩⟩	null
	Pepseeker → *GS2*	
add	⟨⟨GS2_peptide, GS2_start⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit′, d}, x}| {d, x} ← ⟨⟨pepseeker_peptidehit, pepseeker_startresidue⟩⟩]
add	⟨⟨GS2_peptide, GS2_end⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit′, d}, x}| {d, x} ← ⟨⟨pepseeker_peptidehit, pepseeker_endresidue⟩⟩]
add	⟨⟨GS2_aa⟩⟩	Void
add	⟨⟨GS2_aa, GS2_aaid⟩⟩	Void

add	⟨⟨GS2_aa, GS2_pepid⟩⟩	Void
add	⟨⟨GS2_aa, GS2_type⟩⟩	Void
add	⟨⟨GS2_aa, GS2_at⟩⟩	Void
add	⟨⟨GS2_aa, GS2_modified⟩⟩	Void
add	⟨⟨GS2_aa, GS2_pm⟩⟩	Void
add	⟨⟨GS2_peptide, GS2_mh⟩⟩	Void
add	⟨⟨GS2_peptide, GS2_charge⟩⟩	Void
add	⟨⟨GS2_peptide, GS2_delta⟩⟩	Void
add	⟨⟨GS2_proseq, GS2_rf⟩⟩	Void
add	⟨⟨GS2_protein, GS2_expect⟩⟩	Void
add	⟨⟨GS2_aa_pkey, GS2_aa, ⟨⟨GS2_aa, GS2_aaid⟩⟩⟩⟩	null
add	⟨⟨GS1_fk_proteinhit_protein1, GS1_proteinhit, ⟨⟨GS1_proteinhit, GS1_id⟩⟩, GS1_protein, ⟨⟨GS1_protein, GS1_id⟩⟩⟩⟩	null
	GS12_Union → *GS12_Improved*	
add	⟨⟨GS2_GS1_peptidehit, GS2_GS1_pep_start⟩⟩	[{id, start}|{id, start} ← ⟨⟨GS2_peptide, GS2_start⟩⟩; id ← ⟨⟨GS2_GS1_peptidehit⟩⟩]
add	⟨⟨GS2_GS1_peptidehit, GS2_GS1_pep_end⟩⟩	[{id, end}|{id, end} ← ⟨⟨GS2_peptide, GS2_end⟩⟩; id ← ⟨⟨GS2_GS1_peptidehit⟩⟩]

Table E.2: Mappings for Iteration 2

	gpmDB → *GS3*	
add	⟨⟨GS3_iontable⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_TOLU⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_ITOLU⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_CLE⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_ICAT⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_INSTRUMENT⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_Useremail⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Id⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Immon⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_A⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_AStar⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_B⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Bstar⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Bstarplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Bzero⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_BZeroplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Y⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Yplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Ystar⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Ystarplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_YZero⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_YZeroplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Bplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Aplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Astarplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Azero⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Matches⟩⟩	Void
add	⟨⟨GS3_peptidehit, GS3_MassNo⟩⟩	Void
add	⟨⟨GS3_peptidehit, GS3_MissCleav⟩⟩	Void
add	⟨⟨GS3_peptidehit, GS3_MrExpct⟩⟩	Void
add	⟨⟨GS3_searchmasses, GS3_Id⟩⟩	Void
add	⟨⟨GS3_iontable_pkey, GS3_iontable, ⟨⟨GS3_iontable, GS3_Id⟩⟩⟩⟩	null

	PEDRo → *GS3*	
add	⟨⟨GS3_iontable⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_TOLU⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_ITOLU⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_CLE⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_ICAT⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_INSTRUMENT⟩⟩	Void
add	⟨⟨GS3_fileparameters, GS3_Useremail⟩⟩	Void

add	⟨⟨GS3_iontable, GS3_Id⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Immon⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_A⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_AStar⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_B⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Bstar⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Bstarplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Bzero⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_BZeroplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Y⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Yplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Ystar⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Ystarplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_YZero⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_YZeroplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Bplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Aplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Astarplusplus⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Azero⟩⟩	Void
add	⟨⟨GS3_iontable, GS3_Matches⟩⟩	Void
add	⟨⟨GS3_peptidehit, GS3_MassNo⟩⟩	Void
add	⟨⟨GS3_peptidehit, GS3_MissCleav⟩⟩	Void
add	⟨⟨GS3_peptidehit, GS3_MrExpct⟩⟩	Void
add	⟨⟨GS3_searchmasses, GS3_Id⟩⟩	Void
add	⟨⟨GS3_iontable_pkey, GS3_iontable, ⟨⟨GS3_iontable, GS3_Id⟩⟩⟩⟩	null

Pepseeker → *GS3*

add	⟨⟨GS3_iontable⟩⟩	[{′URN : LSID : ispider.man.ac.uk : pepseeker : iontable′, d}|d ← ⟨⟨pepseeker_iontable⟩⟩]
add	⟨⟨GS3_fileparameters, GS3_TOLU⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters′, d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_TOLU⟩⟩]
add	⟨⟨GS3_fileparameters, GS3_ITOLU⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters′, d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_ITOLU⟩⟩]
add	⟨⟨GS3_fileparameters, GS3_CLE⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters′, d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_CLE⟩⟩]
add	⟨⟨GS3_fileparameters, GS3_ICAT⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters′, d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_ICAT⟩⟩]
add	⟨⟨GS3_fileparameters, GS3_INSTRUMENT⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters′, d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_INSTRUMENT⟩⟩]
add	⟨⟨GS3_fileparameters, GS3_Useremail⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : fileparameters′, d}, x}|{d, x} ← ⟨⟨pepseeker_fileparameters, pepseeker_Useremail⟩⟩]
add	⟨⟨GS3_iontable, GS3_Id⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : iontable′, d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Id⟩⟩]
add	⟨⟨GS3_iontable, GS3_Immon⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : iontable′, d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Immon⟩⟩]
add	⟨⟨GS3_iontable, GS3_A⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : iontable′, d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_A⟩⟩]
add	⟨⟨GS3_iontable, GS3_AStar⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : iontable′, d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_AStar⟩⟩]
add	⟨⟨GS3_iontable, GS3_B⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : iontable′, d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_B⟩⟩]
add	⟨⟨GS3_iontable, GS3_Bstar⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : iontable′, d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Bstar⟩⟩]
add	⟨⟨GS3_iontable, GS3_Bstarplusplus⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : iontable′, d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Bstarplusplus⟩⟩]
add	⟨⟨GS3_iontable, GS3_Bzero⟩⟩	[{{′URN : LSID : ispider.man.ac.uk : pepseeker : iontable′, d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Bzero⟩⟩]

add	⟨⟨GS3_iontable, GS3_BZeroplusplus⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_BZeroplusplus⟩⟩]
add	⟨⟨GS3_iontable, GS3_Y⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Y⟩⟩]
add	⟨⟨GS3_iontable, GS3_Yplusplus⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Yplusplus⟩⟩]
add	⟨⟨GS3_iontable, GS3_Ystar⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Ystar⟩⟩]
add	⟨⟨GS3_iontable, GS3_Ystarplusplus⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Ystarplusplus⟩⟩]
add	⟨⟨GS3_iontable, GS3_YZero⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_YZero⟩⟩]
add	⟨⟨GS3_iontable, GS3_YZeroplusplus⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_YZeroplusplus⟩⟩]
add	⟨⟨GS3_iontable, GS3_Bplusplus⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Bplusplus⟩⟩]
add	⟨⟨GS3_iontable, GS3_Aplusplus⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Aplusplus⟩⟩]
add	⟨⟨GS3_iontable, GS3_Astarplusplus⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Astarplusplus⟩⟩]
add	⟨⟨GS3_iontable, GS3_Azero⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Azero⟩⟩]
add	⟨⟨GS3_iontable, GS3_Matches⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : iontable', d}, x}|{d, x} ← ⟨⟨pepseeker_iontable, pepseeker_Matches⟩⟩]
add	⟨⟨GS3_peptidehit, GS3_MassNo⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit', d}, x}|{d, x} ← ⟨⟨pepseeker_peptidehit, pepseeker_MassNo⟩⟩]
add	⟨⟨GS3_peptidehit, GS3_MissCleav⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit', d}, x}|{d, x} ← ⟨⟨pepseeker_peptidehit, pepseeker_MissCleav⟩⟩]
add	⟨⟨GS3_peptidehit, GS3_MrExpct⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : peptidehit', d}, x}|{d, x} ← ⟨⟨pepseeker_peptidehit, pepseeker_MrExpct⟩⟩]
add	⟨⟨GS3_searchmasses, GS3_Id⟩⟩	[{{'URN : LSID : ispider.man.ac.uk : pepseeker : searchmasses', d}, x}|{d, x} ← ⟨⟨pepseeker_searchmasses, pepseeker_Id⟩⟩]
add	⟨⟨GS3_iontable_pkey, GS3_iontable, ⟨⟨GS3_iontable, GS3_Id⟩⟩⟩⟩	null
	GS123_Union → GS123_Improved	
add	⟨⟨GS3_GS2_GS1_dbsearch, GS3_GS2_GS1_CLE⟩⟩	[{id, cle}|{id, cle} ← ⟨⟨GS3_fileparameters, GS3_CLE⟩⟩; id ← ⟨⟨GS3_GS2_GS1_dbsearch⟩⟩]
add	⟨⟨GS3_GS2_GS1_dbsearch, GS3_GS2_GS1_usermail⟩⟩	[{id, umail}|{id, umail} ← ⟨⟨GS3_fileparameters, GS3_Useremail⟩⟩; id ← ⟨⟨GS3_GS2_GS1_dbsearch⟩⟩]
add	⟨⟨GS3_GS2_GS1_peptidehit, GS3_GS2_GS1_MissCleav⟩⟩	[{id, MissCleav}|{id, MissCleav} ← ⟨⟨GS3_peptidehit, GS3_MissCleav⟩⟩; id ← ⟨⟨GS3_GS2_GS1_peptidehit⟩⟩]
add	⟨⟨GS3_GS2_GS1_peptidehit, GS3_GS2_GS1_MrExpct⟩⟩	[{id, MrExpct}|{id, MrExpct} ← ⟨⟨GS3_peptidehit, GS3_MrExpct⟩⟩; id ← ⟨⟨GS3_GS2_GS1_peptidehit⟩⟩]
add	⟨⟨GS3_GS2_GS1_dbsearchparameters, GS3_GS2_GS1_TOLU⟩⟩	[{id, tolu}|{id, tolu} ← ⟨⟨GS3_fileparameters, GS3_TOLU⟩⟩; id ← ⟨⟨GS3_GS2_GS1_dbsearchparameters⟩⟩]
add	⟨⟨GS3_GS2_GS1_dbsearchparameters, GS3_GS2_GS1_ITOLU⟩⟩	[{id, itolu}|{id, itolu} ← ⟨⟨GS3_fileparameters, GS3_ITOLU⟩⟩; id ← ⟨⟨GS3_GS2_GS1_dbsearchparameters⟩⟩]
add	⟨⟨GS3_GS2_GS1_dbsearchparameters, GS3_GS2_GS1_ICAT⟩⟩	[{id, icat}|{id, icat} ← ⟨⟨GS3_fileparameters, GS3_ICAT⟩⟩; id ← ⟨⟨GS3_GS2_GS1_dbsearchparameters⟩⟩]
add	⟨⟨GS3_GS2_GS1_dbsearchparameters, GS3_GS2_GS1_INSTRUMENT⟩⟩	[{id, instrument}|{id, instrument} ← ⟨⟨GS3_fileparameters, GS3_INSTRUMENT⟩⟩; id ← ⟨⟨GS3_GS2_GS1_dbsearchparameters⟩⟩]
add	⟨⟨GS3_GS2_GS1_peak, GS3_GS2_GS1_spectrum⟩⟩	[{id, spectrum}|{id, spectrum} ← ⟨⟨GS3_searchmasses, GS3_Id⟩⟩; id ← ⟨⟨GS3_GS2_GS1_peak⟩⟩]

Table E.3: Mappings for Iteration 3

E.3 Concept Coverage

Concept	gpmDB	Pedro	Pepseeker
Peptide	<<gpmdb_peptide>>	<<pedro_peptidehit>>	<<pepseeker_peptidehit>>
	<<gpmdb_peptide,gpmdb_pepid>>	<<pedro_peptidehit,pedro_id>>	<<pepseeker_peptidehit,pepseeker_Id>>
	<<gpmdb_aa,gpmdb_pepid>>		<<pepseeker_mascotexpect,pepseeker_peptidehit_Id>>
	<<gpmdb_fullpeptide,gpmdb_pepid>>		<<pepseeker_peptideprophet,pepseeker_peptidehit_Id>>
	<<gpmdb_fullpeptidediagnostic,gpmdb_pepid>>		
Protein	<<gpmdb_protein>>	<<pedro_protein>>	<<pepseeker_proteinhit>>
	<<gpmdb_protein,gpmdb_proid>>	<<pedro_protein,pedro_id>>	<<pepseeker_proteinhit,pepseeker_Id>>
	<<gpmdb_peptide,gpmdb_proid>>	<<pedro_proteinhit,pedro_id>>	<<pepseeker_proteinhit,pepseeker_ProteinID>>
	<<gpmdb_fullpeptide,gpmdb_proid>>	<<pedro_proteinhit,pedro_protein>>	<<pepseeker_species,pepseeker_ProteinID>>
	<<gpmdb_fullpeptidediagnostic,gpmdb_proid>>		
	<<gpmdb_proseq,gpmdb_label>>		
Result	<<gpmdb_result>>		
	<<gpmdb_result,gpmdb_resultid>>		
	<<gpmdb_protein,gpmdb_resultid>>		
	<<gpmdb_project,gpmdb_resultid>>		
Ion			<<pepseeker_iontable>>
			<<pepseeker_iontable,pepseeker_id>>
			<<pepseeker_peptidehit,pepseeker_ions>>
PeptideSequence	<<gpmdb_peptide,gpmdb_seq>>	<<pedro_peptidehit,pedro_sequence>>	<<pepseeker_peptidehit,pepseeker_pepseq>>
	<<gpmdb_fullpeptide,gpmdb_seq>>		
	<<gpmdb_fullpeptidediagnostic,gpmdb_seq>>		
ProteinSequence	<<gpmdb_protein,gpmdb_proseqid>>	<<pedro_protein,pedro_sequence>>	
	<<gpmdb_proseq>>		
	<<gpmdb_proseq,gpmdb_proseqid>>		
	<<gpmdb_proseq,gpmdb_seq>>		
expect	<<gpmdb_protein,gpmdb_expect>>		<<pepseeker_mascotexpect,pepseeker_expect>>
	<<gpmdb_peptide,gpmdb_expect>>		
Score		<<pedro_peptidehit,pedro_score>>	<<pepseeker_peptidehit,pepseeker_proteinscore>>
			<<pepseeker_proteinscore,pepseeker_Id>>
			<<pepseeker_proteinscore,pepseeker_Score>>
			<<pepseeker_proteinhit,pepseeker_ProteinScore>>
Format	<<gpmdb_result,gpmdb_tandemversion>>	<<pedro_dbsearchparameters,pedro_program>>	<<pepseeker_fileparameters,pepseeker_FORMAT>>
Charge	<<gpmdb_peptide,gpmdb_charge>>		<<pepseeker_fileparameters,pepseker_CHARGE>>
AA	<<gpmdb_aa>>		
	<<gpmdb_aa,gpmdb_aaid>>		
Mass		<<pedro_dbsearchparameters,pedro_mass_value_type>>	<<pepseeker_proteinhit,pepseeker_Mass>>
		<<pedro_peaklist,pedro_mass_value_type>>	<<pepseeker_peptidehit,pepseeker_MassNo>>
		<<pedro_dbsearchparameters,pedro_mass_error_type>>	<<pepseeker_fileparameters,pepseeker_MASS>>
			<<pepseeker_fileparameters,pepseeker_TOLU>>
			<<pepseeker_fileparameters,pepseeker_ITOLU>>

Figure E.4: Concept Coverage of the integrated resource in Iteration 1

Concept	gpmDB	Pedro	Pepseeker
Peptide	<<gpmdb_peptide>>	<<pedro_peptidehit>>	<<pepseeker_peptidehit>>
	<<gpmdb_peptide,gpmdb_pepid>>	<<pedro_peptidehit,pedro_id>>	<<pepseeker_peptidehit,pepseeker_Id>>
	<<gpmdb_aa,gpmdb_pepid>>		<<pepseeker_mascotexpect,pepseeker_peptidehit_Id>>
	<<gpmdb_fullpeptide,gpmdb_pepid>>		<<pepseeker_peptideprophet,pepseeker_peptidehit_Id>>
	<<gpmdb_fullpeptidediagnostic,gpmdb_pepid>>		
Protein	<<gpmdb_protein>>	<<pedro_protein>>	<<pepseeker_proteinhit>>
	<<gpmdb_protein,gpmdb_proid>>	<<pedro_protein,pedro_id>>	<<pepseeker_proteinhit,pepseeker_Id>>
	<<gpmdb_peptide,gpmdb_proid>>	<<pedro_proteinhit,pedro_id>>	<<pepseeker_proteinhit,pepseeker_ProteinID>>
	<<gpmdb_fullpeptide,gpmdb_proid>>	<<pedro_proteinhit,pedro_protein>>	<<pepseeker_species,pepseeker_ProteinID>>
	<<gpmdb_fullpeptidediagnostic,gpmdb_proid>>		
	<<gpmdb_proseq,gpmdb_label>>		
Result	<<gpmdb_result>>		
	<<gpmdb_result,gpmdb_resultid>>		
	<<gpmdb_protein,gpmdb_resultid>>		
	<<gpmdb_project,gpmdb_resultid>>		
Ion			<<pepseeker_iontable>>
			<<pepseeker_iontable,pepseeker_id>>
			<<pepseeker_peptidehit,pepseeker_ions>>
PeptideSequence	<<gpmdb_peptide,gpmdb_seq>>	<<pedro_peptidehit,pedro_sequence>>	<<pepseeker_peptidehit,pepseeker_pepseq>>
	<<gpmdb_fullpeptide,gpmdb_seq>>		
	<<gpmdb_fullpeptidediagnostic,gpmdb_seq>>		
ProteinSequence	<<gpmdb_protein,gpmdb_proseqid>>	<<pedro_protein,pedro_sequence>>	
	<<gpmdb_proseq>>		
	<<gpmdb_proseq,gpmdb_proseqid>>		
	<<gpmdb_proseq,gpmdb_seq>>		
expect	<<gpmdb_protein,gpmdb_expect>>		<<pepseeker_mascotexpect,pepseeker_expect>>
	<<gpmdb_peptide,gpmdb_expect>>		
Score		<<pedro_peptidehit,pedro_score>>	<<pepseeker_peptidehit,pepseeker_proteinscore>>
			<<pepseeker_proteinscore,pepseeker_Id>>
			<<pepseeker_proteinscore,pepseeker_Score>>
			<<pepseeker_proteinhit,pepseeker_ProteinScore>>
Format	<<gpmdb_result,gpmdb_tandemversion>>	<<pedro_dbsearchparameters,pedro_program>>	<<pepseeker_fileparameters,pepseeker_FORMAT>>
Charge	<<gpmdb_peptide,gpmdb_charge>>		<<pepseeker_fileparameters,pepseker_CHARGE>>
AA	<<gpmdb_aa>>		
	<<gpmdb_aa,gpmdb_aaid>>		
Mass		<<pedro_dbsearchparameters,pedro_mass_value_type>>	<<pepseeker_proteinhit,pepseeker_Mass>>
		<<pedro_peaklist,pedro_mass_value_type>>	<<pepseeker_peptidehit,pepseeker_MassNo>>
		<<pedro_dbsearchparameters,pedro_mass_error_type>>	<<pepseeker_fileparameters,pepseeker_MASS>>
			<<pepseeker_fileparameters,pepseeker_TOLU>>
			<<pepseeker_fileparameters,pepseeker_ITOLU>>

Figure E.5: Concept Coverage of the integrated resource in Iteration 2

Concept	gpmDB	Pedro	Pepseeker
Peptide	<<gpmdb_peptide>>	<<pedro_peptidehit>>	<<pepseeker_peptidehit>>
	<<gpmdb_peptide,gpmdb_pepid>>	<<pedro_peptidehit,pedro_id>>	<<pepseeker_peptidehit,pepseeker_Id>>
	<<gpmdb_aa,gpmdb_pepid>>		<<pepseeker_mascotexpect,pepseeker_peptidehit_Id>>
	<<gpmdb_fullpeptide,gpmdb_pepid>>		<<pepseeker_peptideprophet,pepseeker_peptidehit_Id>>
	<<gpmdb_fullpeptidediagnostic,gpmdb_pepid>>		
Protein	<<gpmdb_protein>>	<<pedro_protein>>	<<pepseeker_proteinhit>>
	<<gpmdb_protein,gpmdb_proid>>	<<pedro_protein,pedro_id>>	<<pepseeker_proteinhit,pepseeker_Id>>
	<<gpmdb_peptide,gpmdb_proid>>	<<pedro_proteinhit,pedro_id>>	<<pepseeker_proteinhit,pepseeker_ProteinID>>
	<<gpmdb_fullpeptide,gpmdb_proid>>	<<pedro_proteinhit,pedro_protein>>	<<pepseeker_species,pepseeker_ProteinID>>
	<<gpmdb_fullpeptidediagnostic,gpmdb_proid>>		
	<<gpmdb_proseq,gpmdb_label>>		
Result	<<gpmdb_result>>		
	<<gpmdb_result,gpmdb_resultid>>		
	<<gpmdb_protein,gpmdb_resultid>>		
	<<gpmdb_project,gpmdb_resultid>>		
Ion			<<pepseeker_iontable>>
			<<pepseeker_iontable,pepseeker_id>>
			<<pepseeker_peptidehit,pepseeker_ions>>
PeptideSequence	<<gpmdb_peptide,gpmdb_seq>>	<<pedro_peptidehit,pedro_sequence>>	<<pepseeker_peptidehit,pepseeker_pepseq>>
	<<gpmdb_fullpeptide,gpmdb_seq>>		
	<<gpmdb_fullpeptidediagnostic,gpmdb_seq>>		
ProteinSequence	<<gpmdb_protein,gpmdb_proseqid>>	<<pedro_protein,pedro_sequence>>	
	<<gpmdb_proseq>>		
	<<gpmdb_proseq,gpmdb_proseqid>>		
	<<gpmdb_proseq,gpmdb_seq>>		
expect	<<gpmdb_protein,gpmdb_expect>>		<<pepseeker_mascotexpect,pepseeker_expect>>
	<<gpmdb_peptide,gpmdb_expect>>		
Score		<<pedro_peptidehit,pedro_score>>	<<pepseeker_peptidehit,pepseeker_proteinscore>>
			<<pepseeker_proteinscore,pepseeker_Id>>
			<<pepseeker_proteinscore,pepseeker_Score>>
			<<pepseeker_proteinhit,pepseeker_ProteinScore>>
Format	<<gpmdb_result,gpmdb_tandemversion>>	<<pedro_dbsearchparameters,pedro_program>>	<<pepseeker_fileparameters,pepseeker_FORMAT>>
Charge	<<gpmdb_peptide,gpmdb_charge>>		<<pepseeker_fileparameters,pepseker_CHARGE>>
AA	<<gpmdb_aa>>		
	<<gpmdb_aa,gpmdb_aaid>>		
Mass		<<pedro_dbsearchparameters,pedro_mass_value_type>>	<<pepseeker_proteinhit,pepseeker_Mass>>
		<<pedro_peaklist,pedro_mass_value_type>>	<<pepseeker_peptidehit,pepseeker_MassNo>>
		<<pedro_dbsearchparameters,pedro_mass_error_type>>	<<pepseeker_fileparameters,pepseeker_MASS>>
			<<pepseeker_fileparameters,pepseeker_TOLU>>
			<<pepseeker_fileparameters,pepseeker_ITOLU>>

