DATA QUALITY BY DESIGN: A GOAL-ORIENTED APPROACH

by

Lei Jiang

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Copyright © 2010 by Lei Jiang
Abstract

Data Quality By Design: A Goal-Oriented Approach

Lei Jiang
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2010

A successful information system is the one that meets its design goals. Expressing these goals and subsequently translating them into a working solution is a major challenge for information systems engineering. This thesis adopts the concepts and techniques from goal-oriented (software) requirements engineering research for conceptual database design, with a focus on data quality issues.

Based on a real-world case study, a goal-oriented process is proposed for database requirements analysis and modeling. It spans from analysis of high-level stakeholder goals to detailed design of a conceptual databases schema. This process is then extended specifically for dealing with data quality issues: data of low quality may be detected and corrected by performing various quality assurance activities; to support these activities, the schema needs to be revised by accommodating additional data requirements. The extended process therefore focuses on analyzing and modeling quality assurance data requirements.

A quality assurance activity supported by a revised schema may involve manual work, and/or rely on some automatic techniques, which often depend on the specification and enforcement of data quality rules. To address the constraint aspect in conceptual database design, data quality rules are classified according to a number of domain and application independent properties. This classification can be used to guide rule designers and to facilitate building of a rule repository. A quantitative framework is then proposed for measuring and comparing DQ rules according to one of these properties: effectiveness; this framework relies on derivation of
formulas that represent the effectiveness of DQ rules under different probabilistic assumptions.
A semi-automatic approach is also presented to derive these effectiveness formulas.
Dedication

To my grandparents

To my parents

To my wife
Acknowledgements

I would like to express my gratitude to Professor John Mylopoulos, my supervisor, who helped me discover this beautiful world of scientific research, and whose vision has been guiding me through this long, tough, yet rewarding journey. John is a person with great character; his supervision is full of care, inspiration and intellectual freedom; his passion towards research and life will always be a role model for me.

I am also very grateful to Professor Alex Borgida, my co-supervisor, who has taught me what it takes to be a real scholar. During the early phase of this work, no matter which direction I were trying pursue, Alex always came up with a sizable list of “suggested readings”, putting me into continuous “reading mode”. Alex is also the one who always understood my (sometimes strange) ideas and challenged me with thoughtful questions; these often led me to deeper thinking. Alex’s knowledgeable and rigorous thinking and scientific attitude lay out the direction for my further scientific quest.

I would also like to thank Professor Thodoros Topaloglou, who provided me with enormous input and support to the early case study on the biological sample database. I am particular thankful for his insight and advice on the first part of this thesis on goal-oriented database design process from a practical point of view. Many thanks also go to Dr. Daniele Barone, whose had been collaborating with me on the last part of this thesis on data quality. I enjoyed a lot the countless discussions and arguments between us.

I would also like to thank Professor Renee Miller for the suggestions and feedback, to Tanya Tang and Reza Samavi for the interesting discussions.
## Contents

1 Introduction 1
   1.1 Goal-Oriented Conceptual Database Design 2
   1.2 Data Quality as a Design Criterion 3
   1.3 Data Quality Rule Design 4
   1.4 Main Contributions 5

2 Related Work 7
   2.1 Goal-Oriented Software Requirements Engineering 7
      2.1.1 Nature of Software Requirements 7
      2.1.2 Main Concepts in GORE 9
      2.1.3 Benefits of Goal Modeling 10
      2.1.4 Main Approaches to GORE 11
      2.1.5 Object Modeling in GORE Approaches 19
   2.2 Database Conceptual Schema Design 21
      2.2.1 Requirements Analysis in Database Development 21
      2.2.2 Fundamentals of Semantic Data Modeling 21
      2.2.3 Main Schema Design Approaches 22
      2.2.4 Dimensions of Conceptual Schema Design 28
      2.2.5 Goal-Orientation in Conceptual Schema Design 31
   2.3 Data Quality as a Design Criterion 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Process Overview</td>
<td>39</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Goal-Oriented Requirements Analysis</td>
<td>39</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Goal-Oriented Schema Design</td>
<td>39</td>
</tr>
<tr>
<td>3.2</td>
<td>Process Steps</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>3SDB Case Study</td>
<td>45</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Application Background</td>
<td>45</td>
</tr>
<tr>
<td>3.3.2</td>
<td>User Requirements</td>
<td>47</td>
</tr>
<tr>
<td>3.3.3</td>
<td>The Running Example</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>A Compositional View of DQ Attributes</td>
<td>63</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Motivating Examples</td>
<td>64</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Nature of Data Quality</td>
<td>65</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Data Quality Predicates</td>
<td>69</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Expressing Data Quality Attributes</td>
<td>72</td>
</tr>
<tr>
<td>4.2</td>
<td>Data Quality By Design Process</td>
<td>77</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Quality Assurance Data Requirements</td>
<td>78</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Process Overview</td>
<td>79</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Process Steps</td>
<td>80</td>
</tr>
<tr>
<td>4.3</td>
<td>ExpDB Case Study</td>
<td>82</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Application Background</td>
<td>82</td>
</tr>
</tbody>
</table>
5 Data Quality Rule Design

5.1 Towards a Repository of DQ Rules

5.1.1 Introduction

5.1.2 Classification of DQ Rules

5.1.3 DQ Rules Design Guidelines

5.1.4 A Note on Formalizing DQ Rule Transformations

5.2 Measuring and Comparing Effectiveness of DQ Rules

5.2.1 Main Concepts

5.2.2 Manual Derivation of Effectiveness Formulas

5.2.3 Evaluating and Comparing DQ Rules

5.3 A Semi-automatic Approach for Deriving Effectiveness Formulas

5.3.1 A Vocabulary for Expressing Events

5.3.2 The Approach and Implementation

5.3.3 Simulations and Evaluation

5.3.4 A Comparison with Belief Networks

6 Conclusion and Future Work

6.1 Conclusion

6.2 Future Work

6.2.1 Cost Analysis

6.2.2 Uncertainty in DQ Assessment

6.2.3 Beyond Quality

Bibliography
Chapter 1

Introduction

A successful information system is the one that meets its design goals. Expressing the goals and subsequently translating them to a working solution is a major challenge for information systems engineering. Over the past 15 years, the state-of-the-art in software requirements engineering (RE) has evolved dramatically, thanks to new streams of research on goal modeling, specification and reasoning for software requirements elicitation, specification and verification. Goal-oriented RE (GORE) [171, 34, 124, 159, 75] has added an early phase where one starts from stakeholders and their intentions, and derives from them functional and non-functional requirements through a systematic, tool-supported process. At the same time, the state-of-the-art for designing the database part of an information system has not made much progress, despite the new challenges faced by data management professionals and new requirements set forward by government regulations concerning quality, privacy, security, trust and etc.

This thesis adopts the concepts and techniques developed in GORE research for database design with a focus on data quality (DQ) issues. More specifically, this work explores the feasibility and the extent to which data quality goals can be modeled and addressed at the schema level during conceptual database design; it is based on the premise that the quality of data in a database is greatly influenced by its schema, and therefore DQ problems need to be recognized at the requirements analysis and design stage [164, 149]. This thesis consists of
three main components which are summarized below.

### 1.1 Goal-Oriented Conceptual Database Design

The first part of the thesis starts with an empirical study [98] of the evolving design of a biological database. This database is a part of a commercial system for the management and analysis of gene expression data [114]; it stores information on biological samples and their donors explored during gene expression analysis. During this study, the design process is revisited by introducing a goal analysis step to gather and organize data requirements before conceptual modeling.

This study confirms the hypothesis that, similar to software development, goal analysis in database design supports systematic exploration and evaluation of design alternatives. The goal analysis results in a comprehensive *design space*, which includes not only the different versions of the conceptual schema generated during the evolution of the system, but also additional design alternatives missed by the original designers. This study also demonstrated that goal analysis extends the scope of conceptual schema design from the specification of the meaning of, structure of and the constraints on the data, to the capture of a rich set of *intentional metadata* that provides traceability from high level goals to design decisions to technical requirements, and vice versa.

Based on the observations from this study, a *goal-oriented conceptual database design* (*GODB*) process is proposed [99]. This process begins with a list of stakeholders and their top level goals. Using the goal analysis techniques, these strategic goals are gradually decomposed into concrete subgoals and plans (to fulfill these goals). Goals can be hard or soft: unlike hard goals which are either satisfied or not, softgoals have neither clear cut definitions nor criteria to decide whether they are satisfied completely; they are usually used to model non-functional requirements.

Goals and plans are interrelated to produce a goal model, which captures not a single, but
several alternative ways in which the top level hard goals may be fulfilled. These alternatives are evaluated using the softgoals as criteria, and one is selected as the input for the next phase. From the chosen design alternative, data requirements are elicited and organized into an initial conceptual schema for the database-to-be. Softgoals are then prioritized and considered in more detail to modify existing and/or introduce new data requirements. The final schema is produced based on the initial schema, and the new and/or modified data requirements.

This process is discussed in detail in Chapter 3 Section 3.1 and 3.2, and is illustrated using a running examples, which is taken from the biological database case study (Section 3.3).

1.2 Data Quality as a Design Criterion

The above process is general in the sense that it does not detail how particular types of softgoals are operationalized into technical designs. In the second part of the thesis, the GODB process is extended specifically for dealing with DQ softgoals; this leads to a DQ by design (DQD) process [95]. The basic idea is that data of low quality may be detected and corrected by performing various quality assurance activities. Often these activities require changes in core business activities, and may produce data quality assurance requirements that cannot be recognized by analyzing the core business activities alone.

The DQD process aims at identifying and modeling data quality assurance requirements; it takes as input the initial conceptual schema, which specifies core business data requirements, derived as in the GODB process, and a set of DQ softgoals. For each DQ softgoal, a list of risk factors and mitigation plans are identified. The main component of a mitigation plan is a DQD proposal, consisting of a revised version of the original schema, and a set of quality assurance activities it supports.

To better understand the nature of DQ, a compositional view to define DQ attributes [96] is first offered in Chapter 4, Section 4.1. The DQD process is discussed in detail in Section 4.2 and is illustrated using a running example adopted from Expense Database Case Study in [47]
1.3 Data Quality Rule Design

The DQD process aims at identifying and modeling quality assurance data requirements. In other words, it focuses on the structural aspect of conceptual database design, with DQ as one of the design criteria. The output of the DQD process is a set of DQD proposals, each of which consists of a schema, and a set of quality assurance activities it supports. A QA activity may involve manual work, and/or rely on some automatic techniques, which often depend on the specification and enforcement of DQ rules. The last part of thesis therefore concerns the use of constraints in conceptual database design.

In general, DQ rules are derived from domain and application specific business rules, and their specification requires the involvement of domain experts. Nevertheless, part of the knowledge in rule design is domain and application independent, and could therefore be isolated. Chapter 5 Section 5.1 proposes a classification scheme for DQ rules, based on a number of important domain and application independent properties. This classification contributes to building a DQ rule repository in two ways: (i) to facilitate reuse of domain and application independent knowledge in rule design (this is similar to software design patterns [69] for software design), and (ii) to assist rule designers in acquiring new rules given existing ones.

Section 5.2 then examines in depth one of the properties of DQ rules: effectiveness in context of error detection. A quantitative framework for measuring and comparing DQ rules according to their effectiveness is presented. At the core of this framework lies the concept of effectiveness formula; such a formula represents the effectiveness of a DQ rule under different probabilistic assumptions about the occurrence of errors in data values being assessed and other events that may affect the effectiveness of the rule. Manual derivation of these formulas are illustrated using two examples. Given their effectiveness formulas, DQ rules can be evaluated individually and compared under different scenarios.
Manual derivation of effectiveness formulas is an non-trivial and error prone process. Section 5.3 continues by proposing a semi-automatic approach to derive effectiveness formulas of DQ rules. It starts with the definition of a vocabulary for describing events concerning error sources (Section 5.3.1); given a vocabulary, an effectiveness formula for a DQ rule is derived in the following three phases (Section 5.3.2): (i) constructing a directed acyclic graph of events, expressed using the terms in the vocabulary (Section 5.3.2), (ii) filling in a conditional probability table for each event in the graph (Section 5.3.2), and (iii) formulating the effectiveness formula given the results from the first two phases (Section 5.3.2).

### 1.4 Main Contributions

Building an information system that satisfies user requirements is a major challenge facing systems engineers, especially when these requirements are “soft” in nature. This thesis addresses this challenge from a goal-oriented perspective, with the focus on a particular kind of softgoal, quality of data. Main contributions of this thesis include:

- a goal-oriented conceptual database design process, evolved from a case study of the design of a real-world biological database; this process consists of a number of steps, spanning the spectrum from high-level stakeholder goal analysis to detailed conceptual schema design; it lays the foundation for the rest of the thesis;

- an extension of above process with data quality as one of its main design criteria; it therefore focuses on the structure aspect of schema design: to identify quality assurance data requirements in order to revise the original schema with proper attention to DQ concerns;

- and finally, a set of primitive transformations for guiding DQ rule design, and a workbench approach for systematically measuring and comparing these rules; it provides advanced support for the constraint aspect of schema design: transformation based design guidance helps designers to derive new rules from existing ones and build a rule repository over time; com-
parison among data quality rules is of special importance when alternative rules are available but are associated with different costs of enforcement.
Chapter 2

Related Work

2.1 Goal-Oriented Software Requirements Engineering

2.1.1 Nature of Software Requirements

Research on software requirements can be traced back to mid-70s when a wealth of empirical data confirmed that “the rumored requirements problems are a reality” [21]. The data suggested that software requirements errors were the most numerous and had a significant impact on the quality of resulting software, even more significantly, that they were also the most costly and time-consuming to correct. This recognition of critical nature of requirements established Requirements Engineering (RE) as an important subfield of Software Engineering.

In general, RE deals with *elicitation, analysis, documentation, negotiation and communication* of software systems requirements. The oldest definition of RE already identified the main components of software requirements. In their seminal paper [142], Ross and Schoman pointed out that software requirements include “the real-world goals for, functions of and constraints on software systems” [176]. A number of important properties of software requirements are clearly described in the classic paper by Michael Jackson [89], in terms of several key distinctions.

*System-to-be vs its Environment.* Requirements analysis concerns both the system-to-be
and its environment: requirements describe the purpose of the system; the purpose comes not from the system itself, but from its environment (i.e., the part of the real world with which the system will interact, in which the effects of the system will be observed and evaluated).

**Requirements vs Specification.** Requirements can be stated in a language accessible to stakeholders without referring to the system. But we still need to specify the desired behavior of the system-to-be. According to Jackson, when a system is put into its environment, it can influence and be influence by the environment only because they have some “shared phenomena”. That is, there are events and states that are shared by the system and environment. A software specification is a description of conditions over these shared phenomena.

**Optative vs Indicative Descriptions.** Requirements cannot be put into the specification directly, since in general they are about the private phenomena of the environment. To ensure the satisfaction of these requirements, we need to further distinguish the desired (“optative”) conditions over the environment phenomena (i.e., the requirements) and the given (“indicative”) properties of the environment (i.e., domain properties). Requirements are reduced to specification using domain properties.

Similar distinctions have also been made by other researchers. For example, one of the key principles underlying the RML framework [77] is that a requirements document need to explicitly capture not only the “prescriptive” specification of the desired functionality of a system-to-be but also “descriptive” knowledge about its application domain. Therefore RML distinguish between specification languages

Another important distinction often made for software requirements is between functional vs non-functional requirements. Functional requirements specify the functions that a software system must perform, while non-functional requirements (NFRs) represent global qualities of the system, such as usability, safety and reliability. NFRs are in general more difficult to express in an objective way. Moreover, they are often controversial (e.g., security vs user-friendliness), and are difficult to enforce and validate. Not surprisingly, unmet quality requirements constitute an important failure factor for software development projects.


\section*{2.1.2 Main Concepts in GORE}

Within the RE community, goal-oriented approaches have become increasingly popular in recent years. Traditional system analysis approaches \cite{141, 56, 143} focus on specifying requirements for the software system along, in terms of the \textit{what} (i.e., data) and the \textit{how} (i.e., processes) aspects, and do not capture the rationale for the software systems. Modeling the composite system (i.e., the system-to-be and its operational environment) \cite{57} and emphasizing on the \textit{why} aspect \cite{169} have long been recognized as the contributing factors to better requirements specifications.

In Goal-Oriented Requirements Engineering (GORE), \textit{goals} model the objectives of organizations, interests and motivations of individuals, and purpose and requirements of software systems. They can be \textit{hard} or \textit{soft}: unlike hard goals, a softgoal has no clear cut definition nor criteria to decide whether it is satisfied, and is usually used to model non-functional requirements. Therefore, a softgoal is usually said to be “satisficed” (sufficiently satisfactory) instead of “satisfied”.

Three basic qualitative goal reasoning techniques are available.

- \textit{Decomposition} is used to refine a goal into subgoals, modeling a finer goal structure; an \textit{AND-decomposition} means the goal is achieved if all its subgoals are, while an \textit{OR-decomposition} leads to alternatives fulfillment of the goal (i.e., achieving any of its subgoals is sufficient to fulfill the goal). The goal refinement process terminates when leaf goals are tangible (i.e., can be satisfied though an appropriate course of action) and/or observable (i.e., can be confirmed satisfied/denied by simply observing the application domain) \cite{71}.

- \textit{Means-end analysis} helps to identify alternative \textit{plans} (i.e., activities) to fulfill the leaf goals. A plan is defined abstractly as partially ordered sequence of actions that can be carried out to achieve a goal. For example, to have a “meeting scheduled” one needs to “collect time constraints and preferences from participants” and to “find a time slot and room for the meeting that satisfies these constraints and preferences as much as possible”. AND/OR-
decompositions can be also used to refine plans.

- **Contribution analysis** is used in the “upward” fashion to represent positive / negative influence among goal fulfillment (unlike decomposition and means-end analysis which are “downward” analysis techniques to build a goal hierarchy). Contribution links are characterized by their direction (positive or negative) and the degree of contribution (full or partial). For example, an full positive contribution link (labeled with “++”) from Goal $A$ to $B$ means the latter is achieved whenever the former is, while an partial negative contribution link (labeled with “−”) from Goal $C$ to Softgoal $D$ means, if the former is achieved the latter is partially denied.

The result of a goal analysis is a goal model [75]. A goal model is a forest of goal/plan AND-OR trees with contribution edges between nodes of different trees and means-end edges between goal and plan nodes. Thanks to the presence of OR-decomposition and means-end edges, there are subsets of leaves in the tree that define different ways, i.e., *design alternatives*, to fulfill the aggregate top-level goals. Presence of softgoals and contribution links from hard goals and plans to help to evaluate design alternatives.

### 2.1.3 Benefits of Goal Modeling

The most notable benefits of goal-oriented approaches are summarized below.

1. GORE approaches have a broader modeling scope than traditional RE techniques. Goals are prescriptive assertions that should hold in the system made of the software-to-be and its environment; domain properties and expectations about the environment are explicitly captured during the requirements elaboration process, in addition to the usual software requirements specifications [159]. Moreover, GORE emphasizes both functional and non-functional requirements during requirements analysis [124, 47].

2. GORE approaches add an early phase to software requirements analysis step that (i) focuses on the stakeholders and their intentions, and (ii) supports systematic exploration and eval-
uation of alternative goal refinements, responsibility assignments, obstacle resolutions and conflict resolutions [159, 32].

3. Goals are generally considered more stable than operational requirements [7, 155] and the goal refinement structure provides a rich way of structuring and documenting the entire requirements document [6, 155, 159].

4. Goals act as a correctness criterion for requirements completeness and pertinence [100, 82, 155], providing the rationale for the requirements that operationalize them. Moreover, goal models provide an excellent way to communicate requirements to various stakeholders and help them to understand the requirements in terms of higher level concerns in the application domain [159].

2.1.4 Main Approaches to GORE

Enterprise Knowledge Development

The Enterprise Knowledge Development (EKD) methodology [35, 101, 82, 34] offers a goal-driven approach to requirements acquisition and analysis for information systems development. Its main premise is that a requirements specification of an information system can be constructed as multiple interrelated “sub-models” of (a part of) the enterprise for which the system-to-be will provide support.

An enterprise model in EKD is constructed in terms of a number of interconnected submodels, including the Goal Model, Business Rule Model, Concepts Model and Business Process Model. The Goal Model describe not only the goals of the enterprise, but also the issues (e.g., problems and their causes, constraints, opportunities) associated with achieving these goals. The Goal Model provides the motivation for activities and entities of other submodels. It is also important to point out that the main purpose of the Concept Model is to define a “vocabulary” for “things” and “phenomena” that can be referred to and reasoned about by elements in other submodels precisely and consistently. As a result, inclusion of the informa-
tion in the Concepts Model does not imply realization in a computerized information system. This implies some design decisions need to be made when moving from the Concepts Model to a design model. This feature distinguishes the concepts modeling in EKD with general data modeling using the EER model.

Modeling activities in EKD are only guided by specific “driving questions”. For example, in goal modeling, the driving questions include “what are the strategies of this part of the enterprise?”, “Are there any particular problems hindering this?”, “what actions could be taken to improve the situation?”; in concepts modeling, the driving questions include “which are the main entities in this application?”, “how does an instance of the entity come into existence?”, and “is an entity type generically related to some other type?”

The EKD approach offers a comprehensive language for modeling the enterprise and its related information system requirements. Many of its ideas influence later GORE research. However, its “all-inclusive” nature makes the modeling language complex (6 sub-models with more than 30 different types of modeling components interrelated in a complex way). Moreover, EKD doesn’t offer a rigorous modeling process based on its modeling language.

**Knowledge Acquisition in autOmated Specification**

The Knowledge Acquisition in autOmated Specification (KAOS) (or Keep All Objectives Satisfied) methodology [54, 106, 55, 109, 160, 157, 158, 155, 159] is a goal-oriented software requirements engineering approach, with the extension for software architecture design[156]. In general, each construct in the KAOS language has a two-level structure: an outer graphical layer for modeling concepts, such as goals, objects or agents, and an inner assertion layer for specifying such concepts formally. The assertion layer is optional and used for formal reasoning [109].

The main concepts in the KAOS metamodel includes **objects, operation, agent, goal, constraint, obstacle** and **scenario**.

1. **Objects** are things of interest in the composite system whose instances may evolve from
state to state.

2. An operation is an input-output relation over objects. Operations are characterized by pre-, post- and trigger conditions, which capture the state transition produced by application of the operation.

3. An agent is kind of object which is partly defined by two sets of operations: the set of operations it can perform and the set of operations it must perform after assignment decisions have been made. Agents are active components and have choice on their behavior.

4. A goal in KAOS is a nonoperational objective to be achieved by the composite system. Nonoperational means a goal cannot be established through appropriate state transitions under control of one of the agents. Goals are linked to objects they refer to; these links are used during acquisition to get object descriptions from goal descriptions. Goals are classified according to their pattern (i.e., achieve, cease, maintain, avoid and optimize) and their category (e.g., SatisfactionGoals, InformationGoals, SafetyGoals and PrivacyGoals).

5. A constraint is an operational objective that can be formulated in terms of objects and operations available to some agent in the system and established through appropriate state transitions under control of the agent. Constraints correspond to requirements and assumptions for the composite system [157]. Constraints operationalize goals and can be analyzed in the same way that goals are.

In KAOS, operational software requirements are derived gradually from the underlying goals of the composite system. The elicitation, elaboration and specification of software requirements consist of following main steps [155].

- In the goal elaboration step, a goal model is constructed by identifying relevant goals from input material, such as interview transcripts and available documents. The stop condition for the goal refinement is the leaf-level goals in the refinement tree become operational (i.e., assignable to single agent as constraints).

- In the object modeling step, an object model (can be represented as a UML class diagram) is constructed incrementally from the goal model by identifying “concerned objects” from
goals. An object is concerned by a goal if it is used in the formal specification of the goal.

- In the responsibility assignment step, an agent responsibility model is elaborated by identifying agents together with their potential monitoring/control capabilities, and exploring alternative assignments of goals to agents.

- In the operationalization step, an operation model is constructed by identifying operations and their domain pre- and post-conditions from the goal specifications.

KAOS focuses mainly on the operation aspect of the system-to-be. Goal-oriented object modeling explored in KAOS is not sufficient for database design. Consider an example of a goal specification in KAOS as shown in Table 2.1. Entities such as Initiator, Meeting, Participant, relationships such as requesting, invited and knows, and attributes such as Meeting.feasible and Meeting.scheduled can be derived from the goal and added to the corresponding object model.

Table 2.1: An example of KAOS goal specification

<table>
<thead>
<tr>
<th>SystemGoal</th>
<th>Achieve [MeetingRequestSatisfied]</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormalDef</td>
<td>(∀ r: Initiator, m: Meeting, p: Participant) Requesting (r, m) ∧ Feasible (m) ⇒ ♦Scheduled (m) ∧ Invited (p, m) ⇒ ♦Knows (p, m)</td>
</tr>
</tbody>
</table>

However, not all elements in an object model need persistent storage. For example, for a meeting scheduling system, it may be necessary to record the facts that an initiator has requested a meeting and that a participant has been invited for the meeting (i.e., the relationships requesting and invited); but it is unclear from the goal specification whether the fact that the participant knows the meeting schedule (i.e., the relationship knows) also need to be stored in the database.

Non-Functional Requirements

The NFR (Non-Functional Requirements) framework [122, 124, 47] focuses on the modeling and analysis of nonfunctional requirements in software development. The main idea is to systematically model and refine non-functional requirements and to expose positive and negative
influences of different alternatives on these requirements. This process is visualized in terms of incremental and interactive construction, elaboration, analysis and revision of a *softgoal interdependency graph (SIG)*, which consists of softgoals and interdependencies.

*NFR softgoals* represent non-functional requirements to be satisficed. Each NFR softgoal has a *NFR type*, which indicates the particular non-functional requirement (e.g., accuracy, security, performance) and one or more *topics*, which represent the subject matter of the softgoal (e.g., financial accounts). Examples of NFR softgoals are “Accuracy[AccountInfo]” and “Performance[CheckBalance(AccountInfo)]”. NFR types can be inter-related and arranged into IsA hierarchies called *NFR type catalogs*.

Softgoals are inter-related through *interdependencies*, which are classified into “downward” *refinements* and “upward” *contributions*. Refinements are used in the development process to repeatedly refine parent softgoals into their offsprings. Contributions relates the satisficing of the offsprings to that of the parent. For example, a HELPS contribution from \( S_1 \) to \( S_2 \) means \( S_2 \) is partially satisficeable if \( S_1 \) is satisficed and partially deniable if \( S_1 \) is denied.

In the NFR Framework, reusable techniques for constructing SIGs are encoded as catalogs of *refinement methods* and *correlation rules*. Refinement methods are generic procedures for refining a softgoal into one or more offspring. For example, NFR decomposition methods can break a NFR softgoal into other NFR softgoals, based on its type (called *subtype methods*) or topic (called *subtopic methods*); consider a NFR softgoal “Accuracy[AccountInfo]”, it can be decomposed into its subgoals “Accuracy[RegularAccountInfo]” and “Accuracy[GoldAccountInfo]”; moreover, the second subgoal can be further decomposed into “Accuracy[GoldAccount.name]” and “Accuracy[GoldAccount.balance]”. Correlation rules capture knowledge about generic interactions between softgoals and are used to elaborate any part of an existing graph.

In summary, the NFR Framework provides a *process-oriented*, qualitative approach for dealing with non-functional requirements. This is accomplished by constructing and elaborating a softgoal interdependency graph, with the assist of design knowledge encoded in the catalogs of NFR types, refinement methods and correlation rules. Once the graph is constructed,
the developer chooses alternative combinations of leaf-level nodes softgoals as part of the target system and uses the provided label propagation algorithm to see if the selected alternative is good enough to satisfice the high-level non-functional requirements for the system. The NFR Framework is designed for general software system development and provides an ideal starting point for incorporating non-functional requirements into database design.

### i* Framework

The i* (distributed intentionality) framework was originally developed for modeling and reasoning about organizational environments and their information systems [170], and has been applied to different application areas, including early- and late-phase requirements engineering [171, 172, 59], system and enterprise architectural design [80, 167], business process reengineering [173, 175] and software process modeling [174].

The i* framework offers two inter-related models: the Strategic Dependency (SD) model and the Strategic Rationale (SR) model. The SD model is used to model actors and interdependencies among them. A SD model is a graph where each node in a SD model represents an actor, and each link between two actors represents a dependency between them. The dependency between actors is intentional in the sense that one actor (i.e., depender) depends on the other (dependee) for something (dependum) in order to attain some goal. A dependum can be a (soft)goal to be achieved, a task to be performed or a resource to be furnished; this is used to indicate the type of freedom and constraint the dependee has in delivering the dependum. The SR model is used to model the goals, tasks and resources of actors. A SR model is a graph with four main types of nodes (goal, task, resource and softgoal) and two main types of links (task-decomposition and means-end).

### TROPOS

Tropos [41, 32, 39] is a comprehensive requirements-driven, agent-oriented software engineering methodology. It adopts the primitive concepts offered by the i* framework, and also defines
a formal specification language (called Formal Tropos [67, 66]) that complement i*. In addition, Tropos also provides a formal goal analysis and reasoning in the presence of softgoals and partial contribution relations [74, 146, 75]. Moreover, Tropos defines socially based catalogs of styles and patterns that encode reusable design experience and knowledge for moving from early requirements to Multi-Agent Systems architectures. Finally, Tropos concepts and process is extended and applied to security and trust requirements engineering [73, 72], risk analysis [10] and data warehouse design [112].

The main Tropos modeling constructs are directly adopted from the i* framework, which include actor (agent, role, position), goal (soft, hard), plan (which corresponds to i* task), resource, dependency (goal, task, resource, softgoal). In addition, capability represents the ability of an actor of defining, choosing and executing a plan for the fulfillment of a goal, given certain world conditions and in presence of a specific event (used in architectural and detailed design, and implementation), and belief represents actor knowledge of the world. The Tropos metamodel is describe in detail in [32, 152].

