DEFINITION AND ANALYSIS OF POPULATION-BASED DATA COMPLETENESS MEASUREMENT

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy in the Faculty of Engineering and Physical Sciences

2011

By
Nurul A. Emran
School of Computer Science
Contents

Abstract ......................................................... 10
Declaration ..................................................... 11
Copyright ....................................................... 12
Acknowledgements ............................................. 13

1 Introduction .................................................. 14
   1.1 Research Motivation ..................................... 14
   1.1.1 Population-Based Completeness ...................... 16
   1.2 Aims and Objectives .................................... 18
   1.3 Research Contributions ................................ 19
   1.4 Thesis Outline .......................................... 20

2 Literature Survey ............................................ 21
   2.1 Background ............................................... 21
   2.2 Definitions of Completeness ............................. 25
   2.3 Measuring Completeness ................................ 26
       2.3.1 Null-Based Completeness (NBC) ................. 28
       2.3.2 Tuple-Based Completeness (TBC) ................ 30
       2.3.3 Schema-Based Completeness (SBC) .............. 32
       2.3.4 Towards Population-Based Completeness (PBC) 33
   2.4 Conclusions .............................................. 33

3 Defining Population-Based Completeness .................. 36
   3.1 The PBC Model .......................................... 36
       3.1.1 Populations ....................................... 38
       3.1.2 Reference Populations ......................... 39
List of Tables

2.1 Definitions of Completeness in the Literature .......................... 27

3.1 Basic Configuration of a PBC system (basicConfig) .................. 48

4.1 Examples of Duplicates Using an Adapted Gene Data Set of Zymomonas mobilis [PDL+09] for Setup ................................. 78

4.2 Examples of Errors Using an Adapted Gene Data Set of Zymomonas mobilis [PDL+09] for Setup ........................................ 79

4.3 Implication of CSs Modification on Content of Population Tables 95

4.4 Configuration Information Stored for CSs ............................ 119

5.1 A Proxy Map in a Pure Relational Table ............................... 138

5.2 A Proxy Map in a Multi-valued Table .................................. 138

5.3 Basic Configuration of a PBC System (basicConfig) ................. 150

5.4 Local Variable Definitions for the Query Transformation Algorithm 151

5.5 Proxy Configuration (proxyConfig) ..................................... 153

5.6 Local Variable Definitions for the Proxy Assessment Algorithm .... 157

5.7 An Example of an FD-based Proxy ...................................... 167

5.8 An Example of an AFD-based Proxy .................................... 168

5.9 An Example of The Worst Accuracy Case for an AFD-based Proxy Candidate ............................................................... 170

5.10 An Example of the Tuples that Violate FD ............................ 173

5.11 The TANE Results ......................................................... 175

5.12 An Example of the Best Space Saving Case for an FD-based Proxy Candidate ............................................................... 182

5.13 An Example of the Worst Space Saving Case (Non-composite FD-based Proxy Candidate) ................................................. 183

5.14 An Example of the Worst Space Saving Case (Composite FD-based Proxy Candidate) ......................................................... 185
5.15 An Example of the Best Space Saving Case for an AFD-based Proxy Candidate ........................................ 186
5.16 An Example of the Worst Space Saving Case for an AFD-based Proxy Candidate ........................................ 187
5.17 The Best-Case Space Saving Scenario for FD-based Proxy Candidates ....................................................... 191
5.18 The Best-Case and the Worst-Case Space Saving Scenarios for AFD-based Proxy Candidates .................... 196
List of Figures

3.1 The Components of PBC and Their Relationships .................. 43
3.2 The Relationship Between the Various Subjects of the Microbial Study and the Analyses in the Microbial Study Pathway ....... 50

4.1 Main Use Cases of a PBC System ......................................... 61
4.2 Main Activities of a PBC system ......................................... 61
4.3 A PBC Reference Architecture Using a Virtual Universe .......... 63
4.4 A PBC Reference Architecture Using a Fully Materialised Universe ................................................................. 65
4.5 A PBC Reference Architecture Using a Partially Materialised Universe ................................................................. 66
4.6 Setup in More Detail .............................................................. 72
4.7 Cleaning Tool Component in PBC Reference Architecture ....... 80
4.8 CSs Accessibility Assessor Component in PBC Reference Architecture ................................................................. 83
4.9 Space Optimiser Component in PBC Reference Architecture ... 86
4.10 Measurement in More Detail .................................................... 88
4.11 Format Checker Component in PBC Reference Architecture ... 90
4.12 Maintenance in More Detail ..................................................... 92
4.13 CSs Modification Monitor in PBC Reference Architecture ....... 100
4.14 PBC Reference Architecture With All Added Software Components ................................................................. 103
4.15 PBC Measurement Error for $|D \cap RP| > 0$ in the Case of Reference Population ................................................................. 107
4.16 PBC Measurement Error for $|D \cap RP| = 0$ in the Case of Reference Population ................................................................. 108
4.17 PBC Measurement Error for $|D \cap RP| > 0$ in the Case of Data Set Under Measure ................................................................. 109
4.18 PBC Measurement Error for $|D \cap RP| = 0$ in the Case of Data Set Under Measure ................................................................. 110
4.19 Completeness differences in three human SNP databases involving
74 genes .......................................................... 116
4.20 The Qurator Quality View Layers (Adapted from [EMS+09]) .... 123
4.21 SNP Quality View ............................................. 126
5.1 A Comparison of Proxy Map Size by Table Structure. .......... 142
5.2 Proxy Components in the PBC Reference Architecture. ....... 145
5.3 Maximum Proxy Errors and g3 Scores for AFD-based Proxy Candi-
didates of All Droppable Attributes ................................ 177
5.4 Average Proxy Errors and g3 Scores for AFD-based Proxy Candi-
didates of All Droppable Attributes ................................ 178
5.5 Space Saving and Proxy Errors of FD-based Proxy Candidates .. 192
5.6 The Results of Space Saving Assessment for FD-Based Proxy Candi-
dates ................................................................. 193
5.7 Maximum Proxy Error and Space Saving of AFD-based Proxy Candi-
didates ................................................................. 198
5.8 Average Proxy Error and Space Saving of AFD-based Proxy Candi-
didates ................................................................. 198
5.9 Proxy Error and Space Saving of AFD-based Proxy Candidates for
Droppable Attribute assignby ....................................... 199
5.10 Proxy Error and Space Saving of AFD-based Proxy Candidates for
Droppable Attribute method ........................................ 200
5.11 Proxy Error and Space Saving of AFD-based Proxy Candidates for
Droppable Attribute ev_type ........................................ 201
Abstract

Poor quality data such as data with errors or missing values cause negative consequences in many application domains. An important aspect of data quality is completeness. One problem in data completeness is the problem of missing individuals in data sets. Within a data set, the individuals refer to the real world entities whose information is recorded. So far, in completeness studies however, there has been little discussion about how missing individuals are assessed. In this thesis, we propose the notion of population-based completeness (PBC) that deals with the missing individuals problem, with the aim of investigating what is required to measure PBC and to identify what is needed to support PBC measurements in practice. To achieve these aims, we analyse the elements of PBC and the requirements for PBC measurement, resulting in a definition of the PBC elements and PBC measurement formula. We propose an architecture for PBC measurement systems and determine the technical requirements of PBC systems in terms of software and hardware components. An analysis of the technical issues that arise in implementing PBC makes a contribution to an understanding of the feasibility of PBC measurements to provide accurate measurement results.

Further exploration of a particular issue that was discovered in the analysis showed that when measuring PBC across multiple databases, data from those databases need to be integrated and materialised. Unfortunately, this requirement may lead to a large internal store for the PBC system that is impractical to maintain. We propose an approach to test the hypothesis that the available storage space can be optimised by materialising only partial information from the contributing databases, while retaining accuracy of the PBC measurements. Our approach involves substituting some of the attributes from the contributing databases with smaller alternatives, by exploiting the approximate functional dependencies (AFDs) that can be discovered within each local database. An analysis of the space-accuracy trade-offs of the approach leads to the development of an algorithm to assess candidate alternative attributes in terms of space-saving and accuracy (of PBC measurement). The result of several case studies conducted for proxy assessment contributes to an understanding of the space-accuracy trade-offs offered by the proxies. A better understanding of dealing with the completeness problem has been achieved through the proposal and the investigation of PBC, in terms of the requirements to measure and to support PBC in practice.
Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s policy on presentation of Theses.
Acknowledgements

I would like to gratefully acknowledge my supervisors, Dr. Suzanne Embury and Dr. Paolo Missier for their valuable guidance, insightful supervision, constant support and trust. I also would like to thank my advisor Dr. Alvaro A. A. Fernandes for his suggestions and advice that provide a broad idea on research. Also I thank Dr. Cornelia Hedeler who helps me to improve my understanding on a specific thread of biological study, Dr. Khalid Belhajjame for the technical supports during the setup of my system prototype and experiments, and Dr. Ismael Caballero for valuable discussions on data quality. Thanks also for Ravinder Singh and Javid Akhter for reviewing this thesis.

I would like to express my appreciation to the School of Computer science of the University of Manchester, especially to all members of the Information Management Group for constructive comments that opened an opportunity for me to view research from different contexts.

During my research, I receive very helpful support from my friends including Azlin, Lily, Azlinayati, Wahidah, Haliyana, Shuhaili, Keng Yap, Nicole, Farhana, Luke and Kwanchai who shared their research experience.

My special gratitude to my husband, Anas Malek for his sacrifices, patience and understanding during the substantial time it took to complete this research, and also to my daughters Nur Ainun Najah and Nur Ainun Nadhirah who bring joy in my life. Special thanks to my parents, Hj. Emran Alwi and Hjh. Rohani Jaafar for their constant prayers and motivations for me to regain confidence. I particularly want to thank my brothers and sisters for their unconditional moral supports.

Finally, I would like to acknowledge the financial assistance provided by the Universiti Teknikal Malaysia, Melaka and the Ministry of Higher Education, Malaysia during the course of this research.
Chapter 1

Introduction

1.1 Research Motivation

In many application domains, we can observe the negative consequences of poor data quality (DQ). For example, in the postal service domain, mail either fails to be delivered or is delivered to wrong addresses due to inaccurate, incomplete and outdated addresses [UNI]. In the health care domain, inaccurate and incomplete data have been identified as obstacles to providing antiretroviral (AR) therapy for patients who are infected by retroviruses (primarily HIV) in a timely manner, as more time is taken to resolve errors and missing values in patient records [FCB+08]. Providing the therapy on time is important as most patients, especially those in the severe stage of the disease, are likely to die within weeks or months if the therapy is not continued [FCB+08]. In the retail and marketing domain, the effort to prevent loss caused by retail crime has been reported as difficult because of the presence of errors in the Electronic Point of Sales (EPOS) data, which is used to detect the crime [SM03]. In addition, the EPOS databases are not always accessible by the security specialists who conduct the crime investigation [SM03]. In this case, even though other sources of data such as the corporate database are accessible by the security specialist, the problem of incomplete records of refund and sale transactions hampers the crime detection process [SM03]. Aside from being a complicating factor in the implementation of certain tasks, as just mentioned, poor DQ also imposes financial consequences. For example, a part of a survey conducted on U.K. businesses by Information Difference (a U.K. analyst firm) reports that goods worth about 20 million pounds went unsold because data about them was missing from Information Difference’s database [Cal10].
CHAPTER 1. INTRODUCTION

From the example provided above, we know that poor DQ is a problem and that accuracy is not the only concern. Other criteria such as completeness, consistency and accessibility have also been identified as important by the people who deal with data. A survey by Wang and Strong confirmed that DQ has multiple criteria, identifying fifteen main DQ criteria as being the most important to data consumers [WS96].

Awareness of the unwanted consequences of poor DQ motivates some organisations to assess the quality of data that they want to use in their applications before the decision to accept or reject the data is made (refer, for example, to Data Locator Group Ltd., a U.K.-based direct marketing services firm that uses software tools to assess the quality of its consumers database in terms of the presence of duplicate data entries [Cal10]). An important data quality criterion that is used to determine data acceptance is completeness [FLR94, PLW02]. In assessing the completeness\(^1\), a completeness measurement is conducted.

In the literature, two types of completeness measurement have been identified, namely subjective measurement and objective measurement [PLW02]. Subjective measurement requires people who have experience in dealing with data (such as data providers and data consumers) to make judgments about the quality of the data. A typical method used to conduct the subjective measurement utilises questionnaires. With the questionnaires, DQ ratings or scores, possibly in ratio or percentage units are given by people involved in the measurement [LS03]. Thus, DQ scores of the subjective measurement are based on people’s perception of the quality of data. Objective measurement, on the other hand, usually computes DQ scores based on reference data that can be considered to be of high quality. A few methods are available for objective measurement such as simple ratio, weighted average, min/max operation [PLW02].

In the literature, the simple ratio has been described as a method that “measures the ratio of desired outcomes to total outcomes” [PLW02]. By applying this method to completeness of a data set, we say that the simple ratio measures the ratio of the data set that matches with the ‘complete’ reference data set, to the complete reference data set. Based on the result of the measurement, we know that the data set is complete if it matches with the reference data set in full. If more than one data set is measured, other methods like weighted average and min/max operation might be used to aggregate the completeness ratios, which

\(^1\)For brevity, all references to completeness in this thesis are to data completeness.
individually, are measured based on the simple ratio method. What characterises ‘what is missing?’ from the data set under measure in completeness measurement defines the ‘type’ of completeness.

1.1.1 Population-Based Completeness

One type of completeness that has been mentioned in the literature is population-based completeness (PBC) [PLW02]. A population consists of a set of individuals that represent real world entities, whose information are recorded in a data set. For PBC, the concern is to determine whether the data set consists of a complete set of individuals or not, which requires measuring the individuals that are missing relative to a population.

In the literature, Scannapieco and Batini provide an example of a PBC problem involving a data set of Roman citizens [SB04]. In their example, there is a need to determine the completeness of a data set of Roman citizens, whose information are recorded and maintained by a company for business purposes [SB04]. The personal registry of Rome’s city council is consulted, where a population of Roman citizens that is considered as complete is retrieved as a reference to determine the citizens that are missing from Roman citizen data set that belongs to the company [SB04].

The importance of PBC can be seen in the descriptions of many problems in the literature, as shown in the following examples:

- In bioinformatics, to study the genes that are responsible for certain diseases, a candidate gene set is prepared and validated before more detailed tests are performed to find the disease-causing genes [Ile08]. According to Tiffin et al., because many complex diseases could be linked to multiple gene combinations, determining whether the data set of gene candidates is complete or not is becoming more important in the analysis for the bioinformaticians in order to produce a more reliable set of (potential) disease-causing genes [TANPI09]. Consequently, if some genes are missing from the gene data set used in the analysis, links between those genes and the disease cannot be established.

In this example, completeness of the candidate gene data set used in the analysis is determined by consulting various gene data sources like public
genome databases, gene expression databases, data on gene regulatory networks and pathways, as well as biomedical literature to check whether any gene has been missed from the data set [TANPI09]. In this example the genes reference population is gathered from multiple sources.

- In the study of human mortality levels, an unbiased estimate of the level of mortality can be provided if a complete data set of registered deaths is used [BH81]. This study highlighted the difficulty of gathering information to establish the complete reference population (through census) and proposed a way where census can be avoided.

- In law, incomplete data sets of newborns have caused a hospital in the U.S. to face legal actions [Tam03]. Incomplete data is a serious problem in this context because to treat and to save the lives of newborns with serious health conditions in timely manner, data about the types of treatment received, the patients’ reaction towards the medication and the persons responsible for the treatment must be available when needed. In this case, the completeness of patient data set is of concern. No formal completeness measurement has been given but the completeness of patient data set in this example is determined based on the absence of patient records from the medical treatment time slots. In this example, the non-missing patient records are regarded as a reference population of the patients.

The need for PBC is not limited to the examples as presented above and we can see from the examples that, the usage of reference populations is crucial to determine completeness of data sets under measure. However, little is known about how the reference populations are defined. In addition, how PBC is measured is unclear as the measurement method(s) used has not been described formally.

Ideally, the reference populations used in PBC measurements are the representation of the real world which we would call the true populations. However, using true populations for PBC measurements can be hindered by the lack of knowledge of the individuals of the true populations or by the inaccessibility of the source(s) that provides information about the individuals. The alternatives to true populations are ‘approximate’ populations, the populations that could be accepted by the application domain’s community as ‘complete’ reference populations in PBC measurements. However, even when approximate populations are
adopted, we cannot avoid answering fundamental questions of PBC that have not been addressed by any studies in completeness. These questions are:

- What is a reference population? How can we define a reference population?
- What is needed to make PBC measurement workable in practice?
- What are the technical issues that may arise in PBC measurements? And how can we address the issue(s)?

In the next section, we will provide the aims of our research that are formulated to address the above questions.

1.2 Aims and Objectives

This research aims to explicitly define the elements (i.e., the inputs required to measure PBC) of PBC and to determine whether it is a useful and implementable concept in practice. We set out to meet the following primary research objectives:

1. To propose the definition of PBC.
2. To analyse the types of PBC measurement requests.
3. To define PBC measurement formula.
4. To analyse the software and hardware elements needed to support PBC as a workable solution.
5. To determine the technical issues that are barriers to answering PBC measurement requests accurately.

The analysis of the technical issues of PBC measurements reveals one issue facing PBC measurement providers. When measuring PBC across multiple databases, data from those databases need to be integrated. Materialising the integrated database is an option that PBC designers could choose. Unfortunately, however, this option may lead to an integrated database that is impractically large to maintain.

We propose an approach to test the hypothesis that the available storage space can be optimised by materialising only partial information from the contributing databases, while retaining accuracy of the PBC measurements. Our approach
involves substituting some of the attributes from the contributing databases with smaller alternatives called proxies, by exploiting the functional dependencies (FDs) and approximate functional dependencies (AFDs) that can be discovered within each local database. Thus, alongside the primary objectives, we set some secondary objectives of the research as follows:

1. To analyse how proxies can be useful in optimising the storage space.

2. To propose what is needed in order to implement the proxy-based approach within PBC systems.

3. To observe that accurate PBC measurements can be achieved by using proxies that optimise storage space, by examining the trade-offs between the storage space and the accuracy of PBC measurements within case studies.

1.3 Research Contributions

The contributions of this thesis are:

- The definitions of PBC elements and PBC elements basic configuration.
- The identification of the types of PBC measurement requests.
- The PBC measurement formula that supports the types of PBC measurement requests identified.
- The reference template of the basic configuration of PBC.
- The identification of the types of PBC system reference architecture.
- The list of the technical issues inherent in a PBC system that affect answering accurate PBC measurement requests.
- The identification of the conditions where incomplete and inaccurate reference populations affects PBC measurement accuracy.
- An understanding of the applicability of the approximate functional dependency (AFD) theory in searching for the proxy candidates.
- An example of how a PBC system is implemented in the case of a biological domain.
• The results of storage space optimisation analysis.
• The proxy-based reference architecture of a PBC System.
• The results of the case studies of the space-accuracy trade-offs.

1.4 Thesis Outline

The rest of the thesis is organised as follows.

• Chapter 2 begins by laying out the early studies of completeness, then presents a survey of completeness definitions and a survey of completeness measures. The types of completeness measure will be presented and the gaps in the existing completeness studies will be presented.

• Chapter 3 addresses the questions regarding populations. A detailed description of PBC will be presented where the elements of PBC measurement will be defined. The definition of PBC elements and the definition of the PBC measurement formula will be given in relational database terminology, although the concepts are not restricted to this data model.

• Chapter 4 addresses the question of the practicality of PBC, where the technical issues that affect the accuracy of PBC measurements will be identified. In addition, three types of PBC reference architecture will be proposed. An implementation of a PBC system will be presented, by using a biological domain as a case study.

• Chapter 5 deals with the question of how can we support the designers of PBC systems in optimising use of available storage space. We present a proxy-based approach, to address the storage space constraint issue. Two case studies will be presented in this chapter, with the aim of answering several questions raised regarding the case studies.

• We conclude in Chapter 6 with a discussion of the research contributions, the limitations of our research and proposals for further direction of the research.
Chapter 2

Literature Survey

In this chapter, we provide a review of the literature on completeness with the aim to learn the different forms of completeness measure proposed to date. By learning the features of the completeness measures in the literature, an understanding of the similarities (and differences) of those measures will be provided, and the gaps in the current completeness measure proposals will be identified. We begin with Section 2.1 that provides the background to completeness. Section 2.2 presents the definitions of completeness from the literature. Section 2.3 presents and compares several types of completeness measure proposed to date. In particular, for each proposal, we examine the definition of the reference data set which is used in completeness measurement and the method used to measure completeness. Finally, in Section 2.4, we summarise the chapter and conclude by pointing out the gaps in the current literature that will be addressed in the thesis.

2.1 Background

Studies in data completeness are not new; they have been conducted since at least the 1970s. During this period, the data completeness problem was well known as the problem of missing information among scholars in the database community [Cod79, Cod86, Gra77, Lip79, IL84, Vas79, Zan84] as well as among statisticians [RGA03, DLR77, Roy04, SO98].

For the database community, the early work on completeness largely dealt with the problem of representing missing values (as opposed to ‘empty’ or undefined values) within the relational tables, where *nulls* were usually assigned for the missing values in the tables [Zan84]. Various representations of null have been
used, for example, the @ symbol [Cod75], ω [Cod79] and the use of variables such as x, y and z [IL84]. About 14 manifestations of nulls have been identified in the ANSI/SPARC interim report [Zan84], but the two types of null that can be frequently seen in the literature are: 1) the unknown nulls, where the values are missing because they are unknown, and 2) the non existence nulls, where the values are missing because the attribute of the relation is inapplicable. For example, the attribute ‘property owned’ of the citizens in Rome city council records is assigned a null in the case where a citizen does not own any properties.

The studies in completeness that deal with how to distinguish the types of null just mentioned have been driven by the need to determine whether the completeness problem exists or not. The presence of nulls is regarded as legitimate in the ‘non-existence’ case but not the ‘unknown’ case. Therefore, if ‘non-existence’ nulls are present, no completeness problems arise. But, the presence of unknown nulls signifies the existence of a completeness problem.

In order to distinguish the types of null, different representations have been proposed for each. In 1986, Codd proposed different representations of nulls according to their type, where the unknown nulls are represented by A (denoting applicable, but absent), while I (denoting inapplicable) is used to represent the non-existent nulls [Cod86]. The work assumes that database administrators could provide the information on both types of null and perform the updates (i.e. from I to A), which could be a burden in cases where the volume of data is large.

Furthermore, distinguishing the types of null is not straightforward especially for attributes which are not mandatory. For example, as every registered citizen must have ‘nationality’ information, a null ‘citizen nationality’ attribute in a citizen record means that a citizen’s nationality is ‘unknown’. However, it is hard to distinguish whether the nulls that are present in the ‘property owned’ attribute of citizen records are ‘unknown’ or ‘non-existence’ as having a property is an optional characteristic (rather than mandatory) for the citizens. In this case, additional information is required to distinguish the types of null. For example, the ‘property type’ attribute might be referred to where, if the ‘property type’ attribute has a value, then a null in the ‘property owned’ attribute is an ‘unknown’ (rather than a ‘non-existence’ null). On the contrary, if the ‘property type’ attribute has a null, then a null in the ‘property owned’ attribute might be a ‘non-existence’ type null.

The early studies on completeness also dealt with how to evaluate queries
where nulls are present. One of the questions that arises in the query evaluation is, should a query expression such as \( X > Y \) be evaluated as TRUE or FALSE if either \( X \) or \( Y \) is null? Codd suggests the three-valued logic in treating a query expression involving nulls where the MAYBE state is assigned for such an expression [Cod75]. However, Grant argued that the three-valued logic is only applicable for the unknown nulls but not for the non-existence nulls, and he suggested that expressions involving the non-existence nulls should be evaluated as either TRUE or FALSE only, as there is no uncertainty (denoted by a MAYBE state) issue that arises for the non-existence nulls [Gra77]. In 1979, Vassiliou extended Codd’s work by introducing the four-valued logic to distinguish both types of null using the denotational semantic [Vas79] but, this approach has been criticised by Zaniolo in terms of its computational cost, as for certain queries the cost could be high [Zan84]. Perhaps, because there is no clear consensus on the adoption of either the three-valued logic or the four-valued logic, the two-valued logic remains a common way to treat nulls in query evaluation in most database systems today. In addition, no special symbols are used to distinguish the type of null as the cells containing nulls in relational tables are usually left empty.