Figure E.6: Concept Coverage of the integrated resource in Iteration 3

E.4 Assessments of Quality Factors

E.4.1 Quality Measurement for Iteration 1

Factor 1

⟨⟨gpmdb_protein⟩⟩	⟨⟨pedro_iontrap, pedro_mz_analysis⟩⟩
⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩	⟨⟨pedro_lccolumn⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩	⟨⟨pedro_lccolumn, pedro_id⟩⟩
⟨⟨gpmdb_peptide⟩⟩	⟨⟨pedro_lccolumn, pedro_description⟩⟩
⟨⟨gpmdb_peptide, gpmdb_seq⟩⟩	⟨⟨pedro_lccolumn, pedro_manufacturer⟩⟩
⟨⟨gpmdb_peptide, gpmdb_expect⟩⟩	⟨⟨pedro_lccolumn, pedro_part_number⟩⟩
⟨⟨gpmdb_result⟩⟩	⟨⟨pedro_lccolumn, pedro_batch_number⟩⟩
⟨⟨gpmdb_result, gpmdb_tandemversion⟩⟩	⟨⟨pedro_lccolumn, pedro_internal_length⟩⟩
⟨⟨gpmdb_proseq, gpmdb_label⟩⟩	⟨⟨pedro_lccolumn, pedro_internal_diameter⟩⟩
⟨⟨gpmdb_proseq, gpmdb_seq⟩⟩	⟨⟨pedro_lccolumn, pedro_stationary_phase⟩⟩
⟨⟨pedro_proteinhit⟩⟩	⟨⟨pedro_lccolumn, pedro_bead_size⟩⟩
⟨⟨pedro_proteinhit, pedro_id⟩⟩	⟨⟨pedro_lccolumn, pedro_pore_size⟩⟩
⟨⟨pedro_proteinhit, pedro_all_peptides_matched⟩⟩	⟨⟨pedro_lccolumn, pedro_temperature⟩⟩
⟨⟨pedro_proteinhit, pedro_score⟩⟩	⟨⟨pedro_lccolumn, pedro_flow_rate⟩⟩
⟨⟨pedro_proteinhit, pedro_protein⟩⟩	⟨⟨pedro_lccolumn, pedro_injection_volume⟩⟩
⟨⟨pedro_proteinhit, pedro_db_search⟩⟩	⟨⟨pedro_lccolumn, pedro_parameters_file⟩⟩
⟨⟨pedro_proteinhit, pedro_component_peptides⟩⟩	⟨⟨pedro_lccolumn, pedro_lc_column⟩⟩
⟨⟨pedro_proteinhit, pedro_masses_matched⟩⟩	⟨⟨pedro_lccolumn, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_proteinhit, pedro_score_type⟩⟩	⟨⟨pedro_listprocessing⟩⟩
⟨⟨pedro_peptidehit⟩⟩	⟨⟨pedro_listprocessing, pedro_id⟩⟩
⟨⟨pedro_peptidehit, pedro_id⟩⟩	⟨⟨pedro_listprocessing, pedro_smoothing_process⟩⟩
⟨⟨pedro_peptidehit, pedro_score⟩⟩	⟨⟨pedro_listprocessing, pedro_background_threshold⟩⟩
⟨⟨pedro_peptidehit, pedro_score_type⟩⟩	⟨⟨pedro_listprocessing, pedro_peak_list⟩⟩
⟨⟨pedro_peptidehit, pedro_sequence⟩⟩	⟨⟨pedro_maldi⟩⟩
⟨⟨pedro_peptidehit, pedro_information⟩⟩	⟨⟨pedro_maldi, pedro_id⟩⟩
⟨⟨pedro_peptidehit, pedro_probability⟩⟩	⟨⟨pedro_maldi, pedro_laser_wavelength⟩⟩
⟨⟨pedro_peptidehit, pedro_db_search⟩⟩	⟨⟨pedro_maldi, pedro_laser_power⟩⟩
⟨⟨pedro_dbsearch⟩⟩	⟨⟨pedro_maldi, pedro_matrix_type⟩⟩
⟨⟨pedro_dbsearch, pedro_id⟩⟩	⟨⟨pedro_maldi, pedro_grid_voltage⟩⟩
⟨⟨pedro_dbsearch, pedro_username⟩⟩	⟨⟨pedro_maldi, pedro_acceleration_voltage⟩⟩
⟨⟨pedro_dbsearch, pedro_id_date⟩⟩	⟨⟨pedro_maldi, pedro_ion_mode⟩⟩
⟨⟨pedro_dbsearch, pedro_n_terminal_aa⟩⟩	⟨⟨pedro_maldi, pedro_ion_source⟩⟩
⟨⟨pedro_dbsearch, pedro_c_terminal_aa⟩⟩	⟨⟨pedro_massspecexperiment⟩⟩
⟨⟨pedro_dbsearch, pedro_count_of_specific_aa⟩⟩	⟨⟨pedro_massspecexperiment, pedro_id⟩⟩
⟨⟨pedro_dbsearch, pedro_name_of_counted_aa⟩⟩	⟨⟨pedro_massspecexperiment, pedro_description⟩⟩
⟨⟨pedro_dbsearch, pedro_regex_pattern⟩⟩	⟨⟨pedro_massspecexperiment, pedro_parameters_file⟩⟩
⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩	⟨⟨pedro_massspecexperiment, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_dbsearch, pedro_peak_list⟩⟩	⟨⟨pedro_massspecmachine⟩⟩
⟨⟨pedro_dbsearchparameters⟩⟩	⟨⟨pedro_massspecmachine, pedro_id⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩	⟨⟨pedro_massspecmachine, pedro_manufacturer⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_program⟩⟩	⟨⟨pedro_massspecmachine, pedro_model_name⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_database_name⟩⟩	⟨⟨pedro_massspecmachine, pedro_software_version⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_database_date⟩⟩	⟨⟨pedro_massspecmachine, pedro_ion_source⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_parameters_file⟩⟩	⟨⟨pedro_mobilephasecomponent⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_taxonomical_filter⟩⟩	⟨⟨pedro_mobilephasecomponent, pedro_id⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_fixed_modifications⟩⟩	⟨⟨pedro_mobilephasecomponent, pedro_description⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_variable_modifications⟩⟩	⟨⟨pedro_mobilephasecomponent, pedro_concentration⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_max_missed_cleavages⟩⟩	⟨⟨pedro_mobilephasecomponent, pedro_lc_column⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_value_type⟩⟩	⟨⟨pedro_msmsfraction⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_fragment_ion_tolerance⟩⟩	⟨⟨pedro_msmsfraction, pedro_id⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_peptide_mass_tolerance⟩⟩	⟨⟨pedro_msmsfraction, pedro_target_m_to_z⟩⟩

⟨⟨pedro_dbsearchparameters, pedro_accurate_mass_mode⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_error_type⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_error⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_protonated⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_icat_option⟩⟩
⟨⟨pedro_peak⟩⟩
⟨⟨pedro_peak, pedro_id⟩⟩
⟨⟨pedro_peak, pedro_m_to_z⟩⟩
⟨⟨pedro_peak, pedro_abundance⟩⟩
⟨⟨pedro_peak, pedro_multiplicity⟩⟩
⟨⟨pedro_peak, pedro_peak_list⟩⟩
⟨⟨pedro_protein⟩⟩
⟨⟨pedro_protein, pedro_id⟩⟩
⟨⟨pedro_protein, pedro_accession_num⟩⟩
⟨⟨pedro_protein, pedro_gene_name⟩⟩
⟨⟨pedro_protein, pedro_synonyms⟩⟩
⟨⟨pedro_protein, pedro_organism⟩⟩
⟨⟨pedro_protein, pedro_orf_number⟩⟩
⟨⟨pedro_protein, pedro_description⟩⟩
⟨⟨pedro_protein, pedro_sequence⟩⟩
⟨⟨pedro_protein, pedro_modifications⟩⟩
⟨⟨pedro_protein, pedro_predicted_mass⟩⟩
⟨⟨pedro_protein, pedro_predicted_pi⟩⟩
⟨⟨pedro_analyteprocessingstep⟩⟩
⟨⟨pedro_analyteprocessingstep, pedro_id⟩⟩
⟨⟨pedro_analyteprocessingstep, pedro_input_type⟩⟩
⟨⟨pedro_analyteprocessingstep, pedro_processing_type⟩⟩
⟨⟨pedro_assaydatapoint⟩⟩
⟨⟨pedro_assaydatapoint, pedro_id⟩⟩
⟨⟨pedro_assaydatapoint, pedro_time⟩⟩
⟨⟨pedro_assaydatapoint, pedro_protein_assay⟩⟩
⟨⟨pedro_assaydatapoint, pedro_lc_column⟩⟩
⟨⟨pedro_band⟩⟩
⟨⟨pedro_band, pedro_id⟩⟩
⟨⟨pedro_band, pedro_area⟩⟩
⟨⟨pedro_band, pedro_intensity⟩⟩
⟨⟨pedro_band, pedro_local_background⟩⟩
⟨⟨pedro_band, pedro_annotation⟩⟩
⟨⟨pedro_band, pedro_annotation_source⟩⟩
⟨⟨pedro_band, pedro_volume⟩⟩
⟨⟨pedro_band, pedro_pixel_x_coord⟩⟩
⟨⟨pedro_band, pedro_pixel_y_coord⟩⟩
⟨⟨pedro_band, pedro_pixel_radius⟩⟩
⟨⟨pedro_band, pedro_normalisation⟩⟩
⟨⟨pedro_band, pedro_normalised_volume⟩⟩
⟨⟨pedro_band, pedro_lane_number⟩⟩
⟨⟨pedro_band, pedro_apparent_mass⟩⟩
⟨⟨pedro_band, pedro_gel_1d⟩⟩
⟨⟨pedro_band_otm_analyteps⟩⟩
⟨⟨pedro_band_otm_analyteps, pedro_band_gel_1d⟩⟩
⟨⟨pedro_band_otm_analyteps, pedro_band_id⟩⟩
⟨⟨pedro_band_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_boundarypoint⟩⟩
⟨⟨pedro_boundarypoint, pedro_id⟩⟩
⟨⟨pedro_boundarypoint, pedro_pixel_x_coord⟩⟩
⟨⟨pedro_boundarypoint, pedro_pixel_y_coord⟩⟩
⟨⟨pedro_boundarypoint, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_boundarypoint, pedro_spot_id⟩⟩
⟨⟨pedro_chemicaltreatment⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_id⟩⟩
⟨⟨pedro_msmsfraction, pedro_plus_or_minus⟩⟩
⟨⟨pedro_msmsfraction, pedro_peak_list⟩⟩
⟨⟨pedro_mzanalysis⟩⟩
⟨⟨pedro_mzanalysis, pedro_id⟩⟩
⟨⟨pedro_mzanalysis, pedro_type⟩⟩
⟨⟨pedro_mzanalysis, pedro_detection⟩⟩
⟨⟨pedro_ontologyentry⟩⟩
⟨⟨pedro_ontologyentry, pedro_id⟩⟩
⟨⟨pedro_ontologyentry, pedro_category⟩⟩
⟨⟨pedro_ontologyentry, pedro_value⟩⟩
⟨⟨pedro_ontologyentry, pedro_description⟩⟩
⟨⟨pedro_organism⟩⟩
⟨⟨pedro_organism, pedro_id⟩⟩
⟨⟨pedro_organism, pedro_species_name⟩⟩
⟨⟨pedro_organism, pedro_strain_identifier⟩⟩
⟨⟨pedro_organism, pedro_relevant_genotype⟩⟩
⟨⟨pedro_otheranalyte⟩⟩
⟨⟨pedro_otheranalyte, pedro_id⟩⟩
⟨⟨pedro_otheranalyte, pedro_name⟩⟩
⟨⟨pedro_otheranalyte, pedro_other_analyte_processing_step⟩⟩
⟨⟨pedro_otheranalyte_otm_analyteps⟩⟩
⟨⟨pedro_otheranalyte_otm_analyteps, pedro_other_analyte⟩⟩
⟨⟨pedro_otheranalyte_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_otheranalyte_otm_ontent⟩⟩
⟨⟨pedro_otheranalyte_otm_ontent, pedro_other_analyte⟩⟩
⟨⟨pedro_otheranalyte_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_otheranalyteps⟩⟩
⟨⟨pedro_otheranalyteps, pedro_id⟩⟩
⟨⟨pedro_otheranalyteps, pedro_name⟩⟩
⟨⟨pedro_otheranalyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_otheranalyteps_otm_ontent⟩⟩
⟨⟨pedro_otheranalyteps_otm_ontent, pedro_other_analyte_processing_step⟩⟩
⟨⟨pedro_otheranalyteps_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_otherionisation⟩⟩
⟨⟨pedro_otherionisation, pedro_id⟩⟩
⟨⟨pedro_otherionisation, pedro_name⟩⟩
⟨⟨pedro_otherionisation, pedro_ion_source⟩⟩
⟨⟨pedro_otherionisation_otm_ontent⟩⟩
⟨⟨pedro_otherionisation_otm_ontent, pedro_other_ionisation⟩⟩
⟨⟨pedro_otherionisation_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_othermzanalysis⟩⟩
⟨⟨pedro_othermzanalysis, pedro_id⟩⟩
⟨⟨pedro_othermzanalysis, pedro_name⟩⟩
⟨⟨pedro_othermzanalysis, pedro_mz_analysis⟩⟩
⟨⟨pedro_othermzanalysis_otm_ontent⟩⟩
⟨⟨pedro_othermzanalysis_otm_ontent, pedro_other_mz_analysis⟩⟩
⟨⟨pedro_othermzanalysis_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_peaklist⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_peaklist, pedro_list_type⟩⟩
⟨⟨pedro_peaklist, pedro_description⟩⟩
⟨⟨pedro_peaklist, pedro_mass_value_type⟩⟩
⟨⟨pedro_peaklist, pedro_mass_spec_experiment⟩⟩
⟨⟨pedro_peakspecificchromint⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_id⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_resolution⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_software_version⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_background_threshold⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_area_under_curve⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_peak_description⟩⟩

⟨⟨pedro_chemicaltreatment, pedro_digestion⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_derivatisations⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_chromatogrampoint⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_id⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_time_point⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_ion_count⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_peak⟩⟩
⟨⟨pedro_collisioncell⟩⟩
⟨⟨pedro_collisioncell, pedro_id⟩⟩
⟨⟨pedro_collisioncell, pedro_gas_type⟩⟩
⟨⟨pedro_collisioncell, pedro_gas_pressure⟩⟩
⟨⟨pedro_collisioncell, pedro_collision_offset⟩⟩
⟨⟨pedro_collisioncell, pedro_mz_analysis⟩⟩
⟨⟨pedro_dbsearchpars_otm_ontent⟩⟩
⟨⟨pedro_dbsearchpars_otm_ontent, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearchpars_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_detection⟩⟩
⟨⟨pedro_detection, pedro_id⟩⟩
⟨⟨pedro_detection, pedro_type⟩⟩
⟨⟨pedro_digegel⟩⟩
⟨⟨pedro_digegel, pedro_id⟩⟩
⟨⟨pedro_digegel, pedro_dye_type⟩⟩
⟨⟨pedro_digegel, pedro_excitation_wavelength⟩⟩
⟨⟨pedro_digegel, pedro_exposure_time⟩⟩
⟨⟨pedro_digegel, pedro_tiff_image⟩⟩
⟨⟨pedro_digegel, pedro_gel_1d⟩⟩
⟨⟨pedro_digegel, pedro_gel_2d⟩⟩
⟨⟨pedro_digegelitem⟩⟩
⟨⟨pedro_digegelitem, pedro_id⟩⟩
⟨⟨pedro_digegelitem, pedro_dye_type⟩⟩
⟨⟨pedro_digegelitem, pedro_band_gel_1d⟩⟩
⟨⟨pedro_digegelitem, pedro_band_id⟩⟩
⟨⟨pedro_digegelitem, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_digegelitem, pedro_spot_id⟩⟩
⟨⟨pedro_electrospray⟩⟩
⟨⟨pedro_electrospray, pedro_id⟩⟩
⟨⟨pedro_electrospray, pedro_spray_tip_voltage⟩⟩
⟨⟨pedro_electrospray, pedro_spray_tip_diameter⟩⟩
⟨⟨pedro_electrospray, pedro_solution_voltage⟩⟩
⟨⟨pedro_electrospray, pedro_cone_voltage⟩⟩
⟨⟨pedro_electrospray, pedro_loading_type⟩⟩
⟨⟨pedro_electrospray, pedro_solvent⟩⟩
⟨⟨pedro_electrospray, pedro_interface_manufacturer⟩⟩
⟨⟨pedro_electrospray, pedro_spray_tip_manufacturer⟩⟩
⟨⟨pedro_electrospray, pedro_ion_source⟩⟩
⟨⟨pedro_experiment⟩⟩
⟨⟨pedro_experiment, pedro_id⟩⟩
⟨⟨pedro_experiment, pedro_hypothesis⟩⟩
⟨⟨pedro_experiment, pedro_method_citations⟩⟩
⟨⟨pedro_experiment, pedro_result_citations⟩⟩
⟨⟨pedro_fraction⟩⟩
⟨⟨pedro_fraction, pedro_id⟩⟩
⟨⟨pedro_fraction, pedro_start_point⟩⟩
⟨⟨pedro_fraction, pedro_end_point⟩⟩
⟨⟨pedro_fraction, pedro_protein_assay⟩⟩
⟨⟨pedro_fraction, pedro_lc_column⟩⟩
⟨⟨pedro_fraction_otm_analyteps⟩⟩
⟨⟨pedro_fraction_otm_analyteps, pedro_fraction_lc_column⟩⟩
⟨⟨pedro_fraction_otm_analyteps, pedro_fraction_id⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_sister_peak_reference⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_peak⟩⟩
⟨⟨pedro_peptidehit_mtm_ontent⟩⟩
⟨⟨pedro_peptidehit_mtm_ontent, pedro_peptide_hit_id⟩⟩
⟨⟨pedro_peptidehit_mtm_ontent, pedro_peptide_hit_db_search⟩⟩
⟨⟨pedro_peptidehit_mtm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_percentx⟩⟩
⟨⟨pedro_percentx, pedro_id⟩⟩
⟨⟨pedro_percentx, pedro_percentage⟩⟩
⟨⟨pedro_percentx, pedro_mobile_phase_component⟩⟩
⟨⟨pedro_percentx, pedro_gradient_step_lc_column⟩⟩
⟨⟨pedro_percentx, pedro_gradient_step_id⟩⟩
⟨⟨pedro_quadrupole⟩⟩
⟨⟨pedro_quadrupole, pedro_id⟩⟩
⟨⟨pedro_quadrupole, pedro_description⟩⟩
⟨⟨pedro_quadrupole, pedro_mz_analysis⟩⟩
⟨⟨pedro_relatedgelitem⟩⟩
⟨⟨pedro_relatedgelitem, pedro_id⟩⟩
⟨⟨pedro_relatedgelitem, pedro_description⟩⟩
⟨⟨pedro_relatedgelitem, pedro_gel_reference⟩⟩
⟨⟨pedro_relatedgelitem, pedro_item_reference⟩⟩
⟨⟨pedro_relatedgelitem, pedro_band_gel_1d⟩⟩
⟨⟨pedro_relatedgelitem, pedro_band_id⟩⟩
⟨⟨pedro_relatedgelitem, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_relatedgelitem, pedro_spot_id⟩⟩
⟨⟨pedro_relgelitem_mtm_proteinhit⟩⟩
⟨⟨pedro_relgelitem_mtm_proteinhit, pedro_related_gel_item⟩⟩
⟨⟨pedro_relgelitem_mtm_proteinhit, pedro_protein_hit⟩⟩
⟨⟨pedro_sample⟩⟩
⟨⟨pedro_sample, pedro_sample_id⟩⟩
⟨⟨pedro_sample, pedro_sample_date⟩⟩
⟨⟨pedro_sample, pedro_experimenter⟩⟩
⟨⟨pedro_sample, pedro_experiment⟩⟩
⟨⟨pedro_sample_mtm_sampleorigin⟩⟩
⟨⟨pedro_sample_mtm_sampleorigin, pedro_sample⟩⟩
⟨⟨pedro_sample_mtm_sampleorigin, pedro_sample_origin⟩⟩
⟨⟨pedro_sample_otm_analyteps⟩⟩
⟨⟨pedro_sample_otm_analyteps, pedro_sample⟩⟩
⟨⟨pedro_sample_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_sampleorigin⟩⟩
⟨⟨pedro_sampleorigin, pedro_id⟩⟩
⟨⟨pedro_sampleorigin, pedro_description⟩⟩
⟨⟨pedro_sampleorigin, pedro_sample_condition⟩⟩
⟨⟨pedro_sampleorigin, pedro_condition_degree⟩⟩
⟨⟨pedro_sampleorigin, pedro_environment⟩⟩
⟨⟨pedro_sampleorigin, pedro_tissue_type⟩⟩
⟨⟨pedro_sampleorigin, pedro_cell_type⟩⟩
⟨⟨pedro_sampleorigin, pedro_cell_cycle_phase⟩⟩
⟨⟨pedro_sampleorigin, pedro_cell_component⟩⟩
⟨⟨pedro_sampleorigin, pedro_technique⟩⟩
⟨⟨pedro_sampleorigin, pedro_metabolic_label⟩⟩
⟨⟨pedro_sampleorigin, pedro_organism⟩⟩
⟨⟨pedro_sampleorigin, pedro_tagging_process⟩⟩
⟨⟨pedro_spot⟩⟩
⟨⟨pedro_spot, pedro_id⟩⟩
⟨⟨pedro_spot, pedro_area⟩⟩
⟨⟨pedro_spot, pedro_intensity⟩⟩
⟨⟨pedro_spot, pedro_local_background⟩⟩
⟨⟨pedro_spot, pedro_annotation⟩⟩
⟨⟨pedro_spot, pedro_annotation_source⟩⟩