Tropos software development starts with early requirements analysis where the domain stakeholders and their intentions are identified and modeled as social actors and goals respectively. Goal analysis are then carried out from the perspective of each actor: for each of its goal, the actor has the choices of (a) refine it into subgoals using AND/OR decomposition (or into plans using means-end analysis), (b) accept it (i.e., taking the responsibility of fulfilling it herself), or, (c) delegates it to an existing or new actor. This process ends when all goals have been dealt with to the satisfaction of their actor, and leads to strategic dependencies among the actors. The result of this process is the initial actor diagram (i.e., i* SD model) and goal (or rationale) diagram (i.e., i* SR model) for the organizational environment within which the system-to-be will operate.

In late requirement analysis phase, the actor diagram is extended by introducing the system-to-be as the new actor, which has a number of dependencies with other actors of the organization. Existing dependencies can be revised and new ones can be introduced. Resource, task and
softgoal dependencies correspond naturally to functional and non-functional requirements of
the system. Alternative system requirements can be explored and evaluated by making different
sets of dependencies on the system-to-be.

The Goal-Risk framework [9, 10] extends Tropos metamodel with concepts for analyzing
risks, based on the idea of the three-layer analysis introduced in the Defect Detection and
Prevention (DDP) framework [64]. A risk is understood as an uncertain events with negative
impacts. In the Goal-Risk framework, an event is an uncertain circumstance/state of affair
that may affect (positively or negatively) the achievement of a goal. A treatment is a plan
(in the Tropos terminology) that reduce/treat/mitigate the effects of an event. The Goal-Risk
framework therefore include following three layers: goal layer, event/risk layer and treatment
layer.

This framework is intended to support the deliberation process of autonomous agents. To
the end, a risk analysis process is proposed. Given a Goal-Risk model, an autonomous agent
selects a strategy (i.e., a subset of all possible plans and treatments satisfies the top goals),
such that the risk is below a specific risk level and the total cost is affordable. The risk level is
calculated from the likelihood of events and the success-rate of plans and treatments; the total
cost is the sum of the cost of the plans and treatments in the strategy. Both backward (i.e., to
identify all possible strategies) and forward reasoning (i.e., to compute risk) are used in the
process.

The i*/Tropos framework offers a goal-oriented approach to software development. Goal-
oriented modeling and analysis in i*/Tropos focus on the dynamic part of software systems
(e.g., operationalization of goals into plans using means-end analysis, evaluation of alternative
plans to fulfill goals). Goal-oriented support for modeling the static aspect of the application
domain is limited. The notion of resource is used to represent “a physical or an informational
entity” [169]. A resource can be used in a particular type of intentional dependency between
actors; it can also be used represent data flows (in Tropos) between software agents, and to
decompose tasks (in i*). Resources are transparent in the sense that their internal structure and
external relationships are of no interest to i*/Tropos modelers.

2.1.5 Object Modeling in GORE Approaches

The question about how goal modeling is interacted with more traditional, non-goal-oriented approaches has been long raised [168]. Goal-Scenario coupling proposed in CREWS [140, 139] is one such example. For object-oriented approaches, it would be interesting to see how different goal concepts interact with classification, generalization, aggregation, encapsulation, and other abstraction mechanisms. For example, [3] proposes six guidelines for transforming i* early requirements models to pUML/OCL-based requirements. One such guideline states that the goal dependencies in i* are mapped to Boolean attributes in pUML. Examples of goal-object interactions are also abundant in current GORE approaches. Following are examples of interactions.

In KAOS (see Section 2.1.4), the goal-oriented object modeling starts with the identification of goals and their “concerned objects” and “state variables”. These objects and variables become the conceptual classes, attributes and associations of the object model under construction in parallel with the goal model. The rule of direct reference states that a concept need to be added to the object model if it is directly used in the formulation of a goal specification. For example, the goal “to satisfy meeting requests” reference the concepts “Meeting” and “Meeting Request” [106].

In EKD (see Section 2.1.4), an enterprise model is constructed in terms of several interconnected sub-models, including the goal model, the business rule model, the concepts model, the business process model and the actor and resource model. Goals are related to concepts in terms of static and dynamic constraints. For example, in a concepts model for a library, “Reference Book” is a sub-concept of “Loanable Book”. This modeling decision can be explained by the goal (the borrower) “able to check out any type of books” which contributes positive to the softgoal is “happy customer”. This is an example of an interaction between a goal and

---

1 pUML stands for precise UML.
a concept specialization. As another example, consider the goal (the librarian) “to have high availability of books”. This goal explains the constraint “Loan.duration ≤ 14 days”. In this case, the interaction is between a goal and a static constraint [82].

In NFR (see Section 2.1.4), the NFR decomposition methods are used to refine a NFR softgoal into its subgoals, based on the interaction between the goal model and the structural aspect of the domain. The subclass method encodes the interaction between goal decomposition and concept specialization. For example, a NFR softgoal “Accuracy[Account]” can be decomposed into its subgoals “Accuracy[RegularAccount]” and “Accuracy[GoldAccount]”. The attribute method encodes the interaction between goal decomposition and concept aggregation. For example, the NFR softgoal “Accuracy[GoldAccount]” can be further decomposed into “Accuracy[GoldAccount.name]”, “Accuracy[GoldAccount.balance]” and “Accuracy[GoldAccount.highSpending]” [47].

In TROPOS (see Section 2.1.4), intentional dependencies are used to generate portions of UML class diagrams during detailed design. For example, from the dependency from the agent “Shopping Cart” to the agent “Online Catalogue” on the resource “Shopping Item Detail”, one can derive four classes, “ShoppingCart”, “ItemLine”, “OnlineCatalogue”, “MediaItem”, two aggregations: “partOf(ItemLine, ShoppingCart)”, “partOf(MediaItem, OnlineCatalogue)”, and one association “hasItemDetail(ItemLine, MediaItem)” [40].
2.2 Database Conceptual Schema Design

2.2.1 Requirements Analysis in Database Development

Databases are essential components in information systems. Their design has similar lifecycle as software design. Conventional database design methodologies \([11, 50, 88, 25, 125, 84]\) starts with requirements collection and analysis activities, in which data and transactional requirements are first gather into natural language statements from various sources (e.g., user interviews, existing documentation, earlier applications), and then disambiguated, classified and organized into a requirement specification, possibly expressed in some semi-formal language.

Compared to software requirements, transactional requirements for databases are relative simple: they are often of the simple form of updates and queries. The core part of a database requirements specification is a set of data requirements, expressing the semantics and structure of and constraints on data to be stored.

Formal requirements specifications for database are normally expressed as schemas, whose design is often referred to as data modeling. Database design is traditionally divided into three phases, each of which address specific design issues. For example, at the conceptual level, the main concerns include correctness, completeness and pertinence \([11, 19]\); at the logical and physical levels, the focus is on consistent and efficient storage and performance considerations.

2.2.2 Fundamentals of Semantic Data Modeling

Information modeling is concerned with the construction of computer-based symbol structures (referred as \textit{information bases}) that model some part of the real world (referred as \textit{application}), using the symbol structure types (referred as \textit{terms}) defined by an information model; in additional to symbol structure types, an information model also consists of a collection of operations which can be applied to any valid symbol structure, and a collection of general integrity rules which define the set of consistent symbol structure states, or changes of states \([123]\).
Data modeling is a type of information modeling that focuses on the static, structural aspects of its application by describing what things exist, their attributes and interrelationships, for the purpose of database development. Conceptual (or semantic) data model offer more expressive facilities for modeling applications and structuring information bases than their physical and logical counterparts, using semantic terms and abstraction mechanisms [123]. The result of semantic data modeling is often referred as the conceptual schema for the database-to-be.

### 2.2.3 Main Schema Design Approaches

**EER-based Approaches**

The extended entity-relationship (EER) model refers collectively to various extensions to Chen’s original entity-relationship model [43], which incorporate sophisticated forms of generalization and aggregation. For example, [18] proposed an form of EER that supports generalization hierarchies with coverage properties, and composite attributes. EER has been widely used in conceptual database design [18, 137, 11, 153, 154, 50, 88].

A classic approach to conceptual schema using EER was originally developed in [18] and refined in [11]. The design process is viewed as an incremental transformation process, structured by well-defined design strategies. Each design strategy is defined by a set of transformation primitives and a partial order in which they can be applied. A transformation primitive is a template of a design action that transforms an input schema by adding new or modifying existing schema elements.

1. In the *top-down strategy*, a conceptual schema is produced by a series of successive refinements, starting from an initial schema that describes all the data requirements by means of a few highly abstract concepts, and then gradually refining these concepts into more concrete ones, capable of describing the original concepts in more details. Each refinement is realized by a *top-down transformation primitive*. For example, one such primitive splits one
relationship into two, connected through a new entity.

2. In the **bottom-up strategy**, concrete concepts are identified and added to the schema first. *Bottom-up transformation primitives* are then used to introduce into the schema new concepts, which are capable of describing aspects of the application that have not been taken into account before. For example, one such primitive connects two previously unrelated entities with a new relationship.

3. The **inside-out strategy** is a particular type of bottom-up strategy, where concrete concepts are identified progressively, starting from a few important ones, and spreading outward radically.

In practice it is rare that a pure top-down or bottom-up strategy is followed throughout the design process. A comprehensive conceptual design methodology is offered in [11] based on the **mixed strategy**. The overall design problem is first broken down into smaller pieces, and the principle concepts in each sub-problem are extracted to form a skeleton schema. Concepts in the skeleton schema are then examined separately, and undergone either gradual refinement using top-down primitives or extension using bottom-up primitives.

The design strategies described above structure the design process based on the direction of modeling. Other design methodologies (e.g., [50, 88]) suggest to structure the design process according to the types of modeling constructs (e.g., entities first, relationships next, attributes third, generalization and constraints last).

**OO-based Approaches**

Object-oriented (OO) approaches to data modeling normally adopt the Unified Modeling Language (UML) as the modeling notation (e.g., [125, 121, 4]). General speaking, UML Class Diagrams (without operations) are comparable to EER schemas in terms of their expressive power, although the former allow dynamic classification, and can have associated constraints specified in OCL. On the downside, since UML was originally developed for software design, it has no built-in support for specifying keys. This problem can be solved by using stereotype
on attributes [121].

A classic OO approach to database development is described in [25], where the conceptual schema design process is divided into analysis and design steps. The purpose of analysis is to understand the problem thoroughly and devise a model of the real world (i.e., *analysis model*) from various sources of information about the domain (e.g., problem statement, interviewing). The development of the analysis model follows the construct-based strategy adopted in [50, 88] (i.e., classes, associations, attributes, generalization).

Unlike conceptual analysis, which is focused on the real world modeling, conceptual design focuses on the computer resources. In the detailed design step, the analysis model is gradually transformed into a *design model* (i.e., the conceptual schema of the database), using a series of *primitive transformations*, similar to those used in [11]. The key difference is that in [11] the entire schema design process is transformational-based, where in [25] the transformations are applied only when the analysis model is complete.

The reason for this restriction is that at the initial phase of the schema design when the analysis model is incomplete, flexibility is more important than the formality (rigorous transformation may even be distracting); only when the requirements are properly understood, the model can be evolved in a more disciplined manner. Primitive transformations are specified in terms of source and target model patterns with pre- and post-conditions respectively, and can be classified as equivalence, information losing and information gaining transformations.

[121] provides a more recent account on using UML for data modeling. Transactional and data requirements are first modeled using Use Cases, which are then used to drive the construction of the Class Diagrams. One of its main contributions is a list of design patterns for conceptual schema design. For example, the *Lookup Table* pattern provides a reusable architecture solution to the common requirement to assign a value from a list of possible values, while the *Metamodel* pattern allows one to dynamically add attributes to an entity by representing the names of the attributes as the meta data stored in a lookup table. Other patterns deal with

---

2 Actually, it uses OMT, which is a predecessor of UML
various ways to declare primary keys, reduce redundancy, handle null values, etc.

ORM-based Approaches

Object-Role Modeling (ORM) is a method for modeling and querying an information system at the conceptual level [84]. ORM pictures the world in terms of objects (i.e., entities or values) that play roles (i.e., parts in relationship). Unlike the ER and OO approaches, ORM makes no explicit use of attributes. Instead, it uses elementary fact as its basic and primary modeling unit. A elementary fact is an atomic proposition asserting that a particular object has a property (e.g., Prof. Jones M. is tenured), or that one or more objects participate together in a relationship (e.g., Prof. Jones M. works for the Computer Science department).

A conceptual schema in ORM consists of three components: base elementary fact types, constraints and derivation rules. A 7-step conceptual schema design procedure is defined. In the first step, informal domain examples are transformed into elementary facts. For example, from a directory of academic staff, the elementary facts such as “the Academic with empNr 715 has Name Adams A”, and “the Academic with empNr 715 obtained Degree PhD from University UQ” can be defined. Next, fact types are then abstracted from elementary facts. For example, the second fact above is abstracted into the fact type “Academic obtained Degree from University”.

The first two steps allow the initial conceptual schema to be elaborated from familiar domain examples and later on validated by verbalization in natural language. In the third step, redundant elementary fact types are removed. Part of this step deals with separating derived fact types from base ones, and specifying them through derivation rules. The last four steps focus on refining the conceptual schema with various constraints. For example, a subset constraint specifies that the head of an department must also be an employee of the department.

ORM is considered more suitable for the conceptual analysis due to its features such as elementary fact-oriented, attribute-free modeling and role-based constraint specification, while other data modeling techniques such as the Entity-Relationship and Object-Oriented ap-
proaches are best used for a compact representation of the result of ORM analysis and design [83].

**Conceptual Schema Design using Ontologies**

Domain knowledge of various forms plays an essential role in data requirements analysis and conceptual schema design. A domain ontology encodes commonly shared understanding of domain for the purpose of facilitating reuse of domain knowledge and communication among people and software systems. It contains an extensive and well-defined set of concepts in a specific domain that one can refer to when needed.

Sugumaran and Storey proposed a semi-automatic approach [150, 151] for creating and evaluating conceptual schemas (expressed in EER model) using lightweight domain ontologies. The main constructs of the domain ontologies are terms and (binary) relationships. Two categories of relationships are distinguished: *basic relationships* include is-a, synonym and related-to which are typically appear in an ontology; *domain relationships* capture additional constraints (business rules) in the domain, which include prerequisite, temporal, mutually inclusive and mutually exclusive relationships. For example, a mutually exclusive relationship exists between the term “buyer” and “seller” since an individual cannot be the buyer and seller of the same item.

In this work, domain ontologies help schema design in following way: (a) initial terms extracted from user requirements are first matched with the terms in the domain ontology; and (b) the basic domain relationships are then followed, starting from matched ontological terms to other related ones; finally, (c) related terms and relationships are suggested to the user, who selects and uses them to refine and expand the set of user terms and/or to check its completeness and consistency.

For example, the synonym relationship is used to suggest “best” names for user terms, while the related-to relationship is used to suggest “missing” concepts in user requirements statement. Moreover, domain relationships such as mutually exclusive are used to check the consistency
of the set of user terms. The design process is supported by a set of design rules. For example, the rule used to find related ontological terms is: if (\(X\) is a user term) and (related-to(\(X, Y\)) is in the ontology) and (\(Y\) is not a user term) then suggest \(Y\) to the user.

Essentially, this approach explores the relationships in the domain ontology to guide the schema design in a mechanical way. There are two essential tasks that are not addressed in any detail: determining the relevance between user requirements and ontological terms, and specifying the stop conditions for the automatic propagation of relevance through relationships in the ontology. A domain ontology contains a large amount of concepts that commonly appear in the domain, where a database schema may only require a small portion of it.

Moreover, domain relationships in the ontology capture constraints in the domain, but do not necessarily constrain the schema design. For example, a prerequisite relationship exists between the terms “online payment” and “credit card” because every online payment requires a valid credit card. But this domain constraint does not imply that every online transaction database has to store both payment and credit card information (for security concerns for example).

A more sophisticated approach is presented in [48], where a conceptual schema is developed by specializing a general (i.e., a top-level, domain or task) ontology. Three activities are identified and characterized in this process. In the refinement activity, the general ontology is extended manually by the designer to include all elements of the conceptual schema. The result is an extended ontology, which normally contains many irrelevant concepts. In the prune activity, these irrelevant concepts are automatically removed, given that both the extended ontology and functional requirements of the information system are formally specified in UML and OCL. A pruning algorithm is presented in detail in [49]. In the refactoring activity, the pruned ontology is restructured to produce the final conceptual schema, which is externally equivalent to the pruned ontology, but with redundant concepts removed. Two sufficient conditions for determining redundancy are defined and used to automate part of this activity.


2.2.4 Dimensions of Conceptual Schema Design

One way to view a design problem is as a search of the “optimal” solution in a multi-dimensional space, where each dimension corresponds to one type of design issue to be addressed; this gives rise to a set of design alternatives for addressing that issue. The entire design space is the cross product of all the dimensions. A point in design space corresponds to a complete design. Various types of criteria are used to evaluate and select among design alternatives.

For example, [25] described several design dimensions, including the support for temporal data and units. All data has the temporal dimension whether or not it is explicitly represented. For example, for an investment portfolio management application, both the transaction and valuation of an asset have the associated time. Transactions normally happen at a single moment in time, while the value of assets changes continually. When designing the schema, decisions have to be made whether to represent the transaction time explicitly as an additional attribute, and whether to track multiple values for a given asset.

Units are another type of “secondary data” [25] associated with measurement attribute values. The designer has to choose whether to assume a canonical unit of measure (therefore does not need to represent it explicitly), to introduce a single attribute for unit (therefore allowing any type of unit to be used), or to support multiple units (and the conversion among them) simultaneously. It is suggested that the temporal and secondary dimensions of data should be modeled after the core application data is modeled [25].

Below we discuss other important dimensions for conceptual schema design.

Data Quality

See Section 2.3 for a detailed discussion on the related work on data quality.

Data Privacy

Data privacy is a growing concern among businesses and other organizations in a variety of sectors, such as healthcare, finance, e-commerce, and government. Hippocratic databases [135]
have been proposed as a candidate solution to allow database systems to take the responsibilities for maintaining the privacy of the personal information. Ten privacy principles governing the design of a hippocratic database system is derived from privacy regulations and guidelines. These design principles are: purpose specification, consent, limited collection, limited use, limited disclosure, limited retention, accuracy, safety, openness and compliance.

In accordance with these design principles, a set of privacy metadata is identified, which includes the concepts of purpose, external recipient, retention period and authorized user. The goal is to associate this metadata with each attribute of the schema. A privacy metadata schema is defined with two table for this purpose: for a given purpose, the first table associates individual attribute name with a list of external recipients and a retention period (thus capturing the privacy policy), while the second one associates the attribute name with a list of authorized users (thus capturing the privacy authorization that support the policy). In addition to the privacy metadata schema, each regular table in the database is augmented with a special attribute purpose, which allows tagging each row with the purposes it is intended.

Privacy policy reflects an organization’s general measures for privacy assurance, and has to be specified manually by a privacy officer. Privacy authorization instantiates each external recipient in privacy policy with the corresponding authorized user, thus representing the information that is actually disclosed. [116] proposed a goal-oriented framework to automate the derivation of the privacy authorization table from the corresponding privacy policy table. It extends the original hippocratic database proposal to allow for hierarchies of purposes (i.e., goal hierarchies), distributed authorizations (i.e., multiple stakeholders), and minimal disclosure supporting the business processes of virtual organizations.

The hippocratic databases approach and its extensions (e.g., [108]) aims at providing a comprehensive solution for designing, implementing and querying privacy-aware database systems on top of the relational model. But schema design issues are not fully explored in this approach. For example, [135] itself noted that the fixed privacy metadata schema design implicitly assumes that (a) purpose together with attribute completely determine the set of recipients
and retention period, and (b) the set of attributes collected for a particular purpose is fixed. In situations where these assumptions are not generally true, the privacy metadata schema need to be redesigned. For instance, if the privacy policy varies depending on the owner of the personal information, an additional user concept needs to be introduced in the privacy policy table.

**Data Provenance**

Provenance refers to the metadata that helps determine how a piece of information arrived at its present form. Provenance metadata describes the “why”, “who”, “where” and “how” aspects of, and help to determine the reliability, quality, ownership, etc. of the data being described. If not treated with extra effort, even the creators may not be able to trace the provenance of their information. No existing database design approach provides systematic support for modeling provenance metadata. Below we brief describes various forms of provenance metadata appeared in different context. In most the cases, the provenance metadata can be grouped around three central concepts: the subject for which provenance metadata is maintained, the origin of the subject, and the derivation process describing how the subject arrives at its present form.

In the context of relational databases, the work on data provenance, as first reported by Buneman et al [36], focuses on identifying data elements in the source database that “contribute to” the data elements in the target database, and proposes an approach to compute provenance when the data of interest has been created by a database query. The provenance subject is a single attribute value in the target database; its origin are the attributes in source database from which this value is derived, while the derivation process is a series of database queries that transform data from the source into the target.

In the context of e-science, Greenwood et al [78] view provenance as the metadata that associated with in-silico experiments. The subject here is a data object that results from in-silico experiments. The origin indicates who performed the experiment and for what purpose (e.g., to test a hypothesis), the scientific domain concepts that are associated with the data or experiments, etc. The derivation process is the scientific workflow, which includes database
queries, algorithms and web services.

In the context of Web applications and Semantic Web, the notion of knowledge provenance has been introduced in [51]. The subject is a piece of knowledge one can obtain from some knowledge source (e.g., an assertion from a knowledge base, an answer from a Semantic Web application). Source meta-information records the origin of the piece of knowledge, which is a description of the knowledge source from which this piece of knowledge is derived. This description includes the name, authorship, authoritativeness, degree of belief and completeness of the source. Moreover, source meta-information may also contain the semantic information (i.e., meaning and relationships) of the terms and phrases in natural or a formal language in the knowledge source. The derivation process of knowledge in this case is the reasoning process that is captured by knowledge process information.

### 2.2.5 Goal-Orientation in Conceptual Schema Design

Starting with the notion of goals in database design is not a brand new idea. See for example [88, 50], where the first step is to define a mission statement and a set of mission objectives. A mission statement states the high level purpose of the database and provides a focus for the database developer. A mission objective describes general tasks to be performed against the data collected in the database. But the mission statement is only for scoping the problem space and has no direct influence on the rest of the design steps.

A more sophisticated use of goal modeling in database schema design appeared in [27], where a 3-step process was proposed for data marts design: first the user requirements are elicited using a goal-oriented, top-down analysis; then, a bottom-up analysis brings to the surface the semantics of the existing operational databases; the final step integrates these two viewpoints, and thus generates a feasible solution.

The goal-oriented requirements analysis starts user interviews in which high-level and informal goals are gathered. These goals are then aggregated, refined, fully specified using the GQM paradigm [17]. Quality focuses and their variation factors (i.e., the variables that may
cause changes to quality focuses) in the goal specification are used to extract user requirements in terms of an ideal star schema. One drawback of this approach is that it only considers functional requirements during schema design.

The data quality by design approaches discussed in Section 2.3.2 are good demonstrations for modeling of non-functional requirements in database schema design. Various data quality dimensions (e.g., accuracy, trust, privacy) can be modeled as softgoals in a goal-oriented approach, and the schema design process can be viewed as the operationalization of these softgoals in the context of the database. One distinctive feature of non-functional requirements is that they are rarely satisfied completely. This means there is no ideal design, but instead various trade-offs (e.g., quality vs. cost, conflicts between different quality dimensions) need to be taken into considerations during the design.

In summary, although the notion of goal and the techniques of goal modeling have been used in database schema design, few approaches have taken real advantage of what a goal-oriented approach could offer. These are the benefits of GORE approaches we have discussed in Section 2.1.3; among others, these include exploration of alternative ways to fulfill the top-level (functional and non-functional) goals, and explicit traceability of rationale (from goals to schema elements).

### 2.3 Data Quality as a Design Criterion

#### 2.3.1 Definitions of Data Quality

The quality of any artifact is determined by the degree to which it fulfills its intended use (“fitness for purpose”). For a database, the purpose is answering questions about the application it models. General speaking, quality of data refers to the fitness of data values for question-answering purposes.

More specifically, DQ is widely accepted as a multi-dimensional and hierarchical concept [165, 111, 19]. Significant effort has been dedicated to finding classification schemes and defi-
nition of various “aspects” of DQ (often referred to as DQ dimensions or attributes). Examples of such schemes include (i) accessibility, interpretability, usefulness and believability DQ [165] (ii) intrinsic (accuracy, objectivity, believability, etc.), contextual (relevancy, timeliness, completeness, etc.), representational (format, etc.), and accessibility DQ [166], and (iii) mandatory vs. desirable, primary vs. secondary, and direct vs. indirect DQ [68].

Criticism of these approaches include ambiguity, subjectivity, and even circularity of definitions within a single classification [28], as well as inconsistency across multiple classifications [111]. As an example of a circular definition, “credibility” in [165] is considered as a sub-attribute of “believability”, but it is itself defined as having sufficient evidence to be believed; as an example of incompatible definitions, in [166] “completeness” and “believability” belong to two disjoint categories, while being related through a specialization link in [165].

Other approaches to DQ take the view that generic quality attributes (e.g., accuracy, completeness) may be understood in terms of more primitive quality constructs. In the Qurator project [117], such constructs (called quality characterizations or QC) are concrete, operational level quality attributes defined by scientists. For example, “accuracy” can be defined in terms of “confidence” QC, which can then be quantified using calculated numeric experimental errors, or as a function of the type of experimental equipment.

Relationships among individual quality dimensions/attributes have also been studied. For example, [13] investigated the accuracy-timeliness tradeoff, while [14] presented a framework for systematic exploration of tradeoff between completeness and consistency. In [68], a set of logical interdependencies among quality attributes is presented. However, these approaches relate quality attributes based either on commonly accepted intuitions or the authors’ subjective judgment.
2.3.2 Data Quality Requirements in Design

Data Quality Requirements Analysis and Modeling

It has long been recognized that DQ problems need to be addressed at the requirements analysis stage. For example, [164] introduces a set of concepts and premises, as well as a process for DQ requirements for DQ modeling and analysis. In [164], data requirements are divided into application data requirements (called *application attributes*), such as *personal address* and *stock price*, and quality data requirements (called *quality attributes*), such as accuracy, timeliness.

The DQ requirements modeling and analysis process starts with a conventional conceptual data modeling step, where the application requirements are analyzed to elicit application attributes that constitute the first version of the conceptual schema. The elicitation of quality attributes is then carried out through two separate steps: first, qualitative and subjective dimensions by which a user evaluates DQ are identified from the quality requirements from a library of generic quality attributes, and associated with certain application attributes in the schema; second, each such dimension is refined into one or more quantitative and measurable indicators.

Although a significant first step, there are some limitations to this approach. First, the final result of this process is the initial conceptual schema, tagged with various quality indicators. The process of incorporating these quality indicators to produce a new conceptual schema is missing. Second, as an important step, the transition from subjective quality dimensions into objective quality indicators is left open. Although, some of the refinements are quite straightforward (e.g., from “timeliness” to “age”), others are less obvious (e.g., from “accuracy” to “collection method”).


2.3.3 Quality of Schemas

The schema of a database plays a significant role in ensuring the quality of the data in the database. Researchers in conceptual modeling have worked on the understanding, defining and measuring quality aspects of schemas. Early effort focused on defining criteria, such as (schema) correctness, minimality, (schema) completeness, pertinence, readability [18, 19], to evaluate schema quality in subjective way. Aiming at a more objective way to evaluate schema quality, comprehensive sets of quality measures have also been proposed for ER schemas. For example, the integrity measure in [118] were defined in terms of the number of incorrect integrity constraints and the number of correct integrity constraints that are not enforced.

Moreover, a set of quality measures have been developed for logical database schemas [130, 16, 147]. For example, [130, 38] proposed several measures to analyze quality of a relational database schema, such as number of foreign keys and depth of the referential tree. These measures have been validated formally using measurement theory [177] and empirically using questionnaires.

2.3.4 Measurement of DQ and DQ Techniques

Each DQ dimension aims at capturing and representing a specific aspect of quality in the data, and can be associated with one or more measures according to different factors involved in the measurement process [19]. Software engineering researchers and practitioners have been developing and using numerous metrics for assessing and improving quality of software and its development processes [65]. In comparison, measures for DQ have received less attention. Nevertheless, measures for accuracy, completeness, timeliness dimensions have been proposed in [12, 131, 14]. For example, currency, a sub-dimension of timeliness, can be measured using: 

\[ \text{Currency} = \text{Age} + (\text{DeliveryTime} - \text{InputTime}) \] [12], where \text{InputTime} is the time the value is obtained, \text{DeliveryTime} the time the value is delivered to the user, and \text{Age} measures how old the value is when received.
Performance measures have also been proposed and used for certain types of DQ techniques. Record linkage (object identification) algorithms aim at identifying data in the same or different sources that represent the same real-world entity, and can be used to improve quality and integrity of data [19, 46]. Performance measures for record linkage algorithms are often defined in terms of the number of true positives, false positives, etc [81, 46]. For example, in addition to precision, recall and F-measures, the performance of a record linkage algorithm can also be measured using \[ \text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \] [46]. In these proposals, performance measures are obtained for a particular application of a record linkage algorithm on actual data sets, and are mainly used as a mechanism to tune the parameters (e.g., matching threshold) that affect its performance.

### 2.3.5 Quality Issues in Data Warehouse Design

Data quality issues have also been investigated within the context of data warehouse design. The Foundations of Data Warehouse Quality (DWQ) approach [92, 94, 90, 91] explores the quality dimension of various data warehouse components, including schemas, agents and data. It relies heavily on the modeling of quality metadata, which is structured in terms of two interrelated metamodels. The framework metamodel is used to represent architecture components of a data warehouse. The quality metamodel provides the notation for representing quality goals, queries and measurements.