For statisticians, missing values have been regarded as a nuisance that could prevent them from making reliable analyses. The focus of the studies in statistics is largely deal with the effort to reduce the impact of missing values that appear in statistical surveys or in questionnaire results. The missing values are usually represented by responses like “Don’t know”, “Refused”, “Unintelligible”, and the difficulty to distinguish whether the missing values are legitimate or not has been reported as one of the complexities in handling missing values [SG02].

Deletion of the entire records containing missing values has been identified as one of the methods that has been applied to handle missing values in statistics. By deletion, the missing values are ignored and excluded from the studies, causing only the complete survey items to be retained in the studies. However, in the case where the proportion of missing values is large relative to the number of items in a survey, deletion of missing values is not always preferable as much valuable information will be lost [SO98] and selection bias will be introduced. Thus, the imputation method was proposed in which the values that are missing are substituted. Through imputation, no survey items will be discarded but we need to make sure that the imputed values will not cause bias in the study. Statistical
procedure has been proposed in order to improve estimation of the missing values, where studies conducted by Dempster, Laird and Rubin [DLR77] have been recognised as the earliest work that formalises an expectation-maximization (EM) algorithm that is used to estimate the missing values [SG02]. Imputation techniques which are based on maximum likelihood (ML) and multiple imputations have been used to estimate the missing values [Lit88, SG02].

The following are the studies in which completeness is a key factor in several application domains:

- In biology, missing values in microarray data sets can be caused by experimental faults such as when the measurements needed to derive the values cannot be performed due to technical reasons [Ait10] (i.e. the presence of dust on the slides of the specimen or the probe may not be fixed properly [KGP06]). Using complete microarray data sets is crucial for scientists, as missing data points can hinder downstream analyses, such as unsupervised clustering of genes, detection of differentially expressed genes, construction of gene regulatory networks etc.) [Ait10, TANPI09]. Imputation methods from statistical methods have been applied to estimate the missing values so that analysis and scientific interpretation that rely on the data sets can be performed. In studies conducted by Jörnsten et al., a new imputation method called LinComb has been proposed to estimate missing values in microarray data sets where the method is adaptable to the frequency of missing values and to the distribution of the missing values in the data set (whether the values are heterogeneous or homogeneous) [JWWO05].

- In the government sector, missing values in time series data affect the measurement of the return premiums generated by liquidity differences in bonds for U.S. government notes and bond portfolios [War92]. The EM algorithm as introduced by statisticians (refer to [DLR77]) has been used to estimate the missing values.

- In social sciences, missing values have been identified in surveys caused by several factors such as lack of cooperation from the respondents, or the deletion of inappropriate values in the survey [LR89]. Researchers in this domain study the types of missing value based on the distribution of the missing values in the data set. By identifying the types of missing value (i.e. generic/random, univariate and monotonic), suitable techniques to handle
the missing values are identified. Statistical methods have been applied to impute the missing values.

The studies presented above deal with nulls (that are illegitimately present) in order to produce reliable and complete analyses. Completeness has been defined as the absence of illegitimate nulls in the studies presented in this section. To continue exploring the meanings of completeness, we will next present definitions of completeness as documented in the literature.

2.2 Definitions of Completeness

A survey conducted by Wang and Strong showed that completeness is an important data quality criteria for data consumers [WS96]. However, even though many agree that completeness is important, a standard definition of completeness that can be accepted by the people who are concerned about completeness is not available. We extract a range of completeness definitions from the literature on databases as well as on data quality, proposed from 1985 to 2009 as shown in Table 2.1, in order to examine the various definitions of completeness.

Based on the definitions, we found that, the common question that arises is the question of ‘what is missing?’\(^1\), and that there are several types of missing ‘units’ proposed. We can see in Table 2.1 some missing units appear in multiple definitions (such as ‘value’, ‘tuples’ and ‘information’) and some missing units only appear in a single definition of completeness (such as ‘elements’ and ‘contents’).

We can also see from the definitions that, the ‘view(s)’ of ‘what is missing’ was included by some researchers (notably [BP85],[BP03], [FLR94], [Li03],[PS03], [PLW02] and [WKC\(+07\), WKK\(+09\)]). For example, Fox, Levitin and Redman [FLR94] addressed missing values from the entity attributes view, while Wolf \textit{et al.} [WKC\(+07\), WKK\(+09\)] addressed missing values from the view of a tuple. As more than one view is available, we know that completeness is a multi-dimensional concept that requires some way to measure the missing units from the data set concerned.

Unfortunately, how completeness is measured was described by only some of the studies that provide definitions of completeness (notably [BP03], [FLR94]

\(^1\) ‘What is missing’ in every definition of Table 2.1 is in italics.
and [PLW02]). To learn how completeness is measured, we surveyed the literature where completeness measures were proposed.

### 2.3 Measuring Completeness

Driven by the ‘what is missing?’ question and the types of the missing unit, we divide the completeness measures into the following types:

- **Null-Based Completeness (NBC):** the focus of the measures is on the ‘values’ that are missing from the data set under measure, where the missing values are represented by nulls.

- **Tuple-Based Completeness (TBC):** the focus of the measures is on the ‘tuples’ that are missing from the data set under measure.

- **Schema-Based Completeness (SBC):** the focus of the measures is on the missing ‘schema elements’ (e.g., attributes and entities) from the schema under measure.

- **Population-Based Completeness (PBC):** the focus of the measures is on the missing ‘individuals’ from the data set under measure relative to a reference population.

A simple ratio method is usually applied to measure completeness [PLW02] where completeness of a data set set under measure can be defined as:

\[
\text{completeness}(D, R) = \frac{|(D \cap R)|}{|R|} \in [0, 1],
\]

where \( D \) is the *data set under measure* and \( R \) is the *reference data set*.

Within each proposal of completeness measures, we will observe the following characteristics:

- the view(s) of ‘what is missing’ from the data set under measure (e.g., missing values from a tuple view, missing tuples from a relation view etc.),

- the reference data set used,

- how the reference data set is defined,

- the method used to measure completeness.
Table 2.1: Definitions of Completeness in the Literature

<table>
<thead>
<tr>
<th>Researcher</th>
<th>Completeness definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ballou and Pazer [BP85]</td>
<td>“all values for a certain variable are recorded.”</td>
</tr>
<tr>
<td>Fox, Levitin and Redman [FLR94]</td>
<td>“the degree to which a data collection has values or all attributes of entities that are supposed to have values.”</td>
</tr>
<tr>
<td>Pipino, Lee and Wang [PLW02]</td>
<td>column completeness: “a function of the missing values in a column of a table”</td>
</tr>
<tr>
<td>Wolf et al. [WKC+07, WKK+09]</td>
<td>“Let $R(A_1, \ldots, A_n)$ be a database relation. A tuple $t \in R$ is said to be complete if it has non-null values for each of the attributes $A_i$; otherwise it is considered incomplete. A complete tuple $t$ is considered to belong to the set of completions of an incomplete tuple $\hat{t}$ (denoted $C(\hat{t})$), if $t$ and $\hat{t}$ agree on all the non-null attribute values.”</td>
</tr>
<tr>
<td>Motro [Mot89]</td>
<td>“completeness as constraints correspond to predicates whose interpretation must contain all the tuples that represent real world relationships.”</td>
</tr>
<tr>
<td>Li [Li03]</td>
<td>“query answer that could be computed if we could retrieve all the tuples from the relations in the query.”</td>
</tr>
<tr>
<td>Kahn, Strong and Wang [KSW02]</td>
<td>“the extent to which data is not missing and is of sufficient breadth and depth for the task at hand.”</td>
</tr>
<tr>
<td>Wang and Strong [WS96]</td>
<td>“breadth, depth and scope of information in the data.”</td>
</tr>
<tr>
<td>Jarke and Vassiliou [JV97]</td>
<td>“the percentage of the real world information entered in the sources and/or the warehouse.”</td>
</tr>
<tr>
<td>Motro and Rakov [MR98]</td>
<td>“measures of the true proportion of the information that is stored.”</td>
</tr>
<tr>
<td>Pipino, Lee and Wang [PLW02]</td>
<td>schema completeness: ‘the degree to which entities and attributes are not missing from the schema’.</td>
</tr>
<tr>
<td>Wand and Wang [WW96]</td>
<td>“the ability of an information system to represent every meaningful state of the represented real world system.”</td>
</tr>
<tr>
<td>Bovee, Rajendra and Mak [BRM03]</td>
<td>“having all required parts of an entity’s information present.”</td>
</tr>
<tr>
<td>Naumann and Rolker [NR00]</td>
<td>“quotient of the number of response items and the number of real world items. Also: coverage, scope, granularity, comprehensiveness, density, extent.”</td>
</tr>
<tr>
<td>Pernici and Scannapieco [PS03]</td>
<td>“the degree to which the elements of an aggregated element are present in the aggregated element instance.”</td>
</tr>
<tr>
<td>Ballou and Pazer [BP03]</td>
<td>“presence of all defined content at both data element and data set levels.”</td>
</tr>
</tbody>
</table>
2.3.1 Null-Based Completeness (NBC)

The first proposal for a measure of NBC was made by Fox, Levitin and Redman [FLR94]. In explaining their completeness measures, they described a datum as a triple $< e, a, v >$, where $v$ is the value of the attribute $a$ that belongs to an entity $e$ [FLR94]. Nulls were viewed from two levels of granularity: single datum level and at data collection level. According to the study, different granularity levels require different types of measurement. At the single datum level, a binary measure was proposed which checks whether a datum has a value or not. At the collection level, the study described the completeness measure as an ‘aggregate’ measure that computes the fraction of the data that are null. The reference data set for this study is defined implicitly, which is a data set that has no nulls (where all attributes are mandatory). In this study, a null is represented by an empty slot for $v$ in each datum and all nulls under observation are regarded as illegitimate.

Another NBC measure, by Pipino, Lee and Wang, viewed nulls from the column level of a relational table [PLW02]. ‘Aggregate’ measure is used by taking the ratio of the number of nulls in a column of a table to the number of tuples in the column and subtracting it from 1. In this study, the reference data set is defined implicitly from the column under measure (i.e., it consists of a column with the same number of tuples, none of which are nulls). The values from the column were treated as a bag (rather than a set) where all individual null values are counted in the NBC measurement.

Ballou and Pazer [BP03] however, view NBC from a vector of ‘categories of data’ (e.g. teaching data, research data). Each data category is assigned with a weight that shows its importance relative to the other categories. Each category consists of one or more data sets, and by using the ‘aggregate’ measure, the aggregated completeness is computed to determine the overall completeness of the category. The overall NBC has been defined as an aggregate function where the average of the product of the individual category completeness measurements and their weights is computed. An NBC measure called structural completeness was defined by the authors to measure completeness of a data set, as shown below.

\[
\text{Structural completeness} = \frac{(\text{values that are recorded})}{(\text{values that could have been recorded})},
\]

where the ‘values that are recorded’ are the non-null values in the data set under measure, and the ‘values that could have been recorded’ are the reference data set that has no nulls (where all attributes are mandatory). As in the measures by
Fox, Levitin and Redman (see [FLR94]) and Pipino, Lee and Wang (see [PLW02]), the reference data set is defined implicitly from the data set under measure. The authors suggest that different granularities of views (such as an entire database, a table or a record) and different formats of data sets (i.e. spreadsheet or file) can be supported within a category. However, no further discussion has been provided on how the different requirements for NBC were treated.

Scannapieco and Batini [SB04] extended the NBC measures by introducing more granularity levels from which NBC can be viewed (beyond the column level proposed in [PLW02]). Four granularity levels were introduced, namely the value, tuple, attribute and relation levels. NBC is measured at each granularity level by the binary measure (called as boolean function) or by the ‘aggregate’ measure that computes the percentage of the non nulls within the data set under measure. The reference data set was implicitly defined from the data set under measure, with an assumption that the data set under measure should be free from nulls, which is similar to the NBC measures presented earlier.

A study by Sampaio and Sampaio which was conducted in the context of web query processing also focuses on nulls (called missing instance values), but within XML documents that are queried from a web source [SS07]. Similar to the NBC measures presented so far, ‘aggregate’ measure is used in this study where, NBC is determined based on the information on the number of missing instance values within the query results relative to the total number of instance values (which is the reference data set). In this study, nulls were viewed from the query results (which is the data set under measure), and the reference data set is defined implicitly from the query results, with an assumption that the query results should be free from nulls. The study however, assumes that the web source could provide information on the missing instance values that may be supported by an established collaboration with the web sources (e.g., within the context of a cooperative web information system [MSV+02] and a shared workspace system [BHST95]).

Naumann, Freytag and Leser [NFL04] proposed an NBC measure called a density measure. Within the context of a virtual data integration of multiple data sources, the notion of a union schema that consists of the schema of relations (of all data sources) was used. Each data source that takes part in the integration contributes one or more relations. A universal relation was introduced, which consists of the union of relations of data sources whose schemas are present in
the union schema. The density measure is an ‘aggregate’ measure which is based on counts of non-nulls in the data set under measure, viewed from attribute view, source view and query view. The density of an attribute was defined as “the ratio of non-null values in the attribute to all values in the attribute”. The density of a source was defined as “the average density of all attributes that appear in the global schema”, while the density of a query was defined as “the average density of all attributes that appear in the query”. Similar to the other NBC measures presented in this section, the reference data set (that comes from the universal relation) is implicitly defined as a data set that has no nulls (with attribute(s) that is/are mandatory).

2.3.2 Tuple-Based Completeness (TBC)

Most studies on TBC have been performed within the context of the relational model.

The TBC measure proposed by Motro and Rakov [Mot89, MR98] is not only useful for detecting missing tuples, but it also helps to determine whether the tuples are accurate. TBC, in their proposal was viewed from a database level and is defined as an ‘aggregate’ measure as follows:

\[
\text{Completeness (of the database relative to the real world)} = \frac{|D \cap W|}{|W|},
\]

where \(D\) is the actual stored database instance while \(W\) is the ideal, real world database instance. From this definition, we gain an important insight into completeness which is completeness can be affected by the presence of errors in the data set. \(W\) in the definition represents not only a reference data set that is complete, but also a reference data set that is accurate. Nevertheless, because \(W\) is very unlikely to be acquired, the measure used the sample of \(W\) which came from alternative databases or judicious sampling (where the verification of the samples is made by humans) [MR98].

Fox, Levitin and Redman also proposed a TBC measure (in addition to NBC) where, in their study, a tuple is defined as a collection of triples \(<e,a,v>\) that belong to the same entity (denoted as \(e\)), where \(v\) is the value for the entity’s attribute \(a\) [FLR94]. The authors stated that, a tuple is missing if the triplet is missing entirely. An ‘aggregate’ measure is used to measure TBC, where the authors stated that the fraction of triples that are missing from the ‘data collection’ is computed against the number of triples in the data collection. The reference
data set in this measure is the data collection, however, no further description of how the data collection was obtained was documented.

Naumann, Freytag and Leser proposed a TBC measure called a coverage measure (in addition to the density measure as described earlier in Section 2.3.1) [NFL04] which is an ‘aggregate’ measure. As in the density measure, the reference data set used in the coverage measure is derived from the universal relation. TBC was viewed from the data source level, where the coverage measure was defined as the ratio of the number of records in the data source to the number of records of the universal relation.

Scannapieco and Batini dealt with TBC which was viewed from a relation level [SB04]. Like other TBC measures presented earlier in this section, an ‘aggregate’ measure is used in which, the completeness of a data set under measure (a relation) was defined as the ratio of the number of tuples it contains to the number of tuples of the reference data set (called a reference relation). By assuming the number of tuples of the reference data set is known, this study extended TBC by proposing ways to determine completeness of a data set under measure based on the knowledge on the completeness of the relations from which the data set under measure is derived (called defining relations by Scannapieco and Batini). This extension of the TBC measure is particularly useful within the context where the reference data set for the data set under measure is unknown or not available (that prevents a TBC measure), and the information about the completeness of its defining relations is available to determine completeness of the data set under measure. Taking the example given by the authors, given that $D$ is the data set under measure, $R_1$ and $R_2$ are the defining relations of $D$, where, $D = R_1 \cup R_2$. Consider further that the completeness ratio of $R_1$ is 0.5 and the completeness ratio of $R_2$ is 0.5. In the case where $R_1$ and $R_2$ are disjoint, completeness of $D$ is defined as the sum of the completeness ratio of $R_1$ and $R_2$, which is equal to 1.

However, the extension to TBC proposed by Scannapieco and Batini may introduce TBC measurement complexity as knowledge about how those defining relations are related (i.e., overlap or disjoint) and about how the data set under measure is derived from the defining relations (i.e., through union, intersection and cartesian product operators) is required. In addition, we also need to know whether the same reference data set is used to measure completeness of the defining relations or not, as how TBC is determined for the data set under measure is
not the same for both cases. For example, if different reference data sets are used to measure completeness of $R_1$ and $R_2$, completeness of $D$ cannot be determined by simply adding the completeness ratio of $R_1$ and $R_2$, even though $R_1$ and $R_2$ are disjoint.

In a study of TBC by Fan and Geerts, the notion of master data was used as the reference data set in determining TBC from the view of a query level [FG10]. The authors stated that, if a query is submitted against the reference data set, the result of the query (which is a set of tuples) is complete. The study addressed the problem of determining TBC for queries submitted against a database that consists of not only the tuples that fully overlap with the reference data set but also other data sets which may be incomplete. Nevertheless, this study does not attempt to compute how many tuples are missing from the query answer (which is the data set under measure), but rather it defines the characteristics of queries that will yield complete answers based on the information of the queried data sets. TBC is not measured based on a mathematical equation such as the simple ratios, but is determined based on the evaluation of the content of the queried data sets and the queries.

### 2.3.3 Schema-Based Completeness (SBC)

SBC is called “model completeness” by Sampaio and Sampaio who defined it as “the measure of how appropriate the schema of the database is for a particular application”. From an XML point of view, Sampaio and Sampaio defined SBC as the number of missing attributes relative to the total number of attributes [SS07]. The definition of SBC given by Pipino, Lee and Wang states that SBC is “the degree to which entities and attributes are not missing from the schema” [PLW02].

Both definitions tell us that the SBC measures are the ‘aggregate’ measures, where attributes and entities are the views from which SBC can be assessed. For SBC proposals, there is a notion of database schemas with complete entities and attributes that are used as reference, but the details of how SBC is actually measured in practice are missing from the literature. Another limitation of the SBC literature is that the explanations of where these reference database schemas come from and how they are defined are missing.
2.3.4 Towards Population-Based Completeness (PBC)

To the best of our knowledge, the first recorded use of the term ‘population’ in connection with completeness is in a proposal by Pipino, Lee and Wang [PLW02]. The authors did not provide a formal definition of the PBC measure, but hinted at the presence of this useful concept through an example. In the example, the authors stated that, “If a column should contain at least one occurrence of all 50 states, but only contains 43 states, then we have population incompleteness” [PLW02]. From the example, we observe that there is a data set under measure (from state column) in which its completeness is determined by the number of missing ‘individuals’ (the states) from a ‘reference population’ (a set of 50 states). There is a notion of reference population that is used to represent a population that consists of complete individuals.

Another example, provided by Scannapieco and Batini regarding Rome’s citizens [SB04] might represent a form of PBC. This is because there is a notion of a reference population, which is a citizen population based on the personal registry of Rome’s city council, and the data set under measure that consists of a set of Rome citizens derived from a company that stores Rome citizen information for the purpose of its business (in which its completeness is of concern). Based on the literature, we say that there are two elements must be present for PBC measurement which are the data set under measure and the reference population.

However, details of how PBC measurement is made in practice are missing from both proposals, especially in terms of how the reference populations are acquired and used. The elaboration of the concept of PBC therefore remains an open question for research in terms of the current literature.

2.4 Conclusions

This chapter provides background on studies that deal with completeness in the database community and in statistics, with some examples of completeness problems in several application domains. We surveyed definitions of completeness with the aim of learning the common question that arises from the definitions and the views from which completeness has been considered. Based on the ‘what is missing?’ question (that is found to be the common question asked), several types of missing ‘units’ (i.e., values, tuples, attributes) are present. We set to
observe several characteristics of completeness measures proposed in the literature such as the view(s) of ‘what is missing’ from the data set under measure, the reference data set used, how the reference data set is defined and the method used to measure completeness. From the survey of completeness measures we conclude that:

- Given the missing ‘unit’ of concern, a data set under measure can be viewed at different granularity levels (i.e., tuple level, relation level and source level) and an ‘aggregate’ measure is the most common method used to measure completeness. Most of completeness measures are objective, mathematical measures but only some of the measures provide a formal definition of the completeness measure formula (notably [BP03, SB04, Mot89, NFL04]).

- Every measure requires a reference data set, that is considered to be complete. However, in most studies, the reference data sets have been assumed to be available and we cannot learn much from the literature about how reference data sets are defined or acquired. Motro and Rakov point out the fact that a true reference data set is difficult to construct, and suggested one method based on judicious sampling of alternative databases [MR98]. While this could be a sensible way to address the complexity of establishing a reference data set, no details were provided about how the sampling can be done.

- For NBC, a separate reference data set is unnecessary as it can be determined from the data set under measure itself, where the reference data set has been implicitly stated in most completeness measures of NBC. Most other completeness measures, however, require reference data sets that are separate from the data set under measure. While the need for a reference data set in completeness measures has been clearly stated in the completeness measure proposals, the complexity of acquiring (or providing) the reference data sets needed in completeness measurements has not been addressed by those proposals, either for reference data sets that are separate from the data set under measure or for reference data sets that are retrievable from the data sets under measure.

New research questions emerge, for example, what are the practical issues that might arise in acquiring (or providing) reference data sets, and how can we deal with those issues?
The gaps in the completeness literature, as stated above, can be seen for completeness measures of all categories. However, we found that the study of PBC is very limited as compared to other categories of completeness measures. Yet, we expect that PBC has much more to offer than its limited coverage suggests. Thus in this thesis, we address the gaps in completeness studies from the context of PBC.

In the next chapter (Chapter 3), we will address the first two gaps laid out in the bullet points above. In particular, Chapter 3 is dedicated to defining PBC beyond the description provided in the literature, where the questions of ‘What is PBC?’, ‘What are the elements required to measure PBC?’ and ‘How can we define PBC measures?’ will be answered. In Chapter 4, the third gap regarding the technical issues of PBC measurements will be addressed.
Chapter 3

Defining Population-Based Completeness

In this chapter, we will propose a model of population-based completeness (PBC) and describe its characteristics as a form of quality measurement. In the previous chapter, we identified what is required to measure PBC at a very basic level, where the data set under measure and the reference population were identified as the ‘elements’ of PBC. In this chapter, we aim to identify other elements of PBC in addition to the data set under measure and the reference population that are essential for PBC measurements (Section 3.1). To present precise definitions of PBC elements, we will instantiate the elements of PBC in terms of a specific data model (the relational model) and identify the possible types of PBC measurement request with the aim of understanding the information that we need to store about PBC elements to fulfill each type (Section 3.2). We will propose a general PBC measurement formula that supports all the types of PBC measurement request identified in this chapter (Section 3.2). To illustrate the usefulness of the PBC model, we will provide a PBC measurement example in the microbial genomics domain (Section 3.3). We will summarise the whole chapter with the questions that have been addressed, highlight several questions that remain unanswered and conclude (Section 3.4).

3.1 The PBC Model

In this section, we present a model of PBC that describes the elements that are essential for PBC measurement.
CHAPTER 3. DEFINING POPULATION-BASED COMPLETENESS

The examples that hinted at PBC given by Pipino, Lee and Wang [PLW02] and Scannapieco and Batini [SB04] help us to understand that the authors have a similar concern to each other, which is on the ‘individuals’ that are missing from a population. They help us to see that completeness is not only about counting nulls or missing tuples in data sets (which receives the most literature coverage). We observe from the examples that, to measure PBC, we need data sets (to measure) and ‘reference’ populations. An explanation of how data sets under measure and their reference populations are used in terms of a formal measurement definition is, however, missing from the literature.

As presented in Chapter 2, Motro and Rakov proposed a TBC measure [Mot89, MR98], where the formal definition of the measure is as a simple ratio method. We apply a similar form of simple ratio method in our PBC measure and define a basic PBC measurement as:

\[ \text{Completeness}(D, RP) = \frac{|D \cap RP|}{|RP|} \in [0, 1], \]  

(3.1)

where \( D \) is the data set under measure, and \( RP \) is the reference population.