⟨⟨pedro_fraction_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_gel1d⟩⟩
⟨⟨pedro_gel1d, pedro_id⟩⟩
⟨⟨pedro_gel1d, pedro_description⟩⟩
⟨⟨pedro_gel1d, pedro_raw_image⟩⟩
⟨⟨pedro_gel1d, pedro_annotated_image⟩⟩
⟨⟨pedro_gel1d, pedro_software_version⟩⟩
⟨⟨pedro_gel1d, pedro_warped_image⟩⟩
⟨⟨pedro_gel1d, pedro_warping_map⟩⟩
⟨⟨pedro_gel1d, pedro_equipment⟩⟩
⟨⟨pedro_gel1d, pedro_percent_acrylamide⟩⟩
⟨⟨pedro_gel1d, pedro_solubilization_buffer⟩⟩
⟨⟨pedro_gel1d, pedro_stain_details⟩⟩
⟨⟨pedro_gel1d, pedro_protein_assay⟩⟩
⟨⟨pedro_gel1d, pedro_in_gel_digestion⟩⟩
⟨⟨pedro_gel1d, pedro_background⟩⟩
⟨⟨pedro_gel1d, pedro_pixel_size_x⟩⟩
⟨⟨pedro_gel1d, pedro_pixel_size_y⟩⟩
⟨⟨pedro_gel1d, pedro_denaturing_agent⟩⟩
⟨⟨pedro_gel1d, pedro_mass_start⟩⟩
⟨⟨pedro_gel1d, pedro_mass_end⟩⟩
⟨⟨pedro_gel1d, pedro_run_details⟩⟩
⟨⟨pedro_gel1d, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_gel2d⟩⟩
⟨⟨pedro_gel2d, pedro_id⟩⟩
⟨⟨pedro_gel2d, pedro_description⟩⟩
⟨⟨pedro_gel2d, pedro_raw_image⟩⟩
⟨⟨pedro_gel2d, pedro_annotated_image⟩⟩
⟨⟨pedro_gel2d, pedro_software_version⟩⟩
⟨⟨pedro_gel2d, pedro_warped_image⟩⟩
⟨⟨pedro_gel2d, pedro_warping_map⟩⟩
⟨⟨pedro_gel2d, pedro_equipment⟩⟩
⟨⟨pedro_gel2d, pedro_percent_acrylamide⟩⟩
⟨⟨pedro_gel2d, pedro_solubilization_buffer⟩⟩
⟨⟨pedro_gel2d, pedro_stain_details⟩⟩
⟨⟨pedro_gel2d, pedro_protein_assay⟩⟩
⟨⟨pedro_gel2d, pedro_in_gel_digestion⟩⟩
⟨⟨pedro_gel2d, pedro_background⟩⟩
⟨⟨pedro_gel2d, pedro_pixel_size_x⟩⟩
⟨⟨pedro_gel2d, pedro_pixel_size_y⟩⟩
⟨⟨pedro_gel2d, pedro_pi_start⟩⟩
⟨⟨pedro_gel2d, pedro_pi_end⟩⟩
⟨⟨pedro_gel2d, pedro_mass_start⟩⟩
⟨⟨pedro_gel2d, pedro_mass_end⟩⟩
⟨⟨pedro_gel2d, pedro_first_dim_details⟩⟩
⟨⟨pedro_gel2d, pedro_second_dim_details⟩⟩
⟨⟨pedro_gel2d, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_gradientstep⟩⟩
⟨⟨pedro_gradientstep, pedro_id⟩⟩
⟨⟨pedro_gradientstep, pedro_step_time⟩⟩
⟨⟨pedro_gradientstep, pedro_lc_column⟩⟩
⟨⟨pedro_hexapole⟩⟩
⟨⟨pedro_hexapole, pedro_id⟩⟩
⟨⟨pedro_hexapole, pedro_description⟩⟩
⟨⟨pedro_hexapole, pedro_mz_analysis⟩⟩
⟨⟨pedro_ionsource⟩⟩
⟨⟨pedro_ionsource, pedro_id⟩⟩
⟨⟨pedro_ionsource, pedro_collision_energy⟩⟩
⟨⟨pedro_ionsource, pedro_type⟩⟩
⟨⟨pedro_ionsource, pedro_mz_analysis⟩⟩
⟨⟨pedro_spot, pedro_volume⟩⟩
⟨⟨pedro_spot, pedro_pixel_x_coord⟩⟩
⟨⟨pedro_spot, pedro_pixel_y_coord⟩⟩
⟨⟨pedro_spot, pedro_pixel_radius⟩⟩
⟨⟨pedro_spot, pedro_normalisation⟩⟩
⟨⟨pedro_spot, pedro_normalised_volume⟩⟩
⟨⟨pedro_spot, pedro_apparent_pi⟩⟩
⟨⟨pedro_spot, pedro_apparent_mass⟩⟩
⟨⟨pedro_spot, pedro_gel_2d⟩⟩
⟨⟨pedro_spot_otm_analyteps⟩⟩
⟨⟨pedro_spot_otm_analyteps, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_spot_otm_analyteps, pedro_spot_id⟩⟩
⟨⟨pedro_spot_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_taggingprocess⟩⟩
⟨⟨pedro_taggingprocess, pedro_id⟩⟩
⟨⟨pedro_taggingprocess, pedro_lysis_buffer⟩⟩
⟨⟨pedro_taggingprocess, pedro_tag_type⟩⟩
⟨⟨pedro_taggingprocess, pedro_tag_purity⟩⟩
⟨⟨pedro_taggingprocess, pedro_protein_concentration⟩⟩
⟨⟨pedro_taggingprocess, pedro_tag_concentration⟩⟩
⟨⟨pedro_taggingprocess, pedro_final_volume⟩⟩
⟨⟨pedro_tandemsequencedata⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_id⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_source_type⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_sequence⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_db_search_parameters⟩⟩
⟨⟨pedro_tof⟩⟩
⟨⟨pedro_tof, pedro_id⟩⟩
⟨⟨pedro_tof, pedro_reflectron_state⟩⟩
⟨⟨pedro_tof, pedro_internal_length⟩⟩
⟨⟨pedro_tof, pedro_mz_analysis⟩⟩
⟨⟨pedro_treatedanalyte⟩⟩
⟨⟨pedro_treatedanalyte, pedro_id⟩⟩
⟨⟨pedro_treatedanalyte, pedro_chemical_treatment⟩⟩
⟨⟨pedro_treatedanalyte_otm_analyteps⟩⟩
⟨⟨pedro_treatedanalyte_otm_analyteps, pedro_treated_analyte⟩⟩
⟨⟨pedro_treatedanalyte_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pepseeker_proteinhit⟩⟩
⟨⟨pepseeker_proteinscore, pepseeker_Score⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_ProteinID⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_peptidehit⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_proteinscore⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_pepseq⟩⟩
⟨⟨pepseeker_mascotexpect, pepseeker_expect⟩⟩
⟨⟨pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Username⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Id⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Filename⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TAXONOMY⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_MODS⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_IT_MODS⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_PFA⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ITOL⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TOL⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SEG⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_CHARGE⟩⟩
⟨⟨pepseeker_searchmasses⟩⟩
⟨⟨pepseeker_searchmasses, pepseeker_Products⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_Proteinname⟩⟩

⟨⟨pedro_iontrap⟩⟩	⟨⟨pepseeker_species, pepseeker_species⟩⟩
⟨⟨pedro_iontrap, pedro_id⟩⟩	⟨⟨pepseeker_proteinhit, pepseeker_Mass⟩⟩
⟨⟨pedro_iontrap, pedro_gas_type⟩⟩	
⟨⟨pedro_iontrap, pedro_gas_pressure⟩⟩	
⟨⟨pedro_iontrap, pedro_rf_frequency⟩⟩	
⟨⟨pedro_iontrap, pedro_excitation_amplitude⟩⟩	
⟨⟨pedro_iontrap, pedro_isolation_centre⟩⟩	
⟨⟨pedro_iontrap, pedro_isolation_width⟩⟩	
⟨⟨pedro_iontrap, pedro_final_ms_level⟩⟩	

Table E.4: Elements Satisfying Quality Factor 1 in Iteration 1

⟨⟨gpmdb_aa⟩⟩	⟨⟨pepseeker_db, pepseeker_location⟩⟩
⟨⟨gpmdb_aa, gpmdb_aaid⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_ProcessNo⟩⟩
⟨⟨gpmdb_aa, gpmdb_pepid⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_mp⟩⟩
⟨⟨gpmdb_aa, gpmdb_type⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_NM⟩⟩
⟨⟨gpmdb_aa, gpmdb_at⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_SEGT⟩⟩
⟨⟨gpmdb_aa, gpmdb_modified⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_SEGTU⟩⟩
⟨⟨gpmdb_aa, gpmdb_pm⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_LTOL⟩⟩
⟨⟨gpmdb_bad_file_del⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_TOLU⟩⟩
⟨⟨gpmdb_bad_file_del, gpmdb_file⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_ITH⟩⟩
⟨⟨gpmdb_distinctseq⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_ITOLU⟩⟩
⟨⟨gpmdb_distinctseq, gpmdb_seq⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_DB⟩⟩
⟨⟨gpmdb_fullpeptide⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_MASS⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_pepid⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_CLE⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_proid⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_PEAK⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_seq⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_QUE⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_mh⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_TWO⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_expect⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_SEARCH⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_start⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_INTERMEDIATE⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_end⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_REPORT⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_charge⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_OVERVIEW⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_delta⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_FORMAT⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_dida⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_FORMVER⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_aastring⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_FRAG⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_mh2⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_PRECURSOR⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_ACCESSION⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_pepid⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_REPTYPE⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_proid⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_SUBCLUSTER⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_seq⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_ICAT⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_mh⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_INSTRUMENT⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_expect⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_ERRORTOLERANT⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_start⟩⟩	⟨⟨pepseeker_fileparameters, pepseeker_Useremail⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_end⟩⟩	⟨⟨pepseeker_fp_to_db⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_charge⟩⟩	⟨⟨pepseeker_fp_to_db, pepseeker_fp_id⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_delta⟩⟩	⟨⟨pepseeker_fp_to_db, pepseeker_db_id⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_dida⟩⟩	⟨⟨pepseeker_iontable⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_aastring⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Id⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_mh2⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Immon⟩⟩
⟨⟨gpmdb_paths⟩⟩	⟨⟨pepseeker_iontable, pepseeker_A⟩⟩
⟨⟨gpmdb_paths, gpmdb_pathid⟩⟩	⟨⟨pepseeker_iontable, pepseeker_AStar⟩⟩
⟨⟨gpmdb_paths, gpmdb_protocol⟩⟩	⟨⟨pepseeker_iontable, pepseeker_B⟩⟩
⟨⟨gpmdb_paths, gpmdb_server⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Bstar⟩⟩
⟨⟨gpmdb_paths, gpmdb_localpath⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Bstarplusplus⟩⟩
⟨⟨gpmdb_paths, gpmdb_relpath⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Bzero⟩⟩
⟨⟨gpmdb_pep_temp⟩⟩	⟨⟨pepseeker_iontable, pepseeker_BZeroplusplus⟩⟩
⟨⟨gpmdb_pep_temp, gpmdb_pepid⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Y⟩⟩
⟨⟨gpmdb_pep_temp, gpmdb_seq⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Yplusplus⟩⟩
⟨⟨gpmdb_peptide, gpmdb_pepid⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Ystar⟩⟩
⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Ystarplusplus⟩⟩
⟨⟨gpmdb_peptide, gpmdb_mh⟩⟩	⟨⟨pepseeker_iontable, pepseeker_YZero⟩⟩
⟨⟨gpmdb_peptide, gpmdb_start⟩⟩	⟨⟨pepseeker_iontable, pepseeker_YZeroplusplus⟩⟩
⟨⟨gpmdb_peptide, gpmdb_end⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Bplusplus⟩⟩
⟨⟨gpmdb_peptide, gpmdb_charge⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Aplusplus⟩⟩
⟨⟨gpmdb_peptide, gpmdb_delta⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Astarplusplus⟩⟩
⟨⟨gpmdb_peptide, gpmdb_dida⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Azero⟩⟩
⟨⟨gpmdb_peptide, gpmdb_didb⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Matches⟩⟩
⟨⟨gpmdb_peptide, gpmdb_didc⟩⟩	⟨⟨pepseeker_lastsession⟩⟩
⟨⟨gpmdb_peptide_word_index⟩⟩	⟨⟨pepseeker_lastsession, pepseeker_Id⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_keyid⟩⟩	⟨⟨pepseeker_lastsession, pepseeker_url⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_word⟩⟩	⟨⟨pepseeker_mascotexpect⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_pepid_list⟩⟩	⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩

⟨⟨gpmdb_peptide_word_index, gpmdb_ts_created⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩
⟨⟨gpmdb_peptide_words⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_MassNo⟩⟩
⟨⟨gpmdb_peptide_words, gpmdb_seq_word⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_MissCleav⟩⟩
⟨⟨gpmdb_peptide_words, gpmdb_pepid_list⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_MrCalc⟩⟩
⟨⟨gpmdb_project⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Delta⟩⟩
⟨⟨gpmdb_project, gpmdb_projectid⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_MrExpct⟩⟩
⟨⟨gpmdb_project, gpmdb_resultid⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_startresidue⟩⟩
⟨⟨gpmdb_project, gpmdb_name⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_endresidue⟩⟩
⟨⟨gpmdb_project, gpmdb_institution⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Score⟩⟩
⟨⟨gpmdb_project, gpmdb_email⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_ions⟩⟩
⟨⟨gpmdb_project, gpmdb_project⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Rank⟩⟩
⟨⟨gpmdb_project, gpmdb_description⟩⟩	⟨⟨pepseeker_peptideprophet⟩⟩
⟨⟨gpmdb_proseq⟩⟩	⟨⟨pepseeker_peptideprophet, pepseeker_peptidehit_Id⟩⟩
⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩	⟨⟨pepseeker_peptideprophet, pepseeker_tpp_pvalue⟩⟩
⟨⟨gpmdb_proseq, gpmdb_label_aux⟩⟩	⟨⟨pepseeker_proteinhit, pepseeker_Id⟩⟩
⟨⟨gpmdb_proseq, gpmdb_rf⟩⟩	⟨⟨pepseeker_proteinhit, pepseeker_ProteinScore⟩⟩
⟨⟨gpmdb_protein, gpmdb_proid⟩⟩	⟨⟨pepseeker_proteinscore⟩⟩
⟨⟨gpmdb_protein, gpmdb_expect⟩⟩	⟨⟨pepseeker_proteinscore, pepseeker_Id⟩⟩
⟨⟨gpmdb_protein, gpmdb_pida⟩⟩	⟨⟨pepseeker_proteinscore, pepseeker_querypeptides⟩⟩
⟨⟨gpmdb_protein, gpmdb_pidb⟩⟩	⟨⟨pepseeker_proteinscore, pepseeker_Matchedpeptides⟩⟩
⟨⟨gpmdb_protein, gpmdb_uid⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Id⟩⟩
⟨⟨gpmdb_proteinrevision⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_queryno⟩⟩
⟨⟨gpmdb_proteinrevision, gpmdb_prorevid⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Precursor⟩⟩
⟨⟨gpmdb_proteinrevision, gpmdb_proid⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Title⟩⟩
⟨⟨gpmdb_proteinrevision, gpmdb_label⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Mass_min⟩⟩
⟨⟨gpmdb_result, gpmdb_resultid⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Mass_Max⟩⟩
⟨⟨gpmdb_result, gpmdb_pathid⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_int_min⟩⟩
⟨⟨gpmdb_result, gpmdb_file⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Int_max⟩⟩
⟨⟨gpmdb_result, gpmdb_completed⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_num_vals⟩⟩
⟨⟨gpmdb_result, gpmdb_active⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_num_used⟩⟩
⟨⟨gpmdb_result, gpmdb_rating⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_fileparameters⟩⟩
⟨⟨gpmdb_result, gpmdb_comments⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_searchmassescol⟩⟩
⟨⟨pepseeker_db⟩⟩	⟨⟨pepseeker_species⟩⟩
⟨⟨pepseeker_db, pepseeker_id⟩⟩	⟨⟨pepseeker_species, pepseeker_ProteinId⟩⟩
⟨⟨pepseeker_db, pepseeker_name⟩⟩	⟨⟨pepseeker_species, pepseeker_fp_id⟩⟩
⟨⟨pepseeker_db, pepseeker_release⟩⟩	⟨⟨pepseeker_species, pepseeker_species_id⟩⟩
⟨⟨pepseeker_db, pepseeker_release_date⟩⟩	⟨⟨pepseeker_yeastacc⟩⟩
⟨⟨pepseeker_db, pepseeker_system_date⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_id⟩⟩
⟨⟨pepseeker_db, pepseeker_entries⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_accession⟩⟩
⟨⟨pepseeker_db, pepseeker_residue_type⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_oln⟩⟩
⟨⟨pepseeker_yeastacc, pepseeker_yeast_acc_id⟩⟩	

Table E.5: Elements not-Satisfying Quality Factor 1 in Iteration 1

Factor 4

Satisfying Items	not-Satisfying Items
⟨⟨gpmdb_peptide, gpmdb_seq⟩⟩	⟨⟨gpmdb_peptide, gpmdb_pepid⟩⟩
⟨⟨gpmdb_peptide⟩⟩	⟨⟨gpmdb_aa, gpmdb_pepid⟩⟩
⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩	⟨⟨gpmdb_fullpeptide, gpmdb_pepid⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩	⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_pepid⟩⟩
⟨⟨pedro_peptidehit, pedro_db_search⟩⟩	⟨⟨pedro_peptidehit⟩⟩
⟨⟨pedro_peptidehit, pedro_score⟩⟩	⟨⟨pedro_peptidehit, pedro_id⟩⟩
⟨⟨pedro_peptidehit, pedro_sequence⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩
⟨⟨pedro_protein, pedro_accession_num⟩⟩	⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩
⟨⟨pedro_protein, pedro_description⟩⟩	⟨⟨pepseeker_peptideprophet, pepseeker_peptidehit_Id⟩⟩
⟨⟨pedro_protein, pedro_organism⟩⟩	⟨⟨gpmdb_protein⟩⟩
⟨⟨pedro_proteinhit, pedro_db_search⟩⟩	⟨⟨gpmdb_protein, gpmdb_proid⟩⟩

⟨⟨pedro_proteinhit, pedro_protein⟩⟩	⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_pepseq⟩⟩	⟨⟨gpmdb_fullpeptide, gpmdb_proid⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_proteinscore⟩⟩	⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_proid⟩⟩
⟨⟨pepseeker_peptidehit⟩⟩	⟨⟨gpmdb_proseq, gpmdb_label⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_fileparameters⟩⟩	⟨⟨pedro_protein⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_ProteinID⟩⟩	⟨⟨pedro_protein, pedro_id⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_value_type⟩⟩	⟨⟨pedro_proteinhit, pedro_id⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_error_type⟩⟩	⟨⟨pepseeker_proteinhit⟩⟩
	⟨⟨pepseeker_proteinhit, pepseeker_Id⟩⟩
	⟨⟨pepseeker_species, pepseeker_ProteinID⟩⟩
	⟨⟨gpmdb_fullpeptide, gpmdb_seq⟩⟩
	⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_seq⟩⟩
	⟨⟨gpmdb_aa, gpmdb_type⟩⟩
	⟨⟨gpmdb_aa⟩⟩
	⟨⟨gpmdb_aa, gpmdb_aaid⟩⟩
	⟨⟨gpmdb_proseq⟩⟩
	⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩
	⟨⟨gpmdb_proseq, gpmdb_seq⟩⟩
	⟨⟨pedro_protein, pedro_sequence⟩⟩
	⟨⟨pepseeker_proteinscore, pepseeker_Id⟩⟩
	⟨⟨pepseeker_proteinscore, pepseeker_Score⟩⟩
	⟨⟨pepseeker_proteinhit, pepseeker_ProteinScore⟩⟩
	⟨⟨pedro_peaklist, pedro_mass_value_type⟩⟩
	⟨⟨pepseeker_proteinhit, pepseeker_Mass⟩⟩
	⟨⟨pepseeker_peptidehit, pepseeker_MassNo⟩⟩
	⟨⟨pepseeker_fileparameters, pepseeker_ITOLU⟩⟩
	⟨⟨pepseeker_fileparameters, pepseeker_MASS⟩⟩
	⟨⟨pepseeker_fileparameters, pepseeker_TOLU⟩⟩

Table E.6: Elements Satisfying and not-Satisfying Quality Factor 4 in Iteration 1

Factor 7

⟨⟨pedro_fk_dbsearch_dbsearchparameters1, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩, pedro_dbsearchparameters, ⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩⟩⟩
⟨⟨pedro_fk_proteinhit_dbsearch1, pedro_proteinhit, ⟨⟨pedro_proteinhit, pedro_db_search⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_id⟩⟩⟩⟩
⟨⟨gpmdb_result⟩⟩
⟨⟨pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_fileparameters, Id⟩⟩
⟨⟨gpmdb_protein⟩⟩
⟨⟨gpmdb_protein, proseqid⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_fileparameters⟩⟩
⟨⟨gpmdb_peptide, gpmdb_pepid⟩⟩
⟨⟨gpmdb_fk_peptide_protein1, gpmdb_peptide, ⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩, gpmdb_protein, ⟨⟨gpmdb_protein, gpmdb_proid⟩⟩⟩⟩
⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩
⟨⟨gpmdb_protein, gpmdb_proid⟩⟩
⟨⟨gpmdb_fk_protein_proseq, gpmdb_protein, ⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩, gpmdb_proseq, ⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩⟩⟩
⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩
⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩
⟨⟨gpmdb_fk_protein_result1, gpmdb_protein, ⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩, gpmdb_result, ⟨⟨gpmdb_result, gpmdb_resultid⟩⟩⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩
⟨⟨gpmdb_result, gpmdb_resultid⟩⟩
⟨⟨pedro_fk_dbsearch_dbsearchparameters1, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩, pedro_dbsearchparameters, ⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩⟩⟩
⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩
⟨⟨pedro_fk_chromatogrampoint_peak1, pedro_chromatogrampoint, ⟨⟨pedro_chromatogrampoint, pedro_peak⟩⟩, pedro_peak, ⟨⟨pedro_peak, pedro_id⟩⟩⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_peak⟩⟩
⟨⟨pedro_peak, pedro_id⟩⟩
⟨⟨pedro_fk_dbsearch_peaklist1, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_dbsearch, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_listprocessing_peaklist1, pedro_listprocessing, ⟨⟨pedro_listprocessing, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_listprocessing, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_msmsfraction_peaklist1, pedro_msmsfraction, ⟨⟨pedro_msmsfraction, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_msmsfraction, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_peak_peaklist, pedro_peak, ⟨⟨pedro_peak, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_peak, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_peaklist_massspecmachine1, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_mass_spec_experiment⟩⟩, pedro_massspecmachine, ⟨⟨pedro_massspecmachine, pedro_id⟩⟩⟩⟩
⟨⟨pedro_peaklist, pedro_mass_spec_experiment⟩⟩
⟨⟨pedro_massspecmachine, pedro_id⟩⟩
⟨⟨pedro_fk_peptidehit_dbsearch1, pedro_peptidehit, ⟨⟨pedro_peptidehit, pedro_db_search⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_id⟩⟩⟩⟩