A **quality goal** is a natural-language statement about a quality requirement. Quality goals cannot usually be assessed directly, but their meaning is circumscribed by quality queries that need to be answered when evaluating the quality [90]. A **quality query** is a formal and executable specification on how to check whether the quality goal for which it provides evidence is achieved, or how the measured quality has changed in a certain period of time. Quality queries cannot usually be answered directly but rely on metrics applied to data warehouse components in question [90].

One important assumption of DWQ is that any information necessary to provide answers
to the quality queries is stored in the data warehouse in terms of *quality measurements*, which is the documented activity to measure the quality of various data warehouse components. For example, the quality goal “increase completeness” of a table can be operationalized into a query which returns the number of null values of that table.

The DWQ’s approach allows to express a wide range of stakeholders’ subjective quality requirements. Quality goals are “operationalized” by multiple executable quality queries, which access documented quality measurements attached to these data warehouse components. The answers to quality queries provide *evidence* for the fulfillment (or otherwise) of quality goals. Therefore the DW’s data quality management solution is limited to the representation, measurement and evaluation of quality of data. Unfulfilled quality goals may cause improvement actions, but only in an *ad hoc* fashion.
Chapter 3

Goal-Oriented Conceptual Database Design

In this chapter, a goal-oriented process for database requirements analysis and modeling (GODB) is presented. This process consists of a number of steps, spanning the spectrum from high-level stakeholder goal analysis to detailed conceptual schema design. It shows how goal modeling contributes to systematic scoping and analysis of the application domain, and subsequent formal specification of database requirements based on this domain analysis. Moreover, a goal-oriented design strategy is proposed to structure the transformation from the initial to the final conceptual schema, according to a set of user defined design issues, also modeled as goals.

The rest of the chapter is organized as follows. An overview of the main phases and concepts in the proposed process is given in Section 3.1. This is followed by a detailed step-by-step description of the process in Section 3.2. This process is illustrated using a running examples taken from the biological database case study (Section 3.3).
3.1 Process Overview

An overview of the proposed GODB process is shown in Figure 3.1. It covers both the analysis of initial requirements (goal-oriented requirements analysis phase) and the specification of these requirements in terms of a conceptual schema (goal-oriented schema design phase).

Figure 3.1: overview of the proposed process

3.1.1 Goal-Oriented Requirements Analysis.

This phase starts with a list of stakeholders and their high-level goals, which are refined and interrelated to produce a goal model. One of the claimed advantages of GORE approaches is the support for systematic exploration and evaluation of alternatives in fulfilling the top-level goals in the goal model. In context of database design, the goal model captures not a single, but several alternative sets of data requirements, from which a particular one is chosen to generate the conceptual schema for the database-to-be. These steps roughly correspond to the early and late RE phases in the TROPOS [32] methodology, but here the focus is on the data representation rather than the software specification part of the system.

The main concepts in this phase are adopted from the TROPOS metamodel [152], including hard goal, softgoal, plan, and the relationships among them (i.e., AND/OR-decomposition, means-end and contribution). Refer to Section 2.1.2 for a detailed discussion on these concepts.

3.1.2 Goal-Oriented Schema Design.

This phase is further divided into two sub-phases: (i) the modeling of the application domain, and (ii) the detailed design of conceptual schema.
In the first sub-phase, one design alternative is chosen from the goal model; application data requirements are extracted from the descriptions of goals and plans in the chosen alternative. These data requirements are used to generate the initial version of the conceptual schema. In the second sub-phase, this initial schema is transformed into a final one, in response to a set of design issues. Design issues mentioned in [25] include persistence, time and units. In this thesis, design issues of interest are those derived from softgoals. For example, in a hospital setting, the initial conceptual schema may only include patients, their medical measurements (e.g., blood pressure); But the equipments/methods may or may not be part of the final conceptual schema depending on design issues relating data provenance and data quality (see Chapter 4, Section 4.1 for more details on this).

It is important to note that feedback may be necessary from the detailed schema design phase to the requirements analysis and domain modeling phases. This is because additional plans, which have not been considered during goal analysis, may be identified in detailed schema design due to closer consideration of some softgoals; these plans are then used to extend the original goal model and lead to identification of additional concepts to be modeled. See Section 4.2 and 4.3 for a detailed discussion and examples of this feedback step in context of designing for data quality softgoals.

### 3.2 Process Steps

The proposed GODB process is divided into eight steps. Each step is described by its input, output, and a brief description.

**Step 1 Identify stakeholder goals.**

*Input:* A list of stakeholders.

*Output:* A list of high-level goals of the stakeholders.

*Description:* We assume there are a number of key roles exist with respect to a database, i.e.,
data provider, data owner, data administrator/regulator, data consumer and data standardization body. Furthermore, we assume that the list of stakeholders is given, instantiating these roles. Stakeholders express a variety of goals depending on their backgrounds, responsibilities and agendas. The objective of this step is to identify the top-level (strategic) goals of each stakeholder, including both hard goals and softgoals.

**Step 2 Generate a goal model.**

*Input*: A list of high-level goals produced in Step 1.

*Output*: A goal model.

*Description*: High level goals give the overall agenda promoted/pursued by stakeholders, but lack details. In this step, these goals are first refined through AND/OR-decomposition. Refined goals are then interrelated through contribution analysis [32]. On one hand, it helps identify positive/negative contributions among the fulfillment of hard goals. This type of contributions are normally absolute, meaning that for example if there is an absolute positive (++) contribution edge from Goal $A$ to $B$, $B$ is fulfilled whenever $A$ is; on the other hand, it helps identify positive/negative contributions to softgoals. By definition, the degree of contribution to softgoals is normally soft ($+/−$).

**Step 3 Select a design alternative.**

*Input*: The goal model obtained in Step 2.

*Output*: A set of the leaf-level goals in the goal model.

*Description*: A goal model captures not a single, but several design alternatives, thanks to goal OR-decomposition (i.e., goal level variability) and means-end analysis (i.e., process level variability). In this step we resolve the first type of variability by selecting a set of leaf-level goals whose collective fulfillment achieves the aggregate top-level goals; we use softgoals as evaluation criteria. On one hand, the contribution analysis performed in Step 2 helps to identify positive/negative contributions from the leaf-level hard goals to the softgoals. On the
other hand, softgoals may also conflict with each other, meaning there is usually no uniformly “best solution”. Conflicts among softgoals can be resolved by ranking softgoals into a partially ordered list according to stakeholders’ preferences. This step can be supported by backward goal reasoning from Tropos [74] (i.e., given a goal model, finding a set of leaf-level goals that together fulfill all root goals and also satisfy a number of desired constraints).

**Step 4 Identify concepts from goals.**

*Input:* The goals in the chosen design alternative.

*Output:* A list of concepts extracted from these goals.

*Description:* In this step, we extract the initial set of concepts from the descriptions of goals in the chosen design alternative. Collecting and analyzing initial data requirements from various sources is an important task that precedes conceptual schema design [11, 84, 125, 88, 50]; these sources include existing documentation, transcripts from user interviews, screen forms and reports of any pre-existing systems, and reference manual and domain ontologies. The goal-oriented approach complements the conventional ones by adding an early phase for data requirements elicitation that, among other benefits, captures the rationale behind collected data requirements.

**Step 5 Identify and select plans.**

*Input:* The goals in the chosen design alternative.

*Output:* A set of plans that collectively fulfill these goals.

*Description:* So far, the goals in the chosen design alternative are not actionable. The process of gradually analyzing goals to identify operational specifications is often referred to as goal operationalization [54, 6, 140]. In this step we operationalize goals into plans using the means-end analysis [32]. If more than one plan is identified to fulfill a goal, we select one using the same evaluation process as the goals (i.e., through their contributions to softgoals).

**Step 6 Identify concepts from plans.**
Input: The plans in the chosen design alternative.

Output: A list of concepts extracted from these plans.

Description: In this step, we expand the list of concepts extracted from goals in Step 4 using the input plans. We first produce for each leaf-level plan a detailed process description that characterizes its constituent operations in terms of pre-/post-conditions. Following [144, 54], we consider both inspecting operations (i.e., queries) and modifying operations (i.e., actions). This is particularly important to database design, since high-level queries can be treated in the same way as actions in requirements analysis. There are two ways in which concepts are identified from descriptions of operations: as the information needs to be available for the operations to be carried out (or generated as the result of the operations), or as the information that needs to be recorded to serve as evidence that the operations were carried out.

Step 7 Construct the initial conceptual schema.

Input: The expanded list of concepts.

Output: The initial version of the conceptual schema.

Description: In this step, the concepts are organized into the initial version of the conceptual schema, using a diagrammatic notation such as ER or UML diagrams, or a formal language such as first-order logic. The analysis is similar to Object-Oriented approaches [144, 25, 33] to domain modeling; it includes identifying missing concepts, resolving ambiguous, redundant and overlapping concepts, distinguishing between entities, relationships and attributes, and forming generalization hierarchies. It is important to notice that in a design process (as opposed to a compilation/translation one), the designer has to add information to move on from earlier stages to later ones. Some of this information may come from external sources of domain knowledge that are not linked to the goal model explicitly in the early steps, including reference models, such as domain ontologies.

Step 8 Construct the final conceptual schema.
**Input:** The initial conceptual schema from Step 7.

**Output:** The final conceptual schema.

**Description:** For this step, the initial conceptual schema from Step 7 is gradually transformed into the one. We propose a design strategy for structuring this transformation process. This design strategy is goal-oriented because it is based on a set of design issues which are modeled as softgoals in earlier steps, and operationalized in this step into technical decisions. The proposed design strategy consists of three sub-steps as explained below:

- **8.1 Instantiation.** In this step, the designer first select from softgoals identified in Step 1 as design issues of concern, and associate them with individual elements in the initial schema. Next, for each design issue associated with an element the designer choose to resolve it now at design time ($R$), or leave it open ($O$). Note that most design issues normally can only be address partially at design time (since they corresponds to softgoals); and a design issue marked with $R$ may still need operation-time intervention. In any case, design decisions are carefully documented.

- **8.2 Prioritization.** In this step, design issues marked with $R$ are prioritizes according to users’ preference. For example, a university may spend more effort in ensuring the accuracy of student grades than that of student addresses, while timeliness may be a more important issue than completeness for course schedule information.

- **8.3 Resolution.** In this step, the designer address each design issue marked with $R$ in the specified order. One way to do this is by defining (or selecting from a library) and applying design operations for each design issue. The DQ by Design process in Section 4.2 is about how such design operations can be derived for the DQ issues.
3.3 3SDB Case Study

In this section, the GODB process is demonstrated using a running example built on the 3SDB case study [98]. This case study is based on a real-world industrial gene expression application. The goal of the case study is to reconstruct the design process of its component biological sample database, and to show how a goal-oriented design could have worked based on evidence of the actual design process. First, the background and motivation of the original application is described in Section 3.3.1; then the definitions of relevant terms and main user requirements for the biological sample database is given in Section 3.3.2; and finally the running example is presented in Section 3.3.3.

3.3.1 Application Background

Gene expression applications involve three data spaces [114]: sample data space, gene annotation data space and gene expression data space. In the case study and as well of the running example, we focus on a subset of the first data space, i.e., the sample data space, hereafter called 3SDB (“Small Subset of Sample Database”).

Sample Data Space

The main concept in the sample data space is biosample, which represents the biological material that is screened in a microarray experiment. A biosample can be of tissue, cell or processed RNA type, and originates from a species-specific (e.g., human, animal) donor. Samples are associated with attributes that describe properties useful for gene expression analysis, such as sample structural and morphological characteristics (e.g., organ site, diagnosis, disease, stage of disease), donor data (e.g., demographic and clinical record for human donors, or strain, genetic modification and treatment information for animal donors). Samples may also be involved in studies and therefore can be grouped into several time/treatment groups.
Genomics 101

Gene expression systems, simply put, measure the level of the activity of genes in biological samples. For the software engineer, an analog of the genome is the source code of a very large and poorly documented concurrent program, where each gene is a “function”. We identify the genes by looking for matching pairs of begin and end statements. Each cell is an interpreter that has a copy of the full genome. At a given state of the execution, functions that are running represent genes that are expressed in the cell.

A biological sample contains millions of cells. A gene microarray is a camera that takes a picture of the sample and counts all the pixels corresponding to each gene for all cells, and thus reports an expression value for each gene. Given that we only have partial knowledge of what genes do, the significance of these pictures (experiments) is that they allow us to infer the function of the genes by correlating them with the conditions of the samples. In other words, if we know something about the samples and the identity of genes on the array, we can make sense of what these functions do.

A gene expression database is an invaluable research tool for studying the biology of genes and how gene expression changes in the presence of diseases or drug treatments. For example, such a tool can handle queries like “find genes whose expression goes up or down in samples derived from prostate biopsies of donors with PSA > 4, relative to normal tissues”. An answer to this query requires knowing about the origin of the sample, including the donor and his/her medical information. The goal of a sample data space is to maintain comprehensive information about the experimental samples.

---

1. In fact, a gene array does not recognize all the genes of an organism, but recognizes several thousands of them that are printed on the array.
2. PSA (Prostatic Antigen) is an early diagnostic marker for prostate cancer.
3.3.2 User Requirements

Sample

Samples are related to other samples in ways that depend on the collection process. Time series are special case of sample relatedness, where an ordered set of samples all related to the same sample source or subject with a dependency marked by a time interval. For example, samples $R_1$, $R_2$, and $R_3$ are derived from a group of similar rats after being treated with a compound for 1, 6 and 24 hours respectively.

To monitor/control the quality of biological sample and subsequently generated gene expression data, certain sample collection and processing parameters need to be recorded as well. These include, for tissue samples, the time it takes after the sample is excised and before it is frozen, and for cell culture samples, the primary tissue from where the cell is harvested and the method used to harvest and isolate the cell.

We also need to control access to biological sample and subsequently generated gene expression intensity data. A biological sample may be derived from a sample source that is provided by a specific collaborator. In this case this collaborator owns the biological sample and its gene expression data. A biological sample, whether it is owned by a specific collaborator, can be classified as public (i.e., can be accessed by all collaborators) or private (can be accessed only by specified collaborators). The owner of biological sample always has access the data generated from the sample.

Animal Model Study

In animal model studies, animal subjects are first measured for common conditions. Then tissue or cell samples originated from these animal subjects are treated and are observed at certain time points. Before the samples are send to gene expression analysis, quality / disease verification tests are also performed on them. Following is a process description for performing an animal model study.
P1.1.1: Generate a study design

*Precondition*: study purpose, financial constraints

*Postcondition*: study design (e.g., the number of subjects, type of treatment, number of time points of treatments and observations), animal subjects

Example: Compare two different strains of mice, one that is generically engineered and a disease (e.g., obesity) is inserted, and one that is normal (design will indicate whether samples are pooled from all animals or individual animal samples are screened?)

P1.1.2: Perform measurements on animal subjects

*Precondition*: animal subjects, study design

*Postcondition*: measurements on the conditions of the animal subjects (e.g., strain, weight, gender)

P1.1.3: Excise samples from animal subjects

*Precondition*: animal subjects, study design

*Postcondition*: untreated animal tissue or cell samples (e.g., from some organ such as brain, liver or heart)

P1.1.4: Perform treatments on animal samples

*Precondition*: untreated animal tissue or cell samples, study design

*Postcondition*: (genetic, compound or surgical) treated animal samples, disease, medicine, surgery

P1.1.5: Perform measurements on animal samples

*Precondition*: treated animal samples, study design

*Postcondition*: measurements performed on samples. Some sample measurement are performed only once per sample (e.g., weight), while others may be performed multiple time points per sample (e.g., dose per unit weight) depending upon the study design.

P1.1.6: Perform quality / disease verification tests on samples

*Precondition*: treated animal samples

*Postcondition*: quality / disease verification tests results
PI.1.7: Perform gene expression analysis on samples

Precondition: treated and verified animal samples

Postcondition: gene expression intensity on the samples

Sample Treatment

Samples taken from animal subjects can be treated before gene expression analysis. Sample treatment is a tuple that consists of a type and a detail description. The type of treatment can be ”Genetic” if the sample has been genetically modified (including diseases), ”Compound”, if the sample has been treated by some treatment agent at a given dose and time, or ”Surgical”, if the sample has been surgically modified.

Donor and Sample Measurement

Measurements on animal subjects and samples have associated units. Although measurements are standardized to single unit when stored (e.g., weight in grams), the user may still want to see the units explicitly in order to, for example, query these attributes in different units or use them in plotting tools. A general principle is that the values of measurement attributes should be tuples with a value and unit component. Separation of the numeric value form the unit string will allow order condition or even querying by different unit than the coding one.

Disease

A healthy animal sample may be treated with one or more diseases. A disease is described by a “term” and “code” in some vocabulary as well as a longer string “description”. The vocabulary can be either “informal” or a formal one like “SNOMED”. If “informal” is specified as vocabulary, then term can be any value (and code is not required). If a formal vocabulary is used, then the term and code values must be valid term and code values in the vocabulary. A further complication of this is there may be multiple standardization bodies. It is important to indicate the source of the vocabulary.
We are interested a particular type of sample-disease association: a disease is considered a “contributing” factor to the gene expression intensity of a sample. This can be partially achieved by comparing the temporal dimension of the disease and sample data. More specifically, we are interested in distinguishing between diseases diagnosed before, at and after sample donation time. For example, a liver tumor diagnosed at (or immediate before) sample-collecting time is considered more relevant than a liver tumor diagnosed 5 year before sample donation. But sometimes time stamp is not the only consideration for the “contributing” association. For example, a sample may be excised from a normal kidney of a diabetic donor who also has high blood pressure. In this case, although diabetes and high blood pressure are two diseases diagnosed at sample collection time, only one may considered as the main contributing factor to the sample. Furthermore, we need to keep track of the stages (e.g., early, late) of the “contributing” diseases.

It is very important to maintain the accurate description of donor disease information.

**Human Tissue Survey**

Unlike animal model study where treatments are performed under controlled conditions, in human tissue survey diseases are the properties of the samples. Following is a process description for performing human tissue survey studies.

**P1.2.1**: Collect human tissue samples from sources

_**Precondition:**_

_**Postcondition:**_ patient, organ, human tissue samples

**P1.2.2**: Collect patient demographic information

_**Precondition:**_ patient

_**Postcondition:**_ patient demographic information including name, gender, race, date of birth

**P1.2.3**: Collect donor medical information

_**Precondition:**_ patient

_**Postcondition:**_ patient disease information
**P1.2.4:** Perform quality / disease verification tests on samples  
*Precondition:* human tissue samples  
*Postcondition:* quality / disease verification tests results

**P1.2.5:** Perform gene expression analysis on samples  
*Precondition:* human tissue samples  
*Postcondition:* gene expression intensity on the samples

### 3.3.3 The Running Example

**Step 1 Identify stakeholder goals.**

In our example, three main domain stakeholders are given: Genomics Information Sponsor, Chief Scientist, and Pharmaceutical Partner (hereafter called *Sponsor*, *Scientist* and *Partner* respectively). The *Sponsor* plays the role of data provider who is responsible for creating a research tool for subscription by making available a high quality gene expression (GX) reference dataset (*G1*). She is also the decision maker who has the financial responsibility for the project (i.e. controlled budget of the experiments, *S1*). The *Scientist* and *Partner* are application domain experts in molecular biology and drug discovery respectively, whose daily work depends on GX reference datasets. The ultimate goal of the *Scientist* is to obtain a comprehensive understanding the biology of genes (*G3*) and have publishable research results (*S4*), while for the *Partner*, the top priority is to discover and validate drug targets, focusing only on a deep understanding of the properties of disease-specific genes (*G2*). Consequently, the *Partner* wants the GX reference dataset that has deep coverage of diseases (*S2*) and has high biological relevance to drug discovery (*S3*). These goals are shown in Figure 3.3.

Furthermore, the softgoals shared by all stakeholders in this case include data accuracy (*QS1*), data security (*QS2*), flexible representation (*QS3*) and provenance of measurement data (*QS4*). The stakeholders and their high-level goals are shown in Table 3.1 and as well as in Figure 3.2.
Table 3.1: stakeholders and their high-level goals

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Hard goal</th>
<th>Softgoal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sponsor:</td>
<td>G1</td>
<td>S1, QS1 - QS4</td>
</tr>
<tr>
<td>- data provider</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- decision maker</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partner:</td>
<td>G2</td>
<td>S2, S3, QS1 - QS4</td>
</tr>
<tr>
<td>- data consumer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- domain expert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scientist:</td>
<td>G3</td>
<td>S4, QS1 - QS4</td>
</tr>
<tr>
<td>- data consumer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- domain expert</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2: stakeholders and their high-level goals
**Step 2 Generate a goal model.**

In order for the Sponsor to achieve its top goal $G1$ of commercializing a GX reference dataset, she could *either* provide comprehensive GX data ($G1.1$) *or* provide disease-focused GX data ($G1.2$) (i.e., using OR-decomposition). On the contrary, the Partner’s top goal $G2$ of drug discovery is AND-decomposed into its subgoals $G2.1$ - $G2.3$, meaning that she has to achieve all these subgoals in order to fulfill the top goal. If we continue the decomposition process for each stakeholder, we produce a goal model, a portion of which is shown in Figure 3.3.

Going down in the tree, as goals become more specific, we can start recognizing the lateral influence between goals through contribution analysis. For example, Goal $G2.1.1$ of the Partner to subscribe to a disease specific disease collection becomes feasible if the Sponsor pursues Goal $G1.2$ which aims at providing disease specific GX data. Moreover, the contribution edge from Goal $G2.1.1$ to Softgoal $S3$ with label “+”, indicates that subscription to a disease specific GX collection contributes positively to the fulfillment of Partner’s softgoal of drug discovery relevance, while contribution edges labeled with “−” between $S1$ (control of budget) and $S2$ (coverage of diseases) models the fact that these two softgoals conflict with each other by nature.

**Step 3 Select a design alternative.**

Given the goal model from Step 2, there are total $2 \times 2 \times 3 = 12$ alternative ways to fulfill the top-level goals \{$G1, G2, G3$\}, as shown in Table 3.2. Assuming a partial ordering of softgoals \{$S2, S3\} \succ S1 \succ S4$ and the contribution edges as shown in Figure 3.3, the following choices can be made for the three variability points: disease focused ($G1.2$), multiple, both $G1.2.1.1.2$, subscription ($G2.1.1$). This produces the design alternative $DAI_{(Step3)} = \{G1.2.1.1.2, G1.2.1.1, G1.2.1.2, G1.2.1, G1.2, G1\} \cup \{G2.1.1, G2.1, G2.2, G2.3, G2\} \cup \{G3.1, G3.2, G3.3, G3\} \cup \{S1, S2, S3, S4\} \cup \{QS1, QS2, QS3, QS4\}$. \(^3\)

**Step 4 Identify concepts from goals.**

\(^3\)We simply consider all softgoals belong to every design alternative.
Figure 3.3: A portion of the goal model

Table 3.2: Design alternatives at the goal level

<table>
<thead>
<tr>
<th>Variability Points</th>
<th>Number of Choices</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage of the GX dataset</td>
<td>2</td>
<td>Budget (S1),</td>
</tr>
<tr>
<td>- Comprehensive (G1.1)</td>
<td></td>
<td>Publishable (S4)</td>
</tr>
<tr>
<td>- Disease focused (G1.2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Choice of data sources</td>
<td>2</td>
<td>Budget (S3)</td>
</tr>
<tr>
<td>- Single, own (G1.2.1.1.1)</td>
<td></td>
<td>Coverage (S2)</td>
</tr>
<tr>
<td>- Multiple, both (G1.2.1.1.2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method of gene discovery</td>
<td>3</td>
<td>Coverage (S2),</td>
</tr>
<tr>
<td>- Subscription (G2.1.1)</td>
<td></td>
<td>Relevance (S5)</td>
</tr>
<tr>
<td>- Inhouse screening (G2.1.2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Public DB (G2.1.3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The set of concepts, $DNC_I(Step_4)$, corresponding to the design alternative $DA_I(Step_3)$ is shown in Table 3.3. Of course, had a different design alternative been selected, this list would be different. For example, if Goal $G1.2.1.1.2.1$ is selected, instead of $G1.2.1.1.2.3$, the concept of collaborator will not be present; if $G1.1$ is selected instead of $G1.2$, additional concepts such as organism, body part, biological function need to be introduced.

Table 3.3: Concepts extracted from $DA_I(Step_3)$

<table>
<thead>
<tr>
<th>Goals</th>
<th>Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G1$</td>
<td>gene, gene expression</td>
</tr>
<tr>
<td>$G1.2$</td>
<td>disease</td>
</tr>
<tr>
<td>$G1.2.1$</td>
<td>linked(gene expression, disease)</td>
</tr>
<tr>
<td>$G1.2.1.1$</td>
<td>biological sample, donor</td>
</tr>
<tr>
<td>$G1.2.1.2$</td>
<td></td>
</tr>
<tr>
<td>$G1.2.1.2$</td>
<td>sample source, collaborator</td>
</tr>
</tbody>
</table>

**Step 5 Identify and select plans.**

At least two well-established procedures can be carried out to acquire biological sample and donor data needed for gene expression analysis: perform animal model study ($P1.1$) and perform human tissue survey ($P1.2$). The first choice studies animal models in a controlled environment, while the second relies on available tissue samples excised from human patients during a surgical operation. An animal model study is considered significantly more costly ($S1$) compared to a human tissue survey since it yields more samples and therefore requires more analysis resources. However, a disease model study gives better coverage of diseases ($S2$) than tissue surveys, because they are easier to manipulate in a controlled environment. With respect to higher biological relevance ($S3$), human data are obviously more valuable than animal data. This portion of the goal model is shown in Figure 3.4.

Means-end analysis introduces additional design alternatives. The total number of alternatives we have identified so far is $12 \times 2$ (for the type of studies) $\times 2$ (for choice of disease descriptions) $= 48$ as shown in Table 3.4.

---

4To avoid unnecessary repetition and simplify the discussion, we now focus on the part of the goal model rooted at Goal $G1$. 
Figure 3.4: A portion of the goal model enriched with plans

Table 3.4: Design alternatives at the plan level

<table>
<thead>
<tr>
<th>Variability Points</th>
<th># of Choices</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure used to acquire sample and donor data:</td>
<td>2</td>
<td>Budget (S1), Coverage (S2)</td>
</tr>
<tr>
<td>- Animal model study (P1.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Human tissue survey (P1.2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description of diseases:</td>
<td>2</td>
<td>Accuracy of Disease (Q51.1)</td>
</tr>
<tr>
<td>- Single, informal terms (P2.1),</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Multiple, standard vocabularies (P2.2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assuming the same partial ordering of softgoals as before, and the contribution edges shown in Figure 3.4, $P1.1$ and $P2.2$ are selected during the evaluation process. This extends the design alternative we obtained in Step 3: $DA1 = DA1(\text{Step3}) \cup \{P1.1, P2.2\}$. Other design alternatives we will refer to later in the discussion are $DA2 = DA1(\text{Step3}) \cup \{P1.2, P2.2\}$ and $DA3 = DA1(\text{Step3}) \cup \{P1.1, P1.2, P2.2\}$.

**Step 6 Identify concepts from plans.**

Animal model studies ($P1.1$) are normally carried out in the following steps before gene expression analysis: generating a study design ($P1.1.1$), measuring animal subjects ($P1.1.2$), excising ($P1.1.3$) and performing treatments ($P1.1.4$) on animal samples, obtaining planned sample measurements ($P1.1.5$), and performing quality and disease verification tests ($P1.1.6$). The concepts that are derived from these sub-plans are listed in Table 3.5. The final result of this step is the expanded list of concepts, corresponding to the design alternative $DA1$ from Step 5: $DNC1 = DNC1(\text{Step4}) \cup \{\text{concepts extracted from } P1.1 \text{ and } P2.2\}$. The same process would be carried out for $DA2$ (or $DA3$), if it were selected instead of $DA1$.

<table>
<thead>
<tr>
<th>Plans</th>
<th>Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P1.1.1$</td>
<td>study, study purpose, study design, animal donor;</td>
</tr>
<tr>
<td>$P1.1.2$</td>
<td>animal measurements (e.g., strain, weight, gender)</td>
</tr>
<tr>
<td>$P1.1.3$</td>
<td>organ, organ type, tissue, cell culture</td>
</tr>
<tr>
<td>$P1.1.4$</td>
<td>sample treatment, treatment type, treatment description), disease description</td>
</tr>
<tr>
<td>$P1.1.5$</td>
<td>sample measurement (e.g., weight, dose per unit weight)</td>
</tr>
<tr>
<td>$P1.1.6$</td>
<td>test (test type, test result)</td>
</tr>
</tbody>
</table>

**Step 7 Construct initial conceptual schema.**

Given the expanded list of concepts, $DNC1$ (from Step 6), a portion of the initial conceptual schema is sketched in Figure 3.5, using a UML Class Diagram.  

---

5 For the sake of space, we do not give the detailed descriptions for the steps.

6 For clarity, some of details are omitted in the diagram, such as association and role names, and attributes that are not relevant for the discussion.
Step 8  Construct the final conceptual schema.

**Instantiation.** According the softgoals identified in Step 1, there are four design issues to be addressed: data accuracy (QS1), data security (QS2), flexibility representation (QS3) and measurement provenance(QS4). Table 3.6 shows the instantiation of these design issues.\(^7\)

**Prioritization.** In this example, the design issues marked with \(R\) are prioritized in the following order: QS1, QS4, QS2.

**Resolution.** In what follows, a few examples of design operations for addressing design issues QS1, QS4 and QS2 are shown, as how to transform the initial conceptual schema (from Step 7, Figure 3.5) into the final conceptual schema. Notice we assume these design operations have already been defined; deriving design operations for the DQ issues is the main topic of Section 4.2.