We can see from Equation (3.1) that measuring PBC is conceptually simple as we only need a data set to measure and a reference population. Nevertheless, to make the measurement workable in practice, we need to know more about the populations. The question of how the reference populations can be acquired is essential, especially in the context of PBC measurement providers. The literature neither offers answers for this question, nor for the other basic research questions that we identified as listed below:

1. What is a population?

2. What characterises individuals that are suitable to act as the members of populations for PBC?

3. Is it feasible to gather true, complete reference populations?

4. What information must be stored about a reference population?

In the next section, we will address the first two research questions stated above.
3.1.1 Populations

The term population is used widely, especially in statistical studies. Statisticians define a population as the entire collection of items that form the subject of a study and that share common features [Knu71]. Within the statistical studies themselves, the definition of a population however is often specific to the application domain. For example, statistical studies in the biological and ecological domains define a population as a group of organisms of the same species in a given area ([Has06, DD62, MWU93, Nei73]). In census studies, a population is defined as the people who inhabit a territory or a state [LM04]. These items, species or people are the ‘individuals’ that belong to their defined population. In philosophy, the term natural kind is used for ‘grouping or ordering that does not depend on humans’, which is the opposite for the term artificial kind used for grouping of arbitrary things made by human [BT10]. Inspired from the observation of how populations are defined in the literature and from the philosophical domain, we define population as a set of individuals of a natural kind and these individuals are the real world individuals (not the artificial individuals created by humans).

In statistics, populations are used as a source of samples from which an inference about the characteristic of the entire population can be made [Knu71]. A question that arises is: what characterises the individuals that are suitable to act as the members of populations for PBC?

Pipino, Lee and Wang pointed out in their example that, the data set that they examined are retrieved from a specific column (states) [PLW02]. This provides us with a hint that only certain attribute of a data set might be of interest and will ‘make sense’ as the basis of a completeness measure. The instances in state column are therefore the data set under measure that is of interest in terms of its completeness. Thus in the example, the individuals that are suitable to act as the members of a population are a set of states. To identify the characteristics of the individuals that will act as the members of the population in addition to subjective judgment, we will next observe how data set under measure can be acquired from an artificial table.

We observe that not all columns consist of individuals that are suitable to act as the members of populations for PBC. For example, suppose that a resident table consists of resident records in the U.S, with schema:

\langle SSN, name, birthdate, contact number, house number, street, state, postcode, job \rangle,
where every resident is identified by a social security number (SSN). We identify
the possible population individuals that can be extracted from the columns by
examining whether, if taken individually, the columns could represent real world
entity types.

A column like house number even though it seems like a possible representa-
tive of an entity type (‘house’), might not consist of useful population individuals.
This is because, multiple houses may have the same house number, and therefore
it is hard to distinguish these houses based on the house number alone. The
same problem applies for street and postcode columns. Columns state and job,
however, if taken individually, can be used to represent real world entity types
(i.e., state and job). From the context of relational databases, these columns are
usually the ‘surrogate keys’ that match with the ‘primary key’ of another table.

Columns with the characteristics just described, could be used to extract the
individuals of a population that are useful for PBC measures. For example,
measuring completeness of the ‘state’ will help us to answer the question of ‘How
complete are the resident records in regard to the states?’ (i.e., do we have
resident records for all US states?) We say that the columns that consist of
individuals that will act as the members of populations are used to represent the
real world entity types.

So far, we answered the first two questions raised earlier that contribute to an
understanding of the populations. To answer the questions regarding the reference
populations, in the next section we will examine how the reference populations
can be acquired resulting in identification of all elements of PBC.

3.1.2 Reference Populations

The notion of reference populations is proposed as an essential element of PBC to
represent populations that are ‘complete’, i.e., that have no missing individuals.
The question is, how can we obtain the reference population? To answer this
question, we examine the possible forms of reference population motivated by
examples from the literature.

- In determining completeness of the Roman citizens data set, the set of
citizens who are registered by Rome’s city council is used as the reference
population [SB04]. In this example, the reference population is obtained
based on the citizens who are recorded in the registry.
However, if one wishes to use the *true* Roman citizen population as a reference, then the citizens who are registered by the city council might be incomplete to form a reference population. This is the case when there exist citizens who are not registered with the city council, but with another institution that handles citizenship applications from people of other countries (such as The Italian Ministry of Interior\(^1\)). By definition, the true Roman citizen population must consist of *all* real world citizens of Rome that exist. If the true Rome citizen population is desired, then we need to acquire the citizens from *all sources* that have them. Gathering the true set of Roman citizens could be difficult especially in the case where existing citizens are not registered with any governing bodies (thus no records about them exist), as it may require manual searching for those citizens, and manual data entry. Therefore in this case, we need an approximate reference population that could represent the true Rome citizen population.

- Recall from Chapter 1 that bioinformaticians were concerned about completeness of the human gene population that is used for an analysis for genes that cause diseases [TANPI09]. Several gene sources such as public genome databases, gene expression databases, data on gene regulatory networks and pathways, as well as biomedical literature were used to retrieve the list of genes that became the reference population [TANPI09]. The reference population used in this study is the integration of several sets of genes from a variety of gene sources (identified by the bioinformaticians in the domain).

However, the reference population used in the analysis is not the *true* human gene population unless it consists of all real world human genes that exist. Because there is still a debate on the actual number of human genes among the scientists, and more time is required to discover true human genes, due to the complexity of the gene discovery process [Sci08], the usage of the true gene population is not possible in this example. Therefore, as the alternative, we must find an approximate population to represent the true gene population.

From the examples, we observe that two forms of reference populations are possible: 1) the *true* populations that consist of all real world individuals that exist, and, 2) the *approximate* populations that are used to represent the true

\(^1\)http://www.interno.it/mininterno/export/sites/default/en/index.html
populations and which are more easily available. In addition, we also observe that, the individuals of a reference population may come from multiple sources.

Within an application domain, we say that the decision regarding which form of reference population is to be used must be made by the domain experts (e.g., by bioinformaticians) due to their knowledge of the sources of the populations. The decision to use approximate populations in the examples above is driven by the costs/difficulties of acquiring the true populations, and the questionable benefits of the small differences in measurements that would result. If this is the case, the approximate populations used must be adequate for determining completeness of data sets within the domain. However, we suspect that the main reason approximate populations are used is that true populations are not feasible to obtain, even though there may be a need to use them.

For example, the usage of the true human gene population is hindered by the challenging task of discovering all human genes that exist (i.e., through microarray analysis [YISH06]). In addition, there is also uncertainty as to when information about the true gene population will become available (as gene discovery can take a long time [Sci08]). Ideally, the true human gene population would be used in the analysis of the genes that cause diseases. For example, it was stated in ScienceDaily news that: “An accurate gene count can help identify the locations of genes and their functions, an important step in translating genomic information into biomedical advances” [Sci08]. This statement pointed to the importance of the discovery effort to find ‘accurate’ genes, that hinted the need to use true populations, at least in the context of genome research. Therefore, in many cases where true populations are not feasible, approximate populations are used as alternatives that are convincing enough to represent the true populations within an application domain.

The situation where there exists a single source that contains a good approximation of the true population is limited however (an exception being the Genbank database\(^2\) that contains the genes with good evidence of their existence). We propose that good approximate populations should be established by integrating individuals from a range of sources. To describe approximate populations established from integrated sources, we adopt the term *universe of discourse* (UoD) or in short *universe*\(^3\). Conceptually, a universe consists of a collection of


\(^3\)The term *universe* was introduced by Augustus De Morgan in 1846 in formal logics to represent the collection of objects under discussion of a specific discourse [Mad91].
approximate populations within an application domain used for PBC measurements, that is built by integrating individuals from several incomplete sources for the populations.

We use the term contributing sources (CSs) for sources that contribute to the reference populations in the universe. The CSs could be in multiple forms, such as databases (private and public) e.g., observation databases from gene regulatory networks and pathways [TANPI09] or published literature. As it is crucial to understand (and to manage) the relationship between the CSs and the reference populations for successful integration, we propose a structure called a population map. Conceptually, a population map consists of a mapping between a reference population and its CSs. If a reference population is stored as a table, and the CSs are databases, we say that a population map is a mapping between a reference population table and queries over tables on CSs. Note that, as the schema of the universe may not be the same as the schema of the CSs, the designers of the population maps must consider the differences.

We use an ER diagram as shown in Figure 3.1 to illustrate the relationships among the elements of PBC. The relationships among the elements of the PBC model are as follows:

- Each reference population is stored in a universe, and each universe consists of one or more reference populations.

- Each reference population has a population map for each CS from which its individuals are taken. Every population map must refer to at least one reference population.

- Each CS has a population map for each reference population to which it contributes. Every population map must refer to at least one CS.

We answered in this section, the questions of how we define reference populations (by examining their types), how to acquire them and also whether it is feasible to gather ‘complete’ reference populations. However, knowing the elements of the PBC model is not enough to measure PBC until we know about ‘what information must be stored about them and how to use the information’. To facilitate precise definition of PBC elements, in the next section, we will present the instantiation of PBC elements in terms of a data model.
3.2 Instantiation of PBC Elements

In this section, descriptions of the information that we need to store about the PBC elements will be given in relational database terms. However, this does not mean that PBC is restricted to a particular data model. Using the relational model is only a pragmatic choice that aims to show how the conceptual PBC elements can be instantiated in concrete terms. We begin this section by identifying the types of PBC measurement request that can be asked with the aim of determining the information that we need to store about the reference populations. A general form PBC measurement formula will be proposed to support all types of PBC measurement requests identified. Following a description of the information that we need for reference populations is the proposal of the information that we need for other elements of the PBC model. As a result of the identification of the information that we need for the PBC elements, a template for basic configuration of the elements in the PBC model will be presented at the end of this section.
3.2.1 Types of PBC Measurement Requests

Identification of the possible types of PBC measurement requests could guide us to determine the information that we need to store about PBC elements. We start with identifying the possible types of PBC measurement requests using the bioinformatics example (by Tiffin et al. [TANPI09]) given in the previous section as a motivation for our search in determining the information that we need to store about reference populations.

Suppose that the reference gene population is stored as a table in the universe, and consists of genes that are integrated from multiple CSs. In a normal ‘base’ case, the person requesting the measurement provides his/her own data sets to be measured, and specifies the universe’s reference population to use for comparison. In this normal case, we regard the data set under measure as ‘external’ to the universe because the data set is not obtained from the universe; the reference population as ‘internal’ as it is obtained from the universe.

The simplest type of PBC measurement request is to measure completeness of an external data set under measure, relative to a reference population in the universe. For example, as our bioinformatics example, the request is to measure completeness of a gene data set (e.g., the data set that a scientist has produced to use as input to some analyses gathered from a file in his/her own machine), relative to a reference gene population already set up in the universe. This simplest type of PBC measurement request only requires the identifier of the population as the information that must be stored about the reference population. For this type of PBC measurement, the schema of a reference population table $p$ in the universe must consist of a set of identifier attributes $I$ (that could be a singleton set), where $P = \langle I \rangle$. As genes are usually identified by their names (e.g., “REV7” is a name used for a gene belonging to a species of yeast called *Saccharomyces cerevisiae* [Gen]), we add gene name in the schema of the reference population table that represents the identifier of the gene population. The PBC measurement formula used to answer this type of request is:

$$\text{Completeness}(\langle ExtD \rangle, (p)) = \frac{|ExtD \cap \Pi_{\text{key}(p)}(p)|}{|\Pi_{\text{key}(p)}(p)|}, \quad (3.2)$$

where $ExtD$ is the external data set under measure specified in form of an actual set of identifiers of the population and $p$ is the name of the reference population (i.e., the name of the reference population table) and $key(p)$ is a function that...
retrieves the identifiers (I) from p where information about I is in the configuration information stored in the metadata about the universe (we will discuss about the configuration information of PBC elements later in this section).

The next plausible type of PBC measurement request is to measure completeness of an external data set under measure relative to a reference sub-population in the universe. A sub-population is a ‘subset’ of individuals of a full population. For example, as opposed to all genes of all species in the previous example, the request is to measure completeness of a mouse gene data set (e.g., the data set gathered from scientists’ own source) relative to a reference mouse gene population (i.e., a gene sub-population) already set up in the universe. For this type of request, we need information regarding the species in addition to the gene names, in order to retrieve the reference sub-population. Therefore, the schema of the table where the reference gene population is stored must have both gene name and species information. This type of PBC measurement request requires the identifier of the population and the information that can be used to extract the sub-population. Therefore, we must add another component to the schema for each reference population: the attribute(s) containing the instances used to extract the sub-population. Considering the additional attribute(s), the schema of a reference population table p is $P = \langle I, A \rangle$, where A is the set of attributes used to extract sub-populations from the population table. The PBC measurement formula used to answer this type of request is:

$$\text{Completeness}((\text{ExtD}), (p, \text{COND})) = \frac{|\text{ExtD} \cap (\Pi_{\text{key}}(\sigma_{\text{COND}}p))|}{|\Pi_{\text{key}}(\sigma_{\text{COND}}p)|}, \quad (3.3)$$

where COND is a conjunction of conditions on p in which A is used as the predicate in COND.

Another plausible type of PBC measurement request is to measure completeness of an external data set relative to a reference sub-population whose individuals originated from only a ‘subset’ of the CSs. This type of PBC measurement is driven by the need to use specific CSs that are preferred by the person requesting the measurement (for example due to known source’s reputation and quality), instead of other CSs identified by the PBC measurement provider. For this type of request, we need information regarding the CSs that can be used to extract the individuals set. Therefore, we must add another component to the schema for each reference population: the source attribute containing the instances of the CSs used to extract the individuals of a ‘subset’ of the CSs. Considering
the additional source attribute, the schema of a reference population table $p$ is $P = \langle I, A, \text{source} \rangle$, where \text{source} is the attribute used to extract individuals that originated from specific CSs. The PBC measurement formula used to answer this type of request is identical with the one presented previously in Equation (3.3), where \text{source} is used as the predicate in $COND$.

Beyond the normal base case of PBC measurement requests, we envisage the case where the data set under measure is a combination of the individuals gathered from external sources (e.g., from scientists’ own lab’s file) and the individuals that can be gathered from the universe. For example, in addition to his/her own mouse gene data set, the person requesting PBC measurement might be interested with the mouse gene data set gathered from certain CSs to be measured. The schema of a reference population table $p$ is $P = \langle I, A, \text{source} \rangle$ is adequate to answer this type of PBC measurement request. The PBC measurement formula used to answer this type of request is:

$$\text{Completeness}(\langle \text{ExtD}, p, COND \rangle, \langle p, COND \rangle) = \frac{|(\text{ExtD} \cup \Pi_{\text{key}(p)}(\sigma_{\text{COND}})) \cap (\Pi_{\text{key}(p)}(\sigma_{\text{COND}}))|}{|\Pi_{\text{key}(p)}(\sigma_{\text{COND}})|}. \quad (3.4)$$

Another case that we envisage is to measure the data set under measure (i.e. external data set or mix of external and internal data set) relative to a reference population whose individuals are obtained from the universe and from external sources. For example, bioinformaticians may want to measure completeness of mouse genes relative to mouse genes that can obtained from both: the universe and external source e.g., private labs that recently discovered new mouse genes that have not been covered yet by the universe. The PBC measurement formula used to answer this type of request is:

$$\text{Completeness}(\langle \text{ExtD}, p, COND \rangle, \langle \text{ExtRP}, p, COND \rangle) = \frac{|(\text{ExtD} \cup \Pi_{\text{key}(p)}(\sigma_{\text{COND}})) \cap (\text{ExtRP} \cup \Pi_{\text{key}(p)}(\sigma_{\text{COND}}))|}{|\text{ExtRP} \cup \Pi_{\text{key}(p)}(\sigma_{\text{COND}})|}, \quad (3.5)$$

where $\text{ExtRP}$ is an external reference population specified in form of an actual set of identifiers of the population.

Depending on the type of PBC measurement request, the person requesting the PBC measurement needs to provide the required parameters stated in the PBC function. The internal parameters are specified by their names (e.g., the name of population and the name of the source) by the person requesting the PBC measurement, while the external data sets (i.e. data set under measure or
reference population) are specified as a set of identifiers of the chosen population.

By identifying the possible types of PBC measurement request, we come to conclude that: a single schema form for all reference populations that can support all envisaged types of PBC measurement request is \( P = \langle I, \text{source}, A \rangle \); the PBC measurement formula presented in Equation (3.5) is the general formula to be used to compute all envisaged types of PBC measurement request identified in this section.

Note that the types of request identified are in cases where at least either the data set under measure or the reference population requested consists of individuals that can be provided by the universe. Having said that, it does not mean that measurement requests involving data set under measure and reference populations that are both external to the universe cannot be supported. This case is excluded from the types of PBC measurement requests identified above as determining what information must be stored about those populations is unnecessary. This is because, the populations are provided by the person requesting the measurements (instead of the universe).

The lesson learnt from identification of the PBC measurement types above is that the reference population in the universe is either used in full or partially. Unlike the reference population, a reference sub-population that is partially used has either one or more constraints (conditions) specified upon it, through attributes other than the identifiers. In addition, for every PBC measurement that requires using individuals from the universe, the person requesting the PBC measurement must provide some information such as the name of the reference population, and the conditions of interest.

In addition to the information about populations, we also need to identify the information that we need for the universe, the CSs and the population maps before the PBC measurements can be provided. For each universe, information that we need is about the reference populations that it provides. Information about the universe indicates the types of PBC measurement requests that are supported (and those unsupported) which will determine whether the person requesting the PBC measurements needs to provide his/her own populations. If the reference populations are stored as tables, we need to know the schema of the tables, and also the CSs of the individuals for each population. To track the mapping between the CSs and the tables of the universe, a population map must include the names of the tables and the queries on the tables of the CSs to
retrieve population individuals. As for the CSs themselves, we need information about the names of the CSs and the mappings they have with the tables in the universe. We also need information of how the CSs can be accessed. For example, if a CS is web database, we need to keep its URI (Uniform Resource Identifier) that will be used during the process to extract the population individuals (e.g. during the setup or maintenance of the universe).

We define information about PBC elements in the form of ‘variables’ and ‘types’, and call them the basic configuration of the PBC model. Table 3.1 presents the reference template of the basic configuration of PBC. In particular, Table 3.1(a) shows two basic variables for population tables and CSs, while Table 3.1(b) presents the definitions of the types of the basic variables as well as other types. These variables when given values describe a particular domain specific PBC system. The template of the basic configuration of PBC serves as a reference for PBC measurement providers to configure PBC elements that are specific to each application domain. The implementation of PBC as a system in which the reference template of the basic configuration of PBC will be described in the next chapter. PBC elements’ configuration information are kept as metadata that will be used during the implementation of PBC.

Table 3.1: Basic Configuration of a PBC system (basicConfig)

<table>
<thead>
<tr>
<th>The variables and their types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS : P(SOURCE)</td>
<td>The set of all CSs that populate the universe.</td>
</tr>
<tr>
<td>UP : P(NAME × POPSCHEMA)</td>
<td>The set of all population tables in the universe, where each element is a double representing the name of the population table and its schema.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The basic types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME: P(STRING)</td>
<td>The set of all legal identifying names of CSs, tables and attributes.</td>
</tr>
<tr>
<td>QUERY: P(STRING)</td>
<td>The set of all queries on population tables or on CSs in the form of SQL strings. For brevity, we assume that all queries in QUERY are legal queries against the schema of the population tables (or the schema of CSs). In practice, checks for this would have to be made in the algorithms.</td>
</tr>
<tr>
<td>POPSCHEMA:</td>
<td>The set of all possible population table schemas where each element is a triple representing the name(s) of population’s identifier attributes, the name of an attribute for the CS and the names of attributes other than identifiers and CS.</td>
</tr>
<tr>
<td>SOURCE: P(NAME × URI × POPMAP)</td>
<td>The set of all possible source descriptions where each element is a triple representing a specific source, and consisting of the name of the CS, its URI for executing queries, and a mapping between the population tables in the universe and the tables in the source’s schema.</td>
</tr>
</tbody>
</table>

To show how PBC measurement requests can be supported by the information
that we proposed to be included in the general reference population table schema, we will provide an example of PBC measurements in Microbial Genomics in the next section.

### 3.3 Example of PBC Measurements in Microbial Genomics

In this section, we will first give a background of microbial genomics where concerns regarding PBC problems arise. To show how the basic configuration of the PBC model and the general PBC measurement formula proposed can be used to support PBC measurement providers answering PBC measurement requests, we need to identify the types of requests that are inherent in the microbial domain.

#### 3.3.1 Background

The study of the genomes of microbes, called microbial genomics, helps pharmaceutical researchers gain a better understanding of how pathogens cause disease [Fal98]. By understanding the association between pathogens and diseases, further analyses, such as regarding pathogens' resistance to drugs or antibiotics, can be conducted in search of a cure for specific diseases.

To explain the PBC problems in the microbial domain, we observed the relationships among the subjects of microbial studies that have been documented in the literature and we produced Figure 3.2 as the result of these observations. The left side of the figure depicts an ER diagram with five subjects of microbial studies, namely *Microbe, Genome sequence, Gene, Infectious disease* and *Antimicrobial agent/vaccine*, and their relationships. Each relationship between the subjects is related to an analysis within the pathways of microbial genomics (the right side of the figure), shown by a dotted line. The analyses are conducted by the scientists in the wet lab through experiments, or by the bioinformaticians in the dry lab with the support of computational tools [FES00, Ile08].

Microbial genome sequencing analysis is conducted in order to determine the genome sequence for the microbes. At this stage, scientists in the wet lab perform several tasks, such as DNA isolation and purification, DNA fragmentation and DNA assembly, before the complete genome sequence can be discovered using an automated tool (such as shotgun sequencing) based on the raw DNA sequence
Figure 3.2: The Relationship Between the Various Subjects of the Microbial Study and the Analyses in the Microbial Study Pathway
Based on the microbial genome sequence, the genes that exist in the microbes are determined through gene mapping analysis, conducted by bioinformaticians using computational gene finder tools [FES00, Jen98].

In order to identify the genes associated with the infectious disease, microbial genome-wide association analysis is conducted. Before any conclusion can be drawn regarding the genes that cause the infectious disease, the data analyst must identify the genes that they think are plausible to be the cause of the disease, called as ‘candidate genes’. Association analysis relies on the results of a genome context analysis, that provides the functions of the genes [OO03]. However, if the resultant set of functions is incomplete, it is hard for the data analyst to generate a complete set of candidate genes for further testing. As a consequence, incomplete gene candidates for association analysis will be used and some diseases might not be associated with their actual causal genes.

Given the set of candidate genes and their functions, the bioinformaticians try to link or associate each candidate gene with a disease based on the frequency of the candidate gene occurrences for a targeted disease [OO03]. Even though complete information on gene function can be provided, problems in this area have been described as “the persistent lack of replicability...of linkage between loci and complex diseases” [RM96] which means an incomplete set of gene-disease associations will make detection of genetic factors of complex diseases (such as schizophrenia, bipolar disorder, and diabetes) more difficult. The awareness of the completeness problem in the analysis motivates the researcher to favour the gene detection model used in association analysis over the detection model used in the linkage analysis, on account of the more complete results it provides [Ile08, RM96].

Another related analysis in the microbial study pathway is conducted to search for the cure for the diseases caused by pathogens. In addition to the microbial resistance analysis [Dav07], protein analysis has been used to search for antimicrobial agents and vaccines for the infectious diseases caused by the pathogens [FES00, Jen98].

The results of the analyses in the microbial study pathway are deposited into public databases (owned by research organisations, universities or health institutes) and private databases (owned by small or large private companies) [Jen98]. Microbial genome sequencing analysis, for example, produces microbial genome sequence data sets, which are deposited into a database that belongs to

---

4An analysis conducted to show how each protein works in a cell or tissue works [Vin06].
the research institution carrying out the study (e.g., Comprehensive Microbial Resource\textsuperscript{5}, Microbial Genome Database\textsuperscript{6}) or into a public database like Genbank\textsuperscript{7}. Furthermore, the availability of sophisticated experimental methods has increased the number of results of genome analyses [TANPI09] and consequently, the databases where the results are deposited are growing.

We observe that for every analysis in the microbial study pathway shown in Figure 3.2, the scientists/bioinformaticians need to prepare an input data set describing the subjects of interest (e.g., \textit{microbe and gene}) for the analysis. In general, in these analyses, the completeness of the input data set determines the completeness of the analysis result. Therefore, an important question that arises in this domain is regarding the completeness of these input data sets. However, not all information in these input data sets are of interest (in terms of completeness) as scientists often look at specific information that is important to them (i.e., completeness in regards to certain genes or species) - a scenario that hinted PBC problems that are inherent in the multiple stages of analysis in microbial domain as described.

The key lesson that we learnt based on the observation in microbial domain is on the applicability of the PBC concept to support answering PBC measurement requests from this domain.

### 3.3.2 PBC Measurement Requests in Microbial Genomics

The observation of the PBC problem faced by bioinformaticians in microbial genomics in the previous section suggests that the PBC measurement requests that are present in the domain fall at least into the first type of PBC measurement request. In handling PBC measurement requests for this domain, PBC measurement providers need to configure the elements of the PBC model that are specific for this domain.