⟨⟨pedro_peptidehit, pedro_db_search⟩⟩
⟨⟨pedro_dbsearch, pedro_id⟩⟩
⟨⟨pedro_fk_proteinhit_dbsearch1, pedro_proteinhit, ⟨⟨pedro_proteinhit, pedro_db_search⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_id⟩⟩⟩⟩
⟨⟨pedro_proteinhit, pedro_db_search⟩⟩
⟨⟨pedro_dbsearch, pedro_id⟩⟩
⟨⟨pedro_fk_tandemsequencedata_dbsearch1, pedro_tandemsequencedata, ⟨⟨pedro_tandemsequencedata, pedro_db_search_parameters⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩
⟨⟨pepseeker_fk_mascotexpect_peptidehit1, pepseeker_mascotexpect, ⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩, pepseeker_peptidehit, ⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩⟩⟩
⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩
⟨⟨pepseeker_fk_searchmasses_fileparameters, pepseeker_searchmasses, ⟨⟨pepseeker_searchmasses, pepseeker_fileparameters⟩⟩, pepseeker_fileparameters, ⟨⟨pepseeker_fileparameters, pepseeker_Id⟩⟩⟩⟩
⟨⟨pepseeker_searchmasses, pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Id⟩⟩

Table E.7: Elements Satisfying Quality Factor 7 in Iteration 1

⟨⟨GS1_fk_proteinhit_protein1, GS1_proteinhit, ⟨⟨GS1_proteinhit, GS1_protein⟩⟩, GS1_protein, ⟨⟨GS1_protein, GS1_id⟩⟩⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_ProteinID⟩⟩
⟨⟨pepseeker_proteinhit⟩⟩
⟨⟨GS1_proteinhit, GS1_protein⟩⟩
⟨⟨GS1_protein, GS1_id⟩⟩

Table E.8: Elements not-Satisfying Quality Factor 7 in Iteration 1

E.4.2 Quality Measurement for Iteration 2

Factor 1

⟨⟨gpmdb_aa, gpmdb_aaid⟩⟩	⟨⟨pedro_listprocessing, pedro_background_threshold⟩⟩
⟨⟨gpmdb_aa, gpmdb_at⟩⟩	⟨⟨pedro_listprocessing, pedro_id⟩⟩
⟨⟨gpmdb_aa, gpmdb_modified⟩⟩	⟨⟨pedro_listprocessing, pedro_peak_list⟩⟩
⟨⟨gpmdb_aa, gpmdb_pepid⟩⟩	⟨⟨pedro_listprocessing, pedro_smoothing_process⟩⟩
⟨⟨gpmdb_aa, gpmdb_pm⟩⟩	⟨⟨pedro_maldi, pedro_acceleration_voltage⟩⟩
⟨⟨gpmdb_aa, gpmdb_type⟩⟩	⟨⟨pedro_maldi, pedro_grid_voltage⟩⟩
⟨⟨gpmdb_bad_file_del, gpmdb_file⟩⟩	⟨⟨pedro_maldi, pedro_id⟩⟩
⟨⟨gpmdb_distinctseq, gpmdb_seq⟩⟩	⟨⟨pedro_maldi, pedro_ion_mode⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_aastring⟩⟩	⟨⟨pedro_maldi, pedro_ion_source⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_charge⟩⟩	⟨⟨pedro_maldi, pedro_laser_power⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_delta⟩⟩	⟨⟨pedro_maldi, pedro_laser_wavelength⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_dida⟩⟩	⟨⟨pedro_maldi, pedro_matrix_type⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_end⟩⟩	⟨⟨pedro_massspecexperiment, pedro_analyte_processing_step⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_expect⟩⟩	⟨⟨pedro_massspecexperiment, pedro_description⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_mh⟩⟩	⟨⟨pedro_massspecexperiment, pedro_id⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_mh2⟩⟩	⟨⟨pedro_massspecexperiment, pedro_parameters_file⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_pepid⟩⟩	⟨⟨pedro_massspecmachine, pedro_id⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_proid⟩⟩	⟨⟨pedro_massspecmachine, pedro_ion_source⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_seq⟩⟩	⟨⟨pedro_massspecmachine, pedro_manufacturer⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_start⟩⟩	⟨⟨pedro_massspecmachine, pedro_model_name⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_aastring⟩⟩	⟨⟨pedro_massspecmachine, pedro_software_version⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_charge⟩⟩	⟨⟨pedro_mobilephasecomponent, pedro_concentration⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_delta⟩⟩	⟨⟨pedro_mobilephasecomponent, pedro_description⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_dida⟩⟩	⟨⟨pedro_mobilephasecomponent, pedro_id⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_end⟩⟩	⟨⟨pedro_mobilephasecomponent, pedro_lc_column⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_expect⟩⟩	⟨⟨pedro_msmsfraction, pedro_id⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_mh⟩⟩	⟨⟨pedro_msmsfraction, pedro_peak_list⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_mh2⟩⟩	⟨⟨pedro_msmsfraction, pedro_plus_or_minus⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_pepid⟩⟩	⟨⟨pedro_msmsfraction, pedro_target_m_to_z⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_proid⟩⟩	⟨⟨pedro_mzanalysis, pedro_detection⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_seq⟩⟩	⟨⟨pedro_mzanalysis, pedro_id⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_start⟩⟩	⟨⟨pedro_mzanalysis, pedro_type⟩⟩
⟨⟨gpmdb_paths, gpmdb_localpath⟩⟩	⟨⟨pedro_ontologyentry, pedro_category⟩⟩
⟨⟨gpmdb_paths, gpmdb_pathid⟩⟩	⟨⟨pedro_ontologyentry, pedro_description⟩⟩
⟨⟨gpmdb_paths, gpmdb_protocol⟩⟩	⟨⟨pedro_ontologyentry, pedro_id⟩⟩
⟨⟨gpmdb_paths, gpmdb_relpath⟩⟩	⟨⟨pedro_ontologyentry, pedro_value⟩⟩
⟨⟨gpmdb_paths, gpmdb_server⟩⟩	⟨⟨pedro_organism, pedro_id⟩⟩
⟨⟨gpmdb_pep_temp, gpmdb_pepid⟩⟩	⟨⟨pedro_organism, pedro_relevant_genotype⟩⟩
⟨⟨gpmdb_pep_temp, gpmdb_seq⟩⟩	⟨⟨pedro_organism, pedro_species_name⟩⟩
⟨⟨gpmdb_peptide, gpmdb_charge⟩⟩	⟨⟨pedro_organism, pedro_strain_identifier⟩⟩
⟨⟨gpmdb_peptide, gpmdb_delta⟩⟩	⟨⟨pedro_otheranalyte, pedro_id⟩⟩
⟨⟨gpmdb_peptide, gpmdb_dida⟩⟩	⟨⟨pedro_otheranalyte, pedro_name⟩⟩
⟨⟨gpmdb_peptide, gpmdb_didb⟩⟩	⟨⟨pedro_otheranalyte, pedro_other_analyte_processing_step⟩⟩

⟨⟨gpmdb_peptide, gpmdb_didc⟩⟩
⟨⟨gpmdb_peptide, gpmdb_end⟩⟩
⟨⟨gpmdb_peptide, gpmdb_expect⟩⟩
⟨⟨gpmdb_peptide, gpmdb_mh⟩⟩
⟨⟨gpmdb_peptide, gpmdb_pepid⟩⟩
⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩
⟨⟨gpmdb_peptide, gpmdb_seq⟩⟩
⟨⟨gpmdb_peptide, gpmdb_start⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_keyid⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_pepid_list⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_ts_created⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_word⟩⟩
⟨⟨gpmdb_peptide_words, gpmdb_pepid_list⟩⟩
⟨⟨gpmdb_peptide_words, gpmdb_seq_word⟩⟩
⟨⟨gpmdb_project, gpmdb_description⟩⟩
⟨⟨gpmdb_project, gpmdb_email⟩⟩
⟨⟨gpmdb_project, gpmdb_institution⟩⟩
⟨⟨gpmdb_project, gpmdb_name⟩⟩
⟨⟨gpmdb_project, gpmdb_project⟩⟩
⟨⟨gpmdb_project, gpmdb_projectid⟩⟩
⟨⟨gpmdb_project, gpmdb_resultid⟩⟩
⟨⟨gpmdb_proseq, gpmdb_label_aux⟩⟩
⟨⟨gpmdb_proseq, gpmdb_label⟩⟩
⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩
⟨⟨gpmdb_proseq, gpmdb_rf⟩⟩
⟨⟨gpmdb_proseq, gpmdb_seq⟩⟩
⟨⟨gpmdb_protein, gpmdb_expect⟩⟩
⟨⟨gpmdb_protein, gpmdb_pida⟩⟩
⟨⟨gpmdb_protein, gpmdb_pidb⟩⟩
⟨⟨gpmdb_protein, gpmdb_proid⟩⟩
⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩
⟨⟨gpmdb_protein, gpmdb_uid⟩⟩
⟨⟨gpmdb_proteinrevision, gpmdb_label⟩⟩
⟨⟨gpmdb_proteinrevision, gpmdb_proid⟩⟩
⟨⟨gpmdb_proteinrevision, gpmdb_prorevid⟩⟩
⟨⟨gpmdb_result, gpmdb_active⟩⟩
⟨⟨gpmdb_result, gpmdb_comments⟩⟩
⟨⟨gpmdb_result, gpmdb_completed⟩⟩
⟨⟨gpmdb_result, gpmdb_file⟩⟩
⟨⟨gpmdb_result, gpmdb_pathid⟩⟩
⟨⟨gpmdb_result, gpmdb_rating⟩⟩
⟨⟨gpmdb_result, gpmdb_resultid⟩⟩
⟨⟨gpmdb_result, gpmdb_tandemversion⟩⟩
⟨⟨gpmdb_aa⟩⟩
⟨⟨gpmdb_bad_file_del⟩⟩
⟨⟨gpmdb_distinctseq⟩⟩
⟨⟨gpmdb_fullpeptide⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic⟩⟩
⟨⟨gpmdb_paths⟩⟩
⟨⟨gpmdb_pep_temp⟩⟩
⟨⟨gpmdb_peptide_word_index⟩⟩
⟨⟨gpmdb_peptide_words⟩⟩
⟨⟨gpmdb_peptide⟩⟩
⟨⟨gpmdb_project⟩⟩
⟨⟨gpmdb_proseq⟩⟩
⟨⟨gpmdb_protein⟩⟩
⟨⟨gpmdb_proteinrevision⟩⟩
⟨⟨gpmdb_result⟩⟩
⟨⟨pedro_analyteprocessingstep, pedro_id⟩⟩
⟨⟨pedro_otheranalyte_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_otheranalyte_otm_analyteps, pedro_other_analyte⟩⟩
⟨⟨pedro_otheranalyte_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_otheranalyte_otm_ontent, pedro_other_analyte⟩⟩
⟨⟨pedro_otheranalyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_otheranalyteps, pedro_id⟩⟩
⟨⟨pedro_otheranalyteps, pedro_name⟩⟩
⟨⟨pedro_otheranalyteps_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_otheranalyteps_otm_ontent, pedro_other_analyte_processing_step⟩⟩
⟨⟨pedro_otherionisation, pedro_id⟩⟩
⟨⟨pedro_otherionisation, pedro_ion_source⟩⟩
⟨⟨pedro_otherionisation, pedro_name⟩⟩
⟨⟨pedro_otherionisation_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_otherionisation_otm_ontent, pedro_other_ionisation⟩⟩
⟨⟨pedro_othermzanalysis, pedro_id⟩⟩
⟨⟨pedro_othermzanalysis, pedro_mz_analysis⟩⟩
⟨⟨pedro_othermzanalysis, pedro_name⟩⟩
⟨⟨pedro_othermzanalysis_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_othermzanalysis_otm_ontent, pedro_other_mz_analysis⟩⟩
⟨⟨pedro_peak, pedro_abundance⟩⟩
⟨⟨pedro_peak, pedro_id⟩⟩
⟨⟨pedro_peak, pedro_m_to_z⟩⟩
⟨⟨pedro_peak, pedro_multiplicity⟩⟩
⟨⟨pedro_peak, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_description⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_peaklist, pedro_list_type⟩⟩
⟨⟨pedro_peaklist, pedro_mass_spec_experiment⟩⟩
⟨⟨pedro_peaklist, pedro_mass_value_type⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_area_under_curve⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_background_threshold⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_id⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_peak_description⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_peak⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_resolution⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_sister_peak_reference⟩⟩
⟨⟨pedro_peakspecificchromint, pedro_software_version⟩⟩
⟨⟨pedro_peptidehit, pedro_db_search⟩⟩
⟨⟨pedro_peptidehit, pedro_id⟩⟩
⟨⟨pedro_peptidehit, pedro_information⟩⟩
⟨⟨pedro_peptidehit, pedro_probability⟩⟩
⟨⟨pedro_peptidehit, pedro_score_type⟩⟩
⟨⟨pedro_peptidehit, pedro_score⟩⟩
⟨⟨pedro_peptidehit, pedro_sequence⟩⟩
⟨⟨pedro_peptidehit_mtm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_peptidehit_mtm_ontent, pedro_peptide_hit_db_search⟩⟩
⟨⟨pedro_peptidehit_mtm_ontent, pedro_peptide_hit_id⟩⟩
⟨⟨pedro_percentx, pedro_gradient_step_id⟩⟩
⟨⟨pedro_percentx, pedro_gradient_step_lc_column⟩⟩
⟨⟨pedro_percentx, pedro_id⟩⟩
⟨⟨pedro_percentx, pedro_mobile_phase_component⟩⟩
⟨⟨pedro_percentx, pedro_percentage⟩⟩
⟨⟨pedro_protein, pedro_accession_num⟩⟩
⟨⟨pedro_protein, pedro_description⟩⟩
⟨⟨pedro_protein, pedro_gene_name⟩⟩
⟨⟨pedro_protein, pedro_id⟩⟩
⟨⟨pedro_protein, pedro_modifications⟩⟩
⟨⟨pedro_protein, pedro_orf_number⟩⟩
⟨⟨pedro_protein, pedro_organism⟩⟩
⟨⟨pedro_protein, pedro_predicted_mass⟩⟩

⟨⟨pedro_analyteprocessingstep, pedro_input_type⟩⟩
⟨⟨pedro_analyteprocessingstep, pedro_processing_type⟩⟩
⟨⟨pedro_assaydatapoint, pedro_id⟩⟩
⟨⟨pedro_assaydatapoint, pedro_lc_column⟩⟩
⟨⟨pedro_assaydatapoint, pedro_protein_assay⟩⟩
⟨⟨pedro_assaydatapoint, pedro_time⟩⟩
⟨⟨pedro_band, pedro_annotation_source⟩⟩
⟨⟨pedro_band, pedro_annotation⟩⟩
⟨⟨pedro_band, pedro_apparent_mass⟩⟩
⟨⟨pedro_band, pedro_area⟩⟩
⟨⟨pedro_band, pedro_gel_1d⟩⟩
⟨⟨pedro_band, pedro_id⟩⟩
⟨⟨pedro_band, pedro_intensity⟩⟩
⟨⟨pedro_band, pedro_lane_number⟩⟩
⟨⟨pedro_band, pedro_local_background⟩⟩
⟨⟨pedro_band, pedro_normalisation⟩⟩
⟨⟨pedro_band, pedro_normalised_volume⟩⟩
⟨⟨pedro_band, pedro_pixel_radius⟩⟩
⟨⟨pedro_band, pedro_pixel_x_coord⟩⟩
⟨⟨pedro_band, pedro_pixel_y_coord⟩⟩
⟨⟨pedro_band, pedro_volume⟩⟩
⟨⟨pedro_band_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_band_otm_analyteps, pedro_band_gel_1d⟩⟩
⟨⟨pedro_band_otm_analyteps, pedro_band_id⟩⟩
⟨⟨pedro_boundarypoint, pedro_id⟩⟩
⟨⟨pedro_boundarypoint, pedro_pixel_x_coord⟩⟩
⟨⟨pedro_boundarypoint, pedro_pixel_y_coord⟩⟩
⟨⟨pedro_boundarypoint, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_boundarypoint, pedro_spot_id⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_derivatisations⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_digestion⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_id⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_id⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_ion_count⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_peak⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_time_point⟩⟩
⟨⟨pedro_collisioncell, pedro_collision_offset⟩⟩
⟨⟨pedro_collisioncell, pedro_gas_pressure⟩⟩
⟨⟨pedro_collisioncell, pedro_gas_type⟩⟩
⟨⟨pedro_collisioncell, pedro_id⟩⟩
⟨⟨pedro_collisioncell, pedro_mz_analysis⟩⟩
⟨⟨pedro_dbsearch, pedro_c_terminal_aa⟩⟩
⟨⟨pedro_dbsearch, pedro_count_of_specific_aa⟩⟩
⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearch, pedro_id_date⟩⟩
⟨⟨pedro_dbsearch, pedro_id⟩⟩
⟨⟨pedro_dbsearch, pedro_n_terminal_aa⟩⟩
⟨⟨pedro_dbsearch, pedro_name_of_counted_aa⟩⟩
⟨⟨pedro_dbsearch, pedro_peak_list⟩⟩
⟨⟨pedro_dbsearch, pedro_regex_pattern⟩⟩
⟨⟨pedro_dbsearch, pedro_username⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_accurate_mass_mode⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_database_date⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_database_name⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_fixed_modifications⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_fragment_ion_tolerance⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_icat_option⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_error_type⟩⟩
⟨⟨pedro_protein, pedro_predicted_pi⟩⟩
⟨⟨pedro_protein, pedro_sequence⟩⟩
⟨⟨pedro_protein, pedro_synonyms⟩⟩
⟨⟨pedro_proteinhit, pedro_all_peptides_matched⟩⟩
⟨⟨pedro_proteinhit, pedro_component_peptides⟩⟩
⟨⟨pedro_proteinhit, pedro_db_search⟩⟩
⟨⟨pedro_proteinhit, pedro_id⟩⟩
⟨⟨pedro_proteinhit, pedro_masses_matched⟩⟩
⟨⟨pedro_proteinhit, pedro_protein⟩⟩
⟨⟨pedro_proteinhit, pedro_score_type⟩⟩
⟨⟨pedro_proteinhit, pedro_score⟩⟩
⟨⟨pedro_quadrupole, pedro_description⟩⟩
⟨⟨pedro_quadrupole, pedro_id⟩⟩
⟨⟨pedro_quadrupole, pedro_mz_analysis⟩⟩
⟨⟨pedro_relatedgelitem, pedro_band_gel_1d⟩⟩
⟨⟨pedro_relatedgelitem, pedro_band_id⟩⟩
⟨⟨pedro_relatedgelitem, pedro_description⟩⟩
⟨⟨pedro_relatedgelitem, pedro_gel_reference⟩⟩
⟨⟨pedro_relatedgelitem, pedro_id⟩⟩
⟨⟨pedro_relatedgelitem, pedro_item_reference⟩⟩
⟨⟨pedro_relatedgelitem, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_relatedgelitem, pedro_spot_id⟩⟩
⟨⟨pedro_relgelitem_mtm_proteinhit, pedro_protein_hit⟩⟩
⟨⟨pedro_relgelitem_mtm_proteinhit, pedro_related_gel_item⟩⟩
⟨⟨pedro_sample, pedro_experiment⟩⟩
⟨⟨pedro_sample, pedro_experimenter⟩⟩
⟨⟨pedro_sample, pedro_sample_date⟩⟩
⟨⟨pedro_sample, pedro_sample_id⟩⟩
⟨⟨pedro_sample_mtm_sampleorigin, pedro_sample_origin⟩⟩
⟨⟨pedro_sample_mtm_sampleorigin, pedro_sample⟩⟩
⟨⟨pedro_sample_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_sample_otm_analyteps, pedro_sample⟩⟩
⟨⟨pedro_sampleorigin, pedro_cell_component⟩⟩
⟨⟨pedro_sampleorigin, pedro_cell_cycle_phase⟩⟩
⟨⟨pedro_sampleorigin, pedro_cell_type⟩⟩
⟨⟨pedro_sampleorigin, pedro_condition_degree⟩⟩
⟨⟨pedro_sampleorigin, pedro_description⟩⟩
⟨⟨pedro_sampleorigin, pedro_environment⟩⟩
⟨⟨pedro_sampleorigin, pedro_id⟩⟩
⟨⟨pedro_sampleorigin, pedro_metabolic_label⟩⟩
⟨⟨pedro_sampleorigin, pedro_organism⟩⟩
⟨⟨pedro_sampleorigin, pedro_sample_condition⟩⟩
⟨⟨pedro_sampleorigin, pedro_tagging_process⟩⟩
⟨⟨pedro_sampleorigin, pedro_technique⟩⟩
⟨⟨pedro_sampleorigin, pedro_tissue_type⟩⟩
⟨⟨pedro_spot, pedro_annotation_source⟩⟩
⟨⟨pedro_spot, pedro_annotation⟩⟩
⟨⟨pedro_spot, pedro_apparent_mass⟩⟩
⟨⟨pedro_spot, pedro_apparent_pi⟩⟩
⟨⟨pedro_spot, pedro_area⟩⟩
⟨⟨pedro_spot, pedro_gel_2d⟩⟩
⟨⟨pedro_spot, pedro_id⟩⟩
⟨⟨pedro_spot, pedro_intensity⟩⟩
⟨⟨pedro_spot, pedro_local_background⟩⟩
⟨⟨pedro_spot, pedro_normalisation⟩⟩
⟨⟨pedro_spot, pedro_normalised_volume⟩⟩
⟨⟨pedro_spot, pedro_pixel_radius⟩⟩
⟨⟨pedro_spot, pedro_pixel_x_coord⟩⟩
⟨⟨pedro_spot, pedro_pixel_y_coord⟩⟩
⟨⟨pedro_spot, pedro_volume⟩⟩