<table>
<thead>
<tr>
<th>Design Issue</th>
<th>Schema elements, etc.</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>QS1</td>
<td>Disease.description, etc.</td>
<td>R</td>
</tr>
<tr>
<td>QS2</td>
<td>Biological Sample, Repeated Observation, etc.</td>
<td>R</td>
</tr>
<tr>
<td>QS3</td>
<td>Measurement, etc.</td>
<td>O</td>
</tr>
<tr>
<td>QS4</td>
<td>Tissue, Cell Culture, etc.</td>
<td>R</td>
</tr>
</tbody>
</table>

**Addressing accuracy issue.** Figure 3.6 shows how a design operation decomposes the attribute Disease.description in initial conceptual schema into Disease.name (i.e., a standardized

---

\(^7\)Marking of design issues can be done at a finer level, i.e., for each (design issue, schema element) pair.
name for the disease), Disease.code (i.e., the corresponding standardized disease code), Disease.source (i.e., the standardization source) and Disease.description (i.e., an informal disease description). The entity SNOMED_CT represents a controlled vocabulary for disease names and codes, derived from the SNOMED Clinical Terms, one of the standardized health care terminologies. This design operation contributes positively to the syntactic accuracy of Disease.description values.

Addressing provenance issue. In the context of e-science, measurement provenance refers to the metadata that describes in-silico experiments, including the purpose, creator and design of the experiments, and the parameters used in the data generation processes. In this example, measurement provenance is an important issue to be addressed. For example, in order to monitor / control the quality of biological samples and subsequently generated gene expression data, the information about the experimental process needs to be recorded as well. For Tissue, the process parameters that need to be stored include “the amount of time it takes after the tissue sample is excised and before it is frozen”; for Cell Culture, it includes “the method used to harvest and isolate the cell”. This issue can be resolved by an operation that adds the attributes time-to-freeze and isolation-method to Tissue and Cell Culture respectively (see Figure 3.7).

Addressing security issue. Data governance concerns additional data requirements that enforce the policies governing the use of application data. Being one type of data governance, the security issue concerns security assurance mechanisms which can be enforced at the schema level. In this example, security concern is based on following requirements statement: “A bio-

---

8http://www.snomed.org
logical sample may be derived from a sample source that is provided by a specific collaborator. In this case, this collaborator owns the biological sample data. The owner of biological sample always has unlimited access to all the data related to the sample, while other collaborator may also have certain access privileges based on mutual agreements.” Figure 3.8 shows how the security issue is addressed for the entity Biological Sample, by a design operation that introduces the entity User Group and the relationship access with the attribute privilege.

The final conceptual schema after applying the above and other operations is shown in Figure 3.9.

---

9It is important to note that (i) the sample operations are used for demonstration purpose; in general there is no warranty to find optimal or suboptimal solutions, and (ii) the proposed design strategy applies to any design issue that the stakeholders feel relevant to the problem at hand, not just those we show in the example.
Figure 3.9: a portion of the conceptual schema for DA1
Chapter 4

Data Quality as a Design Criterion

The GODB process described in Chapter 3 is general in the sense that it does not detail how particular types of softgoals are operationalized into technical designs. In other words, it is missing details in the operationalization of softgoals (see the NFR approach [47] for how it is done for software design).

The purpose of this chapter is to extend the GODB process to deal with DQ softgoals. This leads to a DQ by design (DQD) process. The basic idea is that data of low quality may be detected and corrected by performing various quality assurance activities that rely on techniques with different efficacy and cost under different circumstances. These activities may lead to quality assurance data requirements that cannot be acquired by analyzing the core business activities alone. The proposed DQD process aims at identifying and modeling quality assurance data requirements.

This chapter is organized as follows. Before describing the DQD process in detail, it is necessary to be more specific about what does it mean to have high quality data. For this, first, a compositional view to define DQ attributes is offered (Section 4.1); this allows to specify DQ requirements in a precise way. The main concepts and detailed steps of the DQD process are then discussed (Section 4.2), and illustrated using a running example adopted from the Expense Database Case Study [47] (Section 4.3).
4.1 A Compositional View of DQ Attributes

As reviewed in Section 2.3, DQ is a multi-dimensional and hierarchical concept [165, 111, 19]. Although significant amount of work has been devoted to defining DQ attributes (or DQ dimensions), ambiguity, subjectiveness and inconsistency in DQ definitions still presents a widely acknowledged problem [28, 111].

In this section, a less explored view to DQ is taken, where each DQ attribute is considered as a complex expression; the meaning of an attribute is therefore captured in terms of the meaning of its constituents and the structure of the expression. Instead of defining each DQ attribute separately, one could seek to answer the following questions: (i) What are the primitive constituents from which DQ attributes can be expressed? and (ii) How can these constituents be combined in a meaningful way? The concept of “sign” provides such a notion for the investigation of these questions. Data values in a database are above all linguistic signs that convey meaning from their producer to their user; DQ issues arise when discrepancies occur during this communication. Based on these observations, this section offers a compositional view to understand and define DQ attributes in terms of a variety of primitive relationships between data values and their senses.

The rest of the section is structured as follows. First, a motivation of this work is given in Section 4.1.1, using a few concrete examples where traditional views on DQ are considered unsatisfactory in determining defective values. Second, the nature of DQ requirements is analyzed using the notion of signs in Section 4.1.2. Next, based on the characterization of DQ requirements, four categories of DQ predicates are discussed in Section 4.1.3. Finally, these DQ predicates are used to express existing DQ attributes proposed in the literature in Section 4.1.4. This exercise allows to (i) reveal and differentiate multiple, sometimes conflicting, definitions of a DQ attribute; (ii) accommodate competing views on how these attributes should be related; and (iii) point to possible new attributes.
4.1.1 Motivating Examples

Consider a table *Patient* (Table 4.1) that records body temperatures for patients in a hospital. Suppose that each row in the table records the temperature value of a particular patient at a specific time point. First, let us consider accuracy, one of the most studied DQ attributes. It has been defined as a measure of “the closeness between a value \( v \) and a value \( v' \), considered as the correct representation of the real-life phenomenon \( v \) aims to represent” [136, 19]. For example, if the patient’s real name is \( v' = 'Ben Cheung' \), but was recorded as \( v = 'Ben Franklin' \) instead, we may conclude that \( v \) is inaccurate.

Table 4.1: The Patient table

<table>
<thead>
<tr>
<th>Name</th>
<th>Temperature</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben Cheung</td>
<td>37.2°C</td>
<td>2007/11/05 13:05</td>
</tr>
<tr>
<td>Ben Cheung</td>
<td>38.5°C</td>
<td>2007/11/06 12:00</td>
</tr>
<tr>
<td>Ben Cheung</td>
<td>37.0°C</td>
<td>2007/11/07 11:55</td>
</tr>
</tbody>
</table>

**Example 1.**

In some cases, our judgment of accuracy does not rely on syntactic proximity of data values, but is affected instead by our interpretation of their meanings. For example, it would have been no less accurate to have ‘98.6°F’ instead of ‘37.0°C’ in the last row, as long as we understand that these two values represent the same temperature reading using different scales.

**Example 2.**

Moreover, whether a data value is considered accurate often depends on both its interpreted and intended meaning. For example, if there is no agreement on how the temperature should to be measured, we may interpret ‘37.2°C’ in the first row as Ben’s temperature measured under normal conditions, while it really represents his temperature after aspirin was administered. Inaccuracy caused by such a mismatch is no less a problematic than a typographical error (e.g., entering ‘36.2°C’ instead of ‘37.2°C’).
**Example 3.**

Furthermore, accuracy cannot be considered in isolation: our judgment on accuracy of a value depends on the judgment of that of its related values. For example, consider ‘38.5°C’ and ‘2007/11/06 12:00’ in the second row. If we know that Ben’s temperature was 39 degree Celsius on Nov. 6, 2007 at 12:00, we may want to conclude that ‘38.5°C’ represents inaccurately the real-world phenomenon (i.e., 39 degree Celsius). But, in doing so we have already made an assumption that ‘2007/11/06 12:00’ is accurate! What if we instead know that Ben’s temperature was 38.5 degree Celsius on Nov. 6, 2007 at 11:45? In this case, are we willing to believe that it is the time not the temperature value that was inaccurately recorded?

Consider next completeness, another commonly studied DQ attribute, which has been defined as the percentage of all tuples satisfying the relational schema of a table (i.e., tuples in the true extension of the schema) which are actually presented in Table 4.1 [19].

**Example 4.**

Actually, it is impossible to talk about the “true” extension of a relational schema without knowing what the user’s requirements are. Accordingly, the above data about Ben Cheung could be complete or incomplete depending on whether Ben’s temperature is required to be measured only once or twice a day.

**4.1.2 Nature of Data Quality**

In this subsection, we describe our view of DQ, founded on the notion of signs [128]. Generally speaking, a sign is something that stands to someone for something else. Accordingly, we see values (together with their metadata) in databases as primarily linguistic signs standing for real world phenomena. Information processing is a form of communication realized by creating, passing and utilizing signs [110]; DQ issues arise when discrepancies occur during this communication.
In the meaning triad [110], a triadic sign model, a symbol (e.g., 'Ben Cheung’) is connected to a referent (e.g., a particular person in the world), and a sense understood by its interpreter (e.g., the concept of that person in the interpreter’s mind). The difference between the referent and sense of a symbol could be understood in analogy to that of the extensional and intensional definitions of a term. Moreover a symbol may have more than one “valid” sense (and referent), under different circumstances, according to different interpreters.

Types of Senses

We find it useful to distinguish four types of senses (and referents) of a symbol:

- The intended sense is the sense of the symbol according to its producer. It is the meaning the producer intends to communicate, and is determined exclusively by the producer.

- The interpreted sense is the sense of the symbol according to its user. It is the meaning the user recognizes, and is determined exclusively by the user.

- The supposed sense is the sense, determined exclusively by the requirements for production of the symbol, such as conventions and regulations the producer has to comply with, ethical and social norms, etc.

- The expected sense is the sense, determined exclusively by the conditions for use of the symbol, such as the tasks, purposes and goals of the user.

To illustrate this distinction, consider the temperature value ’37.2°C’ in Table 4.1. Suppose Sudha, the doctor of Ben, needs to know his temperature, not lowered by an antipyretic, and measured around noon every day (because he is plotting a graph with X-axis points every 24 hours). She also expects the measurement to be taken using a thermometer in the mouth.

A new nurse, Catherine, running late, measured Ben’s temperature at 13:05, with a thermometer in the ear. Moreover, Catherine is unaware of the fact that Ben had taken an antipyretic at 12:40. As a result, by recording ’37.2°C’, Catherine intended to say “Ben’s temperature without antipyretic, measured at 13:05 with a tympanal thermometer”.
If Catherine had been more careful, this value’s supposed meaning would be “Ben’s temperature after antipyretic, measured at 13:05 with some thermometer”.

On the other hand, Sudha may interpret this value as “Ben’s temperature without antipyretic, measured at 13:05 (because he saw the time value in the table) with an oral thermometer”, which is different from what he expected: “Ben’s temperature without antipyretic, measured around noon with an oral thermometer”.

Ideally, total DQ means that the four types of senses must match for each data value individually, and certain constraints must hold among the same types of senses for related values, especially ones in different fields of the same row. DQ issues arise when this does not hold. For example, when Sudha expects oral measurements, but this requirement is not specified explicitly, discrepancy is likely to exist between the expected and supposed senses. More generally, if some sources of variability (e.g., the type of thermometer used and patient conditions) are not captured in the data (or metadata), the communication between the producer and user will be ambiguous. Of course, whether or not such ambiguity is considered problematic depends on the purpose for which the data is to be used, and it is the role of the requirements specification to eliminate these problems.

**Description of Senses**

Senses can be described according to an upper ontology. To illustrate, consider the DOLCE ontology [115], which views the world as populated by entities, including concrete physical objects (e.g., persons) as well as abstract regions (e.g., distance values); the latter can appear as the values of properties, called qualia, for objects. To help communication, entities have names that allow them to be uniquely identified within some more or less restricted context: ‘Ben Cheung’ is presumably sufficient to identify the patient currently in the hospital in the previous example. Qualia are associated with properties at specific times (which are also

---

1DOLCE calls properties “qualities”, but we find this too confusing in our context, where we are talking about data quality. Also, DOLCE refines properties into entities that “inhere” in objects – a complication that is unnecessary in our context.
treated as qualia), allowing property values to change.

The fundamental premise of databases is that one can associate a semantics with a relational table such as $\text{Patient}(\text{NM, } TPTR, TM)$ along the lines of “the unique person named $\text{NM}$ has temperature property value $\text{TPTR}$ at time $\text{TM}$”, a semantics that must be shared by data producer and user for proper communication. Given the DOLCE ontology, this might be written in FOL as

\[
\text{Patient}(\text{NM, } TP, TM) \rightarrow \exists!p: \text{Person, tm:TimeQuale, tp:TemperatureQuale.}
\]

\[
\text{hasName}(p, \text{NM}) \land \text{hasName}(tm, TM) \land \text{hasName}(tp, TP) \land \text{temptrOf}(p, tm) = tp.
\]

Based on this, the interpreted senses of the value ‘37.2° C’ in $\text{Patient(‘Ben Cheung’, ‘37.2° C’, ‘2007/11/05 13:05’)}$ could be $m =$ “the temperature quale for the unique person named Ben Cheung at time quale 2007/11/05 13:05”. We assume that functions exist to access different components of a sense. For example, $t(m)$ returns the time component of $m$.

The above account is idealized, since it is usually necessary to observe or measure properties. This introduces a process of measurement, which allows the semantic specification to capture additional requirements. For example, the following formula specifies the kind of instrument to measure the temperature with, and a constraint on the time when measurements are to be taken:

\[
\text{Patient}(\text{NM, } TP, TM) \rightarrow \exists!p: \text{Person, instr:OralThermometer.}
\]

\[
\text{hasName}(p, \text{NM}) \land \text{measures}(\text{temptrOf}(p, TM), TP, instr, TM) \land \text{closeToNoon}(TM)
\]

in which additional variables for qualia to be “named” by $\text{TM}$ and $\text{TPTR}$ are omitted. Moreover, measurements are almost never exact, so the precise semantics may need to talk about accuracy and precision errors for measurements or the instruments involved, the subject of metrology.

**Comparison of Senses**

The above considerations allow us to see a basis for distinguishing different degrees of match between two senses, $m_1$ and $m_2$, of a data value $s$. 
• At one extreme, there is an ideal exact match \( \text{match}_{\text{exact}}(m_1, m_2) \), when the senses are identical.

• At the other extreme, there is a total mismatch \( \text{match}_{\text{mismatch}}(m_1, m_2) \), when for example \( m_1 \) is a temperature quale while \( m_2 \) is a person.

• In between, there are partial matches \( \text{match}_{\text{partial}}(m_1, m_2) \) where \( \text{attr} \) is the attribute of which \( s \) is a value; for example, the four senses of Ben’s temperature value ’37.2°C’ discussed in the previous section would match partially.

• It is also useful to have a more precise variant of partial match, \( \text{closer}_{\text{attr}}(m, m_1, m_2) \), which indicates that \( m_1 \) is closer to \( m \) than \( m_2 \) is; it allows to find that, all other things being equal, a 13:05 measurement of a particular property is closer to a noon one than a 14:30 measurement.

4.1.3 Data Quality Predicates

According to the sign-based view of DQ, primitives for defining DQ attributes could be defined in terms of the relationships between symbols and their senses. These primitives are referred to as \( \text{DQ predicates} \) thereafter. According to the types of senses involved in the relationships, DQ predicates are grouped into four categories: symbol, meaning, purpose and trust. To illustrate the idea, in what follows each category is described using a few examples of DQ predicates in that category. This is, however, by no means an exhaustive list of possible DQ predicates that could be defined.

Symbol Predicates

DQ predicates in this category concern the relationships involving symbols only, without explicitly mentioning their senses. Let \( S \) be a set of symbols of interest. First we may be interested in the membership of a symbol \( s \in S \) in a subset \( S_{\text{accept}} \) of \( S \). Let us denote this using the predicate \( \text{sym}_{\text{member}}(s, S_{\text{accept}}) \Leftrightarrow s \in S_{\text{accept}} \). For example, \( \text{sym}_{\text{member}}('50\,^\circ C', S_{\text{body-\text{temp}}}) \)
does not hold, assuming $S_{body-temp}$ is the set of symbols representing the acceptable human body temperatures.

For acceptable symbols, we may now consider a variety of relationships between them. The simplest such relationship is sameness: let $sym_{match}(s_1, s_2)$ hold whenever $s_1$ and $s_2$ have exactly the same syntactic form. When two symbols do not match exactly, we may consider which are closer syntactically, based on some distance function $distance_f$ (such as edit distance [19]). Let us write this using the predicate $sym_{closer}(s, s_1, s_2) \iff distance_f(s, s_1) < distance_f(s, s_2)$. For example, $sym_{closer}('Cheng', 'Cheung', 'Chiang$') is true because changing from 'Cheng' to 'Cheung' requires fewer edits than to 'Chiang'.

Another interesting predicate, $sym_{more-detail}(s_1, s_2)$, concerns level of detail; for real numbers we might have $sym_{more-detail}(3.1415926', '3.14')$ indicating that, in normalized scientific notation, (i) the two arguments have the same exponent, (ii) the first argument has at least as many digits as the second one in the coefficient, and (iii) the coefficients agree in the digits presented.

**Meaning Predicates**

DQ predicates in this category concern the relationships involving the interpreted and intended senses of a symbol. According to H.P. Grice’s classical account of speaker meaning, we rely on the recognition of our intention to communicate and we use that very recognition to get our message across [79]. In the context of DQ, this implies that in an ideal communication, there should be an exact match between intended and interpreted senses.

Let $M$ be the set of senses to which the symbols in $S$ may refer. First of all, we need to know whether for each symbol there is an interpreted (or intended) sense assigned to it by its user (or producer). Let us use the predicate $mea_{has-intp}(s, m)$ (respectively, $mea_{has-intd}(s, m)$) to indicate that a sense $m \in M$ is an interpreted (respectively, intended) sense of a symbol $s \in S^2$.

---

2 Throughout the rest of the paper, when we mention symbol $s$, we mean a symbol token - its occurrence in a field of a particular table tuple. So '37.2°C' is the occurrence of this symbol in row 1, column 2 of Table 4.1.
For example, $\exists m \in M.\text{mea}_{\text{has-intp}}(\text{`}37.2^\circ C', m)$ probably does not hold for a physician who doesn’t work in Ben’s hospital, because she will not have a way to identify the person named ‘Ben Cheung’ in that hospital.

When a symbol has an interpreted and intended sense, we are mostly interested in whether there is a match between them. First we want to know if they match exactly

$$\text{mea}_{\text{match}}(s, m_1, m_2) \iff \text{mea}_{\text{has-intp}}(s, m_1) \land \text{mea}_{\text{has-intd}}(s, m_2) \land \text{match}_{\text{exact}}(m_1, m_2).$$

For example, $\text{mea}_{\text{match}}(\text{`}37.2^\circ C', m_1, m_2)$ does not hold when $m_1$ and $m_2$ are temperatures of a patient measured at different time points.

For partially matched senses, we may want to know how closely they match. For example, when two symbols $s_1$ and $s_2$ share their intended senses (e.g., because people recorded the same value with different precision), we can state the fact that “the interpreted sense of $s_1$ is closer than that of $s_2$ to their shared intended sense” using following predicate

$$\text{mea}_{\text{closer}}(s_1, s_2, m, m_1, m_2) \iff \text{mea}_{\text{has-intd}}(s_1, m) \land \text{mea}_{\text{has-intd}}(s_2, m) \land \text{mea}_{\text{has-intp}}(s_1, m_1) \land \text{mea}_{\text{has-intp}}(s_2, m_2) \land \text{match}_{\text{attr}}(m_1, m) \land \text{match}_{\text{attr}}(m_2, m) \land \text{closer}_{\text{attr}}(m, m_1, m_2).$$

**Purpose Predicates**

DQ predicates in this category concern the relationships that involve the interpreted and expected senses of a symbol from the user perspective. As we have mentioned, an ultimate criterion for data quality is fitness for purpose. Here the intended use of data values is captured through their expected senses. Therefore, quality issues arise when the interpreted and expected senses of a data value do not match exactly.

Predicates such as $\text{pur}_{\text{match}}(s, m_1, m_2)$ (for indicating the interpreted sense $m_1$ and expected sense $m_2$ of the symbol $s$ match exactly), and $\text{pur}_{\text{closer}}(s_1, s_2, m, m_1, m_2)$ (for indicating the interpreted sense $m_1$ of $s_1$ is closer than the interpreted sense $m_2$ of $s_2$ to their shared intended sense $m$) are defined in a similar way to their counterparts in the meaning category. The existence of expected sense, however, deserves more discussion.
Unlike the interpreted sense which is determined by the user directly, the expected sense is determined by a particular application. If a doctor is only interested in studying the effect of psychotherapy on the temperature of the patient, the blood pressure (or more obviously the number of chairs in the room) have no expected senses to that doctor. To formalize this, let $M_e$ denote a subset of $M$, determined by the tasks and goals the user has to fulfill. As an example, $M$ might have temperatures and blood pressures taken at any time, while $M_e$ might only have temperatures taken around noon. Therefore $m \in M_e$ is an expected sense of a symbol $s$ if $m$ matches, at least partially, with the interpreted sense of $s$. This can be stated using following predicate

$$pur_{\text{has-exp}}(s, m) \iff m \in M_e \land \exists m' \in M. mea_{\text{has-ntp}}(s, m') \land \left(\text{match}_{\text{attr}}^\text{partial}(m, m') \lor \text{match}_{\text{exact}}^\text{exact}(m, m')\right).$$

**Trust Predicates**

DQ predicates in this category concern the relationships involve the intended and supposed senses of a symbol from the producer perspective. According to [132], in order to establish audience trust, both the sincerity and authority conditions have to hold. In the context of DQ, this means the user has to believe that the producer is neither a liar (i.e., no discrepancy caused intentionally, e.g., due to falsification) nor a fool (i.e., no discrepancy caused unintentionally, e.g., due to observation bias). Trust issues arise therefore when there is discrepancy between intended and supposed sense. Predicates in the aspects, such as $\text{tru}_{\text{has-sup}}$, $\text{tru}_{\text{comparable-sup}}$ and $\text{tru}_{\text{match}}$ are defined in the similar way as their counterparts in the purpose category.

### 4.1.4 Expressing Data Quality Attributes

Each DQ attribute can be expressed using DQ predicates from one or more categories. In what follow, a number of well-known DQ attributes are used as examples to show how this can be done. One observation from this exercise is that a single DQ attribute may have been
assigned multiple, sometimes conflicting, definitions in the literature; this is because it can be expressed using different DQ predicates. Moreover, there are often competing views on how DQ attributes are related; this can be accommodated by making explicit the exact meaning of the attributes involved, and by distinguishing relationships that exist by definition and those that exist based on assumptions. Finally, this exercise also help identify possibly new DQ attributes that have been ignored before.

Accuracy, Precision and Currency

Accuracy is normally understood as free of defects or correspondence to reality [166, 111]. In [19], it is defined formally as the closeness between two representations \( s \) and \( s' \), where \( s' \) is the correct representation of the real-life phenomenon \( s \) aims to represent. If we accept that “correctness” here means “justified by some accepted standards or conventions”, and make “closeness” be “identity” to get a Yes/No predicate, then this definition can be stated in terms of predicates in the symbol, meaning and trust categories:

\[
\text{accuracy}_{\text{symbol}}(s) \Leftrightarrow \exists m \in M, s' \in S. \text{sym}_{\text{match}}(s, s') \land \text{mea}_{\text{has-intd}}(s, m) \land \text{tru}_{\text{has-sup}}(s', m).
\]

According to this definition, we cannot have synonyms such as '37.0\(^\circ\)C' and '98.6\(^\circ\)F', which may have been desired. To accommodate this, we can change the perspective from a fixed phenomenon to a fixed representation [162]; it defines accuracy as the closeness between two real-life phenomena \( m \) and \( m' \), where \( m \) is what the symbol \( s \) aims to represent and \( m' \) is what \( s \) appears to represent. This view requires only predicates in the meaning category:

\[
\text{accuracy}_{\text{meaning}}(s) \Leftrightarrow \exists m_1, m_2 \in M. \text{mea}_{\text{match}}(s, m_1, m_2).
\]

The fact that \( s_1 \) is more accurate than \( s_2 \) can then be represented in this view as:

\[
\text{accuracy}_{\text{meaning-compare}}(s_1, s_2) \Leftrightarrow m, m_1, m_2 \in M. \text{mea}_{\text{closer}}(s_1, s_2, m, m_1, m_2).
\]
A typical understanding of precision as a DQ attribute is the degree of details data values exhibit. For example, precision of numeric values is often measured by the number of significant digits used [68]. A number (e.g., ’3.1415926’) is more precise than another one (e.g., ’3.14’), assuming both represent the same phenomenon (e.g., the mathematical constant $\pi$), can be stated as:

$$\text{precision}_{\text{symbol-compare}}(s_1, s_2) \Leftrightarrow \text{sym}_{\text{more-detail}}(s_1, s_2) \land \exists m_1, m_2 \in M.$$ 

$$\text{mea}_{\text{has-intd}}(s_1, m_1) \land \text{mea}_{\text{has-intd}}(s_2, m_2) \land \text{match}_{\text{exact}}(m_1, m_2)$$

Precision is often considered in close relation to accuracy. A typical intuition is that low precision leads to inaccuracy [162, 68], which however cannot be accommodated by $\text{precision}_{\text{symbol-compare}}$ alone. This is because having greater degree of details doesn’t guarantee a better interpretation towards the intended meaning. In order to support this intuition, we need a strengthened version of precision:

$$\text{precision}_{\text{symbol-strengthened}}(s_1, s_2) \Leftrightarrow \text{precision}_{\text{symbol-compare}}(s_1, s_2) \land \text{accuracy}_{\text{meaning-compare}}(s_1, s_2).$$

From a different view, one considers accuracy as a prerequisite for precision: in order to say $s_1$ is a more precise than $s_2$, both have to be accurate (i.e., have matching intended and interpreted senses). This view can be defined as:

$$\text{precision}_{\text{meaning-compare}}(s_1, s_2) \Leftrightarrow \text{sym}_{\text{more-detail}}(s_1, s_2) \land \exists m_{11}, m_{12}, m_{21}, m_{22} \in M.$$ 

$$\text{mea}_{\text{match}}(s_1, m_{11}, m_{12}) \land \text{mea}_{\text{match}}(s_2, m_{21}, m_{22}) \land \text{match}_{\text{exact}}(m_{11}, m_{21}).$$

Now we really have a:

$$\text{precision}_{\text{meaning-compare}}(s_1, s_2) \rightarrow \text{accuracy}_{\text{meaning}}(s_1) \land \text{accuracy}_{\text{meaning}}(s_2).$$

Currency as a DQ attribute is normally understood as the degree to which data are up to date [19, 136]. As a first try, we could represent this understanding as:

$$\text{currency}_{\text{naive}}(s_1, s_2) \Leftrightarrow \exists m_1, m_2 \in M. \text{mea}_{\text{has-intd}}(s_1, m_1) \land \text{mea}_{\text{has-intd}}(s_2, m_2) \land t(m_1) > t(m_2),$$
where \( t \) returns the time component of a sense. One might notice that this definition allows us to compare the currency of the temperatures of different patients.

When this is not desired, we can strengthen it using the notion of partial match:

\[
currency_{\text{strengthened}}(s_1, s_2) \iff \exists m_1, m_2 \in M. \text{mea}_{\text{has-intd}}(s_1, m_1) \wedge \text{mea}_{\text{has-intd}}(s_2, m_2) \wedge \\
\text{match}_{\text{partial}}(m_1, m_2) \wedge t(m_1) > t(m_2).
\]

Currency defined in this way is orthogonal to accuracy. As with precision, some authors consider a value \( s_1 \) is more current than another one \( s_2 \) only when both are accurate at a certain point in time [162]. This view can be captured by:

\[
currency_{\text{meaning}}(s_1, s_2) \iff \exists m_{11}, m_{12}, m_{21}, m_{22} \in M. \text{mea}_{\text{match}}(s_1, m_{11}, m_{12}) \wedge \\
\text{mea}_{\text{match}}(s_2, m_{21}, m_{22}) \wedge \text{match}_{\text{partial}}(m_{11}, m_{21}) \wedge t(m_{11}) > t(m_{21}).
\]

A further complication, which will be discussed below, relates currency to relevance [68].

**Relevance, Completeness and Timeliness**

Relevance considers how data fits its intended use [111]. In its simplest form, it can be defined using the predicates in the purpose category alone (recall \( M_e \) is a subset of \( M \), determined by the tasks, etc. the user of \( s \) has):

\[
\text{relevance}_{\text{purpose}}(s) \iff \exists m \in M_e. \text{pur}_{\text{has-exp}}(s, m).
\]

This definition supports the view that relevance should be evaluated before other quality attributes [68].