To describe the configuration needed for reference populations, assume that the microbial universe consists of reference populations whose individuals are from databases (CSs) in the microbial domain identified by domain experts. PBC measurement providers need to configure the form of reference population table schema, together with the information that must be stored within the tables.

\textsuperscript{5}CMR-http://www.tigr.org/CMR
\textsuperscript{6}MBGD-http://mbgd.genome.ad.jp/
The basic configuration of the PBC model as depicted in Table 3.1 defines the type of reference population table schema (called the POPSHEMA) to be in the general form of: \( \langle I, \text{source}, A \rangle \) as this form of schema can support all types of PBC measurement requests identified in this chapter.

Suppose that the general form of the reference population table schema is adopted and the reference candidate gene population is configured as a table called \textit{gene} with schema: \( \langle \text{geneId, source, species} \rangle \). In addition to the reference population, we also need to configure the microbial universe, its CSs and the population maps that are specific for this domain. Based on the basic configuration of the PBC model, two variables have been defined as shown in Table 3.1(a) namely \( UP \) (the set of reference population tables in the universe and their schema) and \( CS \) (the set of CSs in the universe). The description of the types used for these variables is shown in Table 3.1(b). The following is an example of the instances of the variables configured and stored for PBC measurement in the microbial domain:

- \( UP = \{(\text{gene}, \langle \text{geneId, source, species} \rangle)\} \),
- \( CS = \{(\text{CMR, http://www.tigr.org/CMR, PM\text{CMR}})\} \), where \( PM\text{CMR} \) is a set of population maps for Comprehensive Microbial Resource (CMR) in the form of:

\[
\langle \text{gene, SELECT geneCode, speciesCode FROM microbeGene} \rangle
\]

Every instance of \( PM\text{CMR} \) consists of the name of the population table (that is equivalent to the name of the reference population it contributes), and the query against the table in the CS.

For brevity, we only show an instance for each variable.

Assuming that all elements of PBC have been configured for the microbial domain, we will next present several types of PBC measurement requests that can be supported by the PBC configuration.

We use the example of a request to measure completeness of a candidate gene population relative to the reference candidate gene population. We apply the PBC measurement formula as shown in Equation (3.5) (page 46) to measure PBC based on the request. The instantiation of the PBC measurement formula for the request is:

\[
Completeness(\text{ExtGENE, gene}) = \frac{|\text{ExtGENE}\cap \Pi_{key(\text{gene})}\text{gene}|}{|\Pi_{key(\text{gene})}\text{gene}|},
\]
where $\text{ExtGENE}$ is the external gene data set under measure and $\text{key}(\text{gene})$ is a function that retrieves $\text{geneId}$ (the identifier of genes) from $\text{gene}$. For this type of request, the person requesting the measurement must provide the gene data set under measure and the name of the reference population of interest (which is equivalent to the name of the population table that stores the reference population information).

For a request to measure completeness of a candidate gene population relative to the reference candidate gene population that consists of genes coming from certain CSs of the microbial universe only (e.g., CMR and SwissProt), PBC is measured as:

$$\text{Completeness}(\langle \text{ExtGENE} \rangle, \langle \text{gene}, \text{COND} \rangle) = \frac{|\text{ExtGENE} \cap (\Pi_{\text{key}(\text{gene})}(\sigma_{\text{COND}}\text{gene}))|}{|\Pi_{\text{key}(\text{gene})}(\sigma_{\text{COND}}\text{gene})|},$$

where $\text{COND}$ is a conjunction of conditions on $\text{gene}$ using $\text{source}$ attribute as the predicate. For example, one condition in $\text{COND}$ is specified as $\text{source \ IN (‘CMR’, ‘SwissProt’) in the query.}$

This type of request could be driven by the need to use a reference gene population that comes from a preferred source e.g. based on trust/reputation. Because not all CSs chosen by the PBC measurement provider are preferred by the person requesting the measurement, we need to filter the genes by specifying the condition on the $\text{source}$ predicate in the query. Recall that this type of reference population is called reference sub-population for the constraints (conditions) that are specified on the reference population (see Section 3.2.1).

Similarly, a reference sub-population is used for PBC measurement requests that require the usage of a reference gene sub-population (e.g., sub-population of gene by $\textit{S.bongori}$ species). But for this request, $\text{COND}$ is a conjunction of conditions on $\text{gene}$ using $\text{species}$ as the predicate. For example, $\text{COND}$ is specified as $\text{species=‘S.bongori’ in the query.}$ To use reference sub-populations, the person requesting the measurement must provide the conditions (such as the names of the CSs) in addition to the name of the reference population.

In this section, we showed the usability of the PBC model which supports some types of PBC measurement requests in the microbial domain. We presented how the elements of PBC can be configured and how the general PBC measurement formula can be used to support some plausible PBC measurement requests in this domain. We say that, even though the same elements of PBC are applicable for different application domains, specific configuration of the PBC elements are needed in order to fulfill PBC measurement requests for each domain. Basic
configuration of the PBC model has been proposed to facilitate PBC measurement providers to perform configuration of PBC elements of specific domains.

3.4 Conclusions

In this chapter, we presented answers for basic PBC questions outlined at the beginning of the chapter which contribute to an understanding of the characteristics of PBC that have not been addressed by any literature to date. We defined the elements required to measure PBC in which the identification of the types of PBC measurement requests facilitate determining the information that we need to store about reference populations. In addition, we proposed the basic configuration of the PBC model that will be of value for a PBC system implementation. A general PBC measurement formula is proposed to support the types of PBC measurement requests identified in this chapter. The following conclusions can be drawn from this chapter:

- Approximate reference populations are a typical form of reference populations used in many application domains where true reference populations are difficult to establish. Good approximate reference populations can be established by integrating individuals from a range of sources. The PBC model that we propose supports definition and configuration of such approximate reference populations that are particularly useful in the situation where a single source that contains good approximation of the true populations is limited (or does not exist).

- PBC measurement is conceptually simple because as specified in the basic PBC measurement formula, we only need two elements to make it works namely a data set under measure and a reference population. However, by defining the elements of the PBC model, we learnt that what is needed to measure PBC is beyond the two elements that are visible in the basic PBC measurement formula. Not only do we need to know what information to store about the elements, we also need to know how to configure them. The configuration information that we need to store about PBC elements is specific to a particular application domain. We proposed the basic configuration of the PBC model and a general PBC measurement formula that
is customisable to support PBC measurement providers in responding to PBC measurement requests from different application domains.

- The example of PBC measurements in the microbial domain shows the usability of the PBC model which supports PBC measurement requests within the domain. The basic configuration and the general PBC measurement formula that we proposed will be of benefit in other application domains that have similar concerns about PBC of the data sets that are heavily used for analyses such as in the microbial domain.

Having defined the PBC model, the basic configuration of PBC and the general PBC measurement formula, in the next chapter, we will turn to answer the question of how to make PBC implementable in practice. We set to determine software and hardware components needed to support PBC as a workable solution and also highlight the issues that are barriers to PBC implementation.
Chapter 4

Practical Analysis of PBC

In the previous chapter, we defined the notion of PBC measurement, showing the elements (and the information about them) that are needed to make the computation, and the formula used in PBC measurement. However, the concept of PBC will be of little use if the definitions given in Chapter 3 cannot be implemented in practice, in a usable system at ‘acceptable’ cost.

This is the issue that we explore in this chapter. In particular, we aim to answer the following questions:

- In practice, can the elements required for PBC measurements be acquired in full and with accuracy?
- Can universes be maintained over time, at their accuracy and completeness?
- If complete and accurate elements cannot be obtained, what is the effect on the accuracy of PBC measurements taken using the incomplete/inaccurate versions that are available?

The question of how acceptable the cost is, however, is subjective to the person(s) implementing the PBC system, as it cannot be assessed in absolute terms. Therefore, our focus in this chapter is not to define what is acceptable (or not), but rather to elucidate the costs involved for the PBC measurement providers to decide whether the costs are so high that they would prohibit any practical application of PBC. Among the elements of PBC identified, the main element of PBC that needs attention is the universe. We rely on the universe to determine the types of PBC measurement request that can be offered, based on the reference populations that it covers - the characteristic that makes the
question regarding the costs to build it should receive more attention. Thus, the universe is the main focus of much of this chapter.

The main outcome of this chapter is a list of open problems that must be solved in order for PBC measurements to become implementable in practice. In particular, we focus on the problems that are the barriers to providing accurate PBC measurements. As a side effect, we provide guidelines for PBC system stakeholders in implementing PBC through reference PBC architectures as proposed. The reference architecture outlines the technical requirements which are the inputs and processes required to perform PBC measurements.

The rest of this chapter is organised as follows. Section 4.1 discusses the motivation for conducting the analysis; Section 4.2 describes the types of reference architecture for PBC systems and identifies the technical requirements relative to these architectures; Section 4.3 discusses the problems of a PBC system within the main activities of the PBC system, namely setup, measurement and maintenance in accordance with the identified requirements; Section 4.4 presents the implementation of PBC as a system prototype; and Section 4.5 summarises the problems encountered and draws conclusions regarding the practicality of PBC.

### 4.1 Motivation

Typically, before any software systems can be implemented, system providers need to know about the costs that are involved. Similar to PBC system providers, information about the costs is needed, as they need to judge how acceptable the costs are, before agreeing to implement the system.

Costs are usually viewed as the amount of money that needs to be spent, which could be determined based on the 'efforts' required to set up and maintain the system (i.e. the infrastructure needed for the system). For a PBC system in particular, the efforts taken should support PBC measurement providers to answer PBC measurement requests and to calculating accurate answers for the requested PBC measurements.

In elucidating the efforts that are involved, one question that arises is, ‘What are the problems that will become the barriers for PBC measurement providers to calculate accurate answers?’ By identifying the problems, we could determine the efforts that must be provided to solve them - which can be converted into the financial costs that need to be considered by PBC system providers. The analysis
of the problems is therefore motivated by the need to elucidate the efforts that are plausible for a PBC system implementation.

To identify the problems, we need to understand the inputs and processes that are required within a PBC system. The elements of PBC identified in Chapter 3 are the inputs of a PBC system, in which the interactions among them are made through some processes that are grouped into three main activities that any PBC implementation must support. These activities are Setup, Measurement and Maintenance.

To facilitate identification of the problems, we created a use case diagram to elicit the use cases (that represent the main activities) of a PBC system and the possible interactions among the ‘actors’. To elicit the possible interactions among the main activities, we created a main activity diagram showing the plausible flows through them. For each main activity, we created an activity diagram with the inputs and processes that are specific to it, where the problems and the analysis of their effect on PBC accuracy will be presented in the sections to come.

In the next section, we will describe a PBC system with the aid of a use case diagram and a main activity diagram, followed by a proposal of reference architectures for PBC systems.

### 4.2 A PBC System

We modelled Setup, Measurement and Maintenance as the main activities that must be supported by a PBC system. These activities are shown as use cases in a use case diagram as shown in Figure 4.1. As a result of modelling the main activities of PBC, we identified the ‘actors’ of the system. These actors are the stakeholders of the system whose interest will enable PBC system to be implemented.

The actors and their interactions with the use cases are:

- **Data Analyst**: the end user of the system who initiates PBC measurement in order to support some data processing tasks that he or she is engaged in. For example, data analysts provide the data set to be measured and the specification of the reference population to use for measurement (e.g., the name of the reference population). This actor might also provide some inputs for the system’s maintenance (such as feedback about the performance of the measurement facilities).
• Domain Expert: a person who has knowledge of the populations and the contributing sources of the populations in the application domain, who contributes to Setup and Maintenance.

• System Investor: the person who owns the system and also decides on matters relating to financial cost during Setup and Maintenance.

• PBC Measurement Designer: the person who designs the PBC measures that will be supported by the system and the technical artefacts (e.g., schema) needed to implement them. The PBC Measurement Designer engages in the Setup and Maintenance of the PBC system.

• System Administrator: the person responsible for the maintenance aspect.

• Contributing Source: the source that contributes population individuals in the universe, that interacts during Setup and Maintenance. In some cases, the contributing source might be accessed during Measurement.

We created an activity diagram (Figure 4.2) in order to elicit the possible interactions among the main activities, showing the plausible flows through them. Note that [OK] and [Not OK] notations are used to indicate successful and unsuccessful termination of the activities. Among these activities, Setup is the necessary starting point that leads to Measurement and Maintenance. However, if Setup is unsuccessful, due to for example the schema of one or more CSs being inaccessible or the identified CSs no longer existing, we resume Setup until we can deal with the factors that caused the previous attempts to be unsuccessful.

We allow Measurement and Maintenance to occur concurrently as long as the Maintenance does not involve the data sets required by the Measurement. But in the case where the data sets required by the Measurement need to be maintained, the Maintenance needs to wait until the Measurement which is underway is completed. Maintenance must be resumed if it is interrupted for a reason, such as lack of storage space to add new populations. Later, in Section 4.3, we will present the processes under each main activity to analyse the problems that arise.

We have seen so far the actors of the PBC system and the main activities that the actors undertake. We have also described the interactions among the main activities. However, the questions that arise are: ‘What is the link among the main activities, the actors and the elements of PBC?’, and ‘What are the forms
Figure 4.1: Main Use Cases of a PBC System

Figure 4.2: Main Activities of a PBC system
of architecture of a PBC system that can be used?” To answer these questions, in the next section, we will present several forms of PBC reference architecture, for which the links among the main activities, the actors and the elements of PBC will be described.

### 4.2.1 A PBC Reference Architecture

Basically, all PBC system architectures consist of:

- **Software components:**
  - PBC measurement interface: a software interface for data analysts to issue PBC measurement requests (i.e. by specifying the data set to measure and the specification of the reference population to use) and to receive PBC measurement answers.
  - Measurement processor: a software program that calculates PBC measurement answers based on the PBC measurement requests by instantiating the PBC measurement formula.
  - Wrapper: software that interfaces the universe and the CSs, which is used to transform the queries against CSs that are understandable and executable by the CSs that they wrap.

- **Source components:**
  - Universe: a database that is either ‘virtual’ or ‘materialised’.
  - Contributing source: source of reference populations in the universe that is in certain forms such as databases (e.g., relational and XML), published literature, or observation data (e.g., data from gene regulatory networks and pathways [TANPI09]).
  - Population map: a structure that stores mappings between populations and their CSs.
  - PBC components configuration: information about the CSs (e.g., CSs name and URI) and the universe (e.g., the reference populations it contains).

The form of the universe selected will determine the type of PCB reference architecture used and the additional software component(s) needed for the architecture. We will first describe the types of reference architecture of PBC, before
CHAPTER 4. PRACTICAL ANALYSIS OF PBC

One type of PBC reference architecture, as shown in Figure 4.3, implements the universe as a virtual universe. With the virtual universe, it means that the individuals of the reference populations are extracted from the CSs on demand. In addition to the basic components, for this form of architecture, we need a processing layer called a mediator [Len02, Wie92] that performs data integration tasks on demand. The mediator receives queries for the reference population from the PBC measurement processor (data flow a3), which instantiates the queries based on PBC measurement requests (data flow a1). The mediator executes the queries based on the mappings specified in the population maps and based on CSs information in the PBC components configuration (data flow a5) to extract population individuals from each CS through the wrappers. The queries to extract the populations are shown as data flow a6. These queries are in the form of the schema of the universe (e.g., relational query). Each wrapper will transform the queries into the queries that are understandable by the local CSs (e.g., XML query), which relies on the type of the CSs (e.g., relational or XML databases). The wrappers also transform the query results (population individuals) into the
schema that is similar to the universe's schema (called structural transformation). We may rely on the mediator to perform semantic (and format) transformation on the query results to conform to the universe schema. Population individuals from the CSs (data flow a7) are therefore integrated and transformed by the mediator, before they can be passed to the PBC measurement processor for computation (data flow a4). The results of the measurement are sent to the person requesting the measurement (data flow a2).

Another type of PBC reference architecture, as shown in Figure 4.4, implements the universe as a fully materialised database. Instead of querying reference populations on an on-demand basis, the individuals of the reference populations that are extracted from the CSs (data flow a7) are stored within the universe. For this type of architecture, we need a software that could support up-front extraction of population individuals from multiple CSs, prior to the actual PBC measurements. The extraction process relies on the mappings between the populations and their CSs (retrieved from the population maps), and the information about PBC components configuration (data flow a5). The extracted data sets must also conform to the schema of the universe, before they can be loaded into the universe. Therefore, in addition to the basic components, we propose to include an ETL\(^1\) pipeline within the architecture that will support the tasks just mentioned. The wrappers are plugged into the ETL pipeline and the functionality of the wrappers in this architecture is the same as the wrappers in virtual universe reference architectures. The Transform component is responsible for performing semantic (and formatting) transformation and for integrating the population individuals, which is the same functionality that is performed by the mediator. The Load component loads the integrated reference populations into the universe, through the universe management interface. In this architecture, we also propose to include a software interface, called the universe management interface so that system administrators can interact with the ETL pipeline e.g., in receiving the transformed population individual data sets (data flow a8). Information about modifications on the CSs (data flow a9) is needed in order to perform updates on the universe.

A reference architecture that implements a partially materialised universe, known as a hybrid approach [HZ96, Hul97], as shown in Figure 4.5, is another

\(^1\)Extract, transform and load. A software tool that is usually used in data warehousing and decision support system [CDG01].
type of architecture that is plausible for a PBC system. In this architecture, only some reference populations are stored in the universe while the others are kept in their own CSs and queried on demand through a mediator. A PBC reference architecture with a partially materialised universe must support queries for reference populations (data flow a3) against both the universe and the CSs. We propose to include a software component in this architecture, called the ‘broker’, that will decide whether queries for the reference population should be sent to the universe or to the mediator (or both). The broker must know which reference populations are materialised and which must be fetched on demand. The measurement processor will interact with the broker in issuing the queries instantiated based on the PBC measurement requests. Data flows involving the virtual universe (through the mediator) are illustrated in blue for a clearer distinction between the functionalities of fully materialised and virtual universe architecture within this hybrid architecture.

PBC measurement providers might choose one particular form of PBC reference architecture to implement PBC for a number of reasons.
The architecture with the virtual universe might be chosen because the reference populations are queried on-demand; therefore actual load of data is unnecessary except during the measurement. As a consequence, small data storage allocation is needed to store the system’s components configuration information (such as for the CSs configuration) and population maps. However, the virtual universe is unlikely to be very efficient if PBC measurement needs to be performed frequently, and the number of CSs that must be queried to perform the measurement is large. This is because there is a risk of communication failures, especially if the CSs are prone to experiencing technical problems that cause restricted access. In addition, PBC measurement is likely to be slow, as all the complicated tasks of querying the population individuals from multiple CSs, transforming the query results to conform with the universe schema, and combining the query results must take place before the measurement processor can initiate the PBC computation.

PBC reference architecture with a fully materialised universe serves as a better alternative if quick PBC measurement is needed. Even though an up-front effort to construct and populate the universe is needed, it could be worthwhile if a large
number of data analysts are accessing the reference populations, each wishing to make use of a slightly different combination of the data set under measure and the reference population. However, we need to deal with data freshness, an issue that is common for any approach that materialises copies of data sets from other data sources. Therefore, it is necessary to put effort into detecting or monitoring updates to the contributing data sources, to ensure the materialised data sets are up to date [Wid95].

With additional knowledge regarding which reference populations are frequently accessed (or are not frequently accessed), one might consider PBC reference architecture that implements a partially materialised universe. In terms of the requirements, a partially materialised universe needs support for the functionality required by a fully materialised universe and a virtual universe.

Regardless of the reasons for choosing a particular type of PBC reference architecture, the components in all architectures have their own functionality to support PBC measurement providers to answer PBC measurement requests. The data flows in the architectures are present as the result of the interactions among the components of the system (software and source) that each serve their own functionality (e.g., measurement processor to accept PBC request, instantiate the query and pass it to the universe/mediator). In comparison, with the fully materialised universe, a PBC reference architecture consists of all data flows that are present in the architecture that implements the virtual universe (data flow a1 to a6) with two extra additional data flows (data flow a7 and a8); all data flows that are present in the partially materialised universe architecture can be found in the fully materialised universe architecture. Using the data flows to indicate the results of the functionalities performed by the components, we say that the differences (in terms of functionality) among the three forms of architecture are small. As the functionalities required by the architecture with a fully materialised universe cover the functionalities of the other two forms of architecture, in the rest of this thesis, we will focus on a PBC system with a fully materialised universe.

Having a PBC reference architecture that implements a fully materialised universe, we will next describe, in general, the links among the main activities, the actors and the elements of PBC within the architecture:

- Setup:
  - to set up PBC, domain experts need to identify the reference populations (and reference sub-populations) and their CSs. We rely on
domain experts to perform this task but other people, like data analysts, could participate as well.

– To design the schema of the universe, the measurement designer needs information about the reference populations and their CSs (as provided by the domain expert), as well as the schemas of the CSs. Recall in Chapter 3 that we introduced population maps, which consist of mappings between the population tables in the universe and the queries on the CSs. Based on the schema of the universe and the schema of the CSs, population maps need to be prepared so that we can extract the reference population individuals from the CSs. In addition, the CSs must be configured by storing information about them, such as CSs names and URIs.

– Once the schema of the universe, the population maps and the CSs configuration information are ready, the universe can be created, and the reference population individuals which are extracted from their CSs are loaded into it.

– Queries against the CSs (data flow a6) are submitted to the CSs through the wrappers (that are plugged into the ETL pipeline) through which Extract component of ETL pipeline will interact with. We rely on the wrappers to handle schema heterogeneity among the CSs where each wrapper will use mappings between population tables and the CSs in the population map to issue the local queries against the CSs. The Extract component will get the mappings between population tables and the CSs in the population map and forward it to the wrapper (data flow a5), while CS configuration information (data flow a5) is used to determine which CSs must be queried (through the wrappers). The extracted reference populations (data flow a7) will be transformed into the schema that matches with the type of schema of the universe (e.g., relational schema) by wrappers and then will be passed to the Transform component of the ETL pipeline for further transformation. A further transformation needed is to resolve semantic and format differences among the local schemas in order to conform to the schema of the universe, and to integrate them. The Transform component
must consist of a data validation procedure before population individuals can be integrated and loaded into the universe. A similar validation procedure is implemented by the Joint Genome Institute (JGI) that validates submissions to its Integrated Microbial Genome (IMG) database [MKP+06], which is outlined in the Microbial Data Submission guide [CMMK08]. The validation ensures the data submitted is in the right format and has a valid gene translation, and thus will reject data submissions with these errors. The Load component of the ETL will load the integrated reference populations into the universe, through the universe management interface (data flow a8).

- In the Setup, codes for some of the software components, like the wrappers and the universe management interface, need to be provided by the PBC measurement designer prior to the creation of the universe.

- Measurement:

  - the measurement activity is triggered by PBC measurement requests coming from data analysts (data flow a1). In requesting PBC measurement(s), data analysts need to provide the data set to measure and the specification of reference populations (e.g., the name of the reference population) they want to use through the PBC measurement interface.

  - The measurement requests are forwarded to the measurement processor that will pose the queries needed to extract any data set under measure and the reference population (data flow a3) from the universe. The requested data sets will be returned, as shown by data flow a4.

  - Having acquired the necessary data set under measure and the reference population, the measurement processor will perform the computation and the PBC measurement results (data flow a2) will be sent to data analysts.

- Maintenance:

  - the maintenance of the universe is performed by the system administrator. The principal task is to maintain the universe in the light of updates occurring at the CSs.
Suppose that the information about the modifications to the CSs (e.g., schema modifications or data updates done on the CSs) is available in some forms (e.g., in a log), some reference population tables in the universe might need to be updated. Information about the modifications (data flow a8) is retrieved by the wrappers that are triggered by the request made by the system administrator.

The system administrator will forward modification information to the PBC measurement designer for any schema modifications to the CSs that may be affecting the schema of the universe. For any data updates done on the CSs, reference population tables in the universe must be updated by querying the new/updated versions of population individuals from the CSs (data flow a6 denotes the queries sent to determine these). The interaction between the universe and system administrators is made through the universe management interface.

The maintenance activity could also be triggered by the data analysts. In this case, data analysts might give feedback requesting reference populations which have not been covered by the universe. Updates to the schema of the universe and population maps must be implemented when new reference populations (or sub-populations) are added. PBC components configuration information must be updated when new CSs are added or when the information about the CSs changes (e.g., changes in URI of the CSs).