⟨⟨pedro_dbsearchparameters, pedro_mass_error⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_value_type⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_max_missed_cleavages⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_parameters_file⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_peptide_mass_tolerance⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_program⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_protonated⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_taxonomical_filter⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_variable_modifications⟩⟩
⟨⟨pedro_dbsearchpars_otm_ontent, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearchpars_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_detection, pedro_id⟩⟩
⟨⟨pedro_detection, pedro_type⟩⟩
⟨⟨pedro_digegel, pedro_dye_type⟩⟩
⟨⟨pedro_digegel, pedro_excitation_wavelength⟩⟩
⟨⟨pedro_digegel, pedro_exposure_time⟩⟩
⟨⟨pedro_digegel, pedro_gel_1d⟩⟩
⟨⟨pedro_digegel, pedro_gel_2d⟩⟩
⟨⟨pedro_digegel, pedro_id⟩⟩
⟨⟨pedro_digegel, pedro_tiff_image⟩⟩
⟨⟨pedro_digegelitem, pedro_band_gel_1d⟩⟩
⟨⟨pedro_digegelitem, pedro_band_id⟩⟩
⟨⟨pedro_digegelitem, pedro_dye_type⟩⟩
⟨⟨pedro_digegelitem, pedro_id⟩⟩
⟨⟨pedro_digegelitem, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_digegelitem, pedro_spot_id⟩⟩
⟨⟨pedro_electrospray, pedro_cone_voltage⟩⟩
⟨⟨pedro_electrospray, pedro_id⟩⟩
⟨⟨pedro_electrospray, pedro_interface_manufacturer⟩⟩
⟨⟨pedro_electrospray, pedro_ion_source⟩⟩
⟨⟨pedro_electrospray, pedro_loading_type⟩⟩
⟨⟨pedro_electrospray, pedro_solution_voltage⟩⟩
⟨⟨pedro_electrospray, pedro_solvent⟩⟩
⟨⟨pedro_electrospray, pedro_spray_tip_diameter⟩⟩
⟨⟨pedro_electrospray, pedro_spray_tip_manufacturer⟩⟩
⟨⟨pedro_electrospray, pedro_spray_tip_voltage⟩⟩
⟨⟨pedro_experiment, pedro_hypothesis⟩⟩
⟨⟨pedro_experiment, pedro_id⟩⟩
⟨⟨pedro_experiment, pedro_method_citations⟩⟩
⟨⟨pedro_experiment, pedro_result_citations⟩⟩
⟨⟨pedro_fraction, pedro_end_point⟩⟩
⟨⟨pedro_fraction, pedro_id⟩⟩
⟨⟨pedro_fraction, pedro_lc_column⟩⟩
⟨⟨pedro_fraction, pedro_protein_assay⟩⟩
⟨⟨pedro_fraction, pedro_start_point⟩⟩
⟨⟨pedro_fraction_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_fraction_otm_analyteps, pedro_fraction_id⟩⟩
⟨⟨pedro_fraction_otm_analyteps, pedro_fraction_lc_column⟩⟩
⟨⟨pedro_gel1d, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_gel1d, pedro_annotated_image⟩⟩
⟨⟨pedro_gel1d, pedro_background⟩⟩
⟨⟨pedro_gel1d, pedro_denaturing_agent⟩⟩
⟨⟨pedro_gel1d, pedro_description⟩⟩
⟨⟨pedro_gel1d, pedro_equipment⟩⟩
⟨⟨pedro_gel1d, pedro_id⟩⟩
⟨⟨pedro_gel1d, pedro_in_gel_digestion⟩⟩
⟨⟨pedro_gel1d, pedro_mass_end⟩⟩
⟨⟨pedro_gel1d, pedro_mass_start⟩⟩
⟨⟨pedro_gel1d, pedro_percent_acrylamide⟩⟩
⟨⟨pedro_gel1d, pedro_pixel_size_x⟩⟩
⟨⟨pedro_spot_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_spot_otm_analyteps, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_spot_otm_analyteps, pedro_spot_id⟩⟩
⟨⟨pedro_taggingprocess, pedro_final_volume⟩⟩
⟨⟨pedro_taggingprocess, pedro_id⟩⟩
⟨⟨pedro_taggingprocess, pedro_lysis_buffer⟩⟩
⟨⟨pedro_taggingprocess, pedro_protein_concentration⟩⟩
⟨⟨pedro_taggingprocess, pedro_tag_concentration⟩⟩
⟨⟨pedro_taggingprocess, pedro_tag_purity⟩⟩
⟨⟨pedro_taggingprocess, pedro_tag_type⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_db_search_parameters⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_id⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_sequence⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_source_type⟩⟩
⟨⟨pedro_tof, pedro_id⟩⟩
⟨⟨pedro_tof, pedro_internal_length⟩⟩
⟨⟨pedro_tof, pedro_mz_analysis⟩⟩
⟨⟨pedro_tof, pedro_reflectron_state⟩⟩
⟨⟨pedro_treatedanalyte, pedro_chemical_treatment⟩⟩
⟨⟨pedro_treatedanalyte, pedro_id⟩⟩
⟨⟨pedro_treatedanalyte_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_treatedanalyte_otm_analyteps, pedro_treated_analyte⟩⟩
⟨⟨pedro_analyteprocessingstep⟩⟩
⟨⟨pedro_assaydatapoint⟩⟩
⟨⟨pedro_band_otm_analyteps⟩⟩
⟨⟨pedro_band⟩⟩
⟨⟨pedro_boundarypoint⟩⟩
⟨⟨pedro_chemicaltreatment⟩⟩
⟨⟨pedro_chromatogrampoint⟩⟩
⟨⟨pedro_collisioncell⟩⟩
⟨⟨pedro_dbsearch⟩⟩
⟨⟨pedro_dbsearchparameters⟩⟩
⟨⟨pedro_dbsearchpars_otm_ontent⟩⟩
⟨⟨pedro_detection⟩⟩
⟨⟨pedro_digegel⟩⟩
⟨⟨pedro_digegelitem⟩⟩
⟨⟨pedro_electrospray⟩⟩
⟨⟨pedro_experiment⟩⟩
⟨⟨pedro_fraction_otm_analyteps⟩⟩
⟨⟨pedro_fraction⟩⟩
⟨⟨pedro_gel1d⟩⟩
⟨⟨pedro_gel2d⟩⟩
⟨⟨pedro_gradientstep⟩⟩
⟨⟨pedro_hexapole⟩⟩
⟨⟨pedro_ionsource⟩⟩
⟨⟨pedro_iontrap⟩⟩
⟨⟨pedro_lccolumn⟩⟩
⟨⟨pedro_listprocessing⟩⟩
⟨⟨pedro_maldi⟩⟩
⟨⟨pedro_massspecexperiment⟩⟩
⟨⟨pedro_massspecmachine⟩⟩
⟨⟨pedro_mobilephasecomponent⟩⟩
⟨⟨pedro_msmsfraction⟩⟩
⟨⟨pedro_mzanalysis⟩⟩
⟨⟨pedro_ontologyentry⟩⟩
⟨⟨pedro_organism⟩⟩
⟨⟨pedro_otheranalyte_otm_analyteps⟩⟩
⟨⟨pedro_otheranalyte_otm_ontent⟩⟩
⟨⟨pedro_otheranalyte⟩⟩
⟨⟨pedro_otheranalyteps_otm_ontent⟩⟩

⟨⟨pedro_gel1d, pedro_pixel_size_y⟩⟩
⟨⟨pedro_gel1d, pedro_protein_assay⟩⟩
⟨⟨pedro_gel1d, pedro_raw_image⟩⟩
⟨⟨pedro_gel1d, pedro_run_details⟩⟩
⟨⟨pedro_gel1d, pedro_software_version⟩⟩
⟨⟨pedro_gel1d, pedro_solubilization_buffer⟩⟩
⟨⟨pedro_gel1d, pedro_stain_details⟩⟩
⟨⟨pedro_gel1d, pedro_warped_image⟩⟩
⟨⟨pedro_gel1d, pedro_warping_map⟩⟩
⟨⟨pedro_gel2d, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_gel2d, pedro_annotated_image⟩⟩
⟨⟨pedro_gel2d, pedro_background⟩⟩
⟨⟨pedro_gel2d, pedro_description⟩⟩
⟨⟨pedro_gel2d, pedro_equipment⟩⟩
⟨⟨pedro_gel2d, pedro_first_dim_details⟩⟩
⟨⟨pedro_gel2d, pedro_id⟩⟩
⟨⟨pedro_gel2d, pedro_in_gel_digestion⟩⟩
⟨⟨pedro_gel2d, pedro_mass_end⟩⟩
⟨⟨pedro_gel2d, pedro_mass_start⟩⟩
⟨⟨pedro_gel2d, pedro_percent_acrylamide⟩⟩
⟨⟨pedro_gel2d, pedro_pi_end⟩⟩
⟨⟨pedro_gel2d, pedro_pi_start⟩⟩
⟨⟨pedro_gel2d, pedro_pixel_size_x⟩⟩
⟨⟨pedro_gel2d, pedro_pixel_size_y⟩⟩
⟨⟨pedro_gel2d, pedro_protein_assay⟩⟩
⟨⟨pedro_gel2d, pedro_raw_image⟩⟩
⟨⟨pedro_gel2d, pedro_second_dim_details⟩⟩
⟨⟨pedro_gel2d, pedro_software_version⟩⟩
⟨⟨pedro_gel2d, pedro_solubilization_buffer⟩⟩
⟨⟨pedro_gel2d, pedro_stain_details⟩⟩
⟨⟨pedro_gel2d, pedro_warped_image⟩⟩
⟨⟨pedro_gel2d, pedro_warping_map⟩⟩
⟨⟨pedro_gradientstep, pedro_id⟩⟩
⟨⟨pedro_gradientstep, pedro_lc_column⟩⟩
⟨⟨pedro_gradientstep, pedro_step_time⟩⟩
⟨⟨pedro_hexapole, pedro_description⟩⟩
⟨⟨pedro_hexapole, pedro_id⟩⟩
⟨⟨pedro_hexapole, pedro_mz_analysis⟩⟩
⟨⟨pedro_ionsource, pedro_collision_energy⟩⟩
⟨⟨pedro_ionsource, pedro_id⟩⟩
⟨⟨pedro_ionsource, pedro_mz_analysis⟩⟩
⟨⟨pedro_ionsource, pedro_type⟩⟩
⟨⟨pedro_iontrap, pedro_excitation_amplitude⟩⟩
⟨⟨pedro_iontrap, pedro_final_ms_level⟩⟩
⟨⟨pedro_iontrap, pedro_gas_pressure⟩⟩
⟨⟨pedro_iontrap, pedro_gas_type⟩⟩
⟨⟨pedro_iontrap, pedro_id⟩⟩
⟨⟨pedro_iontrap, pedro_isolation_centre⟩⟩
⟨⟨pedro_iontrap, pedro_isolation_width⟩⟩
⟨⟨pedro_iontrap, pedro_mz_analysis⟩⟩
⟨⟨pedro_iontrap, pedro_rf_frequency⟩⟩
⟨⟨pedro_lccolumn, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_lccolumn, pedro_batch_number⟩⟩
⟨⟨pedro_lccolumn, pedro_bead_size⟩⟩
⟨⟨pedro_lccolumn, pedro_description⟩⟩
⟨⟨pedro_lccolumn, pedro_flow_rate⟩⟩
⟨⟨pedro_lccolumn, pedro_id⟩⟩
⟨⟨pedro_lccolumn, pedro_injection_volume⟩⟩
⟨⟨pedro_lccolumn, pedro_internal_diameter⟩⟩
⟨⟨pedro_lccolumn, pedro_internal_length⟩⟩
⟨⟨pedro_otheranalyteps⟩⟩
⟨⟨pedro_otherionisation_otm_ontent⟩⟩
⟨⟨pedro_otherionisation⟩⟩
⟨⟨pedro_othermzanalysis_otm_ontent⟩⟩
⟨⟨pedro_othermzanalysis⟩⟩
⟨⟨pedro_peak⟩⟩
⟨⟨pedro_peaklist⟩⟩
⟨⟨pedro_peakspecificchromint⟩⟩
⟨⟨pedro_peptidehit_mtm_ontent⟩⟩
⟨⟨pedro_peptidehit⟩⟩
⟨⟨pedro_percentx⟩⟩
⟨⟨pedro_protein⟩⟩
⟨⟨pedro_proteinhit⟩⟩
⟨⟨pedro_quadrupole⟩⟩
⟨⟨pedro_relatedgelitem⟩⟩
⟨⟨pedro_relgelitem_mtm_proteinhit⟩⟩
⟨⟨pedro_sample_mtm_sampleorigin⟩⟩
⟨⟨pedro_sample_otm_analyteps⟩⟩
⟨⟨pedro_sample⟩⟩
⟨⟨pedro_sampleorigin⟩⟩
⟨⟨pedro_spot_otm_analyteps⟩⟩
⟨⟨pedro_spot⟩⟩
⟨⟨pedro_taggingprocess⟩⟩
⟨⟨pedro_tandemsequencedata⟩⟩
⟨⟨pedro_tof⟩⟩
⟨⟨pedro_treatedanalyte_otm_analyteps⟩⟩
⟨⟨pedro_treatedanalyte⟩⟩
⟨⟨pepseeker_proteinhit⟩⟩
⟨⟨pepseeker_proteinscore, pepseeker_Score⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_ProteinID⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_peptidehit⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_proteinscore⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_pepseq⟩⟩
⟨⟨pepseeker_mascotexpect, pepseeker_expect⟩⟩
⟨⟨pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Username⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Id⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Filename⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TAXONOMY⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_MODS⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_IT_MODS⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_PFA⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_MASS⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ITOL⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TOL⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SEG⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TOLU⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_CHARGE⟩⟩
⟨⟨pepseeker_searchmasses⟩⟩
⟨⟨pepseeker_searchmasses, pepseeker_Products⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_Proteinname⟩⟩
⟨⟨pepseeker_species, pepseeker_species⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_Mass⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_startresidue⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_endresidue⟩⟩

⟨⟨pedro_lccolumn, pedro_lc_column⟩⟩	
⟨⟨pedro_lccolumn, pedro_manufacturer⟩⟩	
⟨⟨pedro_lccolumn, pedro_parameters_file⟩⟩	
⟨⟨pedro_lccolumn, pedro_part_number⟩⟩	
⟨⟨pedro_lccolumn, pedro_pore_size⟩⟩	
⟨⟨pedro_lccolumn, pedro_stationary_phase⟩⟩	
⟨⟨pedro_lccolumn, pedro_temperature⟩⟩	

Table E.9: Elements Satisfying Quality Factor 1 in Iteration 2

⟨⟨pepseeker_db, pepseeker_entries⟩⟩	⟨⟨pepseeker_iontable, pepseeker_Ystarplusplus⟩⟩
⟨⟨pepseeker_db, pepseeker_id⟩⟩	⟨⟨pepseeker_iontable, pepseeker_YZero⟩⟩
⟨⟨pepseeker_db, pepseeker_location⟩⟩	⟨⟨pepseeker_iontable, pepseeker_YZeroplusplus⟩⟩
⟨⟨pepseeker_db, pepseeker_name⟩⟩	⟨⟨pepseeker_lastsession, pepseeker_Id⟩⟩
⟨⟨pepseeker_db, pepseeker_release_date⟩⟩	⟨⟨pepseeker_lastsession, pepseeker_url⟩⟩
⟨⟨pepseeker_db, pepseeker_release⟩⟩	⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩
⟨⟨pepseeker_db, pepseeker_residue_type⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Delta⟩⟩
⟨⟨pepseeker_db, pepseeker_system_date⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ACCESSION⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_ions⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_CLE⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_MassNo⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_DB⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_MissCleav⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ERRORTOLERANT⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_MrCalc⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_FORMAT⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_MrExpct⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_FORMVER⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Rank⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_FRAG⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Score⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ICAT⟩⟩	⟨⟨pepseeker_peptideprophet, pepseeker_peptidehit_Id⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_INSTRUMENT⟩⟩	⟨⟨pepseeker_peptideprophet, pepseeker_tpp_pvalue⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_INTERMEDIATE⟩⟩	⟨⟨pepseeker_proteinhit, pepseeker_Id⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ITH⟩⟩	⟨⟨pepseeker_proteinhit, pepseeker_ProteinScore⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ITOLU⟩⟩	⟨⟨pepseeker_proteinscore, pepseeker_Id⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_LTOL⟩⟩	⟨⟨pepseeker_proteinscore, pepseeker_Matchedpeptides⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_mp⟩⟩	⟨⟨pepseeker_proteinscore, pepseeker_querypeptides⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_NM⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_OVERVIEW⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Id⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_PEAK⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Int_max⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_PRECURSOR⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_int_min⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ProcessNo⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Mass_Max⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_QUE⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Mass_min⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_REPORT⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_num_used⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_REPTYPE⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_num_vals⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SEARCH⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Precursor⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SEGT⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_queryno⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SEGTU⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_searchmassescol⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SUBCLUSTER⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Title⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TWO⟩⟩	⟨⟨pepseeker_species, pepseeker_fp_id⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Useremail⟩⟩	⟨⟨pepseeker_species, pepseeker_ProteinId⟩⟩
⟨⟨pepseeker_fp_to_db, pepseeker_db_id⟩⟩	⟨⟨pepseeker_species, pepseeker_species_id⟩⟩
⟨⟨pepseeker_fp_to_db, pepseeker_fp_id⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_accession⟩⟩
⟨⟨pepseeker_iontable, pepseeker_A⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_id⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Aplusplus⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_oln⟩⟩
⟨⟨pepseeker_iontable, pepseeker_AStar⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_yeast_acc_id⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Astarplusplus⟩⟩	⟨⟨pepseeker_db⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Azero⟩⟩	⟨⟨pepseeker_fp_to_db⟩⟩
⟨⟨pepseeker_iontable, pepseeker_B⟩⟩	⟨⟨pepseeker_iontable⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Bplusplus⟩⟩	⟨⟨pepseeker_lastsession⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Bstar⟩⟩	⟨⟨pepseeker_mascotexpect⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Bstarplusplus⟩⟩	⟨⟨pepseeker_peptideprophet⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Bzero⟩⟩	⟨⟨pepseeker_proteinscore⟩⟩
⟨⟨pepseeker_iontable, pepseeker_BZeroplusplus⟩⟩	⟨⟨pepseeker_species⟩⟩

⟨⟨pepseeker_iontable, pepseeker_Id⟩⟩	⟨⟨pepseeker_yeastacc⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Immon⟩⟩	
⟨⟨pepseeker_iontable, pepseeker_Matches⟩⟩	
⟨⟨pepseeker_iontable, pepseeker_Y⟩⟩	
⟨⟨pepseeker_iontable, pepseeker_Yplusplus⟩⟩	
⟨⟨pepseeker_iontable, pepseeker_Ystar⟩⟩	

Table E.10: Elements not-Satisfying Quality Factor 1 in Iteration 2

Factor 4

Satisfying Items	**not-Satisfying Items**
⟨⟨gpmdb_aa, gpmdb_aaid⟩⟩	⟨⟨gpmdb_peptide, gpmdb_pepid⟩⟩
⟨⟨gpmdb_aa, gpmdb_pepid⟩⟩	⟨⟨gpmdb_fullpeptide, gpmdb_pepid⟩⟩
⟨⟨gpmdb_peptide, gpmdb_seq⟩⟩	⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_pepid⟩⟩
⟨⟨gpmdb_peptide⟩⟩	⟨⟨pedro_peptidehit⟩⟩
⟨⟨gpmdb_proseq, gpmdb_label⟩⟩	⟨⟨pedro_peptidehit, pedro_id⟩⟩
⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩	⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩
⟨⟨gpmdb_protein⟩⟩	⟨⟨pepseeker_peptideprophet, pepseeker_peptidehit_Id⟩⟩
⟨⟨pedro_peptidehit, pedro_score⟩⟩	⟨⟨gpmdb_protein, gpmdb_proid⟩⟩
⟨⟨pedro_peptidehit, pedro_sequence⟩⟩	⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩
⟨⟨pedro_proteinhit, pedro_protein⟩⟩	⟨⟨gpmdb_fullpeptide, gpmdb_proid⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_pepseq⟩⟩	⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_proid⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_proteinscore⟩⟩	⟨⟨pedro_protein⟩⟩
⟨⟨pepseeker_peptidehit⟩⟩	⟨⟨pedro_protein, pedro_id⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_ProteinID⟩⟩	⟨⟨pedro_proteinhit, pedro_id⟩⟩
⟨⟨pepseeker_proteinhit⟩⟩	⟨⟨pepseeker_proteinhit, pepseeker_Id⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_value_type⟩⟩	⟨⟨pepseeker_species, pepseeker_ProteinID⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_error_type⟩⟩	⟨⟨gpmdb_result⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_MASS⟩⟩	⟨⟨gpmdb_result, gpmdb_resultid⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TOLU⟩⟩	⟨⟨gpmdb_project, gpmdb_resultid⟩⟩
	⟨⟨gpmdb_fullpeptide, gpmdb_seq⟩⟩
	⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_seq⟩⟩
	⟨⟨gpmdb_proseq⟩⟩
	⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩
	⟨⟨gpmdb_proseq, gpmdb_seq⟩⟩
	⟨⟨pedro_protein, pedro_sequence⟩⟩
	⟨⟨pepseeker_proteinscore, pepseeker_Id⟩⟩
	⟨⟨pepseeker_proteinscore, pepseeker_Score⟩⟩
	⟨⟨pepseeker_proteinhit, pepseeker_ProteinScore⟩⟩
	⟨⟨gpmdb_aa⟩⟩
	⟨⟨pedro_peaklist, pedro_mass_value_type⟩⟩
	⟨⟨pepseeker_proteinhit, pepseeker_Mass⟩⟩
	⟨⟨pepseeker_peptidehit, pepseeker_MassNo⟩⟩
	⟨⟨pepseeker_fileparameters, pepseeker_ITOLU⟩⟩

Table E.11: Elements Satisfying and not-Satisfying Quality Factor 4 in Iteration 2