Intuitively, completeness concerns whether data is missing with respect to some reference set. In the simplest case, value completeness [19, 131] refers to the presence of null values in a reference column, row or table. This definition can therefore be understood as:

\[
\text{completeness}_{\text{symbol}}(S_a) \iff \exists s \in S_a. \text{sym}_{\text{match}}(s, \text{"null"}),
\]
where $S_a$ is the set of data values of interest. In a more complicated situation, population completeness [131] of $S_a$ is defined as the existence of missing values with respect to the reference set $M_e$:

$$completeness_{purpose}(S_a) \Leftrightarrow \forall m \in M_e \exists s \in S_a. \pur_{has-exp}(s, m).$$

While the notion of completeness concerns whether every relevant data value is presented, we may also consider whether every presented value is relevant (the closest terms proposed in the literature for this attribute are “appropriate amount of data” [111] and “conciseness” [162]):

$$completeness_{purpose-reverse}(S_a) \Leftrightarrow \forall s \in S_a \exists m \in M_e. \pur_{has-exp}(s, m).$$

Some authors use timeliness to mean data is sufficiently up to date with respect to its intended use [19, 133]. It can therefore be considered as another variant of currency [68]. The fact that a value $s_1$ is timelier than $s_2$ with respect to $M_e$ can be stated as:

$$currency_{purpose}(s_1, s_2) \Leftrightarrow currency_{meaning}(s_1, s_2) \land relevance_{purpose}(s_1) \land relevance_{purpose}(s_1).$$

**Reliability and Believability**

There is no generally accepted notion of reliability as a DQ attribute: some definitions overlap with that of accuracy [1], others are linked to dependability of the data producer [111], while still others are based on verifiability [126]. If we choose the last view — that data is reliable if it can be verified (i.e., generated independently by different producers, possibly using different tools, methods, and etc.), we can define, given expect senses $M_e$:

$$reliability_{trust}(s) \Leftrightarrow \exists m_1 \in M, m_2 \in M_e. \tru_{match}(s, m_1, m_2).$$

This means what is intended to be represented by $s$ matches exactly with what is supposed to be represented by it, according to the obligations the producer has. A violation of this condition may be caused by bias (i.e., lack of objectivity [19, 166]) or intention (i.e., intentional
falsification [111]) of the producer, or limitation of instrumentation, method, etc. Notice that reliability defined in this way is independent of accuracy.

On the contrary, believability defined in [19, 166] as “the extent to which data are accepted or regarded as true, real, and credible”, clearly concerns both the meaning and trust predicates:

$$believability_{meaning-trust}(s) \leftrightarrow accuracy_{meaning}(s) \land reliability_{trust}(s).$$

4.2 Data Quality By Design Process

Broadly speaking, to maintain high quality data, two alternative approaches exist. Curative approaches focus on detecting and correcting existing errors, using probabilistic, empirical or knowledge-based techniques [19]. Most of the existing work on DQ focuses exclusively on curative approaches. For example, Object Identification is a well-studied approach to improve the accuracy of existing data, by looking for the same data in different data sources and finding the correct data by comparison [19].

On the other hand, preventive approaches aim at preventing errors from occurring, or at least, reducing the chance of their occurrence; they rely on improved practice of data acquisition, manipulation and dissemination, within both the system and its environment. The idea of “quality information by design” has long been raise in the [163] (as an analog to that of quality-by-design in product manufacturing); it studies DQ problems by viewing information as products of an information manufacturing system, where each stage of the manufacturing process can be analyzed for quality concerns [12, 163, 148].

A pure curative approach is not enough to address all DQ problems. Often some DQ problems that exist in a database cannot be repaired [127]: in some cases, although we can detect that errors exist in the database (e.g., violations of some rules), it may be impossible to pinpoint the exact location of these errors; in other cases, the number of errors makes it impractical to seek out and repair the wrong ones; still in other cases, it may be no longer possible to obtain the correct information.
A pure preventive approach is also not enough. This is because, in some case, an effective and practical preventive measure cannot be defined until the DQ problem to be prevented has occurred, detected and cured at least once; in other cases, preventive measures can only prevent certain type of DQ problems. For example, defensive checkers can prevent values that are invalid individually or in combination, but not values that are just plain wrong [127].

Also note that experience from the practitioners [127] shows that it is unlikely to achieve 100% DQ in practice. Nevertheless, improvements in DQ are worthwhile since most applications do not demand 100% percent DQ to satisfy their requirements. For example, most decision support applications have a tolerance level for inaccurate data: inaccuracies up to the tolerance level allow the application to provide high-quality decision.

In this section, the GODB process proposed in Chapter 3 is extended specifically for dealing with DQ softgoals. This leads to a DQ by design (DQD) process. The rest of the section is organized as follows. First, the main concepts involved in the DQD process are discussed in Section 4.2.1. Then an overview of the process is presented in Section 4.2.2, which is followed by a detailed step-by-step description of the process 4.2.3.

### 4.2.1 Quality Assurance Data Requirements

Errors in data values may be detected, corrected and prevented by performing various quality assurance (QA) activities, which may require and/or produce additional data. In other words, these activities may lead to additional QA data requirements that cannot be acquired by analyzing the core business activities alone. Whether or not these requirements are considered during schema design may influences the ability to carry out the QA activities later on during operation. The DQD process aims at identifying and modeling these QA data requirements, which can be classified into following categories.

1. The most common way to provide QA to data values is to pose restriction on them. This normally results in integrity constraints to be specified on the schema, or other more complicated rules implemented by stored procedures or triggers. This may also lead to metadata
to be stored locally or at least accessed remotely (e.g., lookup tables).

2. Quality in data largely depends on the business activities in which it is produced and used [12, 163, 148, 47]. These include initial acquisition, subsequent manipulation, and final dissemination of data. Any additional data tracking these activities provides evidences (to a certain degree) that they were carried out with proper attention to DQ requirements. (For a more detailed discussion and examples on use of metadata for error detection, see Section 5.1.2).

3. If necessary, DQ measurements can be stored together with data. How to design schemas that allow recording DQ measurements have been explored both for conceptual database design in general and data warehousing design in particular (see Section 2.3.5).

In summary, DQD broadens the scope of database design. Databases need to maintain not only application data derived from core business activities, but also QA data derived from data and QA activities. Both types of data requirements need to be considered at the design time and subject to similar modeling and analysis process. The difference is that the modeling and analysis of QA data requirements tends to be more reusable across applications and domains. See Section 5.1 for a discussion on building a repository of DQ rules.

4.2.2 Process Overview

The DQD process extends the GODB process, by detailing its Step 8.3 (Resolution) for addressing DQ softgoal. More specifically, it takes as input the conceptual schema, which specifies application data requirements (as derived from the GODB process, Step 1 - 7), e.g., Student(id, name, address), and a list of DQ related design issues, associated with individual elements in the schema (as derived from the GODB process, Step 8.1 (Instantiation) and Step 8.2 (Prioritization)), e.g., Accurate[Student.name].

For each design issue that is chosen to be resolved, a list of risk factors, e.g., the clerk misspells a name value, and mitigation plans, e.g., to require and store duplicated name entries are identified. The mitigation plans model QA activities to be introduced, and often lead
to a revised version of the original schema, e.g., \textit{Student}'(id, name\textsubscript{1}, name\textsubscript{2}, address), in order to support these QA activities. The schema and its supported QA activities are called a \textit{DQD proposal}. QA activities may involve manual work, e.g., clerk enters name values twice, and/or rely on DQ techniques, e.g., to enforce the rule: if $\exists id, name\textsubscript{1}, name\textsubscript{2}, address. Person\textsubscript{2}(id, name\textsubscript{1}, name\textsubscript{2}, address) \land name\textsubscript{1} \neq name\textsubscript{2}$, then mark name\textsubscript{1} as possibly erroneous.

### 4.2.3 Process Steps

As for the GODB process, each step is described in detail below by its input, output, and a brief description.

**Step 8.3.1 Identify Risk Factors.**

\textbf{Input:} The initial conceptual schema; a list of DQ related design issues.

\textbf{Output:} A list of risk factors for each design issue.

\textbf{Description:} To identify how quality of data could be compromised, one way is to understand the nature of data values, such as their types (e.g., numeric vs. non-numeric, primary vs. derived), domains (e.g., standardized vs. non-standardized, bounded vs. unbounded), distributions, and if available the types and likelihood of errors associated with them.

Another way to identify risk factors is to analyze various activities that produce data. For example, in a data acquisition activity, an observer makes an observation which may be recorded; note that (i) the observer and recorder may have their own goals that may affect the objectivity of their respective tasks, and (ii) various instruments that are used during the process (e.g., observation instrument, recording media) may have certain limitations (or biases) due to their intrinsic properties (e.g., number of significant digits, error margin) or environment factors (e.g., time, location, altitude).

**Step 8.3.2 Identify and Select Mitigation Plans.**

\textbf{Input:} A list of risk factors from previous step.

\textbf{Output:} A list of mitigation plans for each risk factor.
Description: For each risk factor identified above, one or more mitigation plans can be identified to either (a) reduce the likelihood of occurrence of the risk factor, or (b) reduce its impact on the quality of data. It is not always possible or desirable to simply integrate all identified mitigation plans into the core business activities, due to following reasons:

- For any DQ problem, the levels of tolerance may vary depending on the type of application or type of data. For example, a 30-minute delay in stock price is more critical to a stock trading system than to market analysis application. Likewise, for a student registration system the accuracy of academic history data is more important than that of demographical data.

- Different risk factors may have different likelihoods of occurrence.

- A mitigation plan may have positive contributions to some design issue and negative ones to others; moreover, design issues may conflict with one another.

Similar techniques for evaluating plans used in the GODB process (Step 5) can be used here.

Step 8.3.3 Define (Select) and Apply Design Operations.

Input: A list of selected mitigation plans.

Output: Final conceptual schema, a list of QA activities it supports.

Description: Selected mitigation plans model QA activities to be performed in addition to the core business activities. As mentioned before, QA activities may involve manual work and/or rely on DQ techniques to assess, improve and monitor quality of data. In either case, these QA activities are analyzed to identify QA data requirements, which are used to modify the initial conceptual schema.
4.3 ExpDB Case Study

4.3.1 Application Background

Employees of the particular organization travel to various cities in different countries to participate in various meetings. A travel expense management system helps to monitor the spending on meetings in order to estimate the meeting budget for following fiscal year. The system generates monthly expense reports for each employee and meeting, and issues cheques to employees with the correct amount. The system has to maintain accurate expense information.

The case study concerns the design of the database component for the travel expense management system (called ExpDB thereafter), with and without the accuracy concern, to illustrate the various schema design operations that operationalize the accuracy softgoal. This case study is adopted from [47] (Chapter 6). A portion of the goal model for ExpDB is shown in Figure 4.1.

![Figure 4.1: a portion of the goal model for ExpDB](image)

The employees will be reimbursed for both the travel related cost and meeting registration fee. The (simplified) reimbursement process works as follows. The employee submit an expense summary form based on the original copies of all the expense vouchers. A secretary checks the information in summary form (possibly with the original vouchers) and enters it into the system. The system then issues a check with appropriate amount.

The conceptual schema shown in Figure 4.2 allows one to record the fact the an employee
submitted an expense summary amount occurred at a certain time, and the fact that the expense resulted from attending a particular meeting. It thus supports the execution of all the plans (i.e., $P1.1.1$, $P1.2.1$, $P1.2.2$ and $P1.3.1$), and therefore the fulfillment of the top-level goal $G1$. However, we do not have any “built-in” support for Softgoal $SG1$ (i.e., to maintain accurate expense summary data). In other words, this schema design does not increase our degree of confidence in the accuracy of expense data stored in the database.

![Initial conceptual schema for ExpDB](image)

**Figure 4.2: the initial conceptual schema for ExpDB**

### 4.3.2 The Running Example

**Step 8.3.1 Identify risk factors.**

In this example, the Softgoal $SG1$ leads to a design issue (i.e., accuracy) that is associated with two attributes of the entity $ExpenseSummary$. To identify risk factors, first these two attributes are analyzed. For example, $ExpenseSummary.amount$ models the monetary amount of the total expense; it has numeric and derived data values, which are associated with reasonable semantic bounds. Moreover, it is reasonable to assume that $ExpenseSummary.date$ is always syntactically valid (i.e., there is no syntactic errors in date values).

Next, data activities are analyzed. For example, the employee plays both the roles of data observer and recorder, with the goal of maximizing expense reimbursement. She (a) “observes” various expenses concerning meeting related events, such as airline ticket purchase, hotel booking and meeting registration, (b) calculates the total amounts during the reimbursement request event, and (c) “records” detailed expenses on the original vouchers (or a separate piece of paper), and expense summary on the reimbursement request form. The secretary plays the role
of data enterer whose main concern is to finish assigned tasks in an efficient and correct way. Therefore, she (d) usually enters the meeting summary data in a batch mode. All these factors have the potential to reduce the quality of the observation being finally stored in the database.

Based on the analysis of these two attributes and their acquisition process, the risk factors are identified. Table 4.2 list a few examples of these risk factors, some of which are further explained below.

Table 4.2: Examples of risk factors for ExpDB

<table>
<thead>
<tr>
<th>During observation time:</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: The employee considers expenses that do not result directly from the meeting (e.g., visiting a nearby place or friend, before or after the meeting)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>During recording and manipulation time:</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2: The employee miscalculates the total amount of the expense summary</td>
</tr>
<tr>
<td>R3: The employee fills in incorrect summary data in the request form</td>
</tr>
<tr>
<td>R4: The employee makes a reimbursement request long time after the trip and the original expense vouchers are lost</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>During data entry time:</th>
</tr>
</thead>
<tbody>
<tr>
<td>R5: The secretary enters summary data incorrectly</td>
</tr>
</tbody>
</table>

During observation time, because the ultimate goal of the employee is to maximize meeting expense reimbursement (which conflicts with one of the softgoals of ExpDB: accurate estimation of meeting budget for the next year fiscal year), the employee may report expenses that do not result directly from the meeting (R1). During recording and manipulation time, since the meeting expense summary data is derived from expense detail data, there is a potential that the calculation may be wrong (R2). During data entry time, typographical errors are the most common sources of data defects in the database (R5). This is especially true when the secretary’s private goal (i.e., efficiency) conflicts with the accuracy softgoal.

**Step 8.3.2 Identify and Select Mitigation Plans.**

Table 4.3 shows a few mitigation plans that are defined for the risk factors identified in the previous step.

Note that MP1, MP2, MP3 and MP5 can be performed either manually by the manager or secretary before data entry, or automatically by the system at data entry time (or periodically).
Table 4.3: Examples of mitigation plans for ExpDB

<table>
<thead>
<tr>
<th>Mitigation Plan</th>
<th>Risk Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP1</td>
<td>R1</td>
</tr>
<tr>
<td>MP2</td>
<td>R2 - R4 (performed manually), R2 - R5 (performed automatically)</td>
</tr>
<tr>
<td>MP3</td>
<td>R4</td>
</tr>
<tr>
<td>MP4</td>
<td>R5</td>
</tr>
<tr>
<td>MP5</td>
<td>R1 - R5</td>
</tr>
</tbody>
</table>

In most the cases, the automatic versions of these plans require additional data to be maintained by ExpDB. For example, for MP2, it is necessary to enter both the meeting expense summary and expense voucher data into ExpDB in order to perform the automatic verification of consistency between these two. Moreover, the manual versions of these plans may also lead to additional data requirements. The correspondence between mitigation plans and the associated risk factors are shown in Table 4.4.

Table 4.4: Correspondence between mitigation plans and risk factors

Step 8.3.3 Define (Select) and Apply Design Operations.

For demonstration purpose, we will discuss design operations for all mitigation plans MP1 - MP5; in practice, only select ones need to be considered. Figure 4.3 shows a portion of a new goal model (rooted at Goal G1.1), in which a new plan P1.1.2 is created by merging P1.1.1 with MP1 (manual), MP2 (manual), MP3 (automatic), MP4 and MP5 (manual). Plan P1.1.2 provides an alternative way to achieve Goal G1.1 with better attention to accuracy.

The improved reimbursement process can be described as follows: the employee collects the vouchers of expenses related directly to the meeting, and fills in a reimbursement request
form with a summary of all the expenses. The employee then submits the form and all expense vouchers to the secretary who first checks (a) if any expense voucher date is within the meeting date \( \pm \) one day, and (b) if meeting expense summary is the sum of all the expense voucher amounts. If no error is found, the secretary is then responsible for entering the form into the system at least twice. The system finally generates an expense report at a specified time in a particular format, and issue reimbursement cheques accordingly. The manager occasional goes through a selected sample set of the expense summary data newly entered into the database to identify suspicious expense patterns.

This new process requires new entities, relationships and attributes to be included in the conceptual schema, in addition to those shown in Figure 4.2. At minimum, a verification mechanism requires that the employees report meeting expenses in both the summarized (using the attribute \textit{amount\ of\ Expense\ Summary}) and breakdown form (using the attributes \textit{registration\ fee\ and\ travel\ cost\ of\ Meeting\ Expense}). This allows the system or a third party to verify the accuracy of summarized data in terms of the breakdown data. Figure 4.4 shows this design operation.

Another way to increase the degree of confidence in accuracy, while avoiding redundant
data, is validation at data entry time. In this case, the employees are required to not only submit the expense summary form, but also make available the original expense vouchers supporting the reimbursement request. The secretary validates the information in summary form using the original vouchers before entering it into the database. It requires the secretary to record the validation event along with the date being audited (using the attributes *signature* and *date* of the relationship *validate*). Figure 4.5 shows this design operation.

A more “aggressive” approach is to have a manager periodically go through a selected sub-set of the expense summary data in the system to identify suspicious expense patterns, possibly with reference to the original expense vouchers. Figure 4.6 shows this design operation. The attributes *adjusted* and *adjustment date* help to trace adjustment to erroneous expense summary data detected during audit; the attribute *last audited date* helps the manager to select the sample population.

The above examples depict generic design operations that can be used to address the accuracy issue. There are also application-specific operations. For example, we can impose the constraint that any meeting expense needs to occur within the period of the meeting. This
Figure 4.6: the conceptual schema for ExpDB with audit by manager

requires adding \textit{begin} and \textit{end} attributes to both Meeting and Expense Summary. As another example, the attribute \textit{budget} can be added to the entity employee. If the meeting expense is within budget, no further actions are taken; otherwise, the manager needs to sign off on the expense summary before it can be reimbursed.
Chapter 5

Data Quality Rule Design

The DQD process described in Chapter 4 concerns the GODB process for addressing DQ softgoals; it focuses on the structural aspect of conceptual database design with DQ as one of the design criteria. The output of the DQD process is a set of DQD proposals, each of which consists of a schema and a set of QA activities supported by the schema.

These QA activities may involve manual work, and/or rely on some automatic DQ techniques. An important type of DQ techniques relies on the specification and enforcement of DQ rules. This chapter concerns the constraint aspect of conceptual database design, with focus on DQ rules for error detection.

This chapter is organized as follows. Section 5.1 proposes a classification scheme for DQ rules, based on a number of important domain independent properties; this contributes to building of a repository that facilitates accumulation of DQ rules over time. Section 5.2 examines one of these properties, effectiveness, in depth, and presents a quantitative framework for measuring and comparing DQ rules according to their effectiveness. Such comparison relies on the derivation of effectiveness formulas by making probabilistic assumptions about the occurrence of errors in data values and other special events. Manual derivation is an non-trivial and error prone process. Section 5.3 therefore proposes a semi-automatic approach to derive effectiveness formulas of DQ rules.
5.1 Towards a Repository of DQ Rules

5.1.1 Introduction

Integrity constraints were introduced in databases for the purposes of capturing data semantics. In the form of dependency theory, they play a role in relational schema design. As constraints that are checked at run time, they play an essential role in data cleaning [63], the activity of detecting and removing errors and inconsistencies. On the practical side, data quality assessment relies heavily on a large number of DQ rules (or simply rules) that may or may not be expressed as integrity constraints [8].

In any case, DQ rules, defined according to domain and application specific business rules, are meant to preserve data consistency and accuracy. Deriving a complete set of rules that accurately reflects an organization’s policies and domain semantics is thus a primary task in improving data quality [44]. The specification of DQ rules however cannot be automated, but requires the involvement of domain experts [120]. Moreover, a comprehensive catalog of DQ rule (types) can only be accumulated in practice over a long period of time.

Nevertheless, part of the above knowledge is domain and application independent, and could therefore be isolated from domain knowledge. This section aims at providing guidance to DQ rule designers in two ways: (i) to facilitate reuse of domain and application independent knowledge in rule design (this is similar to software design patterns [69] for software design), and (ii) to assist rule designers in acquiring new rules given existing ones.

The first goal is achieved by providing a classification of DQ rules according to their domain and application independent properties; this allows to describe a DQ rule at a certain level of abstraction. The second is achieved by exploring several directions in the classification hierarchy. To this end, this section contributes to building of a repository that facilitates accumulation of DQ rules over time.
5.1.2 Classification of DQ Rules

In the most general term, we take a computer-enforceable DQ rule to have the form, \( if P, then Q \), where \( P \) is called the rule’s antecedent and \( Q \) is its consequent. In what follows, a classification scheme is proposed with respect to this general form but we will focus later on a particular class of DQ rules.

Each DQ rule can be described by a variety of facets, including an (i) identifier, (ii) textual description (such as name, informal description, formal definition), (iii) origin (such as motivation, purpose, e.g., types of errors it assesses, and the business rule it originated from), (iv) abstraction level (e.g., location in the generalization hierarchy), and (v) a set of properties (which provides a classification of DQ rules).

Value-based, Aggregation/distribution-based and Probabilistic Patterns.

A DQ rule essentially specifies a pattern on data values, trying to distinguish between valid and unacceptable (combinations) of values. At the top level of this classification scheme, one can distinguish between value-based and aggregation/distribution-based rules.

Value-based rules involve specific data values only. These include (i) patterns within a single data value, such as regular expressions, (ii) patterns on entire individual values, such as domain constraints, and (iii) patterns involving multiple data values, such as data dependencies [62].

It is interesting to notice that data dependencies were traditionally considered as a way to improve the quality of schema, while recently there has been renewed interest in data dependencies for improve the quality of data [63].

Aggregation/distribution-based rules involve aggregations and/or distributions of data values. For example, a FD \( X \rightarrow Y \) can be transformed into a finiteness dependency \( X \overset{fin}{\rightarrow} Y \) which states that for each fixed valuation of the variables in \( X \), there are only finitely many values of the variables in \( Y \) [113]. Indeed, most cardinality constraints specified on a ER schema requires aggregations of data values (e.g., a course meets three times a week). As
another example, a university rule may states that in any undergraduate class, there should be fewer than 10% of the students who get A or above, and fewer than 5% of the students who get an F.

In many situations, patterns cannot be defined in a crisp manner, meaning that they involve a certain degree of uncertainty. Therefore, we may talk about probabilistic value-based patterns and probabilistic aggregate-data-based patterns.

**Error Detection Mechanism.**

One way to further classify DQ rules, is to understand the mechanism they use for error detection. In general, just like software testing, which is a task to detect defects in software by contrasting a computer program’s results actual with its expected results for a given set of inputs, error detection in databases is based on comparison of data values actually stored with what are expected. Different error detection mechanisms use different ways to specify this expectation. In what follows, the main error detection mechanism are discussed. A DQ rule may use more than one type of mechanism.

**Domain theory constraints on data values.** Many constraints are initially derived from domain analysis, and mapping from conceptual models, such as ER diagrams. These include key constraints for identifying objects, cardinality constraints (including so-called Functional Dependencies) on relationships, inclusion dependencies from subclass hierarchies, and domain knowledge such as the fact that `terminationDate` must follow `hiringDate`. For the purposes of the present classification, we syntactically characterize them as those expressible in SQL without the use of constants — just constraints among values.

**Constraints due to introduction of redundancy.** When transmitting text, one could check the correctness of the transmission by verifying that the message received satisfies the grammatical rules of the language – these are what we called above “domain theory constraints”. In electrical engineering, one in fact introduces redundancy (e.g., parity checks, packet retransmission)
in order to increase confidence in the correctness of the message received.

Similarly, an organization may have or can choose to explicitly add redundancy in its data gathering and entry process for the sole/main purpose of error detection and correction.

The simplest example of this would be a duplicated attribute. Such a technique is common in practice for several reasons [8]: (i) they are widespread in legacy databases, (ii) they are used to improve efficiency of data access, (iii) some data across different systems is invariably redundant, (iv) the data can be naturally gathered at different stages of a workflow. A common example of case (iii) is a situation where current values of some attributes are stored in a separate relation in addition to being present in the most recent record of a historical data stack [8]. For example, original date of hire \((\text{hireDate})\) for an employee \(\text{EmployeeInfo}(\text{empID, hireDate, termDate})\) can be also found as the effective date \((\text{effDate})\) of the earliest record in the employee history events relation \(\text{EmpStatusHistory}(\text{empID, effDate, actionCode, stateCode})\).

In other cases, redundancy among attributes is partial. Intuitively, partial redundancy exists between two attributes when knowing the values of one set of attributes restricts the possible values of another. As mentioned before, partial dependency among two sets of attributes can be calculated using the Shannon entropy function [52, 42, 107]. An extension of CFDs, called eCFDs [30], is an example of partial redundancy. For example, the eCFD \(CT = \text{NYC} \rightarrow AC \in \{212, 718, 646, 347, 917\}\) asserts that when the city \((CT)\) is NYC, the area code \((AC)\) must be either 212, 718, 646, 347, or 917. (Note that if this is going to be a redundancy constraint, area codes would be added not because they are needed/useful for the application on their own, but because they help detect errors.)

**Constraints due to use of metadata.** Another type of error detection mechanism relies on the use of metadata, which may be specified during requirements analysis and conceptual database design (see Section 4.2.1 for a list of QA data requirements and their discovery, and Section 4.2.2 and 4.2.3 for the design process), or discovered using data profiling [8]. For the purpose of error detection, the following types of metadata are particular useful.
**Restrictive Metadata.** This type of metadata specifies a restriction on the admissible values of one or more attributes. In its simplest form, such restrictions can be placed on individual attributes, e.g., *domain restrictions*. Restrictions may be also placed on aggregation over the values of an attribute, which are effective ways of detecting not only inaccurate but also missing data. For example, if a company expects the new orders to be around 20,000 each month, a considerable deviance below this level suggests that some data on new orders may be missing.

Data profiling is the first step towards better data quality. Essentially, it is a process to gather statistics and distributions of attribute values. Therefore, data profiling can be considered as a process to gather restrictive metadata. However, due to potential errors in data, these statistics themselves may not be 100% accurate. It is therefore important to realize that restrictive metadata collected using data profiling tools does not offer the final answer to the problems of DQ assessment, but instead provides an opportunity to find the right questions to ask. Answers to these questions will lead to the discovery of DQ rules (based on correct restrictive metadata) to assess the data.

**Descriptive Metadata.** This type of metadata provides additional information on the values of some attribute. One type of descriptive metadata is provenance information on how data values are created/collection (see [85] for examples).

For example, consider the schema $\text{Patient}(\text{name, temperature, time})$ which is used to record body temperatures for patients measured at different time points in a hospital. Information on how temperature values were measured includes who did the measurement (by a registered nurse vs. a student nurse), with what instrument (digital thermometer vs. mercurial thermometer), and under what conditions (with or without taking antipyretic). This type of information provides context within which data values should be assessed.

It is worthwhile to notice that descriptive metadata for an attribute can be easily identified from the description of senses of that attribute. For example, consider the attribute *temperature*.

---

from the previous example, its descriptive metadata include nurse, instrument and condition, which are components of the specification of its senses (See Section 4.1.2 for details).

**Constraints due to use of historical data.** Integrity constraints can be classified into static and dynamic constraints. Unlike static constraints, dynamic ones cannot be checked by solely inspecting the most recent state of a database. One example of dynamic constraint is “employees’ salary never decrease”. One way to turn a dynamic constraint to a static one is to extend every database state with auxiliary relations that contain the historical information necessary for checking the constraint [45].

On the practical side, historical data comprises the majority of data in both operational systems and data warehouses [8]. For a time-dependent attribute (e.g., height, salary), its value history is usually stored using three attributes: an object identifier, a timestamp indicating the measurement date, and the time-dependent attribute itself. For example, a salary history for employees of a company could be stored using a table containing employee id, effective date and salary amount. This allows to treat constraints such as “employees’ salary never decrease” as static constraints, since their enforcement requires only the current state of the database.

In general, historical information of an attribute consists another error detection mechanism. Several types of DQ rules which make use of historical data have been discussed in [8], including timeline pattern rule and value pattern rule; these rules are motivated in DQ assessment in practice. For example, “employees’ salary never decrease” is an example of value pattern rule which restricts the direction of change in attribute values; a slightly more complex rule also restricts the magnitude of value changes (e.g., height changes of a person must be restricted to six inches per year).

**Design Dimensions of DQ Rules.**

Orthogonal to error detection mechanisms, another way to classify DQ rules is to examine a set of dimensions along which the rules may vary. These design dimensions provide an important
vehicle for assisting DQ rule designers (see Section 5.1.3). In what follows, we discuss several important design dimensions.

**Context.** Many DQ rules are known to be contextual in the sense that they hold only over a portion of the data being assessed [44]. Contextual rules have received much attention recently due to the practical needs for data cleaning.

Context can be specified by individual values, such as the constants in a pattern tableau for a CFD [26] (e.g., $CC, AC, PN \rightarrow STR, CT, ZIP, \{(01, 908, _ \parallel _), MH, _\})$ and for a CIND [31] (e.g., $cd[album, price; genre] \subseteq book[title, price; format], \{(book||audio)\}$, i.e., every cd of genre book must also appear as a book with format audio).

Context can also be specified by a (complement of a) set of values as in the antecedent of an eCFD [30] (e.g., $CT \rightarrow AC, \{(S \parallel _)\}$ where $S = \{NYC, LI\}$), or by an arbitrary constraint as in a constrained FD $c \Rightarrow X \rightarrow Y$ [113, 20], which states that the FD $X \rightarrow Y$ holds on the subset of data which satisfies the constraint $c$ [29] (e.g., $year > 2 \Rightarrow courseId \rightarrow Instructor$).

FDs and INDs are not the only types of rules that can have contextual counterparts. Take NOT NULL constraint for example. Consider the relation schema $Student(name, toefl, status)$ which records the information about foreign students registered at some university. Whether or not $toefl$ is allowed to take null value is determined by the values of $status$: ESL students may have not taken TOEFL test, while students who are taking university level courses are required to have a TOEFL score (above certain threshold). As another example, consider the relation schema $Emp(type, annualSalaryRate, scheduledHours, hourlyPayRate)$. The type of an employee (i.e., full time or part time) determines whether other attributes could take null values. For example, if an employee works full time, attribute $annualSalaryRate$ would be required while $scheduledHours$ and $hourlyPayRate$ would be prevented.