In this section, we identified the basic components of a PBC system that are essential in a PBC reference architecture. We proposed three types of PBC reference architecture, each with extra components that must be added along with the basic components of PBC. The choice of the type of PBC reference architecture to use will guide PBC measurement providers to determine the software (and hardware) components that are needed and therefore to make necessary estimates on the financial costs incurred.

However, the question that remains unanswered is regarding the efforts needed to support PBC measurement providers in providing accurate PBC measurement answers. These efforts should be included as the costs of establishing a PBC system, in addition to the costs of building the infrastructure of the PBC system (based on the chosen reference architecture). Thus, in the next section, we will identify the problems that become the barriers to answering PBC measurement
requests and the barriers to providing accurate answers for the measurement. To aid the problem identification, we divide the analysis into the main activities of PBC, creating an activity diagram for each. The processes and the plausible flows through them will be elicited in the activity diagrams, where the plausible problems and their effect on PBC measurement accuracy will be presented.

4.3 The Problems Concerning PBC Measurement Accuracy

The PBC system architecture presented in the previous section is proposed to support PBC measurement providers to answer PBC measurement requests, but accurate PBC measurement answers can only be computed in the absence of the problems that cause the universe to store reference populations that are incomplete or inaccurate. Therefore, in this section, we aim to answer the following questions:

- What are the problems that cause inaccurate and incomplete reference populations to be stored in the universe?
- What are the efforts (i.e., solutions) that must be provided to deal with the problems?
- How do inaccurate and incomplete reference populations affect PBC measurement accuracy?

To answer these questions, we created detailed activity diagrams for the Setup, Measurement and Maintenance and then analysed them to find the possible points where the problems will cause the populations stored in the universe to be inaccurate and incomplete. We will begin our analysis in this section with Setup, followed by Measurement and Maintenance.

4.3.1 PBC Setup Analysis

We created Figure 4.6 to illustrate a set of processes (activities) in Setup and the possible interactions among them. This diagram is an expansion of the setup activity as shown in Figure 4.2 (page 61). Each activity is labelled with an activity number and a description of it is provided in the legend. Note that we
Figure 4.6: Setup in More Detail
do not enforce any particular order for the implementation of activities that are independent of each other (such as A1.1 and A1.2), and therefore, the numbers associated with the activities have no particular meaning other than identification. The figure depicts the case where every unsuccessful termination of the activities (shown as [NOT OK]) will require the activities to be resumed. However, PBC measurement providers might wish not to resume unsuccessful activities once the problems have been discovered, which therefore will cause the unsuccessful activities to be aborted.

Setup is triggered by the decision to implement PBC by the system investors, and performed in accordance with the PBC reference architecture selected by PBC measurement providers (stakeholders). We will first describe the activities in Setup before we can elicit the problems from the activities. These activities will be described in terms of: the actors that perform them, the components in the PBC architecture that support them and the functionalities of the PBC components (i.e., the inputs and outputs produced) that are achieved by the implementation of the activities.

Within the context of an initial PBC setup, the description of the activities involved are:

- In A1.1, domain experts identify the populations and sub-populations that will be stored in the universe. Information about which populations are needed might come from PBC measurement requests collected from data analysts.

- In A1.2, domain experts select the CSs known for the populations in which this activity can be performed concurrently with A1.1. We rely on domain experts’ knowledge to select CSs that they believe can provide the individuals of good approximate reference populations.

- In A1.3, PBC measurement designers retrieve the schemas of the CSs. The schema of the universe is created in A1.4 by the PBC measurement designers, based on the information about the identified populations and sub-populations. As the figure shows, the creation of the universe schema also depends on information from the identified CSs (and their schemas). The universe must be configured with a schema that describes the specific reference populations required.
In A1.5, PBC measurement designers create a population map. Within the reference architecture (as shown in Figure 4.4, page 65) the population map is the component that consists of the mapping between population tables and the CSs, used by the ETL tool to query the populations from the CSs. Like the universe, the population map must be configured specifically for the populations (and CSs) that are required, as outlined in the basic configuration of PBC elements provided in Chapter 3 (Table 3.1, page 48). Every CS must also be configured in this activity, in which CSs configuration information (e.g. the CS name and URI) becomes the input for the ETL tool to extract population individuals.

In A1.6, the universe is created by PBC measurement designers. This activity might be conducted in parallel with the creation of population maps. Within the reference architecture, queries for reference populations are submitted to the universe by the measurement processor, and in return, the universe will pass the requested reference populations to the measurement processor to compute PBC measurement answers.

In A1.7, the population individuals are extracted from their CSs, transformed to conform with the universe’s schema, integrated and loaded into the universe. The component that supports the implementation of these activities within the reference architecture is the ETL pipeline. The Extract component in the ETL pipeline takes the mappings from the population map and CS configuration as the input and sends the queries to the extract reference populations from the CSs through wrappers. Each wrapper will translate the queries in the population map into the queries that are understandable by the CS that it wraps. For example, a wrapper for a CS that is an XML database must translate SQL query in the population map into XQuery to extract population individuals from the CS. The wrapper also needs to transform the result of the queries that are in the form of the schema of its CS into the schema of the universe before they can be sent to the Transform component of the ETL pipeline for semantic and formatting transformation and integration. The Extract component is triggered by the system administrator, who issues the request to extract population individuals to the ETL pipeline, through the universe management interface. CSs configuration becomes the input for the ETL
tool to extract population individuals.

Both the ETL pipeline and the universe management interface are generic components, as they perform the same behaviour across PBC systems of different information needs (which are shaped by application domains). However, the CSs must be configured specifically to the application domains and populations required, while the wrapper must be written specifically for the CS it wraps in order to issue the queries to extract population individuals against the CS.

Based on the reference architecture that we proposed, the universe is a component that is effectively an integrated data set. Therefore, all the problems that beset a standard data integration effort are also potential problems for a PBC system. For example, we could expect schema and data heterogeneity among the CSs. This is especially the case where CSs are autonomous (i.e., “created and administered independently” [ZK99] of the organisation that implements the PBC system) and use different schema and data formats. Schemas’ and data formats’ heterogeneity must be resolved before the population’s individuals can be integrated. Solutions to data integration problems would benefit us in setting up (or maintaining) PBC systems. The state of the art for data integration problems like ETL tools (and wrappers) are included in the reference architecture to support PBC measurement providers in dealing with format differences and the CSs schema heterogeneity problem. From this we can see that the current technology that is available can be used to deal with the standard data integration problems and the PBC system is implementable as long as these technologies can be acquired and implemented within the system. In addition, we say that with data integration solutions, we can support PBC measurement providers to answer PBC measurement requests.

Nevertheless, there are problems that require solutions that support not only the answering of PBC measurement requests, but also the computing of accurate PBC measurement answers. In the next section, we will present the problems that affect the accuracy of PBC measurements and the solutions needed to deal with them. In particular, we look at the problems that cause reference populations to be inaccurate and incomplete. For each problem, we will describe how it occurs and what we can do to deal with it. We will begin with the quality of data, followed by the accessibility of CSs and storage space constraint. In Section 4.3.4 we will describe how reference populations that are affected by the problems can
cause PBC measurement errors.

4.3.1.1 Quality of Data

In Setup, populations are extracted from the CSs, and loaded into the universe. Specifically, what are extracted from the CSs are data sets that consist of population individuals (accompanied by their attributes), which will be loaded into population tables in the universe. One problem that must be dealt with by any application that extracts data from multiple data sources is the presence of duplicates and errors in the data set [BSIBD09, CFG+07]. Therefore, duplicates and errors are also a problem that needs to be dealt with by PBC systems that extract and integrate populations from multiple CSs. In particular, the special issue relating to PBC is that duplicates and errors may cause reference data sets to be inaccurate or incomplete and therefore may affect the accuracy of PBC measurements. The question is, how can duplicates and errors affect the accuracy of PBC measurements? To answer this question, we will first examine how duplicates and errors in reference populations of the universe cause reference data sets to be inaccurate or incomplete. We begin by discussing the duplicates problem in reference populations.

Duplicates in reference populations

Duplicates are defined as entity instances that have multiple representations [EIV07] or multiple references [BG07]. We say that duplicates are present when some population individuals have multiple identifiers. One factor to the duplicates problem according to Chapman, who studies data quality in botanical taxonomy is due to alternative spellings for the name used to identify individual species [Cha05]. He provides an example of duplicates due to alternative spellings that are all accepted for an orchid species as follows:

- Phaius tancarvilleae
- Phaius tankervilliae
- Phaius tankarvilleae
- Phaius tankervilleae
- Phaius tankervillae
In addition to alternative spellings, valid nomenclatural or taxonomic alternative names can cause duplicates, for example, *Brassicaceae* and *Cruciferae* are both valid (and accepted) nomenclatures for a plant family according to the International Botanical Code [Cha05]. Duplicates can also appear within a data source as a result of multiple tuples for the same individuals, but with different identifiers [BGMM+09] or as the result of integrating data from multiple sources where each source uses different conventions to identify the same individuals [MBGM06, PDL+09].

To illustrate how duplicates can occur in a population table of a universe, we use a simplified gene data set which is adapted from a data integration problem involving the bacterium *Zymomonas mobilis* [PDL+09] as shown in Table 4.1.

In a gene population table (which is unrealistically small), the gene individuals are identified by gene codes that are recorded within the *Gene* column and have several attributes.

Table 4.1(a) shows a case where there are no duplicates in the table, as a gene is identified with the same identifier by three different CSs namely *Brenda*, KEGG and BioCyc. However, in Table 4.1(b) duplicates occur in a source (*Brenda*), as the same gene has been identified with three different gene codes in this single source. Table 4.1(c) shows duplicates originating from multiple CSs where different gene codes are used to identify the same gene.

From the example we can see that we can bring any data duplicates problem already present in the individual CS by extracting its population individuals. However, a duplicates problem could also occur by integrating population individuals from CSs that individually appear to be okay. We say that duplicates cause spurious individuals to be added into the reference populations and therefore cause the reference populations to be inaccurate.

**Errors in reference populations** Errors in data sets have been caused largely by mistakes during data entry, where some instances may have been misspelled, or wrong values entered [Cha05]. We say that errors are present when some of the population individuals have incorrect attribute values recorded for them by their CSs. For example, returning to the gene data set example that we used in the discussion of the duplicates issue, Table 4.2(a) presents a gene population table with no errors. Using Table 4.2(a) as a reference, we can see
Table 4.1: Examples of Duplicates Using an Adapted Gene Data Set of *Zymomonas mobilis* [PDL+ 09] for Setup

(a) A Gene Population Table Without Duplicates

<table>
<thead>
<tr>
<th>Gene</th>
<th>Compound type</th>
<th>Enzyme commission number</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>Brenda</td>
</tr>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>KEGG</td>
</tr>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>BioCyc</td>
</tr>
</tbody>
</table>

(b) A Gene Population Table With Duplicates Contributed by a CS

<table>
<thead>
<tr>
<th>Gene</th>
<th>Compound type</th>
<th>Enzyme commission number</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>Brenda</td>
</tr>
<tr>
<td>6390740</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>Brenda</td>
</tr>
<tr>
<td>ZMO1605</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>Brenda</td>
</tr>
</tbody>
</table>

(c) A Gene Population table with Duplicates Contributed by Multiple CSs

<table>
<thead>
<tr>
<th>Gene</th>
<th>Compound type</th>
<th>Enzyme commission number</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>Brenda</td>
</tr>
<tr>
<td>6390740</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>KEGG</td>
</tr>
<tr>
<td>ZMO1605</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>BioCyc</td>
</tr>
</tbody>
</table>

that the population table shown in Table 4.2(b) presents an error in the *Gene* column where an incorrect gene identifier (*pzmob1-po1*) has been recorded for the second record. Table 4.2(c) however shows an incorrect value (*cofactors*) entered for the second record in a non-identifier attribute (*Compound Type*). From this example, we can see that error can occur as a result of integrating population individuals from CSs that individually appear to be accurate. An error that occurs in the identifier attribute is more invisible as compared to error that occurs in non-identifier attributes. This is because, we can check inconsistencies among the individuals that have the same identifier in terms of the values of the non-identifier attributes, but it is hard to distinguish if the individuals that have the same values in the non-identifier attributes are actually the same individual.

We say that errors can cause spurious individuals to be added into a reference population or some individuals to be missing from the reference population. Therefore, as a consequence, errors cause a reference population to be *inaccurate* or *incomplete*.

From the examples we can conclude that extracting population individuals from CSs can cause any DQ problems present in the individual CS to be added
### Table 4.2: Examples of Errors Using an Adapted Gene Data Set of *Zymomonas mobilis* [PDL+09] for Setup

(a) A Gene Population Table without Errors

<table>
<thead>
<tr>
<th>Gene</th>
<th>Compound type</th>
<th>Enzyme commission number</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>Brenda</td>
</tr>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>KEGG</td>
</tr>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>BioCyc</td>
</tr>
</tbody>
</table>

(b) A Gene Population Table with an Error in the Identifier Column Contributed by the CS

<table>
<thead>
<tr>
<th>Gene</th>
<th>Compound type</th>
<th>Enzyme commission number</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>Brenda</td>
</tr>
<tr>
<td>pzmob1-po1</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>KEGG</td>
</tr>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>BioCyc</td>
</tr>
</tbody>
</table>

(c) A Gene Population Table with an Error in the Non-identifier Column Contributed by the CS

<table>
<thead>
<tr>
<th>Gene</th>
<th>Compound type</th>
<th>Enzyme commission number</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>Brenda</td>
</tr>
<tr>
<td>pzmob1-po2</td>
<td>cofactors</td>
<td>1.1.1.267</td>
<td>KEGG</td>
</tr>
<tr>
<td>pzmob1-po2</td>
<td>activators</td>
<td>1.1.1.267</td>
<td>BioCyc</td>
</tr>
</tbody>
</table>

into the universe. However, DQ problems could also occur by integrating population individuals from CSs that individually appear to be okay, which could be more challenging to detect. A solution is needed to detect and to rectify the duplicates and errors problem in the population individuals set extracted from the CSs before they can be used. In the literature, it is usually proposed that, errors coming from multiple databases are handled via a data cleaning process, for example by using data cleaning tools [BSIBD09, CCG00] or by applying a data cleaning algorithm [CFG07, DHY07, GFS01] within the ETL pipeline. In particular, we should add a data cleaning tool to detect and resolve duplicates and errors after the Transform component of ETL performs semantic and formatting transformation. This is because it would be easier to deal with data sets that are already correct semantically and of the same schema and formats rather than in their raw form.

Figure 4.7 shows the additional software component within the PBC reference architecture that deals with duplicates and errors, called the Cleaning tool. The transformed and ‘cleaned’ data sets (data flow a8’) are the output that is sent to the Load component for further processing, while the input for the tool is the
transformed data sets from the Transform component of ETL (a8).

Research is still ongoing to deal with more challenging cases of duplicates and errors problems that are less visible (i.e., in CSs that individually appear to be okay but are problematic) and for which, we cannot rely on the unique identifiers to distinguish the problem [BG07, BGMM+09]. Thus, rectifying duplicates and errors is an open problem and any improved solutions to this problem would benefit us in providing accurate PBC measurement answers.

4.3.1.2 Accessibility of the CSs

Another plausible problem that could affect PBC measurement accuracy is the accessibility of the CSs. During the initial PBC system setup, if the CSs are inaccessible, CSs schema retrieval (A1.3) and population individuals extraction (A1.7) cannot be performed. While successful schema retrieval is crucial for PBC
measurement designers in creating the schema of the universe (A1.4) and the population map (A1.5) that will enable PBC measurement requests to be answered, full extraction of population individuals is crucial in answering PBC measurement accurately. If one or more CSs of a reference population are inaccessible, some population individuals are missing from it. With missing individuals, a reference population is incomplete and therefore may affect the accuracy of PBC measurement. Before we discuss how to deal with the problem, we will briefly describe barriers to CS accessibility, which are of course not limited to the following list:

- The CSs are autonomous: CSs accessibility can be hindered by lack of cooperation from the owner organisations/institutions in granting access to their sources’ schemas and disclosing their contents [IG02]. This is especially the case where the CSs are autonomous. In some cases, as reported by the institutions that deal with a large amount of data, like the Digital Curation Centre\(^2\) and the Data Preservation Alliance for the Social Sciences (Data-PASS)\(^3\), there are legal concerns associated with extracting and storing the data (e.g., confidentiality and copyright problems) [ABHS04, Dat05], besides financial cost for data extraction that needs to be resolved before such cooperation can be granted [Dat05].

- The CSs allow limited access to their schema and data: access to CSs that are public databases usually do not impose financial complications, and access to their contents is done through their query interfaces [Cul07, Li03]. However, the full content of the CSs and their schemas might be beyond what can be provided through the query interfaces. In this case, we need to consult data administrators of the CSs to get the schemas of the CSs as well as to retrieve the population individuals that are not retrievable via the query interfaces.

- Unpredictable CS failure: the identified CSs might be inaccessible for several technical reasons such as:
  - media failure: where the hard disks/servers on which data are stored are corrupted due to magnetic decay or viruses in operating system routines that perform writes to disk [SL76, HR83];

---

\(^2\)http://www.dcc.ac.uk

\(^3\)http://www.icpsr.umich.edu/icpsrweb/DATAPASS/
system failure: where viruses in DBMS code cause data loss, or operating systems error [GMP90, HR83];

user error: where accidental deletion of data causes data loss [SL76];

natural disaster: where disastrous or catastrophic events like fire, flood or electrical outage cause damage and loss of data [GMP90].

The problem of CSs inaccessibility is a common problem that is faced by any systems that require data extraction from multiple sources, which is inevitable for a PBC system with similar requirements. In the case where CSs are autonomous, we have less control in rectifying the technical problems that occur until these problems can be resolved by the organisations that administer the sources; limited access to schema and data can be resolved by gaining the permission to get full access to the CSs. In the case where there are conditions that must be met to gain access, PBC measurement providers need to decide whether these conditions are acceptable.

However, if PBC measurement providers can be informed about CSs that are inaccessible for technical reasons (e.g., system and media failure), this might help them to identify reference populations that are incomplete. Based on the information, PBC measurement provider systems could notify data analysts who are requesting PBC measurement about the accuracy of PBC measurement answers computed based on incomplete reference populations.

Within the reference architecture, a software component can be added to get feedback from each wrapper about the CS that is inaccessible, in which a report in some form is sent to PBC administrators through a universe management interface. As shown in Figure 4.8, a software component called an Accessibility assessor, which is added to the reference architecture, gets an input from the population maps to get information about the CSs that are suppose to contribute to a reference population (data flow a5), in addition to the information provided by the wrappers about the CSs that are accessible (data flow a10). A CS accessibility report is the output produced by an Accessibility assessor (data flow a11), which is received by the administrators. Figure 4.8 is an extension of the previous diagram (as shown in Figure 4.7) that illustrates the Cleaning tool which is added to deal with duplicates and errors problems.

We say that CSs inaccessibility is a problem that may cause reference populations to be incomplete. However, dealing with this problem is an open problem
that requires further research which may suggest solutions to cope with accessibility problems caused in particular by technical factors or natural disasters.

Figure 4.8: CSs Accessibility Assessor Component in PBC Reference Architecture.

4.3.1.3 Storage Space Constraint

Another problem that could cause reference populations to be incomplete is limited storage space for the universe. The amount of storage space allocated for the universe is determined by system investors, who decide on the budget for the storage space of the system. In many applications, concern regarding the size of the data sets to be stored or to be transmitted over a network is typically motivated by the time costs or financial costs [CL03, Mer77, NS90]. These costs introduce constraints on the storage space of the database, which is, in our case, a constraint on the storage space of the universe. Storage space is an inevitable issue in data integration systems for domains where the number of sources is large,
with each source consisting of a large amount of data (e.g., gene data used for a pharmaceutical company must be integrated from 40 databases that individually contribute large number of records [LC03]).

The problem arises in the case where the number of population individuals integrated from CSs is large, and we need adequate space to store all of these large populations in the universe. As a consequence of limited space, some reference populations (or their subsets) cannot be stored in the universe. If reference populations are missing, some PBC measurement requests cannot be answered; however, if some subsets of reference populations are missing (i.e., incomplete), PBC measurements that rely on these incomplete reference populations will not be accurate. In Setup, the activity that will be affected is A1.7, because if the storage space required for the full universe exceeds the space available, some population individuals cannot be loaded. The software component in the reference architecture that is involved for this activity is the Load component.

To deal with the storage space constraint problem, PBC measurement providers may acquire new disks to expand the capacity of the universe. However, an alternative way(s) to deal with this problem is needed if expanding the universe’s storage space by additional disks is not an acceptable option for system investors.

One alternative is to deal with this problem by selecting more important reference populations to be stored instead of all reference populations identified. In this case, domain experts need to elicit which reference populations are more important (and not) based on their knowledge of PBC measurement requests in the domain. Another option is to store incomplete reference populations by omitting some of the individuals. For example, a PBC measurement provider might choose to omit the individuals from CSs that impose conditions that are ‘expensive’ to meet in granting accessibility to their data, or CSs that have major duplicates and errors problems that require more effort from PBC measurement providers in order to fix the problems. This solution is based on the selection options that could best utilise the amount of space available. The solution requires inputs in terms of preference on reference populations (or CSs). A similar approach is used in computational complexity theory for the knapsack problem, where the problem is to determine the number of items to include in a collection so that the total weight is less than or equal to a given limit [Pis05], which is often applicable in addressing the problem of allocating resources with some constraints (e.g., time [GOMO07] and financial [BS02]).
Nevertheless, we regard this alternative as a less optimal solution (in terms of the number of reference populations stored) because the space is ‘optimised’ by excluding some reference populations (or their subsets) in order to meet the constraint, and because storing the full universe is not the goal of this solution. Therefore, we need a way to optimise the universe’s space with the goal of maximising the amount of PBC measurement requests to answer and their accuracy.

Suppose that a Space optimiser based on the goal just specified is available; we would need to add the software component in the reference architecture that can perform the universe’s space optimisation during Setup or Maintenance. As shown in Figure 4.9 (which is an extension to Figure 4.8), the Space optimiser is plugged into the Load component of the ETL pipeline, where the input for it is information on the amount of space allocated for the universe, taken from PBC component configuration (data flow a5) and the transformed query results (data flow a8’). The output of the Space optimiser is the result of an analysis of how much of the space saving gained can be used to store reference populations. Ideally, if all reference populations in the queue can be stored, all the transformed query results (data flow a8’) will be stored.

To conclude, we say that space constraint is a problem that may cause incomplete reference populations. Dealing with storage space constraint is an open problem, whose solution will help PBC system providers to answer PBC measurement requests accurately.

4.3.2 PBC Measurement Analysis

We created Figure 4.10 to show a more detailed breakdown of the Measurement. As with Setup, we first describe the activities in Measurement before we can elicit the problems from the activities. These activities will be described in terms of: the actors that perform them, the components in the PBC architecture that support them and the functionalities of the PBC components (i.e., the inputs and outputs produced) that are achieved by the implementation of the activities. Measurement is triggered by PBC measurement requests made by data analysts. In issuing the requests, data analysts must provide the necessary inputs for the measurement, before the PBC measurement computation can be made. The output of Measurement is PBC measurement results, given in the form of PBC ratios or percentages.

The activities in Measurement can be described as follows:
Figure 4.9: Space Optimiser Component in PBC Reference Architecture
• In A2.1, data analysts specify the data set under measure by providing the data set to be measured (e.g., in form of text files) as the input of the activity. If the data set under measure is available in the universe and data analysts requesting the measurement wish to use it, they must specify some parameters that describe the types of measurement request issued (refer to Chapter 3, page 44 for description of the types of PBC measurement request), such as the name of the reference population, the CSs, and the sub-population. The component of the reference architecture (in Figure 4.4) that supports the specification of the data set under measure is the PBC measurement interface. The interface should have some forms of selective options with which data analysts can specify the parameters needed for the data set under measure that can be provided by the universe. In the case where the data set under measure that data analysts wish to use is not available in the universe (as shown by [Not OK]), we allow them to give feedback and terminate the Measurement, as depicted by A2.5. Alternatively, they can provide their own data set under measure and proceed, as shown by the [Not OK] loop in the diagram.

• In A2.2, data analysts specify the reference population to use by choosing the reference populations in the universe and/or providing their own reference populations. Similar to A2.1, to use reference populations that can be provided by the universe, data analysts must specify some parameters that describe the types of measurement request issued through the PBC measurement interface. Because there is no particular order to follow, both A2.1 and A2.2 can be performed in parallel, or one after the other. We provide the same option in A2.2 in the case where the reference population to use cannot be provided by the universe. In such cases, data analysts may choose to terminate the activity after providing feedback or proceed by specifying their own reference population.