Factor 7

⟨⟨pedro_fk_dbsearch_dbsearchparameters1, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩, pedro_dbsearchparameters, ⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩⟩⟩
⟨⟨pedro_fk_proteinhit_dbsearch1, pedro_proteinhit, ⟨⟨pedro_proteinhit, pedro_db_search⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_id⟩⟩⟩⟩
⟨⟨gpmdb_result⟩⟩
⟨⟨pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_fileparameters, Id⟩⟩
⟨⟨gpmdb_protein⟩⟩
⟨⟨gpmdb_protein, proseqid⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_fileparameters⟩⟩
⟨⟨gpmdb_peptide, gpmdb_pepid⟩⟩
⟨⟨gpmdb_fk_peptide_protein1, gpmdb_peptide, ⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩, gpmdb_protein, ⟨⟨gpmdb_protein, gpmdb_proid⟩⟩⟩⟩
⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩
⟨⟨gpmdb_protein, gpmdb_proid⟩⟩
⟨⟨gpmdb_fk_protein_proseq, gpmdb_protein, ⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩, gpmdb_proseq, ⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩⟩⟩
⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩
⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩
⟨⟨gpmdb_fk_protein_result1, gpmdb_protein, ⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩, gpmdb_result, ⟨⟨gpmdb_result, gpmdb_resultid⟩⟩⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩
⟨⟨gpmdb_result, gpmdb_resultid⟩⟩
⟨⟨pedro_fk_dbsearch_dbsearchparameters1, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩, pedro_dbsearchparameters, ⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩⟩⟩
⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩
⟨⟨pedro_fk_chromatogrampoint_peak1, pedro_chromatogrampoint, ⟨⟨pedro_chromatogrampoint, pedro_peak⟩⟩, pedro_peak, ⟨⟨pedro_peak, pedro_id⟩⟩⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_peak⟩⟩
⟨⟨pedro_peak, pedro_id⟩⟩
⟨⟨pedro_fk_dbsearch_peaklist1, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_dbsearch, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_listprocessing_peaklist1, pedro_listprocessing, ⟨⟨pedro_listprocessing, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_listprocessing, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_msmsfraction_peaklist1, pedro_msmsfraction, ⟨⟨pedro_msmsfraction, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_msmsfraction, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_peak_peaklist, pedro_peak, ⟨⟨pedro_peak, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_peak, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_peaklist_massspecmachine1, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_mass_spec_experiment⟩⟩, pedro_massspecmachine, ⟨⟨pedro_massspecmachine, pedro_id⟩⟩⟩⟩
⟨⟨pedro_peaklist, pedro_mass_spec_experiment⟩⟩
⟨⟨pedro_massspecmachine, pedro_id⟩⟩
⟨⟨pedro_fk_peptidehit_dbsearch1, pedro_peptidehit, ⟨⟨pedro_peptidehit, pedro_db_search⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_id⟩⟩⟩⟩

⟨⟨pedro_peptidehit, pedro_db_search⟩⟩
⟨⟨pedro_dbsearch, pedro_id⟩⟩
⟨⟨pedro_fk_proteinhit_dbsearch1, pedro_proteinhit, ⟨⟨pedro_proteinhit, pedro_db_search⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_id⟩⟩⟩⟩
⟨⟨pedro_proteinhit, pedro_db_search⟩⟩
⟨⟨pedro_dbsearch, pedro_id⟩⟩
⟨⟨pedro_fk_tandemsequencedata_dbsearch1, pedro_tandemsequencedata, ⟨⟨pedro_tandemsequencedata, pedro_db_search_parameters⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩
⟨⟨pepseeker_fk_mascotexpect_peptidehit1, pepseeker_mascotexpect, ⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩, pepseeker_peptidehit, ⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩⟩⟩
⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩
⟨⟨pepseeker_fk_searchmasses_fileparameters, pepseeker_searchmasses, ⟨⟨pepseeker_searchmasses, pepseeker_fileparameters⟩⟩, pepseeker_fileparameters, ⟨⟨pepseeker_fileparameters, pepseeker_Id⟩⟩⟩⟩
⟨⟨pepseeker_searchmasses, pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Id⟩⟩
⟨⟨GS1_fk_proteinhit_protein1, GS1_proteinhit, ⟨⟨GS1_proteinhit, GS1_id⟩⟩, GS1_protein, ⟨⟨GS1_protein, GS1_id⟩⟩⟩⟩
⟨⟨pepseeker_proteinhit⟩⟩
⟨⟨GS1_proteinhit, GS1_id⟩⟩
⟨⟨GS1_protein, GS1_id⟩⟩

Table E.12: Elements Satisfying Quality Factor 7 in Iteration 2

E.4.3 Quality Measurement for Iteration 3

Factor 1

⟨⟨gpmdb_aa, gpmdb_aaid⟩⟩	⟨⟨pedro_othermzanalysis, pedro_name⟩⟩
⟨⟨gpmdb_aa, gpmdb_at⟩⟩	⟨⟨pedro_othermzanalysis_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨gpmdb_aa, gpmdb_modified⟩⟩	⟨⟨pedro_othermzanalysis_otm_ontent, pedro_other_mz_analysis⟩⟩
⟨⟨gpmdb_aa, gpmdb_pepid⟩⟩	⟨⟨pedro_peak, pedro_abundance⟩⟩
⟨⟨gpmdb_aa, gpmdb_pm⟩⟩	⟨⟨pedro_peak, pedro_id⟩⟩
⟨⟨gpmdb_aa, gpmdb_type⟩⟩	⟨⟨pedro_peak, pedro_m_to_z⟩⟩
⟨⟨gpmdb_bad_file_del, gpmdb_file⟩⟩	⟨⟨pedro_peak, pedro_multiplicity⟩⟩
⟨⟨gpmdb_distinctseq, gpmdb_seq⟩⟩	⟨⟨pedro_peak, pedro_peak_list⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_aastring⟩⟩	⟨⟨pedro_peaklist, pedro_description⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_charge⟩⟩	⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_delta⟩⟩	⟨⟨pedro_peaklist, pedro_list_type⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_dida⟩⟩	⟨⟨pedro_peaklist, pedro_mass_spec_experiment⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_end⟩⟩	⟨⟨pedro_peaklist, pedro_mass_value_type⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_expect⟩⟩	⟨⟨pedro_peakspecificchromint, pedro_area_under_curve⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_mh⟩⟩	⟨⟨pedro_peakspecificchromint, pedro_background_threshold⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_mh2⟩⟩	⟨⟨pedro_peakspecificchromint, pedro_id⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_pepid⟩⟩	⟨⟨pedro_peakspecificchromint, pedro_peak_description⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_proid⟩⟩	⟨⟨pedro_peakspecificchromint, pedro_peak⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_seq⟩⟩	⟨⟨pedro_peakspecificchromint, pedro_resolution⟩⟩
⟨⟨gpmdb_fullpeptide, gpmdb_start⟩⟩	⟨⟨pedro_peakspecificchromint, pedro_sister_peak_reference⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_aastring⟩⟩	⟨⟨pedro_peakspecificchromint, pedro_software_version⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_charge⟩⟩	⟨⟨pedro_peptidehit, pedro_db_search⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_delta⟩⟩	⟨⟨pedro_peptidehit, pedro_id⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_dida⟩⟩	⟨⟨pedro_peptidehit, pedro_information⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_end⟩⟩	⟨⟨pedro_peptidehit, pedro_probability⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_expect⟩⟩	⟨⟨pedro_peptidehit, pedro_score_type⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_mh⟩⟩	⟨⟨pedro_peptidehit, pedro_score⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_mh2⟩⟩	⟨⟨pedro_peptidehit, pedro_sequence⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_pepid⟩⟩	⟨⟨pedro_peptidehit_mtm_ontent, pedro_ontology_entry⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_proid⟩⟩	⟨⟨pedro_peptidehit_mtm_ontent, pedro_peptide_hit_db_search⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_seq⟩⟩	⟨⟨pedro_peptidehit_mtm_ontent, pedro_peptide_hit_id⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_start⟩⟩	⟨⟨pedro_percentx, pedro_gradient_step_id⟩⟩
⟨⟨gpmdb_paths, gpmdb_localpath⟩⟩	⟨⟨pedro_percentx, pedro_gradient_step_lc_column⟩⟩
⟨⟨gpmdb_paths, gpmdb_pathid⟩⟩	⟨⟨pedro_percentx, pedro_id⟩⟩
⟨⟨gpmdb_paths, gpmdb_protocol⟩⟩	⟨⟨pedro_percentx, pedro_mobile_phase_component⟩⟩
⟨⟨gpmdb_paths, gpmdb_relpath⟩⟩	⟨⟨pedro_percentx, pedro_percentage⟩⟩
⟨⟨gpmdb_paths, gpmdb_server⟩⟩	⟨⟨pedro_protein, pedro_accession_num⟩⟩
⟨⟨gpmdb_pep_temp, gpmdb_pepid⟩⟩	⟨⟨pedro_protein, pedro_description⟩⟩
⟨⟨gpmdb_pep_temp, gpmdb_seq⟩⟩	⟨⟨pedro_protein, pedro_gene_name⟩⟩
⟨⟨gpmdb_peptide, gpmdb_charge⟩⟩	⟨⟨pedro_protein, pedro_id⟩⟩
⟨⟨gpmdb_peptide, gpmdb_delta⟩⟩	⟨⟨pedro_protein, pedro_modifications⟩⟩
⟨⟨gpmdb_peptide, gpmdb_dida⟩⟩	⟨⟨pedro_protein, pedro_orf_number⟩⟩
⟨⟨gpmdb_peptide, gpmdb_didb⟩⟩	⟨⟨pedro_protein, pedro_organism⟩⟩
⟨⟨gpmdb_peptide, gpmdb_didc⟩⟩	⟨⟨pedro_protein, pedro_predicted_mass⟩⟩
⟨⟨gpmdb_peptide, gpmdb_end⟩⟩	⟨⟨pedro_protein, pedro_predicted_pi⟩⟩
⟨⟨gpmdb_peptide, gpmdb_expect⟩⟩	⟨⟨pedro_protein, pedro_sequence⟩⟩
⟨⟨gpmdb_peptide, gpmdb_mh⟩⟩	⟨⟨pedro_protein, pedro_synonyms⟩⟩
⟨⟨gpmdb_peptide, gpmdb_pepid⟩⟩	⟨⟨pedro_proteinhit, pedro_all_peptides_matched⟩⟩
⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩	⟨⟨pedro_proteinhit, pedro_component_peptides⟩⟩
⟨⟨gpmdb_peptide, gpmdb_seq⟩⟩	⟨⟨pedro_proteinhit, pedro_db_search⟩⟩
⟨⟨gpmdb_peptide, gpmdb_start⟩⟩	⟨⟨pedro_proteinhit, pedro_id⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_keyid⟩⟩	⟨⟨pedro_proteinhit, pedro_masses_matched⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_pepid_list⟩⟩	⟨⟨pedro_proteinhit, pedro_protein⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_ts_created⟩⟩	⟨⟨pedro_proteinhit, pedro_score_type⟩⟩
⟨⟨gpmdb_peptide_word_index, gpmdb_word⟩⟩	⟨⟨pedro_proteinhit, pedro_score⟩⟩

⟨⟨gpmdb_peptide_words, gpmdb_pepid_list⟩⟩
⟨⟨gpmdb_peptide_words, gpmdb_seq_word⟩⟩
⟨⟨gpmdb_project, gpmdb_description⟩⟩
⟨⟨gpmdb_project, gpmdb_email⟩⟩
⟨⟨gpmdb_project, gpmdb_institution⟩⟩
⟨⟨gpmdb_project, gpmdb_name⟩⟩
⟨⟨gpmdb_project, gpmdb_project⟩⟩
⟨⟨gpmdb_project, gpmdb_projectid⟩⟩
⟨⟨gpmdb_project, gpmdb_resultid⟩⟩
⟨⟨gpmdb_proseq, gpmdb_label_aux⟩⟩
⟨⟨gpmdb_proseq, gpmdb_label⟩⟩
⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩
⟨⟨gpmdb_proseq, gpmdb_rf⟩⟩
⟨⟨gpmdb_proseq, gpmdb_seq⟩⟩
⟨⟨gpmdb_protein, gpmdb_expect⟩⟩
⟨⟨gpmdb_protein, gpmdb_pida⟩⟩
⟨⟨gpmdb_protein, gpmdb_pidb⟩⟩
⟨⟨gpmdb_protein, gpmdb_proid⟩⟩
⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩
⟨⟨gpmdb_protein, gpmdb_uid⟩⟩
⟨⟨gpmdb_proteinrevision, gpmdb_label⟩⟩
⟨⟨gpmdb_proteinrevision, gpmdb_proid⟩⟩
⟨⟨gpmdb_proteinrevision, gpmdb_prorevid⟩⟩
⟨⟨gpmdb_result, gpmdb_active⟩⟩
⟨⟨gpmdb_result, gpmdb_comments⟩⟩
⟨⟨gpmdb_result, gpmdb_completed⟩⟩
⟨⟨gpmdb_result, gpmdb_file⟩⟩
⟨⟨gpmdb_result, gpmdb_pathid⟩⟩
⟨⟨gpmdb_result, gpmdb_rating⟩⟩
⟨⟨gpmdb_result, gpmdb_resultid⟩⟩
⟨⟨gpmdb_result, gpmdb_tandemversion⟩⟩
⟨⟨gpmdb_aa⟩⟩
⟨⟨gpmdb_bad_file_del⟩⟩
⟨⟨gpmdb_distinctseq⟩⟩
⟨⟨gpmdb_fullpeptide⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic⟩⟩
⟨⟨gpmdb_paths⟩⟩
⟨⟨gpmdb_pep_temp⟩⟩
⟨⟨gpmdb_peptide_word_index⟩⟩
⟨⟨gpmdb_peptide_words⟩⟩
⟨⟨gpmdb_peptide⟩⟩
⟨⟨gpmdb_project⟩⟩
⟨⟨gpmdb_proseq⟩⟩
⟨⟨gpmdb_protein⟩⟩
⟨⟨gpmdb_proteinrevision⟩⟩
⟨⟨gpmdb_result⟩⟩
⟨⟨pedro_analyteprocessingstep, pedro_id⟩⟩
⟨⟨pedro_analyteprocessingstep, pedro_input_type⟩⟩
⟨⟨pedro_analyteprocessingstep, pedro_processing_type⟩⟩
⟨⟨pedro_assaydatapoint, pedro_id⟩⟩
⟨⟨pedro_assaydatapoint, pedro_lc_column⟩⟩
⟨⟨pedro_assaydatapoint, pedro_protein_assay⟩⟩
⟨⟨pedro_assaydatapoint, pedro_time⟩⟩
⟨⟨pedro_band, pedro_annotation_source⟩⟩
⟨⟨pedro_band, pedro_annotation⟩⟩
⟨⟨pedro_band, pedro_apparent_mass⟩⟩
⟨⟨pedro_band, pedro_area⟩⟩
⟨⟨pedro_band, pedro_gel_1d⟩⟩
⟨⟨pedro_band, pedro_id⟩⟩
⟨⟨pedro_quadrupole, pedro_description⟩⟩
⟨⟨pedro_quadrupole, pedro_id⟩⟩
⟨⟨pedro_quadrupole, pedro_mz_analysis⟩⟩
⟨⟨pedro_relatedgelitem, pedro_band_gel_1d⟩⟩
⟨⟨pedro_relatedgelitem, pedro_band_id⟩⟩
⟨⟨pedro_relatedgelitem, pedro_description⟩⟩
⟨⟨pedro_relatedgelitem, pedro_gel_reference⟩⟩
⟨⟨pedro_relatedgelitem, pedro_id⟩⟩
⟨⟨pedro_relatedgelitem, pedro_item_reference⟩⟩
⟨⟨pedro_relatedgelitem, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_relatedgelitem, pedro_spot_id⟩⟩
⟨⟨pedro_relgelitem_mtm_proteinhit, pedro_protein_hit⟩⟩
⟨⟨pedro_relgelitem_mtm_proteinhit, pedro_related_gel_item⟩⟩
⟨⟨pedro_sample, pedro_experiment⟩⟩
⟨⟨pedro_sample, pedro_experimenter⟩⟩
⟨⟨pedro_sample, pedro_sample_date⟩⟩
⟨⟨pedro_sample, pedro_sample_id⟩⟩
⟨⟨pedro_sample_mtm_sampleorigin, pedro_sample_origin⟩⟩
⟨⟨pedro_sample_mtm_sampleorigin, pedro_sample⟩⟩
⟨⟨pedro_sample_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_sample_otm_analyteps, pedro_sample⟩⟩
⟨⟨pedro_sampleorigin, pedro_cell_component⟩⟩
⟨⟨pedro_sampleorigin, pedro_cell_cycle_phase⟩⟩
⟨⟨pedro_sampleorigin, pedro_cell_type⟩⟩
⟨⟨pedro_sampleorigin, pedro_condition_degree⟩⟩
⟨⟨pedro_sampleorigin, pedro_description⟩⟩
⟨⟨pedro_sampleorigin, pedro_environment⟩⟩
⟨⟨pedro_sampleorigin, pedro_id⟩⟩
⟨⟨pedro_sampleorigin, pedro_metabolic_label⟩⟩
⟨⟨pedro_sampleorigin, pedro_organism⟩⟩
⟨⟨pedro_sampleorigin, pedro_sample_condition⟩⟩
⟨⟨pedro_sampleorigin, pedro_tagging_process⟩⟩
⟨⟨pedro_sampleorigin, pedro_technique⟩⟩
⟨⟨pedro_sampleorigin, pedro_tissue_type⟩⟩
⟨⟨pedro_spot, pedro_annotation_source⟩⟩
⟨⟨pedro_spot, pedro_annotation⟩⟩
⟨⟨pedro_spot, pedro_apparent_mass⟩⟩
⟨⟨pedro_spot, pedro_apparent_pi⟩⟩
⟨⟨pedro_spot, pedro_area⟩⟩
⟨⟨pedro_spot, pedro_gel_2d⟩⟩
⟨⟨pedro_spot, pedro_id⟩⟩
⟨⟨pedro_spot, pedro_intensity⟩⟩
⟨⟨pedro_spot, pedro_local_background⟩⟩
⟨⟨pedro_spot, pedro_normalisation⟩⟩
⟨⟨pedro_spot, pedro_normalised_volume⟩⟩
⟨⟨pedro_spot, pedro_pixel_radius⟩⟩
⟨⟨pedro_spot, pedro_pixel_x_coord⟩⟩
⟨⟨pedro_spot, pedro_pixel_y_coord⟩⟩
⟨⟨pedro_spot, pedro_volume⟩⟩
⟨⟨pedro_spot_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_spot_otm_analyteps, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_spot_otm_analyteps, pedro_spot_id⟩⟩
⟨⟨pedro_taggingprocess, pedro_final_volume⟩⟩
⟨⟨pedro_taggingprocess, pedro_id⟩⟩
⟨⟨pedro_taggingprocess, pedro_lysis_buffer⟩⟩
⟨⟨pedro_taggingprocess, pedro_protein_concentration⟩⟩
⟨⟨pedro_taggingprocess, pedro_tag_concentration⟩⟩
⟨⟨pedro_taggingprocess, pedro_tag_purity⟩⟩
⟨⟨pedro_taggingprocess, pedro_tag_type⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_db_search_parameters⟩⟩

⟨⟨pedro_band, pedro_intensity⟩⟩
⟨⟨pedro_band, pedro_lane_number⟩⟩
⟨⟨pedro_band, pedro_local_background⟩⟩
⟨⟨pedro_band, pedro_normalisation⟩⟩
⟨⟨pedro_band, pedro_normalised_volume⟩⟩
⟨⟨pedro_band, pedro_pixel_radius⟩⟩
⟨⟨pedro_band, pedro_pixel_x_coord⟩⟩
⟨⟨pedro_band, pedro_pixel_y_coord⟩⟩
⟨⟨pedro_band, pedro_volume⟩⟩
⟨⟨pedro_band_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_band_otm_analyteps, pedro_band_gel_1d⟩⟩
⟨⟨pedro_band_otm_analyteps, pedro_band_id⟩⟩
⟨⟨pedro_boundarypoint, pedro_id⟩⟩
⟨⟨pedro_boundarypoint, pedro_pixel_x_coord⟩⟩
⟨⟨pedro_boundarypoint, pedro_pixel_y_coord⟩⟩
⟨⟨pedro_boundarypoint, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_boundarypoint, pedro_spot_id⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_derivatisations⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_digestion⟩⟩
⟨⟨pedro_chemicaltreatment, pedro_id⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_id⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_ion_count⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_peak⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_time_point⟩⟩
⟨⟨pedro_collisioncell, pedro_collision_offset⟩⟩
⟨⟨pedro_collisioncell, pedro_gas_pressure⟩⟩
⟨⟨pedro_collisioncell, pedro_gas_type⟩⟩
⟨⟨pedro_collisioncell, pedro_id⟩⟩
⟨⟨pedro_collisioncell, pedro_mz_analysis⟩⟩
⟨⟨pedro_dbsearch, pedro_c_terminal_aa⟩⟩
⟨⟨pedro_dbsearch, pedro_count_of_specific_aa⟩⟩
⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearch, pedro_id_date⟩⟩
⟨⟨pedro_dbsearch, pedro_id⟩⟩
⟨⟨pedro_dbsearch, pedro_n_terminal_aa⟩⟩
⟨⟨pedro_dbsearch, pedro_name_of_counted_aa⟩⟩
⟨⟨pedro_dbsearch, pedro_peak_list⟩⟩
⟨⟨pedro_dbsearch, pedro_regex_pattern⟩⟩
⟨⟨pedro_dbsearch, pedro_username⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_accurate_mass_mode⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_database_date⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_database_name⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_fixed_modifications⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_fragment_ion_tolerance⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_icat_option⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_error_type⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_error⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_value_type⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_max_missed_cleavages⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_parameters_file⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_peptide_mass_tolerance⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_program⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_protonated⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_taxonomical_filter⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_variable_modifications⟩⟩
⟨⟨pedro_dbsearchpars_otm_ontent, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearchpars_otm_ontent, pedro_ontology_entry⟩⟩
⟨⟨pedro_detection, pedro_id⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_id⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_sequence⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_source_type⟩⟩
⟨⟨pedro_tof, pedro_id⟩⟩
⟨⟨pedro_tof, pedro_internal_length⟩⟩
⟨⟨pedro_tof, pedro_mz_analysis⟩⟩
⟨⟨pedro_tof, pedro_reflectron_state⟩⟩
⟨⟨pedro_treatedanalyte, pedro_chemical_treatment⟩⟩
⟨⟨pedro_treatedanalyte, pedro_id⟩⟩
⟨⟨pedro_treatedanalyte_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_treatedanalyte_otm_analyteps, pedro_treated_analyte⟩⟩
⟨⟨pedro_analyteprocessingstep⟩⟩
⟨⟨pedro_assaydatapoint⟩⟩
⟨⟨pedro_band_otm_analyteps⟩⟩
⟨⟨pedro_band⟩⟩
⟨⟨pedro_boundarypoint⟩⟩
⟨⟨pedro_chemicaltreatment⟩⟩
⟨⟨pedro_chromatogrampoint⟩⟩
⟨⟨pedro_collisioncell⟩⟩
⟨⟨pedro_dbsearch⟩⟩
⟨⟨pedro_dbsearchparameters⟩⟩
⟨⟨pedro_dbsearchpars_otm_ontent⟩⟩
⟨⟨pedro_detection⟩⟩
⟨⟨pedro_digegel⟩⟩
⟨⟨pedro_digegelitem⟩⟩
⟨⟨pedro_electrospray⟩⟩
⟨⟨pedro_experiment⟩⟩
⟨⟨pedro_fraction_otm_analyteps⟩⟩
⟨⟨pedro_fraction⟩⟩
⟨⟨pedro_gel1d⟩⟩
⟨⟨pedro_gel2d⟩⟩
⟨⟨pedro_gradientstep⟩⟩
⟨⟨pedro_hexapole⟩⟩
⟨⟨pedro_ionsource⟩⟩
⟨⟨pedro_iontrap⟩⟩
⟨⟨pedro_lccolumn⟩⟩
⟨⟨pedro_listprocessing⟩⟩
⟨⟨pedro_maldi⟩⟩
⟨⟨pedro_massspecexperiment⟩⟩
⟨⟨pedro_massspecmachine⟩⟩
⟨⟨pedro_mobilephasecomponent⟩⟩
⟨⟨pedro_msmsfraction⟩⟩
⟨⟨pedro_mzanalysis⟩⟩
⟨⟨pedro_ontologyentry⟩⟩
⟨⟨pedro_organism⟩⟩
⟨⟨pedro_otheranalyte_otm_analyteps⟩⟩
⟨⟨pedro_otheranalyte_otm_ontent⟩⟩
⟨⟨pedro_otheranalyte⟩⟩
⟨⟨pedro_otheranalyteps_otm_ontent⟩⟩
⟨⟨pedro_otheranalyteps⟩⟩
⟨⟨pedro_otherionisation_otm_ontent⟩⟩
⟨⟨pedro_otherionisation⟩⟩
⟨⟨pedro_othermzanalysis_otm_ontent⟩⟩
⟨⟨pedro_othermzanalysis⟩⟩
⟨⟨pedro_peak⟩⟩
⟨⟨pedro_peaklist⟩⟩
⟨⟨pedro_peakspecificchromint⟩⟩
⟨⟨pedro_peptidehit_mtm_ontent⟩⟩
⟨⟨pedro_peptidehit⟩⟩
⟨⟨pedro_percentx⟩⟩