**Tolerance.** A DQ rule is more tolerant than the other when it allows a larger variety of values to satisfy or violate it. Tolerance can be specified with respect to the consequent, antecedent of a rule, or both.
**Tolerance in the Consequent.** Tolerance in the consequent measures variability in admissible data values (of one or more attributes) in the consequent of the rule. In the extreme case, a rule has zero tolerance when it only allows a single value in its consequent (e.g., FDs and CFDs are zero-tolerance rules). More generally, the consequent of a rule may allow a set or range of values [30].

The concept of tolerance in the consequent also appeared in the study of metric FDs (MFDs) [103]. A MFD $X \xrightarrow{\delta} Y$ holds if

$$\max_{T \in \pi_X} \Delta_d(T[Y]) \leq \delta$$

where $\pi_X$ is a set of equivalence classes each of which contains all tuples that share the same value in $X$, $d$ is a metric function defined on the domain of $Y$, and $\Delta_d(S) = \max_{p,q \in S} d(p,q)$ measures the diameter of a set of points $S$ in a metric space [103]. For example, a MFD may state that “same movie should have almost the same duration recorded in different data source”.

**Tolerance in the Antecedent.** The antecedent of a rule may also exhibit different tolerance. For example, a matching dependency [63] may state that “if the last name, phone number values of two tuples are the same and their first name values are similar, then these tuples refer to the same person in the real world”.

**Tolerance in both.** A fuzzy FD [134] may specify tolerance in both the antecedent and consequent using fuzzy sets. Fuzzy sets are sets whose elements have degrees of membership between 0 and 1, rather than just 0 or 1. For example, the fuzzy FD “Highly qualified employees should have high salaries” uses two fuzzy sets highly qualified employees and high salaries; the latter includes $80,000 as one of its members to the degree of 0.95.

**Uncertainty.** There are at least two types of uncertainty regarding DQ rules which have been extensively investigated.
Unconditional Uncertainty. In practice, we often encounter situations where a rule (as a whole) almost holds, allowing some exceptions. For example, in a company database, “years of experience determines salary level” may hold in all but a few exceptional cases. Unconditional uncertainty of a DQ rule measures the expected degree to which the rule can be violated due to exceptions. In symbols, if we use $d$ to represent the unconditional uncertainty of a DQ rule “if $P$, then $Q$”, we have $pr(Q \rightarrow P) = 1 - d$.

Unconditional uncertainty has been investigated in approximate functional dependencies (AFDs) [70, 60, 102], approximate order dependencies [58], as well as partial functional dependencies [23]. A common approximation measure for an AFD is the fraction of exceptional tuples that must be deleted for the rule to hold [102]. A similar measure, support, has been defined for association rules; it measures the fraction of transactions in a dataset that satisfy both the consequent and antecedent of an association rule [2]. In this case, we have $support = pr(Q \land P)$.

Conditional Uncertainty. In many cases, it is also useful to talk about the “strength” of a rule, i.e., the likelihood that the consequent of a rule is true given the antecedent is true. This is similar to confidence, a strength measure for association rules (i.e., the fraction of transactions in a dataset that satisfy the consequent given that they also satisfy the antecedent of an association rule).

This type of uncertainty has been studied in [129, 104] in the form of partial determination. A partial determination is a generalized FD: $X \xrightarrow{d} Y$, where $d$ is the determination factor; it states that the probability that two randomly chosen tuples have the same values of $Y$, given they have the same values of $X$ [129] is $d$. That is if $s$ and $t$ are two randomly chosen tuples, partial determination of the above form states that $pr(s[Y] = t[Y]|s[X] = t[X]) = d$.

Conditional uncertainty has also been investigated in a more general setting in information dependency [52, 42, 107]. Let $X$ and $Y$ be sets of attributes in the schema. The information dependency (InD) measure, $H_{X \rightarrow Y}$, was proposed to answer the question “how much do we
not know about values in $Y$ provided we know the corresponding values in $X$?[52]. It can be calculated using the lemma $H_{X \rightarrow Y} = H_{XY} - H_X$, where $H_R = \sum_{i=1}^{n} p_i \log 1/p_i$ is the Shannon entropy function defined on the set of attributes $R$, whose values are drawn from a finite set $\{v_1, \ldots, v_n\}$, and $p_i = \text{pr}(R = v_i)$.

These two types of uncertainty are summarized in Table 5.1:

<table>
<thead>
<tr>
<th>Antecedent</th>
<th>Consequent</th>
<th>expected number of tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>$n_1$</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>$n_2$</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>$n_3$</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>$n_4$</td>
</tr>
<tr>
<td>Unconditional: $(n_1 + n_3 + n_4)/(n_1 + n_2 + n_3 + n_4)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditional: $n_1/(n_1 + n_2)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Performance.** In the context of error detection, DQ rules vary in performance in two ways.

**Precision.** A DQ rule is more precise than another, with respect to error detection, when its violation involves less amount of data. Precision is partially affected by the number of tuples involved in a rule violation. For example, the CFD $(CC, AC \rightarrow CT, \{(01, 215||PHI)\})$, which states that in the USA, if the area code is “215”, then the city must by “PHI”, is more precise than the standard FD $[CC, AC] \rightarrow [CT]$. This is because a single tuple can violate the first rule, while a violation of the second rule involves at least two tuples. A multivalued dependency [62] may involve more than two tuples. More generally, constraint-generating $k$-dependency refers to a constraint-generating dependency whose violation involves $k$ tuples [20].

Precision is also partially affected by the number of attribute involved in a rule violation. For example, a violation of the CFD $(CC, AC \rightarrow CT, \{(01, 215||PHI)\})$ involves three attributes $CC, AC$ and $CT$. 
Effectiveness. To consider effectiveness of a DQ rule \( if P then Q \) we need to make explicit another component of the rule, which represents the assessment result one can obtain from a violation of the rule. The effectiveness of a DQ rule measures the closeness between the assessment results according to the rule with the correct assessment results.

For example, consider the CFD \( (CC, AC \rightarrow CT, \{(01, 215\|PHI)\}) \) again. The effectiveness of this rule is 100% if we conclude that the \([CC, AC, CT]\) taken as a whole is erroneous whenever the rule is violated. On the other hand, the effectiveness is likely to be less than 100% if we always blame \([CT]\) whenever the rule is violated. For more details of effectiveness, please refer to Section 5.2 where we have proposed a semi-automatic approach for deriving effectiveness formulas that generate estimated effectiveness scores for a large range of DQ rules.

A Taxonomy of DQ Rules.

To summarize, Figure 5.1 presents a taxonomy of rules for DQ assessments. Recall that the basic action in DQ assessment is comparing data actually stored in a database with what is expected. This taxonomy therefore classifies DQ rules from following three viewpoints: (i) with respect to what is to be assessed (Figure 5.1(a)), (ii) with respect to what is expected (Figure 5.1(b)), and (iii) with respect to how to compare these two (Figure 5.1(c)). Notice categories are not necessary disjoint.

For example, a CFD is a deterministic, hard, and contextual DQ rule between values that relies on full redundancy; an approximate FD is different from a CFD because it is probabilistic and universal, while a fuzzy FD is soft. On the other hand, a regular expression is a deterministic, hard, and universal DQ rule within individual values that relies on restrictive metadata.

5.1.3 DQ Rules Design Guidelines

Starting with a small set of existing rules, a designer may acquire new ones by following design guidelines. Since a DQ rule may be described at different level of abstraction, design guidelines
Chapter 5. Data Quality Rule Design

(a) What is to be assessed?

(b) What is expected?

(c) How to compare?

Figure 5.1: A Taxonomy of DQ Rules.
may be offered by considering a generalization/specialization hierarchy. For example, the rule: *employees’ salary always increases* may be specialized into *employees’ salary increases by 1% to 3% on an annual basis*, in the sense that any data that satisfies the specialized rule also satisfies the general one.

Design guidelines may also exist in the form of instantiation. For example, the rule: *in US, applicable sales tax rate is $t$ if year is $y$ and county is $c$. is abstract because it contains variables to be instantiated; an instantiation of this rule is *in US, applicable sales tax rate is 6.95% if year is 1999 and county is Harkin.***

**Transformation Primitives.**

Design guidelines may also be offered with respect to a classification, such as the one discussed in Section 5.1.2. In particular, a designer may perform *transformations* among DQ rules within a single dimension or between two dimensions. Such a transformation consists of one or more applications of the following primitives (note performance, i.e., precision and effectiveness, of a DQ rule are normally not changed directly, but instead may be affected by changes in other dimensions):

- within context dimension
  - add context
  - remove context

- within tolerance dimension
  - increase tolerance
  - decrease tolerance

- within uncertainty dimension
  - increase uncertainty
– decrease uncertainty

For example, the DQ rule: *applicable sales tax rate is 6.95% if year is 1999* can be transformed into *in US, applicable sales tax rate is 6.95% if year is 1999 and county is Harkin* by scoping down context (i.e., restricting county to Harkin). As another example, the tolerance level of the rule: *total salary of an employee must lie in a range defined by (minimum pay rate, maximum pay rate) and pay period* can be adjusted by changing the metadata used (i.e., *minimum pay rate and maximum pay rate*).

Often the cases in one design dimension may affect other ones. Therefore, two or more primitives may need to be applied at the same time to complete a transformation. For example, it has been argued that tolerance parameter ($\delta$) and approximation parameter ($\epsilon$) move in the opposite direction [103]: when a higher tolerance level is chosen, the expected number of exceptional cases is often reduced, while decreasing tolerance level often leads to more exceptions. As a concrete example, consider the DQ rule: *95% of people ($\epsilon = 1 - 0.95 = 0.05$) lives less than 100 years ($\delta = 100 - 0 = 100$).* Natural transformations of this rule includes: ($\epsilon = 0.01, \delta = 120$), ($\epsilon = 0.20, \delta = 80$), ($\epsilon = 0.99, \delta = 0$).

These primitives are further illustrated with more examples in the next section.

**Examples of DQ Rule Transformations.**

**Domain Restriction Rules.** At the most atomic level, data in a database consists of individual values of various attributes. Values of an attribute often cannot take arbitrary values, but only certain reasonable ones. Restrictions on attribute domains are normally specified using metadata (that either come with the schema or are discovered using data profiling).

One of the most common restriction on attribute values is the NOT NULL constraint, such as the following DQ rule:

- *Each international student has a TOEFL score.*

Sometimes, a NOT NULL constraint only holds for a certain portion of data; the following
is a revised version of the above rule, obtained by scoping down its context.

- Each international student who is taking or has taken some university level course has a TOEFL score. (add context)

  Similarly, the following two rules enforce a NOT NULL constraint for a subset of the data.

- When the status of an order is shipped, a tracking number must be generated for the order. (add context)

- When the status of a payment is received, a confirmation number must be stored for the payment. (add context)

  More often, value restrictions are specified using one (or more) range / set of (in)valid values, as shown in the following DQ rules:

- The duration of each patient visit must be within 2 hours.

- The age of each person must between 0 and 150.

- Year-end bonus payments are always made in December and January.

- Every student is either a graduate or undergraduate student.

- Employee status is one of \{Active, On Leave, Retired, Terminated, Deceased\}.

  This makes it possible to fine tune a rule of this type by increasing or decreasing its tolerance level, as shown below:

- The duration of each patient visit must be within 2.5 hours. (increase tolerance).

- The age of each person must between 0 and 100. (decrease tolerance).

- Every student is either a graduate or undergraduate student or special student. (increase tolerance).
Moreover, the level of tolerance is closely related to the level of uncertainty.

- 85% percent of the time, the duration of a patient visit lasts less than 1.5 hours. (decrease tolerance, increase uncertainty)

Existence Rules. Existence Rules assert that certain data must exist in a database. One use of existence rules is to enforce (pre-/post-) conditions or workflows for events, dependencies (e.g., cause-and-effect) among events, as well as object state transitions, as shown in the following example:

- A laser vision correction surgery is always preceded by a consultation visit.

- An employee who is currently on leave must be active before.

To allow exceptions, these rules could be modified by increasing their uncertainty level. For example, the rule concerning vision correction surgery may be revised to

- 99% of the time, a laser vision correction surgery is preceded by a consultation visit. (increase uncertainty)

Functional Dependencies. As discussed in Section 5.1.2, recently research has been carried out to extend FDs along several of the design dimensions. For example, consider the FD:

- Area code uniquely determines city.

which is not always true in the real world. An attempt to “fix” this FD is to add context as in

- In state of New York of US, area code uniquely determines city. (add context)

However, an examination of New York area codes reveals that most cities in the state have a unique area code, except for the City of New York and Long Island. Therefore, we need to further add the context as in

- In state of New York of US (except for the City of New York and Long Island), area code uniquely determines city. (add context)
To increase the tolerance level of a FD, there are at least two ways: transforming it into a MFD using a metric, and transforming it into a eCFD using disjunction. As an example of the first case, consider following the FD

- **ISBN of a book uniquely determines its number of pages.**

  Which is almost always false when we consider multiple data sources on the Web (e.g., Google Books v.s. Amazon). However, it can be transformed into a MFD by increasing tolerance level in the consequent with a tolerance parameter \( n \), as shown below.

- **When two books have the same ISBN, the difference of their number of pages must be within \( n \). (increase tolerance)**

  As an example of the second case, consider following the FD

- **Area code uniquely determines city.**

  It can be transformed into an eCFD by scoping down its context and increasing its tolerance level (in the consequent) at the same time, as shown below.

- **In state of New York of US, if the City is New York, the area code must be one of 212, 718, 646, 347, 917. (scoping down context, increase tolerance)**

**Order dependencies.** Another type of dependency among data values that often arises in practice is order dependency. Order dependencies often occur for time-dependent data, such as

- **Height of a person does not decrease in time.**

- **Total copies of book sold must be non-decreasing.**

  Order dependencies also occur among transactional data, such as

- **Larger sales receipt number implies later purchase date and time.**

- **Larger product serial number implies later manufacture date.**
Moreover, order dependencies may also occur without involving time, such as

- **Employee with higher qualification/rank gets higher salary.**

One way to decrease the tolerance of an order dependency is to pose restrictions on the magnitude of a change in values, in addition to the direction of the change, such as

- **Height of a person increase at most six inches per year. (decrease tolerance)**

- **Pay raise of an employee does not exceed 30%. (decrease the tolerance)**

As with other types of DQ rules, an order dependency can be modified to also incorporate context and uncertainty. For example, the previous rule regarding pay raise can be transformed into:

- **Pay raise of an employee who has been promoted to a new position is between 20% to 30%. (add context, decrease tolerance)**

- **95% of the cases, pay raise of an employee does not exceed 25%. (increase uncertainty, decrease tolerance)**

**Frequency Rules.** A frequency rule states that an event happens every $n$ time period. For example,

- **Every patient must visit his/her dentist at least every six months for regular checkups.**

- **Every participant in a medical study is required to take blood pressure readings at least once a week.**

Sometimes, event frequency can be defined as a function of other type of data (instead of time), as shown in the following example:

- **An airplane undergoes extensive maintenance every 50 flights or less.**
Often frequency in a rule is related to the tolerance level of the rule. For example, increasing the frequency of the rule regarding medical study decreases its tolerance level in the consequent of the rule, as shown in the following example; this is because a participant who satisfies the original rule may violate the revised one.

- Every participant in a medical study is required to take blood pressure readings at least twice a week. (decrease tolerance)

Cardinal Rules. A cardinal rule enforces a cardinality constraint which asserts that every value $v_1$ is related to at least $m$ and at most $n v_2$, as shown in following examples:

- A doctor cannot see more than 25 patients a day.
- A car collision accident involves two or more cars.
- Score history must include no less than 20 round for a golfer player.

Since $(m, n)$ specifies a range, a natural transformation of this type of rule is to fine tune its tolerance and uncertainty level like for domain restriction rules. For example, the rule regarding doctor seeing patients could be transformed into

- 80% of the time, a doctor cannot see more than 20 patients a day. (decrease tolerance, increase uncertainty)

5.1.4 A Note on Formalizing DQ Rule Transformations

To provide tool support for applying DQ rule transformations, the primitives discussed in Section 5.1.3 need to be formalized. This section discusses key observations and approach towards such formalization.
A logical form of DQ Rules.

Consider a class of DQ rules that can be expressed in the following logical form

\[ H \leftarrow R_1, \ldots, R_m, M_1, \ldots, M_n \]

where \( R \)'s are database relations, \( M \)'s are arithmetic comparison relations\(^2\), and \( H \) is either a database or arithmetic comparison relation.

Without losing generality, we impose the following two conditions to simplify our discussion: (i) every \( R_i \) contains variables only (so selection with constants are expressed using arithmetic equalities), and (ii) variable occurrences must be unique over all \( R \)'s (so each variable uniquely identifies a relation and a column in it). As a result, all real constraints in a rule are expressed in \( M \)'s and \( H \).

Moreover, we require as usual, that rules be “safe” in the sense that every variable appearing in \( H \) or some \( M_j \) must also appear in some \( R_i \). Furthermore, we assume every DQ rule is put into a standard form where: (i) all tautologies (e.g., \( X = X \)), contradictions (\( X > Y \) and \( X < Y \)) and redundancies (e.g., \( X = Z \) given \( X = Y \) and \( Y = Z \)) are removed, and (ii) a variable which appears only in one \( R_i \) (but is absent from any comparison) is treated as an unnamed variables (denoted by \( _{\cdot} \)). DQ rules defined in this way can be used to express a large and useful class of integrity constraints that includes tuple-generating and equality-generating dependencies.

Formalization of Context Change.

Changing the context of a DQ rule amounts to adding to or removing from the rule arithmetic comparison relations. More specifically, adding an arithmetic relation \( M \) to a rule poses an restriction on the possible values of one or more attributes that satisfy the rule, therefore adding context. For example, consider the following rules

---

\(^2\)which are standard conjunctive-query atoms of the form \(<\text{Variable}> <\text{RelOp}> <\text{Variable}|\text{Constant}>\)
Each student must have a TOEFL score

Each international student must have a TOEFL score

Each international student who is not taking ESL level course must have a TOEFL score

which demonstrate adding context of a student’s status and level of the course taken by the student. Conceptually, these transformations correspond to specialization of the concept of “Student” to “International Student” to “ESL International Student”. This can be expressed syntactically using the logical form as:

\[ X \neq \text{NULL} \leftarrow \text{Student}(\text{name} : \_, \text{status} : \_, \text{toefl} : Y, \text{level} : \_). \]

\[ X \neq \text{NULL} \leftarrow \text{Student}(\text{name} : \_, \text{status} : X, \text{toefl} : Y, \text{level} : \_), X = ' \text{int}'. \]

\[ X \neq \text{NULL} \leftarrow \text{Student}(\text{name} : \_, \text{status} : X, \text{toefl} : Y, \text{level} : Z), X = ' \text{int}', Z \neq ' \text{esl}'. \]

Adding context can be carried out in the following steps:

• selecting an unnamed variable from some database relation in the body of the rule,
• replacing it with a new named variable,
• repeating the previous two steps if necessary
• adding an arithmetic comparison relation, which
  • restricts (prohibits) a variable to take (from taking) a specific value (e.g., \( X = a, X \neq b \))
    or range of values (e.g., \( X < c, X \geq d \)), or
  • restricts (prohibits) a function of variable(s) to take (or from taking) a specific value (e.g., \( X + Y = e \)) or range of values (e.g., \( \text{len}(X) + \text{len}(Y) < f \))

In the opposite direction, removing context is realized by selecting and removing an arithmetic comparison relation from the body of the rule, and replacing any named variable no longer involved in any arithmetic comparison relation with an unnamed variable.
Formalization of Tolerance Change.

Changing the tolerance of a DQ rule amounts to modifying arithmetic comparison relations in the rule. More specifically, the tolerance of a rule is increased by weakening the restrictions on the possible values of one or more attributes that satisfy the rule. For example, consider the following rules

- (In different data sources) Movies with the same title have the same recorded duration.
- (In different data sources) Movies with the same title have recorded durations differing no more than 6 minutes.

which demonstrate increase of tolerance on the difference between recorded durations of the same movie in different data sources. Conceptually, this transformation can be achieved by (i) replacing equalities $V = W$ with the equivalent form, $|V - W| = \delta$, and then (ii) relaxing the constraint $\delta = 0$. This can be seen more clearly when these rules are expressed in the logical form:

$$Y_1 = Y_2 \leftarrow MovieDB_1(title : X_1, dur : Y_1), MovieDB_2(title : X_2, dur : Y_2), X_1 = X_2.$$  
$$\delta \leq 6 \leftarrow MovieDB_1(title : X_1, dur : Y_1), MovieDB_2(title : X_2, dur : Y_2), X_1 = X_2.$$  

More generally, for values for which a metric function $d$ can be defined (such as the absolute difference for numbers, and the edit distance for strings), increasing tolerance can be carried out by:

- selecting an arithmetic comparison relation from the body or head of the rule, and,

- transforming the relation into one of the following forms:

  (i) $t_1 \ op t_2$,

  (ii) $d(t_1, t_2) = 0$,

  (iii) $d(t_1, t_2) \ op \ \delta$, 

where $op$ is a comparison operator (e.g., $\leq$, $\geq$, $\neq$, etc.).
(iv) \(d(t_1, t_2) \ op \leq \delta\)

where \(t_x\) is either a variable, a constant or a function involving variable(s) and constant(s), \(op\) is either < or \(\leq\), \(op\) is either > or \(\geq\), and \(\delta_1\) and \(\delta_2\) are positive numbers,

- performing one of the following modifications:

  - if the relation has the form \(t_1 \ op \leq t_2\), replacing it with \(t_1 \ op \leq t_2 + \delta_1\),
  - if the relation has the form \(d(t_1, t_2) = 0\), replacing it with \(d(t_1, t_2) \ leq \delta_1\),
  - if the relation has the form \(d(t_1, t_2) \ op \leq \delta_1\), replacing it with \(d(t_1, t_2) \ op \leq \delta_1 + \delta_2\),
  - if the relation has the form \(d(t_1, t_2) \ op \geq \delta_1\), replacing it with \(d(t_1, t_2) \ op \geq \max(0, \delta_1 - \delta_2)\).

In the opposite direction, tolerance is decreased by strengthening the restrictions on the possible values of one or more attributes that satisfy the rule; it can be realized by reversing the modifications discussed above.

**Formalization of Uncertainty Change.**

In order to talk about uncertainty of a rule \(\phi : H \leftarrow R_1, \ldots, R_m, M_1, \ldots, M_n\), we need to extend it to accommodate the information about the probability that its consequent is false while its antecedent is true, and the probability that its antecedent is false. It is sufficient to write it as a triple \((\phi, p, q)\) where \(p = \text{pr}(\neg H \land R_1 \land \ldots \land R_m \land M_1 \land \ldots \land M_n)\), and \(q = \text{pr}(R_1 \land \ldots \land R_m \land (\neg M_1 \lor \ldots \lor \neg M_n))\).

For example, consider the rule discussed above:

\[\phi_{\text{student}} : X \neq \text{NULL} \leftarrow \text{Student(name : _, status : X, toefl : Y, level : _), } X = '\text{int}'\]

- \((\phi_{\text{student}}, 0, 0.7)\) states that, if there is no error, 30% chance a student is an international student, and each international student must have a TOEFL score;
• \((\phi_{\text{movie}}, 0.05, 0.7)\) states that, if there is no error, 30% chance a student is an international student, and there is a 5% chance that an international student has no TOEFL score.
5.2 Measuring and Comparing Effectiveness of DQ Rules

This section examines in depth one of the important domain-independent properties of DQ rules, namely effectiveness, and presents a quantitative framework for measuring and comparing such effectiveness. The rest of the section is organized as follows. First, the main concepts involved in the measurement of effectiveness of DQ rules are described in Section 5.2.1. Second, section 5.2.2 illustrates manual derivation of effectiveness formulas using two example. Finally, section 5.2.3 shows how DQ rules can be evaluated individually and compared under different scenarios given their effectiveness formulas.

5.2.1 Main Concepts

In what follows, we consider the class of DQ rules of the form

\[ H \leftarrow R_1, \ldots, R_m, M_1, \ldots, M_n \]

as defined in Section 5.1.4, with each \( M \) being restricted to an arithmetic (in)equality. When used for quality assessment, we make it explicit what values are being assessed by writing:

\[ \text{Assessment}(X_1, \ldots, X_p) \leftarrow R_1, \ldots, R_m, M_1, \ldots, M_n, \lnot H \]

where \( \{X_1, \ldots, X_p\} \) is a subset of the variables in the rule being assessed. Notice that DQ rules defined in the way include both equality- and tuple-generating dependencies.

Example 5.1: Consider the customer schema discussed in [26], which specifies a customer in terms of the customer’s country code (CC), area code (AC), phone number (PN), name (NM), and address (street (STR), city (CT), zip code (ZIP)). The following DQ rules can be specified on the schema:
CHAPTER 5. DATA QUALITY RULE DESIGN

\[ \phi_1 \text{, Assessment}_1(W) \leftarrow \text{Customer}(CC:-, AC: -, PH: -, NM: -, STR: -, CT: W, ZIP: -), \]
\[ \neg (W \in L_{\text{city}}). \]

\[ \phi_2 \text{, Assessment}_2(W) \leftarrow \text{Customer}(CC: X, AC: Y, PH: -, NM: -, STR: -, CT: W, ZIP: -), \]
\[ X = '01', Y = '215', \neg (W = 'PHI'). \]

\[ \phi_3 \text{, Assessment}_3(W_1, W_2) \leftarrow \text{Customer}(CC: X_1, AC: Y_1, PH: Z_1, NM: -, STR: -, CT: W_1, ZIP: -), \]
\[ \text{Customer}(CC: X_2, AC: Y_2, PH: Z_2, NM: -, STR: -, CT: W_2, ZIP: -), \]
\[ X_1 = X_2, Y_1 = Y_2, Z_1 = Z_2, \neg (W_1 = W_2). \]

The first rule checks if a city value is in a list \( L_{\text{city}} \) of valid city names; the second rule corresponds to the conditional functional dependency \([26]\),

\[ [CC = 01, AC = 215] \rightarrow [CT = 'PHI'] \]

which assures that in the US if the area code is 215 the city must be 'PHI'; the last rule corresponds to the ordinary functional dependency,

\[ [CC, AC, PN] \rightarrow [CT] \]

which states that customers who have the same country code, area code and phone number must also have the same city value.

\[ \square \]

Effectiveness of DQ Rules.

In principle, a violation of a DQ rule signals that one or more errors exist in the data used to evaluate the rule. When all named variables in a rule are those being assessed, a definite conclusion could be made. For example, a violation of \( \phi_1 \) indicates that a city name must be erroneous (assuming \( L_{\text{city}} \) is free of errors). Otherwise, no definite conclusion could be reached about the whereabouts of the error(s). For example, a violation of \( \phi_2 \) may be caused not only by an erroneous city name, but also by an erroneous country code or areacode (or some combination of them). Moreover, each additional arithmetic comparison relation \( M_j \) introduces new “chances” for the rule to give undesired results. For example, a violation of \( \phi_3 \)
may be caused by erroneous \( Z_1 \) and \( Z_2 \) values, which happen to be the same, while both the \( W_1 \) and \( W_2 \) values contain no error (and therefore are different from each other).

Moreover, more sources of errors are possible if we remove the assumption that a list of valid values (or a mapping table) is always error-free. For example, we may want to consider the cases where \( L_{city} \) contains erroneous and extraneous values, or some valid values are missing from it. Furthermore, potential sources of errors are introduced if we replace equality (or other comparison) condition in an arithmetic relation with a metric condition. For example, we may replace \( W_1 = W_2 \) in \( \phi_3 \) with \( d(W_1, W_2) \leq \delta \), where \( d \) is a metric function defined on attribute \( city \) and \( \delta \) is tolerance parameter \([103]\); in this case, a violation may be triggered due to an inappropriately chosen tolerance parameter \( \delta \).

More generally, if arbitrary functions (e.g., to compute a value based on a set of other values) and/or predicates (e.g., to determine if there is a match between two values) are allowed in an arithmetic relation, the quality of these functions and predicates constitutes another source of possible errors when a violation of a DQ rule occurs.

**Effectiveness Measures.**

To talk about the effectiveness of DQ rules, the first necessary step is to define *effectiveness measures*. *Precision* and *recall* are well-known effectiveness measures, traditionally used to evaluate information retrieval algorithms \([161]\):

\[
\text{precision} = \frac{\text{num}(TP)}{\text{num}(TP) + \text{num}(FP)} \tag{5.1}
\]

\[
\text{recall} = \frac{\text{num}(TP)}{\text{num}(TP) + \text{num}(FN)} \tag{5.2}
\]

where \( \text{num}(TP) \), \( \text{num}(FP) \) and \( \text{num}(FN) \) represent the number of true positives (\( TP \)), false positives (\( FP \)) and false negatives (\( FN \)) respectively. These measures have also been adopted to evaluate record linkage algorithms \([81, 46]\). These two measures are combined in Equation 5.3
into F-measure [161]

\[
F_\beta = \frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}}
\] (5.3)

where $\beta$ is a constant that represents the importance attached to recall relative to precision.

To adopt precision and recall as the effectiveness measures for DQ rules, we need to give precise definitions to $TP$, $FP$ and $FN$ in context of error detection. Consider a DQ rule $\phi$, $Assessment(X_1, \ldots, X_p) \leftarrow R_1, \ldots, R_m, M_1, \ldots, M_n, \neg H$. We use $F$ to denote the event where at least one of $X_1, \ldots, X_n$ is factually erroneous, and $D$ to denote the event in which $\phi$ is violated (i.e., at least one of $X_1, \ldots, X_n$ is considered as erroneous due to the violation of $\phi$).