• Given the specifications of the data set under measure and the reference population to use as the inputs, in A2.3 the Measurement processor as shown in the reference architecture will instantiate the queries to extract the reference population and/or the data set under measure from the universe. These queries are submitted to the universe. Having both the data
set under measure and the reference population as the inputs, the Measurement processor instantiates the PBC measurement formula and computes PBC, which is illustrated by A2.4. The output of A2.4 (produced by the Measurement processor) is the PBC measurement results that are given to the data analysts.

Figure 4.10: Measurement in More Detail

Recall from Chapter 3 that in the basic case of PBC measurement, data analysts must provide their own data set under measure. It is a convenience for data analysts to use data sets that can be provided by the universe if these data sets are the data sets that they wish to measure.

However, because we cannot expect most of the data sets under measure to be provided by the universe, one functionality which is necessary for a PBC system is to perform a validation on the data sets provided by data analysts for the measurement. These data sets might be a data set under measure or a reference
population (or both). The validation is necessary as the data sets provided by data analysts come from the sources (i.e. private databases) that are beyond a PBC system’s control, and differences in terms of formats and semantics of the data sets are plausible. However, checking the semantics is harder, as it involves additional knowledge on the domain that cannot be resolved quickly by examining the data sets that may appear to be valid in terms of format. For example, a data analyst may provide a data set under measure which consists of gene identifiers for a mouse species by mistake. Because valid gene identifiers were given, it is hard to detect whether the data set is semantically wrong. Therefore, if a human gene population is specified as the reference population to use, we can still proceed with the measurement. The same implication applies if data analysts provide reference populations that are semantically incorrect.

Therefore, the option that we have at the Measurement stage is to at least provide format checking for the data sets provided by data analysts. In the reference architecture, a software procedure can be added in the Measurement processor to validate the input data sets. The validation procedure should check whether the format of the identifiers specified for the data set under measure matches with the format of the identifiers of the chosen reference population. For example, if the format used for the identifiers of a reference gene is six-digit gene codes, a data set under measure specified in the form of gene names will be considered as invalid.

As shown in Figure 4.11, a software component called Format checker is added within the Measurement processor that receives the same inputs as the Measurement processor (data flow a1). Through the PBC measurement interface, Format checker submits a notification to data analysts in case of invalid formats (data flow a12) in the form of a message containing the error and the suggested correction(s) to make. Functionally, we allow the measurement to proceed only if the problem can be resolved by data analysts (as shown by the [Not OK] loop in Figure 4.10), by providing data sets with the correct format. But the measurement can be terminated by data analysts (by following the feedback path) who do not wish to perform the correction. By dealing with format differences, we support the PBC measurement provider to answer PBC measurement requests that are at least syntactically correct. In the next section, we will present the problem that may affect the accuracy of PBC measurement answers during Measurement.
Figure 4.11: Format Checker Component in PBC Reference Architecture
4.3.2.1 Quality of Data

As with Setup, quality problems also arise, but for the external data sets (i.e., the external data set under measure and the external reference population) provided by data analysts. Duplicates and errors, if present, will impose the same implications on these external data sets as those data sets in the universe which, either spurious individuals are added into the reference populations or some individuals are missing from the reference populations. Because data analysts may extract their own data set under measure (or reference populations) from the sources that are not covered by the universe, we say that duplicates and errors is an issue that is beyond the control of PBC measurement providers. Therefore, in order to get accurate PBC measurement answers, data analysts must provide data sets that are free from any quality issues. Nevertheless, it is still useful to understand the implications of duplicates and errors in the external data sets on the accuracy of PBC measurement answers, an analysis of which will be given in Section 4.3.4.

4.3.3 PBC Maintenance Analysis

We created Figure 4.12 to illustrate more detailed activities in Maintenance and the interactions among them. Like all software systems, a PBC system, once set up, must be able to adapt to changes to its technical and user requirements. The maintenance activity is intended to support this adaptation. The activities in Maintenance are grouped according to triggers for maintenance that initiate them:

- Changes in user requirements: some maintenance activities are triggered by the decision to act on feedback supplied by data analysts. One type of feedback is on the request of desired reference populations. With new added reference populations in the universe it means new (and more) PBC measurement requests can be supported.

- Changes in technical requirements: other activities are triggered by modifications in the CSs. Two types of modification are of concern: modification of the schema of a CS and modification of CS contents.

Before we can discuss the problems in Maintenance, we must first describe its activities as follows:
In A3.1, system administrators receive feedback from data analysts requesting new reference populations, new reference sub-populations or new CSs. Within the PBC reference architecture (Figure 4.4 in page 65), the feedback is issued by data analysts through the PBC measurement interface, and is received by system administrators through the Universe management interface. For brevity, the data flows between the PBC measurement interface and the Universe management interface are excluded from the architecture. Based on the feedback, system administrators forward the requests to domain experts.

In A3.3, domain experts identify the reference populations (and sub-populations) requested by data analysts based on the feedback. In A3.4,
domain experts select CSs for the requested reference populations (and sub- 
populations) or select the requested CSs for reference populations that are 
already in the universe. As with Measurement, both the identification of 
reference populations and CSs selection can be performed in parallel, or one 
after another.

- In A3.5, PBC measurement designers retrieve the schema of the selected 
  CSs, before they can update the schema of the universe and the population 
  map.

- In A3.2, system administrators receive information about the modifications 
  of the CSs from the wrappers. Modification information must be requested 
  by the system administrator, where in the reference architecture, the re-
  quest is made through the Extract component of the ETL pipeline against 
  the wrappers. Wrappers get modification information in some forms from 
  the CSs and pass it to the administrators through the Load component 
  and the Universe management interface. The administrators forward the 
  modification information to the PBC measurement designer.

- In A3.6, PBC measurement designers update the schema of the universe, for 
  example by adding new columns (or removing columns) due to modifications 
  made to the schema of the CSs. Another possible universe schema update 
  is adding a new population table due to new reference populations added 
  to the universe, or removing a population table from the universe in the 
  extreme case where the reference population no longer exists (e.g., due to 
  deletion of its CSs).

- In A3.7, PBC measurement designers update the population map due to 
  changes to the CSs (e.g., columns being renamed or removed), for example 
  by updating the queries to extract the population individuals. CSs config-
  uration information needs to be updated as well by the PBC measurement 
  designer, for example due to changes on the CSs URI and population map. 
  However, modifications to the CSs that affect only the content of popu-
  lation tables will cause A3.6 and A3.7 to be skipped. PBC measurement 
  designers notify the system administrator about the changes made to the 
  schema of the universe and the population map so that the extraction of 
  new or updated population individuals can be made.
• Once the universe schema and the population map have been updated, in A3.8 the system administrator sends a request to extract new population individuals (or updates on them) to the Extract component of the ETL pipeline in the reference architecture (see Figure 4.4 in page 65), through the Universe management interface.

• In A3.9, population tables are ‘refreshed’ due to modifications to the CSs that affect their content. In this activity, the Extract component of the ETL pipeline in the reference architecture extracts new population individuals (or updates on them), which is triggered by the system administrators request. The Extract component executes the queries (in the population map) and sends them to the wrappers. The wrappers translate the queries into the queries that are understandable by the CSs, and send the query results received from the CSs to the Transform component. Similar to Setup, the query results will be transformed semantically and syntactically (in terms of format) and integrated by the Transform component. Finally, population tables will be refreshed by the Load component with the new (or updated) population individuals.

Maintenance must be able to respond to both types of triggers that require updates on the schema of the universe, population map and CSs configuration in order to answer PBC measurement requests. However, updates on population tables must be performed in order to provide accurate PBC measurement answers. Therefore, the problem that is specific to Maintenance is to monitor modifications that occur in the CSs and to take necessary actions (e.g., update schema and refresh the population table) so that answers to PBC measurement requests can not only be given, but are also accurate. In the next section, we will begin discussing the problem of CSs updates, followed by other problems (that are similar to Setup and Measurement) that are plausible in affecting PBC measurement accuracy.

4.3.3.1 Modification to the CSs

The implications of modifications that occur in the CSs on the population table are shown in Table 4.3. Based on the table, we divided the types of modification to CSs into schema modification and contents modification. We focus on the modifications that require updates on the content of the population table that
Table 4.3: Implication of CSs Modification on Content of Population Tables

<table>
<thead>
<tr>
<th>Types of modification in the CS</th>
<th>Updates required on a population table</th>
<th>Schema modification</th>
<th>Contents modification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tuples are added</td>
<td>Table is removed</td>
<td>One or more CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Columns are removed</td>
<td>Individuals are added</td>
</tr>
<tr>
<td></td>
<td>Tuples are deleted</td>
<td></td>
<td>One or more CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Individuals are deleted</td>
</tr>
<tr>
<td></td>
<td>Tuples are updated</td>
<td></td>
<td>One or more CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Individuals are updated</td>
</tr>
</tbody>
</table>

affects PBC measurement accuracy. The question that we would like to answer is how inaccuracy and incompleteness of reference populations could occur if updates required on the population table are ignored. To answer this question, we will describe each type of CSs modification and the updates required on the population table as follows:

- New tuples should be added into the population table if one or more CSs have new population individuals added into their tables.

Within the context of a reference population (as opposed to the reference sub-population), the addition of new tuples in the population table means the reference population has new additional individuals. New individuals are indicated by the tuples that represent individuals that do not yet exist in the reference population (i.e., no CSs have contributed these individuals yet). However, adding new tuples has no effect on the reference population if the individuals represented by the new tuples are already in the reference population (i.e., contributed by other CSs).

For example, consider a synthetic mouse strain population table (that is unrealistically small) with the schema: \((\text{strain}, \text{species}, \text{source})\) which consist of the following tuples:

\[
\langle \text{DBA/2J, mus Musculus, Ensembl} \rangle,
\langle \text{DBA/2J, mus Musculus, dbSNP} \rangle,
\langle \text{A/J, mus Musculus, Ensembl} \rangle,
\langle \text{BALB/cByJ, mus Musculus, Ensembl} \rangle.
\]

A mouse strain reference population in this example consists of three individuals: \(\{\text{DBA/2J, A/J, BALB/cByJ}\}\) contributed by two sources, Ensembl and dbSNP. For the first case, if dbSNP has a new mouse strain, called AKR/J
for *Mus musculus* species, the mouse strain population table must be updated by adding a new tuple to it. Adding the new tuple 
\[ \langle AKR/J, Mus Musculus, dbSNP \rangle \]
will therefore add a new individual to the mouse strain reference population as the strain does not yet exist in the reference population.

For the second case, if *dbSNP* has a new mouse strain, called *A/J*, the mouse strain population table must also be updated by adding a new tuple to it, which is \( \langle A/J, Mus Musculus, dbSNP \rangle \). However, adding this new tuple into the population table has no effect on the mouse strain reference population, as the individual represented by the new tuple already exists in the mouse strain reference population (which is contributed by another CS, *Ensembl*). Therefore, if an update on the population table is ignored for the first case, the reference population will be *incomplete*, as some individuals are missing from it. Nevertheless, the reference population is not affected if an update on the population table is not performed for the second case.

Note, however, that the reference sub-population that is constrained by CS(s) will be affected (will be *incomplete*) if an update on the population table is ignored, provided that the CS(s) that is used as the constraint is also the one that has new individuals. For example, suppose there is a request to use a mouse strain reference population from *dbSNP* only; if updates on the population table are ignored, then the mouse strain reference sub-population (which is constrained by *dbSNP*) will be *incomplete*.

- Some tuples should be deleted from the population table if one or more CSs delete their population individuals.

Within the context of a reference population, in contrast to the previous case (where new tuples were added), the deletion of tuples in the population table means the reference population has fewer individuals if it causes one or more individuals to be deleted from the reference population. However, deletion of tuples impose no effect on the reference population if it does not cause any individual to be deleted from the reference population.

For example, we revisit the synthetic mouse strain population table in the previous example, which consists of the following tuples:

\[ \langle DBA/2J, Mus Musculus, Ensembl \rangle, \]
\[ \langle DBA/2J, Mus Musculus, dbSNP \rangle, \]
A mouse strain reference population consists of three individuals: \{DBA/2J, A/J, BALB/cByJ\}. We say that if the third tuple is deleted from this table, the mouse strain reference population will have fewer individual as the deletion causes an individual (A/J) to be deleted from the reference population. Nevertheless, if the second tuple is deleted, the deletion does not cause the individual strain DBA/2J to be deleted entirely from the reference population, because there is another source (Ensembl) that contributes DBA/2J to the reference population.

Therefore, we can say that if an update on the population table is ignored for the first case, the reference population will be inaccurate, as some individuals should be deleted from it. Nevertheless, for the second case, the reference population is not affected if an update on the population table is not performed.

Note, however, that ignoring updates on the population table can cause the reference sub-population that is constrained by CS(s) to be inaccurate, especially if the CS(s) that is used as the constraint is also the one that has deleted individuals.

- Some tuples in the population table should be updated if one or more CSs update one or more attributes of its source table, or one or more CSs remove one or more attributes of its source table. The implications of ignoring the updates on the population table due to updates that occur in the CSs on the reference population are as follows:

  - Updates on the identifier columns will affect the reference population, as well as the reference sub-population.

    If updates on the identifiers will cause the reference population to have additional individuals, then ignoring the updates on the population table will cause the reference population to be incomplete. In contrast, if updates on the identifiers will cause the reference population to have fewer individuals, then ignoring the updates on the population table will cause the reference population to be inaccurate.
- Updates on the non-identifier columns will affect the reference sub-population (rather than the reference population), as the columns are used as the predicate in the query to retrieve the reference sub-population.

If updates on a non-identifier column should cause the reference sub-population to have additional individuals, then ignoring the updates on the population table will cause the reference sub-population to be *incomplete*, as it has missing individuals. However, if updates on a non-identifier column should cause the reference sub-population to have fewer individuals, then ignoring the updates on the population table will cause the reference sub-population to be *inaccurate*, as it contains spurious individuals.

Based on the analysis of the implications of modification to CSs on the population tables, we conclude that the population table must be updated in the cases where ignoring the updates causes the reference population or the reference sub-population to be *incomplete* or *inaccurate*. In most cases, updates cannot be ignored for reference sub-populations, as these populations will be either *incomplete* or *inaccurate* if updates are ignored. However, there are cases where ignoring updates on the population table does not affect reference populations. Understanding the cases where modification to CSs does not affect reference populations is particularly useful in the situation where updates on the population table need to be deferred due to some problems (e.g., lack of storage space). In this situation, the PBC measurement provider can prioritise population tables that must be updated by focusing on the updates that cannot be ignored.

Because it is important to perform updates on the universe, the next question that we would like to answer is ‘How can we support the PBC measurement provider in dealing with modifications to CSs?’. The effort to detect and monitor updates of the data sources is necessary in any systems that retain copies of data from the data sources to ensure the materialised data is up to date [Wid95], which is also an inevitable requirement for PBC systems. Some data sources keep ‘logs’ of the modifications made on their schema or contents. For example, a microbial research institute called Comprehensive Microbial Resource publishes the updates made on its database, which are then accessible publicly⁴; NCBI releases the current changes made on the contents and organisation of the flat files of its

⁴http://cmr.jcvi.org/CMR/CmrUpdates.shtml
Genbank database\textsuperscript{5} on its website. In those examples, information about changes to the CSs can be acquired by accessing the logs provided by the data sources. Regardless of the forms of modification information available, modifications that occur on the data sources must be monitored so that the universe can be updated as needed.

In the description of maintenance activity, information on CSs modification is retrieved based on the request made by the system administrator through the Extract component of ETL that interacts with the individual CS’s wrapper. To support PBC system providers in monitoring CSs modification automatically, a software component that could monitor and detect CSs modification and send the information to the administrator for further action is necessary. Functionally, this software must be able to access CSs modification information and decide whether the modifications that occur in the CSs affect any of the universe’s artefacts (e.g., schema of the universe, population map, CSs configuration and population table). Upon the detection of modifications that affect the universe, this software component must alert the system administrator for further action (i.e., forward the modification information to the PBC measurement designer). Within the PBC reference architecture, as depicted by Figure 4.13, this software component is called the CSs modification monitor. It receives CSs modification information from the wrappers (data flow a9) and performs some procedure to decide whether the modifications to the CSs are applicable, before submitting the required modification (data flow a9') to the system administrator. It would be useful if this software could distinguish between the cases where urgent updates are needed (i.e., for reference populations that are affected) and the cases where less urgent updates are needed (i.e., CSs modification has no implications for the reference populations) so that we could prioritise the update task. However, how to distinguish CSs modifications that cause reference populations to be incomplete or inaccurate (and utilise the information) requires further investigation. In addition, how to deal and manage frequent CSs modification is also a question that arises that may be worth answering.

\textsuperscript{5}ftp://ftp.ncbi.nih.gov/genbank/gbel.txt
4.3.3.2 Quality of Data, Accessibility of the CSs and Storage Space Constraint

As in Setup, we need to deal with the problems of data quality, accessibility of the CSs and storage space constraint in Maintenance. In particular, the activity that could be affected by these problems is A3.9 (as shown in Figure 4.12 in page 92). We will briefly describe these problems as follows:

- **Quality of data**: The same implications for reference populations caused by duplicates and errors can be observed in Maintenance. Duplicates and errors problems are plausible during the updates of the population table, where new population individuals are being added into the universe, or updated individuals are being loaded into the universe. Consequently, a reference population will have spurious individuals if duplicates are present,
while errors will cause a reference population to have spurious or missing individuals. As in other problems discussed earlier, we regard the reference population as being *inaccurate* if it has spurious individuals and *incomplete* if some individuals are missing from it. Therefore, to deal with data quality problems, we need a Cleaning tool component, as proposed in Setup (which is depicted in Figure 4.7, page 80), to detect (and rectify) duplicates and errors within the new additional individuals and the updated individuals during Maintenance.

- **Accessibility of the CSs**: In Maintenance, if CSs are inaccessible (due to the factors described in Section 4.3.1.2), the universe cannot be updated as required. In particular, several activities will be affected e.g., CSs schema retrieval by the PBC measurement designer (A3.5), updates on universe schema (A3.6), updates on population map and refreshing of population tables (A3.9). Unlike in Setup, an inaccessibility problem in Maintenance causes reference populations to be not only incomplete but also inaccurate, as ignoring updates on a population table may cause the reference population to have missing individuals or spurious individuals. Because it is hard to resolve CSs inaccessibility in many cases where CSs are autonomous, as in Setup, we need an Accessibility assessor (as shown in Figure 4.8) to support the PBC measurement provider in determining how many reference populations are affected by the problem. Based on the information, the PBC measurement provider system could notify the person requesting PBC measurement about the accuracy of the PBC measurement answers computed based on ‘stale’ reference populations that cannot be updated due to CSs accessibility problems.

- **Storage space constraint**: Updates on the universe that involve the addition of new additional tuples require additional storage space. As long as the additional storage requirement is within the storage space capacity allocated for the universe, the new tuples can be loaded into the universe as needed. However, the problem is, if the required additional space exceeds the storage space capacity of the universe, the task of loading those additional tuples (see A3.9) into the population tables needs to be deferred until the problem can be resolved. Storage space problems in Maintenance could cause reference populations to be incomplete, as ignoring updates
on population tables may cause the reference population to have missing individuals. To deal with a space problem, we need a Space optimiser as proposed in Setup, as shown in Figure 4.9 (page 86). In the case where the space problem cannot be resolved in full, it would be useful for the PBC measurement provider if the Space optimiser could give feedback in terms of population tables that can be fully updated or not. Based on the information, the person requesting PBC measurement can be notified about the accuracy of the PBC measurement answers computed based on incomplete reference populations that cannot be updated due to the storage space problem.

We have presented so far the problems that are the barriers for PBC measurement providers in answering PBC measurement requests and the problems that cause reference populations to be inaccurate and incomplete. Based on the analysis of the problems, we conclude that those problems are inherent in each main activity of the PBC system, and we can see that some problems must be dealt with in both Setup and Maintenance (i.e., quality of data in the universe, CSs accessibility and storage space constraint). To support PBC measurement providers in dealing with these problems, we suggested some methods to adopt, which result in adding software components to the initial PBC reference architecture proposed in the previous section (as shown in Figure 4.4, page 65). Figure 4.14 depicts the PBC reference architecture with the added software components (shown by red boxes) proposed to deal with the problems in Setup, Measurement and Maintenance. Nevertheless, we must say that some of the problems identified can be addressed by existing technologies (e.g., the ETL tool for standard data integration problems), while for other problems further research is needed in order to deal with them in full. Thus, the additional software components proposed might require more work to address questions that arise as outlined for each. In the next section, we set out to answer the question of how inaccurate and incomplete reference populations affect PBC measurement.
4.3.4 The Effects of Inaccurate and Incomplete Reference Populations on PBC Measurement Accuracy

We envisage the possibility that PBC measurement accuracy will be affected by the usage of a reference population that is inaccurate or incomplete. In the problem analysis presented in the previous sections, we say that a reference population is incomplete if it has some individuals missing from it, while a reference population is regarded as inaccurate if it has spurious individuals that do not belong to it. To analyse the effects of inaccurate and incomplete reference populations on PBC measurements we define PBC measurement error as:

\[
m.e(D, RP, RP') = \frac{|D \cap RP|}{|RP|} - \frac{|D \cap RP'|}{|RP'|},
\]  
(4.1)
where $m.e$ is the PBC measurement error, $D$ is the data set under measure, $RP$ is the reference population, and $RP'$ is the reference population that is inaccurate or incomplete.

In this subsection, we set out to answer the following questions:

- In what conditions do inaccurate and incomplete reference populations cause PBC measurement error?
- In what conditions do inaccurate and incomplete reference populations not cause PBC measurement error?

The outcome of this analysis will contribute to an understanding of the conditions under which PBC measurement accuracy is affected, which may suggest the need to deal with the problems that cause reference populations to be inaccurate or to be incomplete. We divide the analysis into three parts:

- Positive PBC measurement error, in which a PBC measurement result with an inaccurate or incomplete reference population is less than the expected correct PBC measurement result;
- Negative PBC measurement error, in which a PBC measurement result with an inaccurate or incomplete reference population is more than the expected correct PBC measurement result;
- No measurement error, in which a PBC measurement result with an inaccurate or incomplete reference population is equal to the expected correct PBC measurement result.

We envisage the actual PBC measurement results according to the amount of overlaps between the data set under measure and the reference population in the form of: 1) $|D \cap RP| > 0$ and, 2) $|D \cap RP| = 0$. Figure 4.15 illustrates PBC measurement errors in the case where $|D \cap RP| > 0$ while Figure 4.16 illustrates PBC measurement errors in the case where $|D \cap RP| = 0$. Note that the reference population that is incomplete or inaccurate within these figures is distinguished by red.

By considering the two forms of actual PBC measurement results involving inaccurate and incomplete reference populations, we observed that:

- A positive PBC measurement error is possible in conditions where:
- the reference population is incomplete and it causes the number of overlapping individuals between the data set under measure and the reference population to become smaller (see Figure 4.15 (a)), or

- the reference population is incomplete and no changes on the number of overlapping individuals between the data set under measure and the reference population (see Figure 4.15 (c)).

- A negative PBC measurement error is possible in conditions where:
  - the reference population is inaccurate and it causes the number of overlapping individuals between the data set under measure and the reference population becomes bigger (see Figure 4.15 (b) and Figure 4.16 (a)), or
  - the reference population is inaccurate and no changes on the number of overlapping individuals between the data set under measure and the reference population (see Figure 4.15 (d)).

- No measurement error is possible in conditions where:
  - The reference population is either inaccurate or incomplete but the number of overlapping individuals between the data set under measure and the reference population is equal to zero (see Figure 4.16 (b),(c)).
  - The ratio of the actual PBC measurement result is equal to the ratio of the PBC measurement result computed using a reference population that is inaccurate (or incomplete), where \( \frac{|D \cap RP|}{|RP|} = \frac{|D \cap RP'|}{|RP'|} \). This is possible for the cases illustrated in Figure 4.15 (a) and (b).

- It is not possible to have cases where:
  - The reference population is incomplete and it causes the number of overlapping individuals between the data set under measure and the reference population becomes bigger because, \( RP' \) is the subset of \( RP \) and therefore \( (D \cap RP') \subset (D \cap RP) \). Thus, \( |D \cap RP'| \leq |D \cap RP| \) will always hold.
  - The reference population is inaccurate and it causes the number of overlapping individuals between the data set under measure and the reference population to become smaller because \( RP' \) is the superset of
RP and therefore \((D \cap RP') \supset (D \cap RP)\). Thus, \(|D \cap RP'| \geq |D \cap RP|\) will always hold.