⟨⟨pedro_detection, pedro_type⟩⟩
⟨⟨pedro_digegel, pedro_dye_type⟩⟩
⟨⟨pedro_digegel, pedro_excitation_wavelength⟩⟩
⟨⟨pedro_digegel, pedro_exposure_time⟩⟩
⟨⟨pedro_digegel, pedro_gel_1d⟩⟩
⟨⟨pedro_digegel, pedro_gel_2d⟩⟩
⟨⟨pedro_digegel, pedro_id⟩⟩
⟨⟨pedro_digegel, pedro_tiff_image⟩⟩
⟨⟨pedro_digegelitem, pedro_band_gel_1d⟩⟩
⟨⟨pedro_digegelitem, pedro_band_id⟩⟩
⟨⟨pedro_digegelitem, pedro_dye_type⟩⟩
⟨⟨pedro_digegelitem, pedro_id⟩⟩
⟨⟨pedro_digegelitem, pedro_spot_gel_2d⟩⟩
⟨⟨pedro_digegelitem, pedro_spot_id⟩⟩
⟨⟨pedro_electrospray, pedro_cone_voltage⟩⟩
⟨⟨pedro_electrospray, pedro_id⟩⟩
⟨⟨pedro_electrospray, pedro_interface_manufacturer⟩⟩
⟨⟨pedro_electrospray, pedro_ion_source⟩⟩
⟨⟨pedro_electrospray, pedro_loading_type⟩⟩
⟨⟨pedro_electrospray, pedro_solution_voltage⟩⟩
⟨⟨pedro_electrospray, pedro_solvent⟩⟩
⟨⟨pedro_electrospray, pedro_spray_tip_diameter⟩⟩
⟨⟨pedro_electrospray, pedro_spray_tip_manufacturer⟩⟩
⟨⟨pedro_electrospray, pedro_spray_tip_voltage⟩⟩
⟨⟨pedro_experiment, pedro_hypothesis⟩⟩
⟨⟨pedro_experiment, pedro_id⟩⟩
⟨⟨pedro_experiment, pedro_method_citations⟩⟩
⟨⟨pedro_experiment, pedro_result_citations⟩⟩
⟨⟨pedro_fraction, pedro_end_point⟩⟩
⟨⟨pedro_fraction, pedro_id⟩⟩
⟨⟨pedro_fraction, pedro_lc_column⟩⟩
⟨⟨pedro_fraction, pedro_protein_assay⟩⟩
⟨⟨pedro_fraction, pedro_start_point⟩⟩
⟨⟨pedro_fraction_otm_analyteps, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_fraction_otm_analyteps, pedro_fraction_id⟩⟩
⟨⟨pedro_fraction_otm_analyteps, pedro_fraction_lc_column⟩⟩
⟨⟨pedro_gel1d, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_gel1d, pedro_annotated_image⟩⟩
⟨⟨pedro_gel1d, pedro_background⟩⟩
⟨⟨pedro_gel1d, pedro_denaturing_agent⟩⟩
⟨⟨pedro_gel1d, pedro_description⟩⟩
⟨⟨pedro_gel1d, pedro_equipment⟩⟩
⟨⟨pedro_gel1d, pedro_id⟩⟩
⟨⟨pedro_gel1d, pedro_in_gel_digestion⟩⟩
⟨⟨pedro_gel1d, pedro_mass_end⟩⟩
⟨⟨pedro_gel1d, pedro_mass_start⟩⟩
⟨⟨pedro_gel1d, pedro_percent_acrylamide⟩⟩
⟨⟨pedro_gel1d, pedro_pixel_size_x⟩⟩
⟨⟨pedro_gel1d, pedro_pixel_size_y⟩⟩
⟨⟨pedro_gel1d, pedro_protein_assay⟩⟩
⟨⟨pedro_gel1d, pedro_raw_image⟩⟩
⟨⟨pedro_gel1d, pedro_run_details⟩⟩
⟨⟨pedro_gel1d, pedro_software_version⟩⟩
⟨⟨pedro_gel1d, pedro_solubilization_buffer⟩⟩
⟨⟨pedro_gel1d, pedro_stain_details⟩⟩
⟨⟨pedro_gel1d, pedro_warped_image⟩⟩
⟨⟨pedro_gel1d, pedro_warping_map⟩⟩
⟨⟨pedro_gel2d, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_gel2d, pedro_annotated_image⟩⟩
⟨⟨pedro_gel2d, pedro_background⟩⟩
⟨⟨pedro_protein⟩⟩
⟨⟨pedro_proteinhit⟩⟩
⟨⟨pedro_quadrupole⟩⟩
⟨⟨pedro_relatedgelitem⟩⟩
⟨⟨pedro_relgelitem_mtm_proteinhit⟩⟩
⟨⟨pedro_sample_mtm_sampleorigin⟩⟩
⟨⟨pedro_sample_otm_analyteps⟩⟩
⟨⟨pedro_sample⟩⟩
⟨⟨pedro_sampleorigin⟩⟩
⟨⟨pedro_spot_otm_analyteps⟩⟩
⟨⟨pedro_spot⟩⟩
⟨⟨pedro_taggingprocess⟩⟩
⟨⟨pedro_tandemsequencedata⟩⟩
⟨⟨pedro_tof⟩⟩
⟨⟨pedro_treatedanalyte_otm_analyteps⟩⟩
⟨⟨pedro_treatedanalyte⟩⟩
⟨⟨pepseeker_proteinhit⟩⟩
⟨⟨pepseeker_proteinscore, pepseeker_Score⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_ProteinID⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_peptidehit⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_proteinscore⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_pepseq⟩⟩
⟨⟨pepseeker_mascotexpect, pepseeker_expect⟩⟩
⟨⟨pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Username⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Id⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Filename⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TAXONOMY⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_MODS⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_IT_MODS⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_PFA⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_MASS⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ITOL⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TOL⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SEG⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TOLU⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_CHARGE⟩⟩
⟨⟨pepseeker_searchmasses⟩⟩
⟨⟨pepseeker_searchmasses, pepseeker_Products⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_Proteinname⟩⟩
⟨⟨pepseeker_species, pepseeker_species⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_Mass⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_startresidue⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_endresidue⟩⟩
⟨⟨pepseeker_db, pepseeker_entries⟩⟩
⟨⟨pepseeker_db, pepseeker_id⟩⟩
⟨⟨pepseeker_db, pepseeker_location⟩⟩
⟨⟨pepseeker_db, pepseeker_name⟩⟩
⟨⟨pepseeker_db, pepseeker_release_date⟩⟩
⟨⟨pepseeker_db, pepseeker_release⟩⟩
⟨⟨pepseeker_db, pepseeker_residue_type⟩⟩
⟨⟨pepseeker_db, pepseeker_system_date⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ACCESSION⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_CLE⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_DB⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ERRORTOLERANT⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_FORMAT⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_FORMVER⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_FRAG⟩⟩

⟨⟨pedro_gel2d, pedro_description⟩⟩
⟨⟨pedro_gel2d, pedro_equipment⟩⟩
⟨⟨pedro_gel2d, pedro_first_dim_details⟩⟩
⟨⟨pedro_gel2d, pedro_id⟩⟩
⟨⟨pedro_gel2d, pedro_in_gel_digestion⟩⟩
⟨⟨pedro_gel2d, pedro_mass_end⟩⟩
⟨⟨pedro_gel2d, pedro_mass_start⟩⟩
⟨⟨pedro_gel2d, pedro_percent_acrylamide⟩⟩
⟨⟨pedro_gel2d, pedro_pi_end⟩⟩
⟨⟨pedro_gel2d, pedro_pi_start⟩⟩
⟨⟨pedro_gel2d, pedro_pixel_size_x⟩⟩
⟨⟨pedro_gel2d, pedro_pixel_size_y⟩⟩
⟨⟨pedro_gel2d, pedro_protein_assay⟩⟩
⟨⟨pedro_gel2d, pedro_raw_image⟩⟩
⟨⟨pedro_gel2d, pedro_second_dim_details⟩⟩
⟨⟨pedro_gel2d, pedro_software_version⟩⟩
⟨⟨pedro_gel2d, pedro_solubilization_buffer⟩⟩
⟨⟨pedro_gel2d, pedro_stain_details⟩⟩
⟨⟨pedro_gel2d, pedro_warped_image⟩⟩
⟨⟨pedro_gel2d, pedro_warping_map⟩⟩
⟨⟨pedro_gradientstep, pedro_id⟩⟩
⟨⟨pedro_gradientstep, pedro_lc_column⟩⟩
⟨⟨pedro_gradientstep, pedro_step_time⟩⟩
⟨⟨pedro_hexapole, pedro_description⟩⟩
⟨⟨pedro_hexapole, pedro_id⟩⟩
⟨⟨pedro_hexapole, pedro_mz_analysis⟩⟩
⟨⟨pedro_ionsource, pedro_collision_energy⟩⟩
⟨⟨pedro_ionsource, pedro_id⟩⟩
⟨⟨pedro_ionsource, pedro_mz_analysis⟩⟩
⟨⟨pedro_ionsource, pedro_type⟩⟩
⟨⟨pedro_iontrap, pedro_excitation_amplitude⟩⟩
⟨⟨pedro_iontrap, pedro_final_ms_level⟩⟩
⟨⟨pedro_iontrap, pedro_gas_pressure⟩⟩
⟨⟨pedro_iontrap, pedro_gas_type⟩⟩
⟨⟨pedro_iontrap, pedro_id⟩⟩
⟨⟨pedro_iontrap, pedro_isolation_centre⟩⟩
⟨⟨pedro_iontrap, pedro_isolation_width⟩⟩
⟨⟨pedro_iontrap, pedro_mz_analysis⟩⟩
⟨⟨pedro_iontrap, pedro_rf_frequency⟩⟩
⟨⟨pedro_lccolumn, pedro_analyte_processing_step⟩⟩
⟨⟨pedro_lccolumn, pedro_batch_number⟩⟩
⟨⟨pedro_lccolumn, pedro_bead_size⟩⟩
⟨⟨pedro_lccolumn, pedro_description⟩⟩
⟨⟨pedro_lccolumn, pedro_flow_rate⟩⟩
⟨⟨pedro_lccolumn, pedro_id⟩⟩
⟨⟨pedro_lccolumn, pedro_injection_volume⟩⟩
⟨⟨pedro_lccolumn, pedro_internal_diameter⟩⟩
⟨⟨pedro_lccolumn, pedro_internal_length⟩⟩
⟨⟨pedro_lccolumn, pedro_lc_column⟩⟩
⟨⟨pedro_lccolumn, pedro_manufacturer⟩⟩
⟨⟨pedro_lccolumn, pedro_parameters_file⟩⟩
⟨⟨pedro_lccolumn, pedro_part_number⟩⟩
⟨⟨pedro_lccolumn, pedro_pore_size⟩⟩
⟨⟨pedro_lccolumn, pedro_stationary_phase⟩⟩
⟨⟨pedro_lccolumn, pedro_temperature⟩⟩
⟨⟨pedro_listprocessing, pedro_background_threshold⟩⟩
⟨⟨pedro_listprocessing, pedro_id⟩⟩
⟨⟨pedro_listprocessing, pedro_peak_list⟩⟩
⟨⟨pedro_listprocessing, pedro_smoothing_process⟩⟩
⟨⟨pedro_maldi, pedro_acceleration_voltage⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ICAT⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_INSTRUMENT⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_INTERMEDIATE⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ITH⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ITOLU⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_LTOL⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_mp⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_NM⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_OVERVIEW⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_PEAK⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_PRECURSOR⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_ProcessNo⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_QUE⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_REPORT⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_REPTYPE⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SEARCH⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SEGT⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SEGTU⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_SUBCLUSTER⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TWO⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Useremail⟩⟩
⟨⟨pepseeker_fp_to_db, pepseeker_db_id⟩⟩
⟨⟨pepseeker_fp_to_db, pepseeker_fp_id⟩⟩
⟨⟨pepseeker_iontable, pepseeker_A⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Aplusplus⟩⟩
⟨⟨pepseeker_iontable, pepseeker_AStar⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Astarplusplus⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Azero⟩⟩
⟨⟨pepseeker_iontable, pepseeker_B⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Bplusplus⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Bstar⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Bstarplusplus⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Bzero⟩⟩
⟨⟨pepseeker_iontable, pepseeker_BZeroplusplus⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Id⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Immon⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Matches⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Y⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Yplusplus⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Ystar⟩⟩
⟨⟨pepseeker_iontable, pepseeker_Ystarplusplus⟩⟩
⟨⟨pepseeker_iontable, pepseeker_YZero⟩⟩
⟨⟨pepseeker_iontable, pepseeker_YZeroplusplus⟩⟩
⟨⟨pepseeker_lastsession, pepseeker_Id⟩⟩
⟨⟨pepseeker_lastsession, pepseeker_url⟩⟩
⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_Delta⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_ions⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_MassNo⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_MissCleav⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_MrCalc⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_MrExpct⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_Rank⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_Score⟩⟩
⟨⟨pepseeker_peptideprophet, pepseeker_peptidehit_Id⟩⟩
⟨⟨pepseeker_peptideprophet, pepseeker_tpp_pvalue⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_Id⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_ProteinScore⟩⟩
⟨⟨pepseeker_proteinscore, pepseeker_Id⟩⟩

⟨⟨pedro_maldi, pedro_grid_voltage⟩⟩	⟨⟨pepseeker_proteinscore, pepseeker_Matchedpeptides⟩⟩
⟨⟨pedro_maldi, pedro_id⟩⟩	⟨⟨pepseeker_proteinscore, pepseeker_querypeptides⟩⟩
⟨⟨pedro_maldi, pedro_ion_mode⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_fileparameters⟩⟩
⟨⟨pedro_maldi, pedro_ion_source⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Id⟩⟩
⟨⟨pedro_maldi, pedro_laser_power⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Int_max⟩⟩
⟨⟨pedro_maldi, pedro_laser_wavelength⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_int_min⟩⟩
⟨⟨pedro_maldi, pedro_matrix_type⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Mass_Max⟩⟩
⟨⟨pedro_massspecexperiment, pedro_analyte_processing_step⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Mass_min⟩⟩
⟨⟨pedro_massspecexperiment, pedro_description⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_num_used⟩⟩
⟨⟨pedro_massspecexperiment, pedro_id⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_num_vals⟩⟩
⟨⟨pedro_massspecexperiment, pedro_parameters_file⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Precursor⟩⟩
⟨⟨pedro_massspecmachine, pedro_id⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_queryno⟩⟩
⟨⟨pedro_massspecmachine, pedro_ion_source⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_searchmassescol⟩⟩
⟨⟨pedro_massspecmachine, pedro_manufacturer⟩⟩	⟨⟨pepseeker_searchmasses, pepseeker_Title⟩⟩
⟨⟨pedro_massspecmachine, pedro_model_name⟩⟩	⟨⟨pepseeker_species, pepseeker_fp_id⟩⟩
⟨⟨pedro_massspecmachine, pedro_software_version⟩⟩	⟨⟨pepseeker_species, pepseeker_ProteinId⟩⟩
⟨⟨pedro_mobilephasecomponent, pedro_concentration⟩⟩	⟨⟨pepseeker_species, pepseeker_species_id⟩⟩
⟨⟨pedro_mobilephasecomponent, pedro_description⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_accession⟩⟩
⟨⟨pedro_mobilephasecomponent, pedro_id⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_id⟩⟩
⟨⟨pedro_mobilephasecomponent, pedro_lc_column⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_oln⟩⟩
⟨⟨pedro_msmsfraction, pedro_id⟩⟩	⟨⟨pepseeker_yeastacc, pepseeker_yeast_acc_id⟩⟩
⟨⟨pedro_msmsfraction, pedro_peak_list⟩⟩	⟨⟨pepseeker_db⟩⟩
⟨⟨pedro_msmsfraction, pedro_plus_or_minus⟩⟩	⟨⟨pepseeker_fp_to_db⟩⟩
⟨⟨pedro_msmsfraction, pedro_target_m_to_z⟩⟩	⟨⟨pepseeker_iontable⟩⟩
⟨⟨pedro_mzanalysis, pedro_detection⟩⟩	⟨⟨pepseeker_lastsession⟩⟩
⟨⟨pedro_mzanalysis, pedro_id⟩⟩	⟨⟨pepseeker_mascotexpect⟩⟩
⟨⟨pedro_mzanalysis, pedro_type⟩⟩	⟨⟨pepseeker_peptideprophet⟩⟩
⟨⟨pedro_ontologyentry, pedro_category⟩⟩	⟨⟨pepseeker_proteinscore⟩⟩
⟨⟨pedro_ontologyentry, pedro_description⟩⟩	⟨⟨pepseeker_species⟩⟩
⟨⟨pedro_ontologyentry, pedro_id⟩⟩	⟨⟨pepseeker_yeastacc⟩⟩
⟨⟨pedro_ontologyentry, pedro_value⟩⟩	
⟨⟨pedro_organism, pedro_id⟩⟩	
⟨⟨pedro_organism, pedro_relevant_genotype⟩⟩	
⟨⟨pedro_organism, pedro_species_name⟩⟩	
⟨⟨pedro_organism, pedro_strain_identifier⟩⟩	
⟨⟨pedro_otheranalyte, pedro_id⟩⟩	
⟨⟨pedro_otheranalyte, pedro_name⟩⟩	
⟨⟨pedro_otheranalyte, pedro_other_analyte_processing_step⟩⟩	
⟨⟨pedro_otheranalyte_otm_analyteps, pedro_analyte_processing_step⟩⟩	
⟨⟨pedro_otheranalyte_otm_analyteps, pedro_other_analyte⟩⟩	
⟨⟨pedro_otheranalyte_otm_ontent, pedro_ontology_entry⟩⟩	
⟨⟨pedro_otheranalyte_otm_ontent, pedro_other_analyte⟩⟩	
⟨⟨pedro_otheranalyteps, pedro_analyte_processing_step⟩⟩	
⟨⟨pedro_otheranalyteps, pedro_id⟩⟩	
⟨⟨pedro_otheranalyteps, pedro_name⟩⟩	
⟨⟨pedro_otheranalyteps_otm_ontent, pedro_ontology_entry⟩⟩	
⟨⟨pedro_otheranalyteps_otm_ontent, pedro_other_analyte_processing_step⟩⟩	
⟨⟨pedro_otherionisation, pedro_id⟩⟩	
⟨⟨pedro_otherionisation, pedro_ion_source⟩⟩	
⟨⟨pedro_otherionisation, pedro_name⟩⟩	
⟨⟨pedro_otherionisation_otm_ontent, pedro_ontology_entry⟩⟩	
⟨⟨pedro_otherionisation_otm_ontent, pedro_other_ionisation⟩⟩	
⟨⟨pedro_othermzanalysis, pedro_id⟩⟩	
⟨⟨pedro_othermzanalysis, pedro_mz_analysis⟩⟩	

Table E.13: Elements Satisfying Quality Factor 1 in Iteration 3

Factor 4

Satisfying Items	not-Satisfying Items
⟨⟨gpmdb_aa, gpmdb_aaid⟩⟩	⟨⟨gpmdb_peptide, gpmdb_pepid⟩⟩
⟨⟨gpmdb_aa, gpmdb_pepid⟩⟩	⟨⟨gpmdb_fullpeptide, gpmdb_pepid⟩⟩
⟨⟨gpmdb_peptide, gpmdb_seq⟩⟩	⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_pepid⟩⟩
⟨⟨gpmdb_peptide⟩⟩	⟨⟨pedro_peptidehit⟩⟩
⟨⟨gpmdb_proseq, gpmdb_label⟩⟩	⟨⟨pedro_peptidehit, pedro_id⟩⟩
⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩	⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩	⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩
⟨⟨gpmdb_protein⟩⟩	⟨⟨pepseeker_peptideprophet, pepseeker_peptidehit_Id⟩⟩
⟨⟨pedro_peptidehit, pedro_score⟩⟩	⟨⟨gpmdb_protein, gpmdb_proid⟩⟩
⟨⟨pedro_peptidehit, pedro_sequence⟩⟩	⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩
⟨⟨pedro_proteinhit, pedro_protein⟩⟩	⟨⟨gpmdb_fullpeptide, gpmdb_proid⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_pepseq⟩⟩	⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_proid⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_proteinscore⟩⟩	⟨⟨pedro_protein⟩⟩
⟨⟨pepseeker_peptidehit⟩⟩	⟨⟨pedro_protein, pedro_id⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_ProteinID⟩⟩	⟨⟨pedro_proteinhit, pedro_id⟩⟩
⟨⟨pepseeker_proteinhit⟩⟩	⟨⟨pepseeker_proteinhit, pepseeker_Id⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_value_type⟩⟩	⟨⟨pepseeker_species, pepseeker_ProteinID⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_mass_error_type⟩⟩	⟨⟨gpmdb_result⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_MASS⟩⟩	⟨⟨gpmdb_result, gpmdb_resultid⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_TOLU⟩⟩	⟨⟨gpmdb_project, gpmdb_resultid⟩⟩
⟨⟨pepseeker_iontable⟩⟩	⟨⟨gpmdb_fullpeptide, gpmdb_seq⟩⟩
⟨⟨gpmdb_fullpeptidediagnostic, gpmdb_seq⟩⟩	
⟨⟨gpmdb_proseq⟩⟩	
⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩	
⟨⟨gpmdb_proseq, gpmdb_seq⟩⟩	
⟨⟨pedro_protein, pedro_sequence⟩⟩	
⟨⟨pepseeker_proteinscore, pepseeker_Id⟩⟩	
⟨⟨pepseeker_proteinscore, pepseeker_Score⟩⟩	
⟨⟨pepseeker_proteinhit, pepseeker_ProteinScore⟩⟩	
⟨⟨gpmdb_aa⟩⟩	
⟨⟨pedro_peaklist, pedro_mass_value_type⟩⟩	
⟨⟨pepseeker_proteinhit, pepseeker_Mass⟩⟩	
⟨⟨pepseeker_peptidehit, pepseeker_MassNo⟩⟩	
⟨⟨pepseeker_fileparameters, pepseeker_ITOLU⟩⟩	
⟨⟨pepseeker_iontable, pepseeker_id⟩⟩	
⟨⟨pepseeker_peptidehit, pepseeker_ions⟩⟩	