The precise meaning of a $TP$, $FN$ and $FP$ with respect to $\phi$ is given by Table 5.2.

| $D = True$ | $F = True$ | $t[X_1, \ldots, X_n]$ is a TP | $t[X_1, \ldots, X_n]$ is a FP |
| $D = False$ | $t[X_1, \ldots, X_n]$ is a FN |

In words, with respect to the DQ rule $\phi$, a tuple $t[X_1, \ldots, X_n]$ is a

- true positive ($TP$), if it is factually erroneous and it is signaled by $\phi$ as being erroneous;
- false positive ($FP$), if it is factually non-erroneous and it is signaled by $\phi$ as being erroneous;
- false negative ($FN$), if it is factually erroneous and it is not signaled by $\phi$ as being erroneous.

Effectiveness Scores.

Given the effectiveness measures, we obtain an effectiveness score for a DQ rule, by comparing the assessment result obtained by applying this rule to a particular database instance with the ground truth [138] (i.e., true knowledge about the quality) of that instance. The following example illustrates the computation of effectiveness scores.

**Example 5.2:** Consider two relation schemas $Customer(sin, cname, country, city, pcode)$ and $Employee(eid, ename, sin, salary)$, which record information about customers and employees
of a company. These include their social insurance numbers (*sin*), names of customers (*cname*) and employees (*ename*), country, city and postal code (*rcode*) of customers, ids (*eid*) and salaries of employees.

Suppose some customers also work for the company. If we assume *sin* values are error free, one may take advantage of this fact to detected potential erroneous *cname* and *ename* values using the following rule, \( \phi_0 \). 

\[
\text{Assessment}(X) \leftarrow \text{CusEmp}(\text{sin}: \_ , \text{cname}: X, \text{ename}: Y), \neg X = Y
\]

where *CusEmp* is the relation schema produced by performing the natural join on *Customer* and *Employee*, followed by a projection onto attributes *sin*, *cname* and *ename*.

Let \( I_{\text{CusEmp}} \) be an instance of *CusEmp*. To calculate the effectiveness scores of \( \phi_0 \), we obtain both manual and automatic assessment (using \( \phi_0 \)) on the same set of *cname* values. We use manual assessment as an approximation of ground truth. Table 5.3 shows \( I_{\text{CusEmp}} \) and its assessment results on 10 tuples. The attributes *errM* and *errT* are used to indicate the assessment results, with “1” indicating an error. The column *cnameM* is used to record the correct *cname* value obtained by manual assessment.

The effectiveness of \( \phi_0 \) is obtained by comparing values in *errM* and *cnameM* with those in *errT*. This comparison can be quantified in terms of the number of true positives = 2 (due to Tuple 006 and 009), false positives = 2 (due to Tuple 001 and 008), and false negatives = 1 (due to Tuple 004).

Effectiveness Formulas.

There are several limitations for obtaining precise effectiveness scores. First, we need access to a “representative” database instance, which may not always be available (e.g., when designing a new schema), or only partially available (e.g., when modifying an existing schema). Second, an effectiveness score indicates only how a DQ rule performs on one snapshot of the database. However, the database may evolve over time, and change its characteristics (e.g., percentage of erroneous *ename* values) in an unexpected way.
At least two alternatives may be used to overcome these limitations: *formal analysis* and *simulations*. In the first alternative, we estimate the effectiveness of a DQ rule without actually applying it, while in the second alternative, we apply the rule to artificially generated data. The first alternative leads to derivation of *effectiveness formulas* for the rule, which requires making probabilistic assumptions about the occurrence of errors in data values and *confounding factors*. By confounding factors, we mean special events that may “confuse” a DQ rule, making it less effective. For example, the country name “Australia” may be misspelled as “Austria”, which is still a valid country name; such an error cannot be detected using a rule based on a list of valid country names. An important point is that precise numeric probabilities are unlikely to be available, so that often it is desirable to use symbolic variables, which then become parameters to these formulas.

### 5.2.2 Manual Derivation of Effectiveness Formulas

In the previous section, we have shown how to calculate effectiveness scores when data is available. This section illustrates how to derive effectiveness formulas in an ad-hoc fashion, without applying the DQ rule to data. This illustration is done using two examples, each of which is divided into following steps: (1) making probabilistic assumptions, (2) calculating probabilities for the events of interests, and (3) formulating effectiveness formulas.
Use of duplicated attribute.

As discussed in Section 5.1.2, use of dependency underlies many DQ assessment techniques. In this section we consider the most basic form of full dependency: duplication of attributes. More specifically, we consider the DQ rule \( \phi_r \). Assessment\(_r\)(\(X\)) \(\leftarrow R_1(A : X, B : Y), X \neq Y\).

**Step 1: making probabilistic assumptions.** The main factor that affects effectiveness of \( \phi_r \) is the occurrence of errors in the attributes \( A \) and \( B \). For the rest of the paper, we make following assumption: errors in values of different attributes are independent of each other.

To simplify the analysis here, we will assume that the probability of an \( A \) or \( B \) value being incorrect is the same — denoted by \( p \). If we use \( Err^{t,A} \) to name the event that the recorded \( A \) value in the tuple \( t \) does not correspond to the real one, and use \( Cor^{t,A} \) to mean the converse, this assumption can be stated symbolically as \( \Pr(Err^{t,A}) = \Pr(Err^{t,B}) = p \), where \( \Pr(E) \) represents the probability of an event \( E \). Before we proceed further, we need to recognize that there is the possibility that both \( t.A \) and \( t.B \) are incorrect yet contain the same erroneous value; in this case, these errors “cancel out” as far as the DQ rule \( \phi_r \) is concerned (since they cannot be detected by \( \phi_r \)). We call this situation “error masking”, which is a particular type of confounding factors. Let us say that such masking will happen only with probability \( 1 - c_1 \).

**Step 2: calculating probabilities for the events of interests.** To estimate the effectiveness scores, we are interested in events concerning a tuple \( t \) (i) whether \( t.A \) has an error, and (ii) whether a DQ problem is signaled by \( \phi_r \). This estimation has to be adjusted for error masking. To compute the expected values for \( TP \), \( FP \) and \( FN \), we will actually compute the probabilities of events concerning a particular tuple \( t \), and then multiply this by the number of tuples in the relation.

First, true positives occur when \( t.A \) has an error (probability \( p \)) that is correctly signaled by \( \phi_r \). This happens when either \( t.B \) is correct (prob. \( 1 - p \)) or \( t.B \) is incorrect (prob. \( p \))...
but different from \( t.A \) (prob. \( c_1 \)); this yields probability: 
\[
\Pr(E_{rt}t.A \land C_{rt}t.B) + \Pr(E_{rt}t.A \land E_{rt}t.B \land (t.A \neq t.B)) = p \times (1 - p) + p \times p \times c_1.
\]

False negatives occur when \( t.A \) has an error that is not signaled by \( T_B \neq A \), because error masking occurs (which requires \( t.B \) to contain the exact same error); this has probability: 
\[
\Pr(E_{rt}t.A \land E_{rt}t.B \land (t.A = t.B)) = p \times p \times (1 - c_1).
\]

False positives occur when \( t.A \) has no error yet \( T_B \neq A \) signals a problem, which arises according to our rule when \( t.B \neq t.A \) (i.e., when \( t.B \) has an error); this has probability: 
\[
\Pr(t.A_{cor} \land t.B_{err}) = (1 - p) \times p.
\]

**Step 3: formulating effectiveness formulas.** Given the probabilities obtained in Step 3, the expected number of true positives, false positives and false negatives can be calculated as the number of tuples (say \( N \)) times the respective probability as following: 
\[
TP(T_B \neq A, A) = N \times (p(1 - p) + p^2 c_1);
\]
\[
FN(T_B \neq A, A) = N \times p^2 (1 - c_1);
\]
\[
FP(T_B \neq A, A) = N \times (1 - p)p;
\]

Specific effectiveness scores for \( \phi_r \) can then be obtained by plugging these numbers into Equation 5.1, 5.2 and 5.3. Since \( N \) appears both in the numerator and denominator, it will cancel out, resulting in the effectiveness formulas in Table 5.4 (Section 5.2.3).

**Use of metadata.**

For an attribute with a standardized (and finite) domain, such as country name or postal code, a common DQ rule is to check its values against a list of valid values for that attribute. Attributes with enumerated value domains (such as gender) also offer this possibility. In this section we consider the DQ rule \( \phi_m \).  
\[
Assessment_r(X) \leftarrow R_2(A : X), \neg L_A(X), \text{ where } L_A \text{ is a list of valid values for attribute } A.
\]

**Step 1: making probabilistic assumptions.** We make two passes through this analysis, in order to account for two different sources of problems. First, we assume as before there is a probability \( p \) that the recorded value of \( A \) is incorrect. In this case, error masking occurs when this erroneous value is still a valid value in the domain of \( A \) (e.g., “Australia” vs “Austria”)

– an event to which we assign probability $c_2$. If we use $Valid^{t.A}$ to name the event that the value $t.A$ is valid and $Invalid^{t.A}$ to mean the converse, we can represent these assumptions using following conditional probabilities: $\text{pr}(Valid^{t.A}|Err^{t.A}) = c_2$ and $\text{pr}(Invalid^{t.A}|Err^{t.A}) = 1 - c_2$.

Second, we consider the possibility of the valid-value list being imperfect, which is another type of confounding factors. In particular, we allow a probability $s$ that some value (e.g., the name of a newly independent country) is missing from the valid-value list $L_A^3$. If we use $L_{t.A}^A$ to name the event that the value $t.A$ is contained in $L_A$, and $L_{¬t.A}^A$ to mean the converse, we have $\text{pr}(L_{t.A}^A) = 1 - s$ and $\text{pr}(L_{¬t.A}^A) = s$. Notice here we are implicitly assuming that $\text{pr}(L_{t.A}^A)$ is independent from the characteristics of $t.A$ values (e.g., name values of different length or in different languages).

**Step 2: calculating probabilities for the events of interests.** In the first case (i.e., assuming a perfect valid-value list), true positives occur when $t.A$ is incorrect and the value is not in the valid-value list $L_A$ (therefore $t.A$ must be invalid, since all valid values are in $L_A$); this has probability: $\text{pr}(Err^{t.A} \land Invalid^{t.A}) = \text{pr}(Err^{t.A}) \times \text{pr}(Invalid^{t.A}|Err^{t.A}) = p \times (1 - c_2)$.

False negatives occur when the error is masked (i.e., when $t.A$ is incorrect but happens to be valid, and therefore is in $L_A$); this has probability $\text{pr}(Err^{t.A} \land Valid^{t.A}) = \text{pr}(Err^{t.A}) \times \text{pr}(Valid^{t.A}|Err^{t.A}) = p \times c_2$.

Finally, in this case, there can be no false positives: every $A$ value not in $L_A$ is an incorrect $A$ value.

In the second case (i.e., assuming a imperfect valid-value list), false positives show up when $t.A$ is correct, yet the value is missing from $L_A$; this has probability: $\text{pr}(Cor^{t.A} \land L_{¬t.A}^A) = (1 - p) \times s$.

For true positives, another source is possible, i.e., when an incorrect $t.A$ value is valid (due

---

\(^3\) A more thorough, but complex, analysis would allow errors in the table values themselves or extra/out of date values.
to error masking), but is accidentally missing from $L_A$; the total probability for true positives is therefore the one obtained in the first case plus following probability: $\text{pr}(Err^t \land Valid^t \land L \neg t) = \text{pr}(Err^t \land Valid^t) \times \text{pr}(L \neg t) = (p \times c_2) \times s$.

For false negatives, we need to multiply the probability obtained in the first case by $(1 - s)$, since they require the masking values also be in $L_A$.

**Step 3: formulating effectiveness formulas.** Given the probabilities we obtained in Step 3, specific effectiveness scores for $\phi_m$ can be calculated in the same way as for the case of $\phi_r$. See Table 5.4 (Section 5.2.3) for the resulting effectiveness formulas for $\phi_m$.

### 5.2.3 Evaluating and Comparing DQ Rules

The effectiveness formulas are useful for several reasons. First, they identify conditions (e.g., parameters $p$ and $s$ in Table 5.4) that affect the effectiveness of a DQ rule. Second, as shown below, they allow to perform trade-off analyses concerning different scenarios that involve one or more DQ rules. Each scenario produces a plot of expected effectiveness scores by fixing some parameters and allowing the others to vary.

The formulas that represent expected precision, recall and F-measure (when $\beta = 1$) for the DQ rules $\phi_r$ and $T_{LA}$, together with a summary of the parameters used in these formulas, are shown in Table 5.4. This subsection shows these two rules are evaluated individually (in Scenarios 1 - 4) and compared with each other (in Scenarios 5 - 10).

**Scenarios 1 - 4: Evaluating individual DQ rules.**

Scenarios 1 and 2 consider the impact of “error masking” (varying $c_1$) on the effectiveness of $\phi_r$, while Scenarios 3 and 4 consider the impact of the valid-value list’s “coverage” (varying $s$) on the effectiveness of $\phi_m$. For each rule, the evaluation is carried out with respect to a relatively clean database ($p = 0.05$, in Scenarios 1 and 3) and a dirty database ($p = 0.3$, in Scenarios 2 and 4).
Table 5.4: Effectiveness formulas for $\phi_r$ and $\phi_m$.

<table>
<thead>
<tr>
<th>DQ Rule: $\phi_r$</th>
<th>DQ Rule: $\phi_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>precision</strong> $= \frac{1+(c_1-1)p}{2+(c_1-2)p}$</td>
<td><strong>precision</strong> $= \frac{1+(s-1)c_2}{s/p+(s-1)(c_2-1)}$</td>
</tr>
<tr>
<td><strong>recall</strong> $= 1 + (c_1-1)p$</td>
<td><strong>recall</strong> $= 1 + (s-1)c_2$</td>
</tr>
<tr>
<td>$F_1 = \frac{2+2(c_1-1)p}{3+(c_1-2)p}$</td>
<td>$F_1 = \frac{2+2(s-1)c_2}{1+s/p+(s-1)(c_2-1)}$</td>
</tr>
</tbody>
</table>

**Parameters:**
- $p$: the probability that an $A$ value is erroneous
- $c_1$: the probability that both $A$ and $B$ values in a tuple are erroneous, but contain different errors
- $c_2$: the probability that an erroneous $A$ value is valid in the domain of $A$
- $s$: the probability that a valid $A$ (with or without error) is not contained in the valid-value list $L_A$

The results for Scenarios 1 and 2, as given in Figure 5.2(a) and 5.2(b), show that the precision and recall of $\phi_r$ decrease when the chance of “error masking” increases (i.e., as $c_1$ decreases). This corresponds to our intuition. However, a comparison of these two figures also reveals that, in a dirty database (i.e., with a larger $p$), the effectiveness of $\phi_r$ decreases more precipitously as the chance of “error masking” increases. For example, as $c_1$ decreases from 1 to 0, the recall of $\phi_r$ decreases by only 0.05 in the clean database, but by 0.3 in the dirty database.

![Figure 5.2: Evaluation of $\phi_r$.](image)

(a) Scenario 1: a relative clean database  
(b) Scenario 2: a dirty database

The results for Scenarios 3 and 4 are shown in Figure 5.3(a) and 5.3(b) respectively. In
both cases, as the “coverage” of the valid-value list decreases (i.e., as $s$ increases), an decrease in precision can be noticed as intuitively expected; however, the dramatic nature of its drop is not so easily predicted by intuition, and is therefore a benefit of this analysis. Also note that recall is much less affected by the “coverage”. Moreover, by comparing these two figures, it is observed that the probability of errors in $A$ has much greater impact on precision than on recall. More specifically, the recall of $\phi_m$ remains the same when comparing the clean and dirty databases; however, in the dirty database, the precision decreases considerably slower as the “coverage” decreases. For example, when $s$ increases from 0 to 1, the precision of $\phi_m$ decreases by 0.95 in the clean database, and by only 0.7 in the dirty database.

![Graphs showing the impact of errors on precision and recall](image)

(a) Scenario 3: a relative clean database  
(b) Scenario 4: a dirty database

Figure 5.3: Evaluation of $\phi_m$.

**Scenarios 5 and 6: Comparing DQ rules - the impact of errors.**

In what follows, DQ rules $\phi_r$ and $\phi_m$ are compared in two scenarios, by investigating the impact of the probability of errors in $A$ (varying $p$) on the effectiveness of these two rules in an optimistic and a pessimistic setting. In the optimistic case, the chance of “error masking” is very small and the “coverage” of the valid-value list is nearly perfect. More specifically, it is assumed that (i) in 99% of the cases, erroneous $A$ and $B$ values in a tuple contain different
errors (i.e., $c_1 = 0.99$), (ii) only 1% of erroneous $A$ values happen to be other valid values (i.e., $c_2 = 0.01$), and (iii) only 1% of the valid $A$ values are not contained in the valid-value list $L_A$ (i.e., $s = 0.01$). In the pessimistic case, the chance of “error masking” is increased significantly, while the “coverage” of the valid-value list is decreased significantly. More specifically, let us set $c_1 = 0.70$, $c_2 = 0.30$ and $s = 0.30$. Figure 5.4(a) and 5.4(b) compare the F-Measures of $\phi_r$ and $\phi_m$ in these scenarios.

In both settings, the F-measure of $\phi_r$ increases as the number of erroneous $A$ values increases (i.e., $p$ increases). A similar pattern can be observed for $\phi_m$ in the pessimistic setting; in the optimistic setting, the F-measure of $\phi_m$ increases dramatically when $p < 0.05$, and remains almost constant when $p \geq 0.05$. These two figures suggest under what circumstances one DQ rule is preferable to the other one. More specifically, in an optimistic world, $\phi_r$ outperforms $\phi_m$ only when the probability of erroneous $A$ values is quite small (i.e., when $p < 0.01$), while in a pessimistic world, $\phi_r$ is a more effective choice than $\phi_m$ as long as the error rate in $A$ is less than 40% (i.e., when $p < 0.4$).

A briefer summary might be that in a typical situation, where the chance of “error masking” is reasonably small (say, less than 5%) and the “coverage” of valid-value list is nearly perfect (say, more than 95%), $\phi_m$ is generally more effective in detecting errors than $\phi_r$, as long as the database is expected to have more than 5% erroneous values.

**Scenarios 7 - 10: Comparing DQ rules - the impact of “error masking”**

In what follows, the DQ rule $\phi_r$ and $\phi_m$ are compared in another four scenarios, by investigating the impact of the “error masking” (varying $c_1$ and $c_2$) on the effectiveness of these rules. The comparison is carried out with respect to a relatively clean database, i.e., $p = 0.05$ (Scenarios 7 and 9) and a dirty database, i.e., $p = 0.30$ (Scenarios 8 and 10), and as well as with respect to a nearly perfect valid-value list, i.e., $s = 0.01$ (Scenarios 7 and 8), and an imperfect valid-value list, i.e., $s = 0.3$ (Scenarios 9 and 10).

Figure 5.5(a) and 5.5(b) compare the F-Measures of $\phi_r$ and $\phi_m$ in Scenarios 7 and 8, while
Figure 5.4: Comparison of $\phi_r$ and $\phi_m$ - Impact of Errors.

Figure 5.6(a) and 5.6(b) compare them in Scenarios 9 and 10. From these figures, a dominant pattern can be observed: the chance of “error masking” has more impact on $\phi_m$ than on $\phi_r$, and this influence is independent of the probabilities of errors in $A$ and the “coverage” of $L_A$. In other words, the F-measure of $\phi_m$ increases more precipitously than that of $\phi_r$ does (in all four cases) as the chance of “error masking” decreases (i.e., as $c_1$ and $1 - c_2$ increases).

In addition to this general pattern, the following conclusion can be reached according to these figures. For relative clean databases with less than 5% of erroneous values, $\phi_r$ is always more effective than $\phi_m$. When there are more than 5% of erroneous values, $\phi_r$ still outperforms $\phi_m$, unless a relatively low chance of “error masking” is guaranteed and the “coverage” of the valid-value list is nearly perfect. This conclusion, together with the one made in Scenarios 5 and 6, gives us a complete comparison of $\phi_r$ and $\phi_m$.

The general point is that the mathematical assessment of the effectiveness of DQ rules based on probabilistic parameters allows us to make judgments about when to use one rule vs. another, or whether to use one at all – remember that there is an overhead for putting into place DQ rules.
(a) Scenario 7: clean db, good lookup  (b) Scenario 8: dirty db, good lookup

Figure 5.5: Comparison of $\phi_r$ and $\phi_r$ - Impact of “Error Masking” (I).

(a) Scenario 9: clean db, bad lookup  (b) Scenario 10: dirty db, bad lookup

Figure 5.6: Comparison of $\phi_r$ and $\phi_r$ - Impact of “Error Masking” (II).
5.3 A Semi-automatic Approach for Deriving Effectiveness Formulas

From Section 5.2.2, one may already notice that even for a simple DQ rules, manual derivation of effectiveness formulas is a non-trivial and error prone process. This section therefore presents a semi-automatic approach for deriving effectiveness formulas, for the class of DQ rules defined in Section 5.1.4. It starts with the definition of a vocabulary for describing events concerning error sources (Section 5.3.1); given a vocabulary, an effectiveness formula of a DQ rule is derived in following three phases (Section 5.3.2):

• constructing a directed acyclic graph of events, expressed using the terms in the vocabulary,

• filling in a conditional probability table for each event in the graph, and

• formulating the effectiveness formula given the results from the first two phases;

As mentioned in Section 5.2.1, there are two alternatives to estimate effectiveness of a DQ rule without actually applying it: derivation of effectiveness formulas and estimation of effectiveness using simulations. These two alternatives are complementary to each other: if they disagree, we know something is wrong in one or both of them, and needs to be reconsidered. For this reason, this section also reports the setup of the simulations of two databases, in which the effectiveness of DQ rules can be computed according to raw counts (as in Example 5.2), and the evaluation of the proposed approach using these simulations (Section 5.3.3).

A comparison of the proposed approach with Belief Networks is shown in Section 5.3.4.

5.3.1 A Vocabulary for Expressing Events

The effectiveness of a DQ rule is affected by two factors: the nature of the rule, and the occurrences of errors and confounding factors in the values being assessed by the rule. For the first factor, the proposed approach focuses on the classes of DQ rules as defined in Section 5.1.4.
For the second factor, the proposed approach requires a vocabulary of terms that can be used to express the events of interest. Let \( I_t^{A_i} \) denote a specific location in an instance \( I \) of a table with schema \( R \). At minimum, one needs to describe events that concern (i) \( \text{stored}(I_t^{A_i}) \), the actual values stored in \( I_t^{A_i} \), (ii) \( \text{real}(I_t^{A_i}) \), the real values supposed to be stored \( I_t^{A_i} \), and (iii) \( \text{Error}(I_t^{A_i}) \), the fact that a stored value \( \text{stored}(I_t^{A_i}) \) is different from the real value \( \text{real}(I_t^{A_i}) \).

If we were also to derive formulas for more advanced DQ rules, the simple vocabulary described above might need to be extended. For example, an arithmetic comparison operator in DQ rule can be replaced with a more advanced operator, such as a match operator [63]. In this case, one needs to add terms to the vocabulary that talk about the quality of these advanced operators.

### 5.3.2 The Approach and Implementation

Given a vocabulary of terms for expressing events of interest, an effectiveness formula of a DQ rule is derived in following three phases, with each phase providing the input for the phase after:

- constructing a directed acyclic graph of events, expressed using the terms in the vocabulary 5.3.2,
- filling in a conditional probability table for each event in the graph 5.3.2, and
- formulating the effectiveness formula given the results from the first two phases 5.3.2;

In what follows, each phase is described in details.

**Phase 1: Construct a Directed Acyclic Graph of Events**

Given a DQ rule \( \phi \), the first phase constructs a directed acyclic graph (DAG) \( G = (V, E) \), where \( V \) is a set of nodes representing the events considered as having some influence on effectiveness of \( \phi \), and \( E \) is a set of edges representing probabilistic relationships among these
nodes. More specifically, each node in $V$ is labeled by a variable $v_i$ (referred to as an event variable thereafter), whose values correspond to possible outcomes of the event represented by the node. Each edge $(v_i, v_j)$ indicates that the probability distribution of $v_i$ (called the parent) directly influences that of $v_j$ (called the child). An absent edge between two nodes means they are conditionally independent.

Notice that this phase is sensitive to the vocabulary we use for expressing the events of interest. A vocabulary essentially defines the scope of consideration (i.e., which events are considered relevant and which are not) for determining effectiveness of a DQ rule. Once this scope is fixed, the goal is then to capture the knowledge about how those events probabilistically relate to each other. Part of that knowledge is reusable and therefore could be captured to automate the graph construction process.

In what follows, we present such an algorithm (Table 5.5) that generates a graph with respect to the simple vocabulary, $O = \{\text{stored, real, Error} \}$, defined in Section 5.3.1. It is important to notice that the resulting graph may be modified manually (e.g., by adding or removing nodes and/or edges manually), if necessary, before feeding into the second phase.

From the examples in Section 5.1.4, we may observe that effectiveness of a DQ rule

---

Table 5.5: Algorithm - Construction of a DAG of events from a DQ Rule.

| Input: A DQ rule $\phi$: Assessment($X_1, \ldots, X_p$) $\leftarrow R_1, \ldots, R_n, M_1, \ldots, M_m, \neg H$; |
| Output: A DAG $G = (V, E)$ of events |
| 1. let $V = \{H_{\text{real}}, \neg H_{\text{stored}}, \text{Root}\}$ |
| 2. for each named variable $X_i$ in $H$ do |
| 3. let $V = V \cup \{\text{Error}(X_i)\}$; $E = E \cup \{\text{Error}(X_i), \neg H_{\text{stored}}\}$ |
| 4. end for |
| 5. for each $M_j$ do |
| 6. let $V = V \cup \{M_{i,\text{real}}, M_{i,\text{stored}}\}$; |
| 7. for each named variable $X_k$ in $M_j$ do |
| 8. let $V = V \cup \{\text{Error}(X_k)\}$; $E = E \cup \{\text{Error}(X_k), M_{i,\text{stored}}\}$ |
| 9. end for |
| 10. end do |
\( \phi \), \( \text{Assessment}(X_1, \ldots, X_p) \leftarrow R_1, \ldots, R_m, M_1, \ldots, M_n, \neg H \) is affected by\(^5\) (i) the number of arithmetic relations \( M_1, \ldots, M_n \), (ii) the number of named variables in \( \phi \) but not in \( \{X_1, \ldots, X_p\} \), and (iii) the quality of \( H \) if it is a database relation. The algorithm therefore captures knowledge about how these factors.

In Table 5.5, we use \( H^{\text{real}} \) to represent the relation obtained by replacing each variable \( X_k \) in \( H \) with real\( (X_k) \) (similarly for \( H^{\text{stored}}, M_i^{\text{real}} \) and \( M_i^{\text{stored}} \)); we use \( \text{Root} \) to denote \( M_1^{\text{stored}} \land \ldots \land M_m^{\text{stored}} \land \neg H^{\text{stored}} \).

From Step 1 to 4, we add to the graph nodes and edges for \( H \). More specifically, the probability whether or not the rule is violated (i.e., when \( \text{Root} \) is true) is affected partially by the probability of \( \neg H^{\text{stored}} \) being true or false. The latter in turn depends on the probability of \( H^{\text{real}} \) being true or false, as well as the probabilities of named variables in \( \neg H^{\text{stored}} \) being erroneous or not.

From Step 5 to 10, we add to the graph the nodes and edges for \( M \)'s. By the same reasoning, the probability whether \( \text{Root} \) is true is also affected by the probability of each \( M_j^{\text{stored}} \) being true or false; the latter in turn depends on the probability of \( M_j^{\text{real}} \) being true or false, as well as the probabilities of named variables in \( M_j^{\text{stored}} \) being erroneous or not. Moreover, as each \( M_j^{\text{real}} \) acts as a pre-condition for checking \( H^{\text{real}} \), the probability of \( M_j^{\text{real}} \) being true or false also affects that of \( H^{\text{real}} \) being true or false.

Although not shown in Table 5.5, when we add a node or edge to the graph, we need to make sure it is not already in the graph. Also notice that we do not add any node for \( R \)'s; this is because all real constraints are specified in \( M \)'s as discussed in Section 5.1.4, and we have not yet consider imperfect \( R \)'s (i.e., missing tuples or containing extraneous tuples) in our simple vocabulary.

An important property of our algorithm is that the resulting graph stays unchanged if we

\(^5\)it is also affected by the quality of (non built-in) functions and predicates in \( M \)'s which are ignored here.
modify the set of variables, \( \{ X_1, \ldots, X_p \} \), being assessed by the rule. The following example illustrates an application of this algorithm to a particular rule.

**Example 5.3:** Consider the DQ rule \( \phi_4 \), a variation of \( \phi_0 \) as discussed in Section 5.2.1, where we no longer assume \( sin \) values are free of errors. The graph for this rule, according to the algorithm, is shown in Figure 5.7. The question marks emphasize that each node is labeled by an event variable with two possible values: \{true, false\}.

![Figure 5.7: The DAG of events for \( \phi_4 \).](image)

**Phase 2: Generate and Fill in CPTs**

In the second phase, an (empty) conditional probability table (CPT) [93] is first generated for each node in \( G \). A CPT of a node specifies a conditional probability distribution of the node, given its parent nodes. More specifically, a CPT of a node \( v \) is generated by (i) finding the parent nodes \( \text{parents}(v) \) of \( v \) in \( G \), and (ii) inserting a row in the CPT for every possible configuration of states of \( v \) and \( \text{parents}(v) \). Since every node has two possible states in our case, there are \( 2^{n+1} \) possible configurations if \( |\text{parents}(v)| = n \).