- A positive measurement error is yielded for an actual PBC measurement result that is zero (depicted by ‘None’ in Figure 4.16), as a positive error is only yielded for an actual measurement result that is greater than a PBC measurement result where incomplete reference population is used.

• An inaccurate reference population causes a negative measurement error and an incomplete reference population causes a positive measurement error.

• Unlike an incomplete reference population, an inaccurate reference population does not cause a PBC measurement error to occur in the case where \(|D \cap RP| > 0\) only.

Even though our focus in this subsection is to examine the effect of an inaccurate or incomplete reference population provided by the universe, it might be worth seeing the effect of an incomplete or inaccurate data set under measure provided by data analysts (a problem raised in Section 4.3.2.1, page 91). We adapt the definition of a PBC measurement error provided in Equation 4.1 as:

\[
m.e(D', RP) = \frac{|D \cap RP|}{|RP|} - \frac{|D' \cap RP|}{|RP|},
\]

where \(m.e\) is a PBC measurement error, \(D'\) is a data set under measure that is inaccurate or incomplete and \(RP\) is a reference population. To distinguish between the measurement error caused by the data set under measure and the measurement error caused by the reference population, it is important to assume that the relationship \(D \subseteq RP\) does not hold. Therefore, the reference population will not be affected by a data set under measure which is inaccurate.

As in the reference population case, we observe the actual PBC measurement results according to the amount of overlaps between the data set under measure and the reference population in the form of: 1) \(|D \cap RP| > 0\) and 2)\(|D \cap RP| = 0\). Figure 4.17 illustrates PBC measurement errors in the case where \(|D \cap RP| > 0\) while Figure 4.18 illustrates PBC measurement errors in the case where \(|D \cap RP| = 0\). Note that the data set under measure that is incomplete or inaccurate within these figures is distinguished by red.
1. \(|D \cap RP| > 0\) Positive measurement error
   Actual PBC measurement result
   
   \[\begin{array}{c}
   D \cap RP \\
   \end{array}\]

   PBC measurement result with error
   
   \[\begin{array}{c}
   D \cap RP' \\
   \end{array}\]

   Positive measurement error
   
   (a) \(|D \cap RP| > |D \cap RP'|\) and \(|RP| > |RP'|\)
   
   (c) \(|D \cap RP| = |D \cap RP'|\) and \(|RP| > |RP'|\)

   Negative measurement error
   
   (b) \(|D \cap RP| < |D \cap RP'|\) and \(|RP| < |RP'|\)
   
   (d) \(|D \cap RP| = |D \cap RP'|\) and \(|RP| < |RP'|\)

Figure 4.15: PBC Measurement Error for \(|D \cap RP| > 0\) in the Case of Reference Population
### Figure 4.16: PBC Measurement Error for $|D \cap RP| = 0$ in the Case of Reference Population

<table>
<thead>
<tr>
<th>Actual PBC measurement result</th>
<th>PBC measurement result with error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive measurement error</td>
<td>Negative measurement error</td>
</tr>
<tr>
<td>Actual PBC measurement result</td>
<td>PBC measurement result without error</td>
</tr>
</tbody>
</table>

- **2. $|D \cap RP| = 0$**
  - NONE
  - (a) $|D \cap RP| < |D \cap RP'|$ and $|RP| < |RP'|$
  - (b) $|D \cap RP| = |D \cap RP'| = 0$ and $|RP| < |RP'|$
  - (c) $|D \cap RP| = |D \cap RP'| = 0$ and $|RP| > |RP'|$
CHAPTER 4. PRACTICAL ANALYSIS OF PBC

1. $|D \cap RP| > 0$

**Actual PBC measurement result**

- Positive measurement error
  - $|D \cap RP| > |D' \cap RP|$ and $|D| > |D'|$

- Negative measurement error
  - $|D \cap RP| < |D' \cap RP|$ and $|D| < |D'|$

- PBC measurement result without error
  - $|D \cap RP| = |D' \cap RP|$ and $|D| > |D'|$
  - $|D \cap RP| = |D' \cap RP|$ and $|D| < |D'|$

**PBC measurement result with error**

1. $|D \cap RP| > 0$

Positive measurement error

- $|D \cap RP| > |D' \cap RP|$ and $|D| > |D'|$

Negative measurement error

- $|D \cap RP| < |D' \cap RP|$ and $|D| < |D'|$

PBC measurement result without error

- $|D \cap RP| = |D' \cap RP|$ and $|D| > |D'|$
- $|D \cap RP| = |D' \cap RP|$ and $|D| < |D'|$

Figure 4.17: PBC Measurement Error for $|D \cap RP| > 0$ in the Case of Data Set Under Measure
FIGURE 4.18: PBC Measurement Error for $|D \cap RP| = 0$ in the Case of Data Set Under Measure
CHAPTER 4.  PRACTICAL ANALYSIS OF PBC

By considering the two forms of actual PBC measurement results involving an inaccurate and incomplete data set under measure, we observed that:

- **A positive PBC measurement error is possible in conditions where:**
  - The data set under measure is incomplete and it causes the number of overlapping individuals between the data set under measure and the reference population to become smaller (see Figure 4.17 (a)).

- **A negative PBC measurement error is possible in conditions where:**
  - The data set under measure is inaccurate and it causes the number of overlapping individuals between the data set under measure and the population to become bigger (see Figure 4.17 (b) and Figure 4.18 (a)).

- **No measurement error is possible in conditions where:**
  - No changes on the number of overlapping individuals between the data set under measure and the reference population even though the reference population is either incomplete or inaccurate (see Figure 4.17 (c) and (d)).
  - The data set under measure is either inaccurate or incomplete but the number of overlapping individuals between the data set under measure and the reference population is equal to zero (see Figure 4.18 (b) and (c)).
  - The ratio of the actual PBC measurement result is equal to the ratio of the PBC measurement result computed using the data set under measure that is inaccurate (or incomplete), where
    \[ \frac{|D \cap RP|}{|RP|} = \frac{|D' \cap RP|}{|RP|} \]
    This is possible for the cases illustrated in Figure 4.17 (a) and (b).

- **It is not possible to have cases where:**
  - The data set under measure is incomplete and it causes the number of overlapping individuals between the data set under measure and the reference population to become bigger. This is because, \( D' \subset D \) and therefore \( (D' \cap RP) \subset (D \cap RP) \). Thus, \( |D' \cap RP| \leq |D \cap RP| \) will always hold.
- The data set under measure is inaccurate and it causes the number of overlapping individuals between the data set under measure and the reference population to become smaller because, \( D' \) is the superset of \( D \) and therefore \( (D' \cap RP) \supset (D \cap RP) \). Thus, \( |D' \cap RP| \geq |D \cap RP| \) will always hold.

- A positive measurement error is yielded for an actual PBC measurement result that is zero (depicted by ‘None’ in Figure 4.18), as a positive error is only yielded for an actual measurement result that is greater than a PBC measurement result where incomplete reference population is used.

  • Similar to the reference population case, an inaccurate data set under measure causes a negative measurement error and an incomplete data set under measure causes a positive measurement error.

  • Unlike an incomplete data set under measure, an inaccurate data set under measure does not cause a PBC measurement error to occur in the case where \( |D \cap RP| > 0 \) only.

In comparison, we observe that PBC measurement errors caused by an inaccurate or incomplete data set under measure is the same with most of the cases of PBC measurement errors caused by an inaccurate or incomplete reference population. However, it is important to note that no PBC measurement error will be made if the usage of an incomplete or inaccurate data set under measure does not change the number of overlapping individuals between the actual data set under measure and the reference population (see Figure 4.17 (c) and (d)).

Unfortunately, even though the usage of an incomplete or inaccurate reference population does not change the number of overlapping individuals between the data set under measure and the actual reference population, a PBC measurement error will occur (see Figure 4.15 (c) and (d)). This means that, for \( |D \cap RP| > 0 \), if some individuals are missing from (or added by mistake into) the reference population in the universe, PBC measurement error occurrence has a greater chance of occurring than if some individuals are missing from (or added by mistake into) the data set under measure given by data analysts.

Based on the error analysis in this subsection, we conclude that inaccurate and incomplete reference populations affect PBC measurement accuracy in certain
conditions as identified. Knowing how an inaccurate and incomplete reference population cause PBC measurement accuracy (i.e., based on positive or negative measurement error yielded) might be useful for a PBC measurement provider in prioritising the problems that they need to deal with. For example, as presented in the analysis, an inaccurate reference population causes a PBC measurement error to occur in more cases than incomplete reference populations (i.e., in cases where $|D \cap RP > 1|$ and $|D \cap RP = 0|$). Thus, efforts to solve problems that cause a reference population to be inaccurate should receive more attention from PBC measurement providers.

Even though there are cases where PBC measurement accuracy is not affected by an inaccurate and incomplete reference population, these cases are uncertain. Furthermore, we also found that PBC measurement error has a greater chance of occurring if an incomplete (or inaccurate) reference population in the universe is used than if an incomplete (or inaccurate) data set under measure (which is provided by data analysts) is used. Therefore, dealing with the problems that cause reference populations in the universe to be inaccurate or incomplete is necessary in order to support PBC measurement providers to answer PBC measurement requests accurately.

In the next section, we will show how the conceptual idea of the PBC is implemented by putting one particular form of its reference architecture into practice.

4.4 A PBC System Implementation: A Case Study for the SNP Domain

We implemented a PBC system based on the reference architecture proposed in this chapter. The implementation is set to answer the following questions:

- How should the conceptual PBC reference architecture be put into practice?
- Which problems identified in the analysis are encountered in the implementation of PBC?

To answer the questions raised above and to elicit more precise PBC measurement requests, we conducted a case study for the domain of a single nucleotide polymorphism (SNP). In the next section, we will first provide an overview of
the domain and describe the motivation for PBC inherent in this domain. We will then describe how the components within the PBC reference architecture are configured for the SNP domain and how PBC measurement requests are answered by the SNP PBC system prototype.

4.4.1 SNP Domain Overview

In genetics study, it is important for biologists to understand the relationship between the genes and the 'phenotype' (i.e., specific traits and behavioural characteristics) of an organism [Bro99]. One form of phenotype is susceptibility towards certain diseases (e.g., cancer or asthma). In order to test the hypothesis of the relationship between genes and diseases, biologists use 'genetic information' taken from a DNA sequence. A DNA sequence is made up of four types of simple units called nucleotides, which are represented by the “letter” A (adenine), C (cytosine), T (thymine), and G (guanine), which carries the genetic information of an organism [Bro99, Nat]. In particular, biologists are interested in a variation that occurs in the DNA sequences of the organisms under study that they hypothesise to be the cause of the diseases. This variation called SNP (pronounced snip), that occurs in a single nucleotide of a gene, is observed in a sufficiently large proportion of an organism to determine the specific phenotype of interest (e.g., the biological function that causes susceptibility towards diseases and physical appearance [Syv01]) [Bro99].

The underlying assumption behind many SNP studies is that the variation that contributes to an increased risk for a particular disease (or other observed phenotype) should occur at higher rates in the population which exhibits the disease compared to the population which does not [Bro99]. For human genes for example, it has been explicitly stated that the variation must occur in at least 1% of the human population in order to say whether there is a link between SNPs and a phenotype under study [Bro99, Hum]. Because of this requirement, many forms of studies involving SNPs (e.g., genome-wide analysis [Ile08], association analysis and gene mapping [SBE+05, SWB+01]) are sensitive to completeness, as, if some of the SNPs under study are missing, a conclusion about its correlation to a specific trait under study cannot be supported.

Every SNP which is discovered by biologists has some information that is stored about it. For example, the National Center for Biotechnology Information (NCBI) states the information that must be provided about SNPs by the person
submitting SNPs into its public database. The information includes the scientific submitter’s SNP id, name of organism, known genes in the DNA sequence region, sex of organism, tissue type and host (laboratory name) [Inf06a]. Our initial exploration of the SNP domain [EEM08] revealed that in the analysis where SNPs are used, not all the information stored about them is of interest. For example, in association studies to link genes and diseases, biologists are only interested in SNPs that occur in certain chromosomal locations [HKS05, TANPI09] or species [FEK+07]. Therefore, we say that not all information about SNPs is sensitive to completeness. For example, given a SNP data set, its completeness as regards to the chromosomes or species will be of concern, instead of other information that is stored about them.

SNPs discovery effort is not restricted to big institutions that provide public SNPs databases (such as dbSNP by NCBI)[SWB+01] and Ensembl by the European Bioinformatics Institute and The Wellcome Trust Sanger Institute [HAA+09], as other individual researchers who maintain their own private SNPs repository also contribute in discovering SNPs [FSY+02]. As efforts to discover and document new SNPs have gained momentum in recent years [Bro99], a number of public and private databases have been established [Bro99]. For example, in April 1999, a total of 7,000 SNPs had been deposited into the major public databases [Bro99], while by January 2002 some 4 million SNPs had been deposited in the dbSNP database alone [MKM02]. Since then, many other public databases and repositories have been established, including databases like Perlegen, GeneSNP, PharmGKB and HOWDY [MKM02].

The availability of multiple sources for SNPs increases the possibility of supporting many forms of studies involving SNPs, but there is a concern regarding the quality of these data sources in terms of the presence of false-positives in the data sources [Bro99, MKM02, RGA03]. In addition to the accuracy of SNPs, the completeness of SNPs data sources is also the issue that is of concern to the SNPs users community. A study by Marsh et al. compared three well-known human SNP databases for 74 human genes: CGAP-GAI, LEELAB and HOWDY in terms of their SNP coverage [MKM02]. Their work discovered a small overlap

---

6Perlegen - http://www.perlegen.com
7GeneSNP - http://www.genome.utah.edu/genesnps/
8PharmGKB - http://pharmgkb.org
9HOWDY - http://howdy.jst.go.jp/HOWDY
11LEELAB - http://www.bioinformatics.ucla.edu/snp/
Figure 4.19: Completeness differences in three human SNP databases involving 74 genes

(taken from [MKM02])

between the databases, as illustrated by the Venn diagram in Figure 4.19. A similar comparison was made by Missier et al. involving dbSNP, Perlegen and Ensembl, which pointed out the coverage issues of the sources under comparison [MEH+07]. Based on the discovery, we learnt that our confidence in the analyses performed over only a single database (or a small selection) should be reduced, because this would result in incomplete sets of SNPs being covered.

The lesson to be learnt from our observation of the SNP domain is that a concern for SNPs data set completeness is specific to some sets of individuals of interest that are present within the SNP data set itself (e.g., mouse species or certain chromosome). Data completeness requirements in the SNP domain hinted at the applicability of the PBC notion in supporting questions regarding the completeness of SNPs data sets. For example, to answer the question of how complete a SNP data set is as regards to a specific chromosome, we need a reference population for chromosomes to tell us whether some chromosomes are missing from the SNP data set. Without a system that could help to answer this question, the individuals of the desired reference population must be gathered manually, from the source(s) that can provide them. This is a non-trivial task, especially if the number of individuals of the reference population is large or they must be gathered from multiple sources. If the reference population is to be used frequently, we need to help to answer PBC measurement requests that could ease the non-triviality of manual gathering of reference populations. To show how PBC conceptual ideas can be put into practice, in the next section we
will describe an implementation of a SNP PBC system prototype.

4.4.2 A SNP PBC System Prototype

One factor that motivates the notion of the universe as presented in Chapter 3 is the absence of a single, good approximation of the true populations. The universe is therefore proposed to support PBC measurement in such situations in which the individuals of a reference population are integrated from a range of sources. As the SNP domain faces the same situation, where it is hard to find a single SNP source that fully overlaps with other SNP sources, we need to gather the individuals of reference populations that are needed to answer PBC measurement requests.

As we set out to determine which problems identified in the analysis are actually encountered, we developed a SNP PBC system prototype based on the same reference architecture used in the problem analysis, which is PBC reference architecture for a fully materialised universe (see Figure 4.4, page 65). We implemented and configured most of the components of PBC within the architecture and omitted some components for certain reasons. Wrappers are omitted, as the number of CSs in this implementation is small, and therefore query translation against the universe can be made manually, without the wrappers. For brevity, the SNP PBC system developer also opts to omit the interfaces components (the PBC measurement interface and the universe management interface) that, if present, will be useful for data analysts and administrators in providing the inputs into (and receiving outputs from) a PBC system within a more user-friendly environment. We configured SNP PBC system components following the steps in Setup (see Section 4.3.1) and used the reference template of the basic configuration of a PBC system provided in Chapter 3 (Table 3.1, page 48). In no particular order, the description of how we configured these components will be given in the following sections.

4.4.2.1 SNP Universe

Before the SNP universe can be configured, we must identify the reference populations to be included in the universe based on PBC measurement requests in the SNP domain. We considered in the implementation the PBC measurement requests hinted at the SNP literature, which are:
• to measure the completeness of a SNP data set in regards to chromosomal locations [TANPI09],

• to measure the completeness of a SNP data set in regards to chromosomal locations where certain genes are located [HKS05], and

• to measure the completeness of a SNP data set in regards to certain species [FEK+07].

If we focus on SNPs for mouse species, we can reduce these requests to the first and second requests only. To answer these requests, we must provide a SNP’s chromosome reference population that consists of information about its chromosomal locations for mouse species. We also need the identifier for this reference population. The schema of the chromosome population table where information about SNP’s chromosome reference population is stored should be at least in the form of \( \langle I, A \rangle \), where \( I \) is the set of identifier attributes and \( A \) is the set of attributes used to retrieve the reference sub-population. But if there is a request that puts some conditions on the CSs from which the genes are gathered, then we must add a source attribute within the schema, as outlined in the reference template of the basic configuration of PBC.

For this SNP system prototype, we designed the schema of chromosome population table, chromosome, as: \( \langle \text{chromosomeLocation}, \text{geneName} \rangle \), where chromosomeLocation is the identifier for the chromosome reference population and the geneName is the information about the gene that lies within the chromosome region. We created the SNP universe (and the population table) using MySQL\(^{12}\).

4.4.2.2 SNP Contributing Sources

In this SNP system prototype, we selected two primary sources of mouse SNPs as the CSs of a SNP chromosome reference population, namely Ensembl [HAA+09] and dbSNP [SWB+01], for their easy public access, and the amount of SNPs covered by these sources that, together, we suspect could make a good approximation of the universe.

We configured these SNPs CSs based on CSs configuration outlined in the reference template of the basic configuration of a PBC system, where information

\(^{12}\)http://mysql.com
about the CSs name, their URI, and population maps must be provided. Both Ensembl and dbSNP have public databases that are downloadable in a variety of formats (i.e., text (tab-delimited or FASTA\(^{13}\)) and SQL dumps).

### 4.4.2.3 Population Map and PBC Components Configuration

PBC components configuration information consists of information about the reference populations in the universe and the CSs (refer to Table 3.1, page 48 for PBC elements variables and type definitions). For a SNP PBC system prototype, the configuration information stored for the universe is: 

$$UP = \{(\text{chromosome}, \langle \text{chromosomeLocation}, \text{geneName} \rangle)\},$$

where \(UP\) is the set of the universe’s population tables and their schemas, and \text{chromosome} is the name of the population table for the chromosome reference population. Configuration information for the CSs is shown in Table 4.4.

#### Table 4.4: Configuration Information Stored for CSs.

<table>
<thead>
<tr>
<th>CS name</th>
<th>URI</th>
<th>population table Name</th>
<th>Query Against CS</th>
</tr>
</thead>
</table>

### 4.4.2.4 ETL Pipeline

We devise The Clover ETL framework (http://cloveretl.berlios.de/) in order to implement Extract, Transform and Load components as depicted in the PBC reference architecture in Figure 4.4 (page 65). In particular, for our SNP system prototype, we use the CloverETL community Edition\(^{14}\) which is a java-based, open source ETL tool deployed on a Mac OS X operating system.

We downloaded the data sets that consisted of the individuals of the chromosome reference population from Ensembl and dbSNP in the form of SQL dumps.

---

\(^{13}\)text-based format for representing either nucleotide sequences or peptide sequences, in which base pairs or amino acids are represented using single-letter codes.

We executed the SQL dumps and stored these raw data sets in temporary tables for each CS in which those tables matched with the schema of the universe (relational schema). These tables are:

- For Ensembl:
  
  - variation(variation_id, chromosome_position_id, source_id, name, validation_status, ancestral_allele, flipped, class_attrib_id, somatic),
  
  - variation_annotation(variation_annotation_id, variation_id, phenotype_id, study_id, associated_gene, associated_variant_risk_allele, variation_names, risk_allele_freq_in_controls, p_value).

- For dbSNP:

  - subSNP(subsnp_id, variation_id, known_snp_handle, snp_loc_id, batch_id, chr_id, gene_name, create_time, sequence_len, samplesize, ancestral_allele, validation_status).

At this point, format and semantic differences had not been addressed yet. The tasks we just described should be performed by the wrappers, but for the SNP PBC system prototype, we performed them manually for the reason stated at the beginning of Section 4.4.2. We used the Eclipse plug-ins provided by The Clover ETL to perform the subsequent Setup tasks after the ‘raw’ data sets had been downloaded and stored as temporary tables.

Basically, the functions provided by Clover are described as transformation graphs in which the components (such as Copy, Sort, Reformat and Filter) involved interact. To implement the Extract component (of PBC reference architecture), we used the Copy component of Clover to execute the queries in the population map, which takes the raw, temporary tables as the input. The result of this step is the relations that match with the chromosome table in the universe in terms of the table’s structure (i.e. a table with two columns), as follows:

- Ensembl: ensemblTemp(chromosome_position_id, associated_gene),

- dbSNP: dbSNPTemp(snp_loc_id, gene_name).

To implement the Transform component, we used the Reformat component of Clover, for which we resolved format differences among the columns of the extracted table. For example, Ensembl uses ‘integer type’ for
chromosome_position_id, while dbSNP uses ‘varchar’ type for loc.snp_id that both map with the universe’s chromosomeLocation attribute. The Transform component takes ensemblTemp and dbSNPTemp as the inputs and produces the transformed tables for each CSs that conforms to the schema of chromosome table. These tables are:

- Ensembl: ensemblTransformed(chromosomeLocation, geneName),
- dbSNP: dbSNPTransformed(chromosomeLocation, geneName).

We add a function to integrate these transformed tables within the Reformat component of Clover using a simple SQL command with union operator as follows:

```
SELECT chromosomeLocation, geneName FROM ensemblTransformed UNION SELECT chromosomeLocation, geneName FROM dbSNPTransformed;
```

The result of the query, which was the integratedChromosome table, was exported into a temporary text file called integratedChromosome.csv. We loaded the integrated chromosomes from integratedChromosome.csv into the Chromosome table in the SNP universe using the following SQL command:

```
LOAD DATA INFILE ‘integratedChromosome’ INTO TABLE chromosome;
```

About 42,505,315 tuples were loaded into the chromosome table, out of which Ensembl and dbSNP contributed 15,514,284 and 26,991,031 tuples respectively. Even though the SNP universe only consists of a single table, about 40.5 gigabytes of space has been used to store the chromosome reference population alone. Storage space will be an issue for the SNP domain if we wish to include more reference populations for mice (or other species) that individually require a large storage space.

4.4.2.5 PBC Measurement Processor

Having the universe readily loaded with the reference population, the question that arises is how can PBC measurement be implemented (i.e. is there any existing technological framework that we can use to implement it)? We found that one

---

For brevity, we omit the full path of the file location.
DQ framework that is specifically designed for implementing quality measures, called Quality View (QV) \([\text{MEG}^+06]\), could support us in answering the question just raised. QV, which is independent of our SNP PBC system prototype, is a software component proposed based on the DQ framework provided by the Qurator project\(^\text{16}\). Basically, the default behaviour of QV is to:

1. take a data set under measure i.e. \(D=\{a,b,c,d\}\) (where \(D\) is the data set under measure);
2. gather the ‘evidence’ of the quality of each element in the data set;
3. classify each element in the data set according to its evidence (i.e. \(a\) and \(b\) (in \(D\) are good);
4. output a version of the data set, which is ‘transformed’ based on the classification. For example, a common transformation is filtering out the elements of the data set that do not meet some predefined DQ acceptance standard. For example, \(D'=\{a,b\}\), where \(D'\) is the version of \(D\) which was transformed based on exclusion of ‘rejected’ elements.

The evidence is the term used for the result of some forms of domain-specific DQ measure performed on the elements of the data set under measure.