Table E.14: Elements Satisfying and not-Satisfying Quality Factor 4 in Iteration 3

Factor 7

⟨⟨pedro_fk_dbsearch_dbsearchparameters1, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩, pedro_dbsearchparameters, ⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩⟩⟩
⟨⟨pedro_fk_proteinhit_dbsearch1, pedro_proteinhit, ⟨⟨pedro_proteinhit, pedro_db_search⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_id⟩⟩⟩⟩
⟨⟨gpmdb_result⟩⟩
⟨⟨pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_fileparameters, Id⟩⟩
⟨⟨gpmdb_protein⟩⟩
⟨⟨gpmdb_protein, proseqid⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩
⟨⟨pepseeker_proteinhit, pepseeker_fileparameters⟩⟩
⟨⟨gpmdb_peptide, gpmdb_pepid⟩⟩
⟨⟨gpmdb_fk_peptide_protein1, gpmdb_peptide, ⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩, gpmdb_protein, ⟨⟨gpmdb_protein, gpmdb_proid⟩⟩⟩⟩
⟨⟨gpmdb_peptide, gpmdb_proid⟩⟩
⟨⟨gpmdb_protein, gpmdb_proid⟩⟩
⟨⟨gpmdb_fk_protein_proseq, gpmdb_protein, ⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩, gpmdb_proseq, ⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩⟩⟩
⟨⟨gpmdb_protein, gpmdb_proseqid⟩⟩
⟨⟨gpmdb_proseq, gpmdb_proseqid⟩⟩
⟨⟨gpmdb_fk_protein_result1, gpmdb_protein, ⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩, gpmdb_result, ⟨⟨gpmdb_result, gpmdb_resultid⟩⟩⟩⟩
⟨⟨gpmdb_protein, gpmdb_resultid⟩⟩
⟨⟨gpmdb_result, gpmdb_resultid⟩⟩
⟨⟨pedro_fk_dbsearch_dbsearchparameters1, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩, pedro_dbsearchparameters, ⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩⟩⟩
⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearchparameters, pedro_id⟩⟩
⟨⟨pedro_fk_chromatogrampoint_peak1, pedro_chromatogrampoint, ⟨⟨pedro_chromatogrampoint, pedro_peak⟩⟩, pedro_peak, ⟨⟨pedro_peak, pedro_id⟩⟩⟩⟩
⟨⟨pedro_chromatogrampoint, pedro_peak⟩⟩
⟨⟨pedro_peak, pedro_id⟩⟩
⟨⟨pedro_fk_dbsearch_peaklist1, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_dbsearch, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_listprocessing_peaklist1, pedro_listprocessing, ⟨⟨pedro_listprocessing, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_listprocessing, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_msmsfraction_peaklist1, pedro_msmsfraction, ⟨⟨pedro_msmsfraction, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_msmsfraction, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_peak_peaklist, pedro_peak, ⟨⟨pedro_peak, pedro_peak_list⟩⟩, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_id⟩⟩⟩⟩
⟨⟨pedro_peak, pedro_peak_list⟩⟩
⟨⟨pedro_peaklist, pedro_id⟩⟩
⟨⟨pedro_fk_peaklist_massspecmachine1, pedro_peaklist, ⟨⟨pedro_peaklist, pedro_mass_spec_experiment⟩⟩, pedro_massspecmachine, ⟨⟨pedro_massspecmachine, pedro_id⟩⟩⟩⟩
⟨⟨pedro_peaklist, pedro_mass_spec_experiment⟩⟩
⟨⟨pedro_massspecmachine, pedro_id⟩⟩
⟨⟨pedro_fk_peptidehit_dbsearch1, pedro_peptidehit, ⟨⟨pedro_peptidehit, pedro_db_search⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_id⟩⟩⟩⟩

⟨⟨pedro_peptidehit, pedro_db_search⟩⟩
⟨⟨pedro_dbsearch, pedro_id⟩⟩
⟨⟨pedro_fk_proteinhit_dbsearch1, pedro_proteinhit, ⟨⟨pedro_proteinhit, pedro_db_search⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_id⟩⟩⟩⟩
⟨⟨pedro_proteinhit, pedro_db_search⟩⟩
⟨⟨pedro_dbsearch, pedro_id⟩⟩
⟨⟨pedro_fk_tandemsequencedata_dbsearch1, pedro_tandemsequencedata, ⟨⟨pedro_tandemsequencedata, pedro_db_search_parameters⟩⟩, pedro_dbsearch, ⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩⟩⟩
⟨⟨pedro_tandemsequencedata, pedro_db_search_parameters⟩⟩
⟨⟨pedro_dbsearch, pedro_db_search_parameters⟩⟩
⟨⟨pepseeker_fk_mascotexpect_peptidehit1, pepseeker_mascotexpect, ⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩, pepseeker_peptidehit, ⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩⟩⟩
⟨⟨pepseeker_mascotexpect, pepseeker_peptidehit_Id⟩⟩
⟨⟨pepseeker_peptidehit, pepseeker_Id⟩⟩
⟨⟨pepseeker_fk_searchmasses_fileparameters, pepseeker_searchmasses, ⟨⟨pepseeker_searchmasses, pepseeker_fileparameters⟩⟩, pepseeker_fileparameters, ⟨⟨pepseeker_fileparameters, pepseeker_Id⟩⟩⟩⟩
⟨⟨pepseeker_searchmasses, pepseeker_fileparameters⟩⟩
⟨⟨pepseeker_fileparameters, pepseeker_Id⟩⟩
⟨⟨GS1_fk_proteinhit_protein1, GS1_proteinhit, ⟨⟨GS1_proteinhit, GS1_id⟩⟩, GS1_protein, ⟨⟨GS1_protein, GS1_id⟩⟩⟩⟩
⟨⟨pepseeker_proteinhit⟩⟩
⟨⟨GS1_proteinhit, GS1_id⟩⟩
⟨⟨GS1_protein, GS1_id⟩⟩

Table E.15: Elements Satisfying Quality Factor 7 in Iteration 3

Appendix F

Glossary of Terms

Syntax	**Meaning**
$concepts(S, O)$	The set of concepts represented by all schema constructs in schema S with respect to a domain ontology O.
$consistent(extensional(S), O)$	The set of schema constructs that are consistent with the definitions of their corresponding real-world concepts with respect to a domain ontology O.
$consistent(q, S)$	Assigned 1 if q evaluates to true on a schema, or schemas, S, otherwise assigned 0.
$constraints(S)$	The set of constraint constructs of a schema, or schemas, S.
$construct(q)$	The set of schema constructs referenced in q.
$corr(q, reformulate(q', M))$	Assigned the value 1 if q is contained by $reformulate(q', M)$, otherwise assigned 0.
$evaluate(q, q')$	Assigned the value 1 if both q and each member of q' evaluate to true, otherwise assigned 0.
$ext(o)$	The extent of the schema construct o.

$ext(q)$	The result set returned by a query or a set of queries, q.
$extensional(GS, M_{GAV})$	The set of extensional GS constructs derived by M_{GAV}.
$extensional(S)$	The set of extensional schema constructs of a schema, or schemas, S.
GS	The global schema
$LAVdefined(LSs, M_{LAV})$	The set of the local schema constructs, o, such that there is a LAV mapping whose LHS is o.
LSs	The local schemas
M	Mappings
$reduce(C, O)$	The set of unique real-world concepts in a group of concepts, C, obtained by removing concepts that are equivalent to or subsumed by other concepts in the ontology O.
$reformulate(q, M)$	The set of reformulated queries for a query q via mappings M.
$removed(constraints(S), M_{GAV})$	The set of constraint constructs in schema, or schemas, S that are deleted by the GAV mappings.
$source^{corr}(S, M, c)$	The set of constraint constructs in schema, or schemas, S where all constraints o in this set and c are restricting the same set of extents.
$source^{corr}(S, M, o)$	The set of extensional schema constructs in schema, or schemas, S from which $o \in extensional(S')$ is derived via the mappings M.
$sources(S, M, o)$	The set of constructs of schema, or schemas, S, from which construct o is derived via the mappings M.

$sources(S, M, S')$	The set of constructs of schema, or schemas, S, from which constructs of schema S' are derived via the mappings M.
S	A schema or schemas S
T	AutoMed Transformations.

Bibliography

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003.

[2] Matthias Jarke, Y. Vassiliou, P. Vassiliadis, and M. Lenzerini. *Fundamentals of Data Warehouses*. 1999.

[3] Maurizio Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, 2002.

[4] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30:174–210, 2005.

[5] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Logical foundations of peer-to-peer data integration. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 241–251, 2004.

[6] Richard Hull. Managing semantic heterogeneity in databases: a theoretical prospective. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '97, pages 51–61, 1997.

[7] Yaser A. Bishr. Overcoming the semantic and other barriers to gis interoperability. *International Journal of Geographical Information Science*, 12(4):299–314, 1998.

[8] Amit P. Sheth. Changing focus on interoperability in information systems: from system, syntax, structure to semantics. *Interoperating Geographic Information Systems*, pages 5–30, 1999.

[9] Mokrane Bouzeghoub and Maurizio Lenzerini. Introduction to the special issue on data extraction, cleaning, and reconciliation. *Inf. Syst.*, 26(8):535–536, 2001.

[10] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[11] Lucas Zamboulis, Nigel Martin, and Alexandra Poulovassilis. Bioinformatics service reconciliation by heterogeneous schema transformation. In *Proceedings of the 4th international conference on Data integration in the life sciences*, pages 89–104, 2007.

[12] Dionysios C. Tsichritzis and Frederick H. Lochovsky. *Data Models*. Prentice Hall Professional Technical Reference, 1982.

[13] Bogdan Alexe, Wang Chiew Tan, and Yannis Velegrakis. Stbenchmark: towards a benchmark for mapping systems. *PVLDB*, 1(1):230–244, 2008.

[14] Bogdan Alexe, Wang-Chiew Tan, and Yannis Velegrakis. Comparing and evaluating mapping systems with stbenchmark. *Proc. VLDB Endow.*, 1:1468–1471, August 2008.

[15] Elena Beisswanger, Stefan Schulz, Holger Stenzhorn, and Udo Hahn. Biotop: An upper domain ontology for the life sciences: A description

of its current structure, contents and interfaces to obo ontologies. *Appl. Ontol.*, 3:205–212, 2008.

[16] A. L. Rector, J. E. Rogers, P. E. Zanstra, and E. van der Haring. Opengalen: Open source medical terminology and tools. In *Proc AMIA Symp*, volume 982, 2003.

[17] David Aumueller, Hong Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with coma++. In *SIGMOD Conference*, pages 906–908, 2005.

[18] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 49–58, 2001.

[19] Marc Friedman, Alon Levy, and Todd Millstein. Navigational plans for data integration. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 67–73, 1999.

[20] Alexandra Poulovassilis and Peter Mc. Brien. A general formal framework for schema transformation. *Data Knowl. Eng.*, 28:47–71, October 1998.

[21] Alon Y. Halevy, Naveen Ashish, Dina Bitton, Michael Carey, Denise Draper, Jeff Pollock, Arnon Rosenthal, and Vishal Sikka. Enterprise information integration: successes, challenges and controversies. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 778–787, 2005.

[22] Renée J. Miller, Mauricio A. Hernández, Laura M. Haas, Ling-Ling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa. The clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.

[23] Angela Bonifati, Elaine Qing Chang, Aks V. S. Lakshmanan, Terence Ho, and Rachel Pottinger. Heptox: marrying xml and heterogeneity in your p2p databases. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 1267–1270, 2005.

[24] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Paolo Naggar, and Fabio Vernacotola. Ibis: semantic data integration at work. In *Proceedings of the 15th international conference on Advanced information systems engineering*, CAiSE'03, pages 79–94, 2003.

[25] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10:270–294, December 2001.

[26] Diego Calvanese, Domenico Lembo, and Maurizio Lenzerini. Survey on methods for query rewriting and query answering using views, 2001.

[27] Laura Chiticariu and Wang-Chiew Tan. Debugging schema mappings with routes. In *Proceedings of the 32nd international conference on Very large data bases*, pages 79–90, 2006.

[28] Khalid Belhajjame, Norman W. Paton, Alvaro A. A. Fernandes, Cornelia Hedeler, and Suzanne M. Embury. User feedback as a first class citizen in information integration systems. In *CIDR*, pages 175–183, 2011.

[29] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Description logics for information integration. In *Computational Logic:*

Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II, pages 41–60, 2002.

[30] Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *Knowl. Eng. Rev.*, 18:1–31, 2003.

[31] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Rec.*, 35(3):34–41, 2006.

[32] Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

[33] Natalya Fridman Noy and Mark A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *AAAI/IAAI*, pages 450–455, 2000.

[34] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. *SIGMOD Rec.*, 30(2):509–520, 2001.

[35] Chokri Ben Necib and Johann Christoph Freytag. Using ontologies for database query reformulation. In *ADBIS (Local Proceedings)*, 2004.

[36] Luciano Serafini and Andrei Tamilin. Drago: Distributed reasoning architecture for the semantic web. In *ESWC*, pages 361–376, 2005.

[37] Christian Meilicke, Heiner Stuckenschmidt, and Ondrej Sváb-Zamazal. A reasoning-based support tool for ontology mapping evaluation. In *ESWC*, pages 878–882, 2009.

[38] Lucas Zamboulis, Alexandra Poulovassilis, and Jianing Wang. Ontology-assisted data transformation and integration. In *ODBIS*, pages 29–36, 2008.

[39] Michael Gertz. Managing data quality and integrity in federated databases. In *Proceedings of the IFIP TC11 Working Group 11.5, Second Working Conference on Integrity and Internal Control in Information Systems: Bridging Business Requirements and Research Results*, pages 211–230, 1998.

[40] Felix Naumann, Ulf Leser, and Johann Christoph Freytag. Quality-driven integration of heterogenous information systems. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 447–458, 1999.

[41] Ulf Leser, Felix Naumann, and Barbara A. Eckman, editors. *Data Integration in the Life Sciences, Third International Workshop, DILS 2006, Hinxton, UK, July 20-22, 2006, Proceedings*, volume 4075 of *Lecture Notes in Computer Science*. Springer, 2006.

[42] Angela Bonifati, Giansalvatore Mecca, Alessandro Pappalardo, Salvatore Raunich, and Gianvito Summa. Schema mapping verification: the spicy way. In *EDBT*, pages 85–96, 2008.

[43] Maria da Conceição Moraes Batista and Ana Carolina Salgado. Information quality measurement in data integration schemas. In *QDB*, pages 61–72, 2007.

[44] Fabien Duchateau and Zohra Bellahsene. Measuring the quality of an integrated schema. In *Proceedings of the 29th international conference on Conceptual modeling*, pages 261–273, 2010.

[45] Ling Ling Yan, Renée J. Miller, Laura M. Haas, and Ronald Fagin. Data-driven understanding and refinement of schema mappings. *SIGMOD Rec.*, 30:485–496, May 2001.

[46] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the role of integrity constraints in data integration. *IEEE Data Eng. Bull.*, 25(3):39–45, 2002.

[47] Cong Yu and Lucian Popa. Constraint-based xml query rewriting for data integration. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 371–382, 2004.

[48] Todd Millstein, Alon Halevy, and Marc Friedman. Query containment for data integration systems. In *J. Comput. Syst. Sci.*, volume 66, pages 20–39, 2003.

[49] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query containment. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 56–67, 2003.

[50] Alan Nash, Alin Deutsch, and Jeff REmmel. Data exchange, data integration and chase. Technical report, University Of California, San Diego, 2006. http://www-cse.ucsd.edu/Dienst/UI/2.0/Describe/ncstrl.ucsd_cse/CS2006-0859.

[51] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tackling inconsistencies in data integration through source preferences. In *Proceedings of the 2004 international workshop on Information quality in information systems*, IQIS '04, pages 27–34, 2004.

[52] Gianluigi Greco and Domenico Lembo. Data integration with preferences among sources. In *ER*, pages 231–244, 2004.

[53] Luca Cabibbo. On keys, foreign keys and nullable attributes in relational mapping systems. In *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology*, pages 263–274, 2009.

[54] Alessandro Raffio, Daniele Braga, Stefano Ceri, Paolo Papotti, and Mauricio A. Hernandez. Clip: a visual language for explicit schema mappings. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 30–39, 2008.

[55] Alexander Borgida and Luciano Serafini. Distributed description logics: Assimilating information from peer sources. *J. Data Semantics*, 1:153–184, 2003.

[56] Matthias Jarke and Yannis Vassiliou. Data warehouse quality: A review of the dwq project. In *IQ*, pages 299–313, 1997.

[57] Stefan Poslad and Landong Zuo. An adaptive semantic framework to support multiple user viewpoints over multiple databases. In *Advances in Semantic Media Adaptation and Personalization*, pages 261–284. 2008.

[58] Peter McBrien and Alexandra Poulovassilis. A formalisation of semantic schema integration. *Inf. Syst.*, 23(5):307–334, 1998.

[59] Peter McBrien and Alexandra Poulovassilis. A uniform approach to inter-model transformations. In *Proceedings of the 11th International Conference on Advanced Information Systems Engineering*, pages 333–348, 1999.

[60] Alexandra Poulovassilis. The automed intermediate query langauge. Technical report 2, Department of Computer Science & Information Systems, Birkbeck College, June 2001.

[61] Peter Buneman, Leonid Libkin, Dan Suciu, Val Tannen, and Limsoon Wong. Comprehension syntax. *SIGMOD Rec.*, 23:87–96, March 1994.

[62] Leonidas Fegaras and David Maier. Towards an effective calculus for object query languages. *SIGMOD Rec.*, 24:47–58, 1995.

[63] Lucas Zamboulis. *XML Data Transformation and Integration - A Schema Transformation Approach*. Phd thesis, Birkbeck College, 2009.

[64] Peter McBrien and Alexandra Poulovassilis. Data integration by bi-directional schema transformation rules. In *ICDE*, pages 227–238, 2003.

[65] Edgar Jasper, Nerissa Tong, Peter McBrien, and Alexandra Poulovassilis. View generation and optimisation in the automed data integration framework. In *CAiSE Short Paper Proceedings*, 2003.

[66] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18:323–364, 1986.

[67] Philip A. Bernstein. Tool requirements for precise data integration. In *Workshop on Information Integration*, 2006. http://db.cis.upenn.edu/iiworkshop/postworkshop/positionPapers/105.pdf.

[68] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. *Inf. Syst.*, 29:147–163, April 2004.

[69] Georg Lausen. Relational databases in RDF: Keys and foreign keys. In Vassilis Christophides, Martine Collard, and Claudio Gutierrez, editors, *Semantic Web, Ontologies and Databases*, pages 43–56. 2008.

[70] Satya S. Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, S02ren Auer, Juan Sequeda, and Ahmed Ezzat. A

survey of current approaches for mapping of relational databases to rdf. Technical report, 2009.

[71] Donald P. Ballou and Harold L. Pazer. Modeling completeness versus consistency tradeoffs in information decision contexts. *IEEE Trans. on Knowl. and Data Eng.*, 15:240–243, 2003.

[72] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning*, pages 161–180, 1999.

[73] Michael Boyd and Peter McBrien. Comparing and transforming between data models via an intermediate hypergraph data model. *J. Data Semantics IV*, pages 69–109, 2005.

[74] Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

[75] Bogdan Alexe, Phokion G. Kolaitis, and Wang-Chiew Tan. Characterizing schema mappings via data examples. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '10, pages 261–272, 2010.

[76] Guillem Rull, Carles Farré, Ernest Teniente, and Toni Urpí. Validation of mappings between schemas. *Data Knowl. Eng.*, 66:414–437, September 2008.

[77] Luca Cabibbo. On keys, foreign keys and nullable attributes in relational mapping systems. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 263–274, 2009.

[78] Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Supporting manual mapping revision using logical reasoning. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, pages 1213–1218, 2008.

[79] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer, 2007.

[80] Avigdor Gal. The generation y of xml schema matching panel description. In *XSym*, pages 137–139, 2007.

[81] Khalid Saleem, Zohra Bellahsene, and Ela Hunt. Porsche: Performance oriented schema mediation. *Inf. Syst.*, 33:637–657, 2008.

[82] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 117–120, 2002.

[83] Lucas Zamboulis, Hao Fan, Khalid Belhajjame, Jennifer A. Siepen, Andrew C. Jones, Nigel J. Martin, Alexandra Poulovassilis, Simon J. Hubbard, Suzanne M. Embury, and Norman W. Paton. Data access and integration in the ispider proteomics grid. In *DILS*, pages 3–18, 2006.

[84] Lucas Zamboulis, Alexandra Poulovassilis, and George Roussos. Flexible data integration and ontology-based data access to medical records. In *BIBE*, pages 1–6, 2008.

[85] Jean-Luc Hainaut, M. Chandelon, C. Tonneau, and M. Joris. Contribution to a theory of database reverse engineering. In *WCRE*, pages 161–170, 1993.

[86] Jianing Wang. A data integration methodology and architecture with quality assessment functionality. In *BNCOD*, pages 151–154, 2010.

[87] Jianing Wang. A quality framework for data integration. In *BNCOD*, pages 131–134, 2010.

[88] Jianing Wang, Nigel Martin, and Alexandra Poulovassilis. An ontology-based quality framework for data integration. In *Workshops on Business Informatics Research*, pages 196 – 208, 2011.