The CPTs can be filled automatically with distinct probabilistic variables \( s_1, \ldots, s_n \) in all rows, with manual work only required to identify those cases where these \( s_i \) are not indepen-
dent because of the internal logic of concepts like error, real, stored or possibly rare cases when complex application domain semantics relate table columns. For example, an $s_i$ can be

- replaced by 1 (resp. 0) when the correspond event will always (resp. never) happen, or,

- set to a specific value according to the constraint that the conditional probability distribution of $\text{pr}(v \mid \text{parents}(v))$ is always 1.

**Example 5.4:** Consider the event graph for $\phi_4$ as shown in Fig. 5.7. The CPTs for two nodes $\text{real}(X_1) = \text{real}(X_2)$? and $\text{stored}(X_1) = \text{real}(X_2)$? are shown in Table 5.6 and 5.7 respectively.

The CPTs are filled with constants and probabilistic variables (shown in blue). Constants represent known probabilities. For example, consider a customer of a company who also works for the company. The social insurance numbers stored in Customer and Employee tables for that person must be identical, if both values are error-free (i.e., Row 7 of Table 5.7); on the other hand, when only one of the values is erroneous, they must be different (Row 3 and 5 of Table 5.7).

When a probability is not known at the time the CPT is constructed, we represent it using a probabilistic variable. For example, we use $r_2$ to denote the probability that a customer happens to be an employee of the same company (i.e., the only row in Table 5.6); we use $s_2$ to denote the probability that the stored social insurance numbers of a customer and an employee, who happen to be the same person, contain exactly the same error (i.e., Row 1 of Table 5.7).

\[ \begin{array}{|c|c|} 
\hline
\text{real}(X_1) = \text{real}(X_2)? & \text{False} \\
\text{True} & 1 - r_2 \\
\hline
\end{array} \]

Table 5.6: The CPT for $\text{real}(X_1) = \text{real}(X_2)$?.

\[ \emptyset \]
Table 5.7: The CPT for \( stored(X_1) = stored(X_2) \).
\[
\begin{array}{|c|c|c|c|}
\hline
\text{Error}(X_1)? & \text{Error}(X_2)? & \text{real}(X_1) = \text{real}(X_2)? & \text{stored}(X_1) = \text{stored}(X_2)\
\hline
\text{True} & \text{True} & \text{True} & s_2 \\
\text{True} & \text{True} & \text{False} & s_6 \\
\text{True} & \text{False} & \text{True} & 0 \\
\text{True} & \text{False} & \text{False} & s_5 \\
\text{False} & \text{True} & \text{True} & 0 \\
\text{False} & \text{True} & \text{False} & s_5 \\
\text{False} & \text{False} & \text{True} & 1 \\
\text{False} & \text{False} & \text{False} & 0 \\
\hline
\end{array}
\]

Phase 3: Derive effectiveness formulas

Given a rule \( \phi: \text{Assessment}(X_1, \ldots, X_p) \leftarrow R_1, \ldots, R_n, M_1, \ldots, M_m, \neg H \) and the effectiveness measures as defined in 5.2.1, the number of TP, FN and FP can be estimated by calculating the probabilities of the following three events (in what follows we use \( F \) to mean \( F = true \) and \( \neg F \) to mean \( F = false \); similarly for \( D \) and other events):

\[
E_{TP} = F \land D \quad (5.4)
\]
\[
E_{FN} = F \land \neg D \quad (5.5)
\]
\[
E_{FP} = \neg F \land D \quad (5.6)
\]

Recall \( F \) is true when at least one of \( \text{Error}(X_1), \ldots, \text{Error}(X_p) \) is true, and \( D \) is true when \( \text{Root} \) is true.

One may be tempted to obtain these quantities according to the definition of joint probability. For example, since \( \text{pr}(F \land D) = \text{pr}(F) \times \text{pr}(D|F) \)\(^6\), to calculate \( \text{pr}(E_{TP}) \), we need to know \( \text{pr}(F) \) and \( \text{pr}(D|F) \). However, in many cases, especially when the DQ rule is complex, \( \text{pr}(D|F) \) may not be easily obtained.

One way to overcome this difficulty is to divide the calculation into smaller pieces using

\(^6\)if \( F \) and \( D \) are independent events, \( \text{pr}(D|F) = \text{pr}(D) \).
additional events. For example, we may introduce an event \( v \), such that

\[
\text{pr}(E_{TP}) = \text{pr}(F \land D) \\
= \text{pr}(F \land v \land D) + \text{pr}(F \land \neg v \land D) \\
= \text{pr}(D|F \land v) \times \text{pr}(v|F) \times \text{pr}(F) \\
+ \text{pr}(D|F \land \neg v) \times \text{pr}(\neg v|F) \times \text{pr}(F)
\]

The premise is that the calculation of \( \text{pr}(D|v) \) and \( \text{pr}(v|F) \) is easier than that of \( \text{pr}(D|F) \). Of course, there could be a chain of additional events \( v_1, ..., v_n \) being introduced. In the present approach, these additional events are described using the terms from the vocabulary and are added to the graph \( G \) automatically in the first phase.

The more events are added, the more complicated the calculation become. However, using the chain rule for conditional probability [93],

\[
\text{pr}(v_1 \land \ldots \land v_n) = \prod_{i=1}^{n} \text{pr}(v_i|\text{parents}(v_i))
\]

(5.7)

the calculation can be dramatically simplified if the average number of parents of a node in \( G \) is much smaller than the total number of nodes in it\(^7\).

For example, if we knew the only edges in \( G \) are \((F, v)\) and \((v, D)\), the calculation of \( \text{pr}(E_{TP}) \) can be simplified as

\[
\text{pr}(E_{TP}) = \text{pr}(D|v) \times \text{pr}(v|F) \times \text{pr}(F) \\
+ \text{pr}(D|\neg v) \times \text{pr}(\neg v|F) \times \text{pr}(F)
\]

where \( \text{pr}(D|v) \) and \( \text{pr}(D|\neg v) \) are given by the CPT for \( D \) while \( \text{pr}(v|F) \) and \( \text{pr}(\neg v|F) \) are given by the CPT for \( v \).

\(^7\)Each \( \text{pr}(v_i|\text{parents}(v_i)) \) in above equation is given by the CPT of the node \( v_i \).
Example 5.5: Consider the DQ rule $\phi_4$ and its graph as shown in Fig. 5.7. In this case, $F$ and $D$ are defined as follows:

\[
F = \text{Error}(Y_1) \lor \text{Error}(Y_2)
\]
\[
D = \left(\text{stored}(X_1) = \text{stored}(X_2)\right) \land \left(\neg \text{stored}(Y_1) = \text{stored}(Y_2)\right)
\]

Therefore, to estimate the number of $TP$, we calculate

\[
\text{pr}(E_{TP}) = \text{pr}(F \land D) = \text{pr}(\text{Error}(Y_1) \land \text{Error}(Y_2) \land D) + \text{pr}(\neg \text{Error}(Y_1) \land \text{Error}(Y_2) \land D) + \text{pr}(\text{Error}(Y_1) \land \neg \text{Error}(Y_2) \land D)
\]

while the set of additional events to be introduced are \{Error($X_1$), Error($X_2$), (real($X_1$) = real($X_2$)), (stored($X_1$) = stored($X_2$)), (real($Y_1$) = real($Y_2$)), (stored($Y_1$) = stored($Y_2$))\}. Notice that the original event for $F$ (i.e., Error($Y_1$) \lor Error($Y_2$)) is divided into three disjoint sub-events Error($Y_1$) \land Error($Y_2$), \neg Error($Y_1$) \land Error($Y_2$), and Error($Y_1$) \land \neg Error($Y_2$).

A completed (and high level) algorithm for this phase is given in Table 5.8.

Implementation

The 3-phase approach presented in this section has been implemented in three Python modules. The first module `dag.py` parses the DQ rule (such as $\phi_4$) provided in an input file and generates
Table 5.8: Algorithm - Derivation of Effectiveness Formulas from CPTs.

**Input:** A DQ rule $\phi$, $H \leftarrow R_1, \ldots, R_n, M_1, \ldots, M_m$; a DAG $G = (V, E)$ of events; a CPT for each node $v$ in $V$.

**Output:** Effectiveness Formulas (precision, recall and f-measure) for $\phi$.

1. let $F = \text{Error}(X_1) \lor \ldots \lor \text{Error}(X_n)$, where $X_1 \ldots X_n$ are variables in $H$.
2. let $D = M_1 \text{stored} \land \ldots \land M_m \text{stored} \land \neg H \text{stored}$.
3. for each $\text{pr}(E_{TP}), \text{pr}(E_{FN}), \text{pr}(E_{FP})$ (as in Equation 5.4-5.6) do
   4. expand $\text{pr}(E_x)$, by (i) dividing $F$ (or $\neg F$) into disjoint sub-events, and (ii) using additional events from $V$ (e.g., Example 3.3).
   5. for each $P_i$ in $\text{pr}(E_x)$ do
      6. rewrite into $\text{pr}(v_1 | \text{parents}(v_1)) \times \ldots \times \text{pr}(v_m | \text{parents}(v_m))$ (using Equation 5.7).
      7. replace each $\text{pr}(v_i | \text{parents}(v_i))$ with a constant or variable in the CPT for $v_i$.
   8. end for
5. end for
9. calculate precision, recall and f-measure using $E_{TP}, E_{FP}, E_{TP}$ (using Equation 5.1 - 5.3).

a DAG of events in DOT language\(^8\) which can be visualized using graphviz\(^9\) (see Figure 5.7). The second module *cpt.py* reads in graph and generates a file which contains all empty CPTs, which are filled in manually by domain experts (see Table 5.6 and 5.7). Finally, these CPTs are read in by the third module *ef.py*, which generate effectiveness formulas (precision and recall) for the input DQ rule. Figure 5.8 shows a snapshot of a particular run of the code for $\phi_4$. The generated formulas can then be fed into a tool (such as the R tool\(^10\)) for plotting (see the next section for examples of such plotting).

5.3.3 Simulations and Evaluation

This section reports the setup of the simulations of two databases, and the evaluation of the proposed algorithm using these simulations as complementary way to obtain effectiveness scores.

An advantage of using synthesized data (when compared to real-world data) is that the ground truth [138] (i.e., true knowledge about the quality) of the data is controlled by a number of input control parameters. It is therefore much easier to evaluate the algorithm under various scenarios, by generating in a systematic way many different instances of a database with different ground truth.

---

\(^8\)http://www.graphviz.org/doc/info/lang.html
\(^9\)http://www.graphviz.org/
\(^10\)http://www.r-project.org/
Given a DQ rule $\phi$, (i) we first generate an instance of the database by setting a number of control parameters to specific values; (ii) we then apply $\phi$ to that instance and measure its effectiveness score, by comparing its result with the ground truth of the instance (which is obtained during data generation); (iii) finally we obtain the effectiveness score estimated according to the formula, by setting of its probabilistic variables according to the ground truth of that instance. This process is repeated many times; in the end, we plot both the measured and estimated scores onto the same graph.

One observation from this exercise is that in the data generation program the number of control parameters we have to set is much less than the entries in the CPTs. The values of many parameters can be derived from others. For example, the percentage of error masking can be derived from that of erroneous $cname$ and $ename$ values. In this sense, evaluating the algorithm with synthesized data is easier than checking its the correctness in a formal way, just like writing test cases for a computer program is an easier task than writing a formal proof for it.
The Canadian Postal Code Database

A Canadian postal code is a string of six characters, in the format A0A 0A0, where A is a letter and 0 is a digit. The first three characters, called forward sortation area (FSA), are used to identify a geographical region, while the last three digits, called local delivery unit (LDU), are used to denote a specific single address or range of addresses.

Consider a relation schema \( \text{Address}(sin, fsa, ldu, prov, city) \) which records the postal code (fsa and ldu) and province (prov) and city (city) of a person (identified by his/her social insurance number \( sin \)). According to Canada Post’s FSA map\(^{11} \), there are at least 385 distinct FSA codes in Ontario. The following DQ rule can be specified to assess the quality of FSA codes in the \( \text{Address} \) relation

\[
\phi_5. \text{Assessment}_5(Y) \Leftarrow \text{Address}(sin: \_ , fsa: Y, ldu: \_ , prov:X, city: \_ ), X='Ontario', \neg L_{FSAOnt}(Y).
\]

It states that if the province of an address is ‘Ontario’, its FSA code must be in a list \( L_{FSAOnt} \) of valid Ontario FSA codes.

**Data Generation**

To measure effectiveness of \( \phi_5 \), we generate an instance \( I_A \) of \( \text{Address} \), and an instance \( I_L \) of \( L_{FSAOnt} \) in the following steps:

Step 1: populate \( I_A \) and \( I_L \) by selecting values from following lists (which are assumed to be error-free),

- \( I_{Ont} \): a list of all Ontario FSA codes,
- \( I_{NonOnt} \): a list of all non-Ontario FSA codes, disjoint from \( I_{Ont} \), and,
- \( I_{Prov} \): a list of all Canadian province names,

Step 2: introduce errors into \( I_A \) and \( I_L \).

\(^{11}\text{http://www.canadapost.ca/common/tools/pg/fsamaps/pdf/Canada.pdf} \)
In both steps, decisions are made based on a number of control parameters, such as the percentage of persons with an Ontario address in \( I_A \), the percentage of erroneous \( fsa / prov \) values in \( I_A \), and the percentage of \( fsa \) codes in \( I_{Ont} \) (respectively, in \( I_{NonOnt} \)) that are accidentally missing from (respectively, included in) \( I_L \).

Notice these control parameters determine the percentage of certain values and errors, but not their exact locations in an instance; the latter is decided randomly and also affects what is considered as a TP, FN and FP.

**Comparison of Effectiveness Scores**

In what follows, we show the comparisons of the measured and estimated effectiveness scores of \( \phi_5 \) under several scenarios. In all cases, \( I_A \) contains a total of 10,000 tuples, and the effectiveness scores are expressed using F-Measure\(^{12}\).

**Scenario 1.1 and 1.2** In the first two scenarios, we simulate the impact of “the percentage of persons with an Ontario address” on the effectiveness of \( \phi_5 \); Scenario 1.1 is done with respect to a relatively clean database (i.e., the percentage of erroneous values is set to 0.005), while Scenario 1.2 is done with respect to a relative dirty database (i.e., that is set to 0.020). Figure 5.9(a) and 5.9(b) shows the plots of these two scenarios.

**Scenario 1.3 and 1.4.** In the next two scenarios, we simulate the impact of “the percentages of erroneous \( fsa \) codes and Province values” on effectiveness of \( \phi_5 \). Both scenarios are done with respect to a relatively clean database with 15% of persons having an Ontario address. The difference is the ratio of these two percentages: in Scenario 1.3 we set it to 1, while in Scenario 1.4 we set it to 1/2. Figure 5.10(a) and 5.10(b) shows the plots these two scenarios.

**Scenario 1.5 and 1.6.** In the last two scenarios, we simulate the impact of “quality of \( I_L \)” on effectiveness of \( \phi_5 \). Both scenarios are done with respect to a relatively dirty database

\(^{12}\)F-measure\([161]\) is an aggregated score of precision and recall defined as

\[
\text{F}_\beta = \frac{(1+\beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}}
\]

where \( \beta \) is a constant that represents the importance attached to recall relative to precision.
Figure 5.9: Canadian Postal Code Scenarios.
Figure 5.10: Canadian Postal Code Scenarios.
with 45% of person having an Ontario address; in Scenario 1.5, $I_L$ is incomplete (i.e., some fsa codes in $I_{Ont}$ are accidentally missing from it), while in Scenario 1.6, $I_L$ contains some extraneous fsa codes (i.e., some fsa codes in $I_{NonOnt}$ are accidentally included in it). Figure 5.11(a) and 5.11(b) shows the plots these two scenarios.

Figure 5.9(a) - 5.11(b) show that the effectiveness scores of $\phi_5$ calculated using the effectiveness formulas agree with those obtained by actually applying it to the various instances of the Canadian postal code database.

The Restaurant Address Database

Rule-based record linkage algorithms [81] find tuples referring to the same real-world entity in two or more databases by using a set of rules. Restaurant data has been used in testing record linkage algorithms (see [24] for an example).

Consider a relation schema $Restaurant(name, addr, desc)$ which records information about the names, addresses and descriptions of restaurants. Let $I^1_R$ and $I^2_R$ be two instances of $Restaurant$ from different data sources. A (simplified version of) record linkage rule states that tuples with matched (as determined by some similarity function and a threshold) address values must refer to the same restaurant. We further simplify the matter by replacing the match predicate with a equality condition. This leads to the following DQ rule which detects erroneous name values when duplicated tuples are detected (i.e., when they have the same address values),

$$\phi_6. Assessment_6(Y_1, Y_2) \leftarrow \ Restaurant(name: Y_1, addr: X_1, desc: \_),$$
$$\ Restaurant(name: Y_2, addr: X_2, desc: \_),$$
$$X_1 = X_2, \neg(Y_1=Y_2).$$

Data Generation

Similar to the Canadian postal code database, to measure effectiveness of $\phi_6$, we generate $I^1_R$ and $I^2_R$ in following steps:

Step 1: populate $I^1_R$ and $I^2_R$ by selecting values from $I_R$, which is an instance of $Restaurant$ containing restaurant information assumed to be error-free, and
Figure 5.11: Canadian Postal Code Scenarios.
Step 2: introduce errors into $I^1_R$ and $I^2_R$,

where the decisions are made based on a number of control parameters, such as the percentage of tuples in $I_R$ that are shared by $I^1_R$ and $I^2_R$, the percentage of erroneous name (and address) values in $I^1_R$ and $I^2_R$, the percentage of error masking in name (and address) values in $I^1_R$ and $I^2_R$.

**Comparison of Effectiveness Scores**

In what follows, we show the comparisons of the measured and estimated effectiveness scores of $\phi_6$ in several scenarios. In all cases, $I_R$ contains a total of 10,000 tuples.

**Scenario 2.1 and 2.2** In the first two scenarios, we simulate the impact of “the percentage of tuples shared by $I^1_R$ and $I^2_R$ (before errors are introduced)” on effectiveness of $\phi_6$; Scenario 2.1 is done with respect to a relatively clean database, while Scenario 2.2 is done with respect to a relative dirty database. Figure 5.12(a) and 5.12(b) shows the plots of these two scenarios.

**Scenario 2.3 and 2.4** In the second two scenarios, we simulate the impact of “the percentage of erroneous name” (Scenario 2.3) and “the percentage of erroneous address values” (Scenario 2.4) on effectiveness of $\phi_6$; in both case, 50% of tuples in $R$ are shared by $I^1_R$ and $I^2_R$ (before errors are introduced). Figure 5.13(a) and 5.13(b) shows the plots of these two scenarios.

**Scenario 2.5 and 2.6** In the last two scenarios, we simulate the impact of “the percentage of error masking for name values” (Scenario 2.5) and “the percentage of error masking for address values” (Scenario 2.6) on effectiveness of $\phi_6$; in both case, 50% of tuples in $R$ are shared by $I^1_R$ and $I^2_R$ (before errors are introduced). Figure 5.14(a) and 5.14(b) shows the plots of these two scenarios.

Figure 5.12(a) - 5.14(b) show that the effectiveness scores of $\phi_6$ calculated using the effectiveness formulas agree with those obtained by actually applying it to the various instances of the restaurant address database.
Figure 5.12: Restaurant Address Scenarios.
Figure 5.13: Restaurant Address Scenarios.

(a) Scenario 2.3

(b) Scenario 2.4
Figure 5.14: Restaurant Address Scenarios.

(a) Scenario 2.5

(b) Scenario 2.6
5.3.4 A Comparison with Belief Networks

The proposed approach is similar to Belief Networks in that both rely on a probabilistic graphical model (i.e., the DAG of events and CTPs). Moreover, the computation in the last phase of our algorithm is based on Bayes’ Theorem and the chain rule for Belief Networks. In fact, any effectiveness score obtainable from the effectiveness formulas can be computed using Belief Networks, as illustrated in following example.

Example 5.6 Consider the DQ rule, \( \phi_0 \). \( Assessment(X) \leftarrow \text{CusEmp}(sin : \neg, \text{cname} : X, \text{ename} : Y), \neg X = Y \), as discussed in Section . The probability of \( E_{TP} \) for \( \phi_0 \), according to Example 5.2, is

\[
p(1 - p) + p^2 c_1
\]

where \( p \) denotes the probability a \( \text{cname} \) (or \( \text{ename} \)) value is erroneous, while \( c_1 \) denotes the probability that \( \text{cname} \) and \( \text{ename} \) values in a tuple contain different errors. When we set \( p = 0.1, c_1 = 0.9 \), we get \( \text{pr}(E_{TP}) = 0.099 \).

Figure 5.15 shows the calculation of this particular probability in Belief Networks using Netica\(^\text{13}\). It shows the probability of a \( \text{cname} \) value being different from the corresponding \( \text{ename} \) value in the same tuple (i.e., \( D \)), given the observed evidence that the \( \text{cname} \) value is actually erroneous (i.e., \( F \)).

However, the proposed approach differs from Belief Networks in following ways. First of all, unlike in Belief Networks, our approach works \textit{by design} with CPTs that contain mostly variables (instead of constants). This feature allows us to generate effectiveness formulas rather than specific scores. As a byproduct, it minimizes the effort to obtain or estimate various probabilities during model construction\(^\text{14}\). Secondly, the specification of the DAG of events in


\(^{14}\)Belief Networks supports parameter learning, an inference task in which some probabilities are learned from data, for the same purpose.
Belief Networks is hard to automated in general. In contrast, the generation of (a skeleton of) the DAG can be automated in the first phase of our approach. The user can modify the resulting graph if necessary before feeding to the second phase.

More importantly, Belief Networks aim at making probabilistic inference and learning concerning individual entities. For example a Bayesian Network can be used to update the probability of certain event of an entity (e.g., a patient having a disease, a DQ rule being violated) when some evidences are observed (e.g., symptoms of the disease, errors in data values). Our approach, in contrast, provides an convenient way to *compare* two or more DQ rules in terms of their probabilities in some special events (e.g., \( E_{TP} \), \( E_{FN} \), \( E_{FP} \) and their aggregated form) under different valuations of variables in the effectiveness formulas. Interested readers are referred to [97] for examples of such comparison.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis has revisited an old problem: how does one design a database schema that satisfies user requirements, taking a new perspective adopted from recent developments in Goal Oriented Requirements research; in addition, it paid special attention to the data quality softgoal, as a significant design criterion.

A goal-oriented process for database requirements analysis and modeling (GODB) has been presented. This process consists of a number of steps, spanning the spectrum from high-level stakeholder goal analysis to detailed conceptual schema design. The GODB process has advantages over conventional ones, because of its support for systematic exploration of design alternatives and traceability. Goal models define alternatives for fulfilling a goal. Conventional approaches start with a requirement statement that describes one way of solving the problem; therefore, they do not recognize alternatives, and as a result can lead to overly heavy designs (e.g., including all possible data that might be needed).

The general GODB process has also been extended to address specifically Data Quality softgoals. This resulted in a “data quality by design” (DQD) process. The basic idea was that data of low quality might be detected and corrected by performing various quality assurance
activities. By analyzing these activities, additional data requirements for quality assurance purpose could be acquired and stored in the database; these cannot not be identified by analyzing the core business activities alone. The original schema produced from the GODB process could then be revised according these additional data requirements, and therefore support the corresponding quality assurance activities that are not available for the original schema.

The DQD process focuses on the structural aspect of schema design; modeling constraints constitutes the other equally important aspect of schema design. Quality assurance activities supported by a revised schema may involve manual work, and/or rely on some automatic DQ techniques, which often depend on the specification and enforcement of DQ rules. To guide DQ rule design and facilitate the building of a DQ rule repository, a classification of DQ rules has been presented, based on a number of important domain and application independent properties of DQ rules. A quantitative framework has also been proposed for measuring and comparing DQ rules according to one of these properties, namely effectiveness. This framework relies on the derivation of formulas that represent the effectiveness of DQ rules under different probabilistic assumptions. To this end, a semi-automatic approach to derive effectiveness formulas has also been presented.

The scalability of the proposed methodology is supported in the following ways. First of all, as it shares its concepts and techniques for goal modeling and analysis with GORE approaches, tools developed for GORE approaches, such as OpenOME\textsuperscript{1} from i* and TAOM\textsuperscript{2} from Tropos can be readily adopted for the early phase in database schema design. Next, analyzing informal data requirements and transforming them into a formal schema is central problem of classic conceptual database design methodologies [11, 50]; these methodologies address scalability by following a design strategy, such as the top-down, bottom-up or inside-out strategy (see Section 2.2.3 on a discussion of design strategies). Similar strategies could

\textsuperscript{1}http://www.cs.toronto.edu/km/openome/
\textsuperscript{2}http://sra.itc.it/tools/taom4e/
be used for the proposed methodology for the identification of concepts from descriptions of
goals and plans and subsequent organization of them into a conceptual schema. Finally, for the
constraint aspect, scalability is provided through a DQ rule repository and design guidelines.
To realize their full potential, proper tools support can be provided to facilitate the creation,
classification, retrieval, reuse and transformation of DQ rules.

6.2 Future Work

The work in this thesis can extended along following lines.

6.2.1 Cost Analysis

First of all, an important DQ issue that has not been addressed in this thesis is cost analysis. DQ
related costs can be classified into (i) costs caused by low quality of data, (ii) quality assessment
costs, and (iii) quality improvement and prevention costs [61]. An immediate extension to the
last part of this thesis would be to measure and compare DQ rules in terms of their costs (which
are quality assessment costs). These include human/organizational costs (e.g., for duplicate
data entry, for input and maintenance of the lists of (or mappings between) valid values), as
well as computer-related costs (e.g., for extra storage, and the cost of running the SQL check
constraint after every update). Together with the Section 5.2 and 5.3, this extension would
provide the foundation for a workbench that supports cost-benefit analysis in DQ rule design.

Moreover, since quality is widely accepted as “fitness for purpose”, the analysis of costs
associated with low quality data should be carried out within the context of specific use sce-
narios of the data. Examples of such use scenarios include data mining and clustering of web
data. To this end, the goal is to derive cost functions for dirty data within these data activities.
For a data activity $A$, suppose its performance on a particular data set $D$ may be obtained by
the function $p(A, D)$. The cost of dirty data in $D$ within the context of $A$ can be quantified by
$p(A, D) - p(A, D')$ where $D'$ is a clean version of $D$ (i.e., with all the erroneous values in $D$
restored to their corrected version).

### 6.2.2 Uncertainty in DQ Assessment

Dealing with uncertain data [119] remains an active research area for database systems. Probabilistic databases [42, 15, 105, 53, 5] models uncertainty by assigning a confidence value to tuples in a regular relation; alternatively, an uncertain database [22, 145] modifies the relational data model to simultaneously represents a set of possible states of the database. As discussed in Section 5.1, uncertainty is also an important property to consider when designing DQ rules. For the purpose of DQ assessment, uncertainty often arises when the violation of a DQ rule may be caused by either truly erroneous data or exceptional but correct data. Two possible lines of future work concerning DQ rule design are: (i) specification and evaluation of probabilistic DQ rules [37] on a regular database, and (ii) specification and evaluation of regular DQ rules on a probabilistic/uncertain database.

Uncertainty also plays an important role in the cause-effect analysis of DQ assessment. Use of low quality data eventually leads to the violation of higher level business rules, policies and objectives, or causes problems / failures in business operations. For example, an increase in erroneous customer addresses may result in an increase of marketing cost. These violations, problems and failures could be considered as the high level symptoms of the underlying DQ problems. They may be therefore used to build Belief Network models to predict the likelihood of the underlying DQ problems. Such predictions are complimentary to those provided by DQ profiling techniques which look for symptoms directly in data. This is similar to the diagnosis of a patient’s illness based on both his/her medical symptoms and lab tests.

### 6.2.3 Beyond Quality

Quality of data is only one of the challenges facing many information systems engineers and users; others concern data privacy, security, trust and etc. For example, with the exponential
growth of data on the Web, trust becomes an increasingly important issue; [86, 87] proposed to assess trustworthiness of RDF data by considering provenance information about the creation and Web-based access of the data. This approach is therefore based on the same metadata-based mechanism as that used by data quality assessment (provenance information is one type of met data). One possible future research direction is to investigate to what degree rules are useful in trust assessment, and how to effectively derive, manage and measure these rules. According to [76], there are two main components in the definition of trust, namely belief and commitment: a person $P_1$ trusts the other person $P_2$ if $P_1$ commits to a particular action based on her belief that $P_2$ will act in a certain way. In other words, “trust occurs when that belief is used as the foundation for making a commitment to a particular action”. Therefore trust rules are more likely to be contextual and probabilistic.
Bibliography


[55] Robert Darimont and Axel van Lamsweerde. Formal refinement patterns for goal-driven
requirements elaboration. In SIGSOFT ’96: Proceedings of the 4th ACM SIGSOFT
symposium on Foundations of software engineering, pages 179–190, New York, NY,
USA, 1996. ACM Press.


In IWSSD ’91: Proceedings of the 6th international workshop on Software specification
Press.


[59] Eric Dubois, Michael Petit, and Eric Yu. From early to late formal requirements: A


[61] L. P. English. Improving Data Warehouse and Business Information Quality. Wiley &
Sons, 1999.


of database systems, pages 159–170, New York, NY, USA, 2008. ACM. This paper introduced the notion of matching dependencies as a way of enforcing object identification constraints across tables. These dependencies use the idea of similarity measures to generalize the equality relation.


[69] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.


[98] Lei Jiang, Thodoros Topaloglou, Alex Borgida, and John Mylopoulos. Incorporating goal analysis in database design: A case study from biological data management. In Pro-


[100] Yue. K. What does it mean to say that a specification is complete? In Fourth International Workshop on Software Specification and Design (IWSSD-4), Monterey, USA, 1987.