Figure 4.20 illustrates QV as a layered software component that performs the default behaviour just mentioned. In particular, the topmost layer of software components, called \textit{quality evidence functions} (QEF), receives the data set under measure and other parameters needed to gather the evidence for each element of the data set provided by QV users. In the second layer, components called \textit{quality assertions} (QA) apply decision procedures to the evidence in which standard aggregation operations (such as average or maximum) might be used to produce either some sort of score; or a classification of data items into quality classes (i.e., good or poor) based on the evidence. The bottom layer has components that implement the transformation on the data set. The output data set depends on the transformation types specified by QV designers. The transformation type is set during the specification of QV, along with the specification of the conditions that will determine the classification of the items in the data set. In addition to the transformed data sets, QV produces a quality report in XML form that

\(^{16}\text{www.qurator.org}\)
consists of a table, giving the output of each QA class for each item submitted by the user.

The default behaviour of QV is applicable for our PBC context, but to make QV more useful, we must perform some adjustments to the default behaviour of QV and add elements that are necessary to support QV functionalities. One adjustment needed is to the input data set. PBC is an aggregate measure that can be sensibly applied to sets of data, rather than to individual values. Therefore, the input data set for PBC measure must be a set of data sets (sets of the data set under measure), where \( D = \{D_1, D_2, D_3, D_4\} \). Each element in \( D \) is a set of data set under measure, which is in this example, we have four data sets to measure. Of course, the set can be a singleton set if only one data set is to be measured for PBC.

The evidence within the PBC context is the result of PBC measurement computed for each data set under measure in \( D \). As we need reference populations to measure PBC, the evidence can only be gathered if reference populations are present for each data set under measure in the set. Thus, the universe is an added element that is essential in order for the QEF component of QV to perform its evidence-gathering function, from which reference populations must be retrieved.

Figure 4.20: The Qurator Quality View Layers (Adapted from [EMS+09])
for PBC measurement. The evidence for PBC measurement is in the form of a completeness ratio (or percentage) computed based on the PBC measurement formula defined in Chapter 3, which is, in fact, a functionality that is beyond the default behaviour of QV. A function to instantiate the PBC measurement formula must also be added to QEF (i.e. to issue the query to get reference populations from the universe). Some parameters, such as the name of the reference populations, need to be provided by the person requesting PBC measurement.

Other adjustments need to be done if we wish to utilise other layers of QV. For example, the QA component needs a decision procedure to follow based on the PBC measurement result of populations under measure. For our case, the QV designer might set an acceptance threshold for the PBC percentage yielded by data sets under measure and classify which data sets under measure are accepted; or rank the data sets under measure based on their PBC measurement results. As for the transformation layer, QV designer might output the data sets that are within the accepted threshold, or output the first three data sets in the rank. Otherwise, we might skip the assertion and transformation layer, and simply return the PBC measurement results to the person requesting PBC measurement.

For the SNP PBC system prototype, we implemented all three layers of QV to answer the PBC measurement requests posted earlier in Section 4.4.2.1 (page 117). To recall, we set out to answer the following PBC measurement requests:

1. to measure the completeness of a SNP data set in regards to chromosomal locations, and
2. to measure the completeness of a SNP data set in regards to chromosomal locations where certain genes are located.

We call the QV for our SNP system QVsnp. Based on the PBC measurement requests, two data sets are of interest: the data set of the chromosomal region (denoted as \(D_1\)), and the data set of the chromosomal region of specific genes or sub-population of chromosome (denoted as \(D_2\)). For \(D_2\), we used genes Gm11767-001, Gm12390-001 and Gm12401-001 to artificially represent the genes of interest. Therefore, the data set under measure used for the SNP PBC system prototype is \(D = \{D_1, D_2\}\). In addition to \(D\), we need to provide the parameter(s) as the inputs to QVsnp. As both \(D_1\) and \(D_2\) require the same chromosome reference population, then the only parameter needed is the name of the reference population, which is equivalent to the name of the population table stored in
the SNP universe, chromosome. $D_1$ and $D_2$ consist of the chromosome locations (the identifiers) of the chromosome population that was taken from a SNP source called Celera\textsuperscript{17}. As illustrated in Figure 4.21, data flow a1 is a PBC measurement request which consists of the inputs just stated, provided to the QEF component of QVsnp. Based on the inputs, QEF issued the following query against the SNP universe that stores the reference population for $D_1$, $RP_1$:

\[
\text{SELECT chromosomeLocation FROM chromosome;}
\]

Then, to retrieve the reference population for $D_2$, $RP_2$, the following query was issued by QEF:

\[
\text{SELECT chromosomeLocation FROM chromosome WHERE geneName IN ('Gm11767-001', 'Gm12390-001', 'Gm12401-001');}
\]

These queries were denoted as data flow a3 in Figure 4.21. The SNP universe returned the results of the queries (data flow a4), and based on the result of the queries, QEF instantiated the PBC measurement formula to compute PBC measurement results as follows:

- \(\text{Completeness}(D_1, RP_1) = \frac{|(D_1 \cap RP_1)|}{|RP_1|}\),
- \(\text{Completeness}(D_2, RP_2) = \frac{|(D_2 \cap RP_2)|}{|RP_2|}\).

The results of PBC measurement (in the form of percentages) were issued to the second layer of QVsnp (data flow QEF output), in which the QA component will perform a classification upon $D_1$ and $D_2$. We added a decision procedure that classifies the results based on a predefined acceptance threshold. We defined PBC measurement results that are between 50 and 100 per cent as being within the ‘acceptable’ class while the results which are below 50 per cent are defined as being within the ‘rejected’ class. The PBC measurement results and their classification (data flow QA output) were issued to the third layer of QVsnp, where the transformation components were invoked. We set the transformation component to transform $D$ into $D'$, which consists of data sets under measure whose PBC measurement results are within the acceptable class, together with

\textsuperscript{17}Mouse SNPs taken from Celera was downloaded from mouse phenome database, http://phenome.jax.org/db/q?rtn=snps/download
the percentage of their PBC measurement results. The result of the transformation (data flow a2) was issued to the person requesting PBC measurement.

Based on the implementation of the SNP PBC system prototype, we encountered some of the problems identified in the problem analysis (See Section 4.3, page 71). In addition to the format differences, we faced a space constraint problem during the Setup of the SNP PBC system prototype. It is difficult to say whether duplicates and errors problem is present without a proper data cleaning procedure being implemented for the SNP PBC system prototype. However, as far as we know, dbSNP exercises strict data submission regulation so that any detected errors or duplicates will be removed from the dbSNP [Inf06b]. For certain, we did not experience any CSs accessibility problem for both selected CSs that are public databases. Even though we have not encountered the data freshness issue during the implementation of the SNP PBC system, we anticipate this problem occurring as it is inevitable that modifications will be made to SNP CSs. This is based on an observation made on DbSNP and Ensembl that announce any modifications that occur to their databases and the tools that they provide
in their websites on a periodic basis\textsuperscript{18}. For PBC measurement, we encountered a format differences problem, in which the format of the chromosome location for the populations under measure extracted from Celera is different from the format of the chromosome location used for our chromosome reference population in the SNP universe. Celera uses a Mega base pair (Mgp) unit format for its chromosome location, while the chromosome location in our SNP universe is formatted in a base pair (bp) unit. For example, for SNP that occurs in chromosome 11 at location 89916077 (in bp unit) is stated as 89.916077 by Celera. We resolved the format problem by converting the format of chromosome locations in $D_1$ and $D_2$ that conform to the format of the chromosome location in the SNP universe in order to enable successful PBC measurement provided by the SNP PBC system prototype.

In this section, we looked at how the conceptual PBC reference architecture can be put into practice, and showed how PBC measurement requests raised in the SNP domain were answered. We presented the problems encountered in PBC system implementation that overlap with the problems identified earlier (based on the analysis). Although we observed PBC measurement requests in the SNP domain, it seems plausible that the PBC system will be of value in other domains as well, where the need for PBC is present. For example, if we consider the business domain, we might wish to assess the completeness of a marketing database relative to the demographics of the customers, the location of the sale or the types of goods sold - a scenario that we say calls for PBC measurements.

\section{4.5 Conclusions}

In this chapter, we answered the questions raised at the beginning of this chapter by identifying the components of PBC reference architecture that result in three possible types of reference architecture. Given the three types of PBC reference architecture described, we can observe several characteristics of PBC system components. Some components, like the PBC measurement interface, the measurement processor and the universe management interface, are ‘generic’ components as with one implementation of PBC, these components can be reused for

any other PBC systems. Components like the universe, population maps, wrappers and CSs must be configured in some way for specific implementation. For example, the universe must be configured with a schema that describes the specific populations required in an application domain. There are components like the ETL pipeline, mediator and broker that are used to support a specific form of reference architecture. These components must also be configured to suit specific information required (e.g., broker must be configured to tell which populations in an application domain must be retrieved from the universe and which must be gathered directly from the CSs.)

We elicited the efforts in terms of architectural aspects (including software and hardware components) to support PBC measurement providers to answer PBC measurement requests, and to answer PBC measurement requests accurately. We identified the problems that cause inaccurate and incomplete reference populations by examining the activities that are involved in any PBC systems and the components that support the implementation of the activities. Even though some problems can be addressed by current technologies (e.g., a data cleaning tool for duplicates detection), we found that dealing with them in full is an open problem that requires further research.

We analysed the effects of using incomplete and inaccurate reference populations on PBC measurement accuracy. The key lesson drawn from the analysis is the conditions under which incomplete and inaccurate reference populations affect (and do not affect) PBC measurement accuracy. Implementation of PBC for the SNP domain showed how the conceptual PBC reference architecture can be put into practice. The implementation validates the presence of some problems identified based on the analysis conducted prior to the implementation. The principal lesson to be drawn from this chapter is that a PBC system is implementable in practice with proper handling of the problems that are barriers to answering PBC measurement requests and barriers to answering PBC measurement requests accurately. The efforts elicited to deal with the problems should be useful for PBC system providers in deciding whether these efforts are so high that they would prohibit any potential application of PBC.
In Chapter 4, we identified several problems affecting PBC measurement accuracy that must be addressed for complete solutions to PBC systems. PBC systems will be of value if the problems that are barriers to accurately answering PBC measurement requests can be overcome. Time limitations allowed only consideration of one of these problems, which is the storage space constraint. Recall in Chapter 4 that storage space constraints cause a reference population to be incomplete as some individuals that are supposed to be in it are missing. Under certain conditions, measuring PBC using incomplete reference populations results in PBC measurement errors (as presented in Chapter 4, Section 4.3.4 in page 103).

One way to deal with the space constraint issue is by purchasing additional disks. Nevertheless, we argue in this chapter that, optimising the usage of the available space for the universe provides an alternative to purchasing additional disks. However, optimisation of the universe space will only be of benefit for PBC measurement providers if they know the ‘costs’ involved to implement it before deciding whether the costs are acceptable. The effort needed in implementing the optimisation is one type of cost to consider, but the more immediate concern regards accuracy of PBC measurement. The primary question that arises is whether there is a trade-off between the space saved and the accuracy of PBC measurement as a result of the universe optimisation.

To answer this question, in this chapter we explore one space optimisation method that we propose to deal with storage space constraint problem. Basically, this method offers space saving through database schema modification, in
particular by dropping attributes from the schema under consideration. The removal of the attributes, of course, will cause information loss and consequently will affect PBC measurements that rely on those attributes. However, if the missing information can be retrieved from other attribute(s), PBC could still be measured using the smaller universe. We use the term ‘proxies’ for attributes that substitute other attributes in the schema, which is inspired by proxies in other contexts with similar roles (e.g., in voting, a proxy is a person authorised to act on behalf of another [Pet09]). We identified two types of proxy, based on the way these proxies are discovered. These are FD-based proxies (which are discovered from functional dependencies present in the universe schema), and AFD-based proxies (which are discovered from approximate functional dependencies present in the universe schema). An understanding of the space-accuracy trade-offs that both types of proxy could offer is required to facilitate the decisions that PBC measurement designers must make in selecting which attributes can be deleted from the universe schema. Therefore, we set to answer the following questions regarding proxies, with the aim to answer the primary question posted earlier:

- How do proxies contribute to space saving?
- What is the default behaviour of a PBC system that implements proxies?
- How do we select the attributes to drop from the schema?
- What determines the amount of space saving that can be offered by proxies?

This chapter is organised as follows. Section 5.1 outlines the motivations behind the study of space optimisation. Section 5.2 explains our proxy-based approach where an analysis of one factor that determines space saving is presented, along with a case study to support the analysis. Section 5.3 describes how proxies can be implemented in PBC systems, where the components required in PBC reference architecture are identified. The description considers proxies in two phases, namely proxies in use and proxies under consideration. Section 5.4 presents how proxies are assessed where the proxy assessment algorithm is proposed. An analysis of the accuracy of proxies is also presented in this section. Section 5.5 consists of space saving analysis where the best- and the worst-case scenarios of proxy space saving are identified in order to learn about other factors that determine space saving. Section 5.6 presents the cases studies for FD-based proxies and AFD-based proxies in microbial domain. Finally, Section 5.7 concludes.
5.1 Motivation

As discussed in the previous chapter, storing the populations in the universe is problematic if the storage space required is larger than the available space allocated for the universe. A consideration of some sample real world populations shows that data volumes may often be large enough for storage space requirements to become an issue. For example, in a scientific application that studies the commonality of SNPs among species, integration of genes for a single chromosome of a mouse species (Mus musculus) from three public databases, Ensembl\(^1\), NCBI dbSNP\(^2\) and the Perlegen Sciences database\(^3\) produces about 800,000 tuples in total [MEH+07]. More space is required if genes in all chromosomes of the mouse species that making up a gene population in the universe are to be stored. Through the implementation of the SNP PBC system prototype (presented in Chapter 4, Section 4.4 in page 113) about 42 million tuples are stored in SNP universe, covering SNPs for all chromosomes of Mus musculus. Of these tuples, Ensembl contributes 15,514,284 tuples while dbSNP contributes 26,991,031 tuples. In this scenario, at least 40.5 gigabytes of space is needed to store the chromosome population in full and thus avoid using an incomplete chromosome population in PBC measurements.

Storing large populations (such as the gene population) is not an issue if additional space can always be acquired by purchasing new disks every time the space needed exceeds the universe’s space capacity. However, under a strict budget, purchasing new disks might not be feasible especially if frequent space expansion is required (i.e., due to the frequent addition of new population individuals of CSs). Under these circumstances, if we could optimise the space so that free space can be gained, purchasing additional disks might not be necessary.

Space optimisation provides benefits not only when space is highly constrained, such as for handheld devices [PKM03] but also when the concern is to optimise query response time. By reducing the space needed to store the data, we can reduce the time taken for input/output operations for the query, as data are stored in fewer blocks on the disk [CL03, NS90]. In addition, space optimisation could ease the task of administering space expansion that usually requires new infrastructure, increased demand for utilities (for power and cooling), extra floor

---

\(^1\)http://www.ensembl.org/index.html
\(^3\)http://www.perlegen.com/
space, as well as additional staff [Ont06].

In fact, the need to optimise the space is not new, as tools and techniques for this purpose, provided by enterprise data storage vendors (such as Oracle [Ora09, Lai08] and DB2 [Eat06]), have been available in the market for about a decade. At the relational table level, data compression tools, for example, apply a repeated values removal technique to gain free space [Lai08]. In addition, data deduplication techniques remove duplicate records in the table to gain storage space [Fre07]. The idea behind these space optimisation solutions is to exploit the presence of overlaps (of values or records) within tables. Both of these techniques are performed at the level of whole tables. A key (though often unstated) assumption behind these optimisation techniques is that all columns can be exploited for space optimisation. Because of this assumption, knowledge of semantics of applications (i.e., how the columns are used) is ignored. As a result, optimisation is usually performed at a whole table level.

The key lesson we learnt from these optimisation techniques is that, space optimisation techniques that achieve space saving at both schema level and whole tables level are limited. In addition, space optimisation techniques that consider knowledge of semantics of applications are also limited. Because of these limitations, the two techniques described above unfortunately do not fully support solving the storage space constraint issue for a PBC system, where knowledge of how the universe is used for PBC measurements must be considered for space optimisation. Therefore, in this chapter we set to propose a space optimisation technique that addresses the limitations which are barriers solving the PBC storage space constraint issue.

5.2 The Proxy-Based Approach

In this section, we propose a space optimisation technique called the proxy-based approach. In particular, we aim to answer the question of how proxies contribute to space saving. To answer this question, we will identify the types of proxies based on the way the proxies are discovered, and present how the use of proxies contributes to space saving.

The idea behind the technique we propose is to achieve space saving through both schema modification and exploitation of the presence of overlaps. Specifically, space saving through schema modification is achieved by dropping some
attributes from the schema. If some attributes are dropped from the schema, the amount of space saved is roughly determined by the number of attributes being dropped and the number of tuples the table contains. For example, consider a table which consists of 100 tuples, with several attributes in its schema. If we drop an attribute from the schema, then the amount of space saved is 100 units of instances\(^4\) (which is of course, is convertible to disk storage unit in bytes).

The question that arises is whether all attributes in the schema are droppable. To answer this question we need to understand the semantics of the application. As for PBC, we need to understand how the universe is used in answering PBC measurement requests, in particular, how attributes in the schema of the universe’s population tables are used.

In Chapter 3, we presented what information was needed in order to answer a range of possible PBC measurement requests (see Section 3.2.1 in page 44). To recall, we defined the generic schema of a population table that consists of information needed to answer all types of PBC measurement identified as \(P = \langle I, \text{source}, A \rangle\), where \(I\) is the set of identifier attributes, \(\text{source}\) is the attribute used to express the query predicates needed to extract individuals that originated from specific CSs and \(A\) is the set of attributes used to express the query predicates needed to extract sub-populations from the population table. Essentially, the identifier attribute is used in every query that retrieves the reference population or the reference sub-population (i.e., constrained by certain attributes or CSs). The identifier attribute has an essential role in identifying population individuals, in both the reference population and the data set under measure. If we substitute the identifier attribute with a proxy, we need to assume that the same proxy is available in the source from which the person requesting the PBC measurement extracted his/her data set under measure. However, in the situation where there is a lack of overlaps between the source from which the data set under measure is extracted and the universe, substituting the identifier attribute with a proxy will create a barrier for the PBC measurement provider attempting to answer PBC measurement requests. Therefore, because there is uncertainty as to whether the same proxy for the identifier attribute will be available in the source where the data set under measure is gathered, we regard identifier attributes as not droppable from the schema. This leaves the options of droppable attributes to those other than the identifier attribute.

\(^4\)We regard each cell in a common relational table as an instance.
However, the problem is that dropping attributes from their schema will cause information loss. Therefore, we must find other attributes that we refer to as proxies to substitute for the droppable attributes. The proxy and the droppable attribute must be related in some way to compensate for the information loss.

A particular kind of relationship that can be observed among attributes in relational tables is a functional dependency (FD). In the literature, FD is described as a unique-value constraint, commonly used for relational schema design [MUJ08]. If an FD applies to a table, the value of an attribute on the table can be uniquely determined by the values of some other attributes [HKPT99]. An FD over a relation schema $R$ is denoted by an expression $X \rightarrow A$ where $X \subseteq R$ and $A \in R$. $X$ in the expression represents the set of attributes, known as ‘determinant attributes’ whose values can uniquely determine the values of $A$. The rule regarding FDs states that $X \rightarrow A$ is valid within relation $r$ with schema $R$, provided that for all pairs of tuples $t, u \in r$ we have:

$$t[X] = u[X] \Rightarrow t[A] = u[A]$$

where $A, X \subseteq R$ [HKPT99].

If FDs within a table such as $X \rightarrow A$ hold, we suspect that if $A$ is substituted by $X$, the information loss caused by dropping $A$ from the schema can be compensated in full.

Another type of relationship that can be observed among attributes in relational tables is an approximate functional dependency (AFD). An AFD is an FD that holds on all but a small fraction of tuples where, these tuples represent errors, or exceptions to the FD rule [HKPT99]. An FD is in fact an AFD without errors, (i.e., a special case of AFD). An AFD over a relational schema $R$ is written as $X \rightsquigarrow A$ where $X \subseteq R$ and $A \in R$ [WKC+07]. The squiggled arrow in the expression represents the possible non-compliance of some tuples with the dependency rule. The fraction of tuples that violates the functional dependency rule can be measured using the $g_3$ measure [KM92]. The $g_3$ score for an AFD $X \rightsquigarrow A$ on a relation $r$ indicates the minimum number of tuples that need to be removed from $r$ in order for it to comply with the FD $X \rightarrow A$. If AFDs within a table such as $X \rightsquigarrow A$ hold, we suspect that if $A$ is substituted by $X$, the information loss caused by dropping $A$ from the schema can be compensated, but not in full.

Therefore, based on our observations on the properties of FD and AFD, we say that, proxies for the droppable attributes can be found by discovering the relationships among attributes in the population tables where FD and AFD are
present. However, because there is uncertainty about the amount of information loss (or even the amount of space saving) as the result of using proxies especially for AFD-based proxies, we say that the proxies must be assessed before they can be accepted as substitutes for the droppable attribute. We will present the details of proxy assessment later, in Section 5.4.

Discovering proxies based on FD is quite straightforward as they are usually key attributes within a relational table, especially if the information about the ‘constraints’ (i.e., PRIMARY KEY constraint) defined on the table is available. However, discovering AFD is not trivial as it requires information beyond what database designers can provide. In the literature, one AFD discovery algorithm called TANE proposed by Huhtale et al. [HKPT99] was used in several applications such as semantic queries for query evaluation optimisation [NK04], accessing deep web results by relaxing the queries that fail to produce satisfactory results due to the constraints of web query interfaces [MSKL08] and, query rewriting that deals with nulls in query answers [WKC07, WKK09]. We propose that TANE is useful in our context as well because the algorithm does not only offer to discover AFD-based proxies, but also to discover FD-based proxies in the absence of the information about the types of constraints defined on the table; or in the case where FD-based proxies do not fall under the key attributes category.

The question that arises if proxies are accepted to substitute the droppable attributes is, how information loss as the result of dropping the droppable attributes can be compensated by the proxies? Space optimisation techniques described earlier (e.g., based on compression) store the details of the removed overlaps in the meta-data in order to compensate information loss. These techniques implement algorithms that will retrieve the removed overlaps from the meta-data, every time a query is submitted against the compressed (or deduplicated) tables. To answer the question just raised, we adopted a similar way to compensate information loss where we use a proxy map to store the mappings between the values of the droppable attributes and the values of their proxies. For example, consider a and b are the values of a droppable attribute. A proxy map consists of the following mappings:

\[
\begin{align*}
a & \rightarrow \{1, 2, 3, 4\}, \\
b & \rightarrow \{5, 6, 7, 8\},
\end{align*}
\]
where the numeric values are the values of the proxy for the droppable attribute and the arrow illustrates a ‘map to’ relationship. Therefore, every time a query is submitted against a population table, the dropped attribute’s values must be retrieved from the proxy map (for the query that has the dropped attribute in its predicate(s)).

5.2.1 Implementation Options of Proxy Maps

As storing the proxy map requires some space, the amount of space used by the proxy map must be taken into account, rather than considering space saving by the droppable attributes alone. Therefore, we define space saving as:

\[
\text{spaceSaving} = \frac{\text{droppableAttrSize} - \text{proxyMapSize}}{\text{droppableAttrSize}} \times 100, \tag{5.1}
\]

where \text{droppableAttrSize} is the size of the droppable attribute, \text{proxyMapSize} is the size of proxy map. The number of instances is used to represent the size of these variables, that is independent of any specific storage and file organisation system. One key criterion of a useful proxy, in terms of space saving, is that the amount of space required for the proxy map is smaller than the amount of space saved by dropping the attribute it substitutes for. Therefore, space saving through proxies will be of benefit if we could ‘minimise’ the space required to store the proxy map. The question that arises is how can we minimise the space required to store the proxy map? To answer this question, we need to identify the possible structures of a proxy map and compare them in terms of their size. Essentially, each proxy map must consist of the values of the droppable attributes and the values of the proxies that map to the values of the droppable attribute.

We identified two possible structures for a proxy map as follows:

- **A pure relational table**: in its simplest form, a proxy map is in pure relational table structure for schema: \((\text{droppableAttr}, \text{proxy})\). For this structure, each value of the droppable attribute will be mapped to exactly one value of the proxy. Because of this characteristic, each droppable attribute value that is substituted by multiple proxy values will be stored repeatedly in the table. For example, Table 5.1 shows an artificial proxy map in pure relational structure where, \(A\) is the droppable attribute and \(B\) is the proxy. Based on the table, 16 instances are stored within the proxy map.
As it is also possible for a proxy to be a composite proxy, which is a combination of two or more attributes called partial proxies, the schema of the proxy map in pure relational structure is in the form of: 
\[
\langle \text{droppableAttr}, \text{partialProxy}_1, \text{partialProxy}_2...\text{partialProxy}_n \rangle
\]
where \( n \) is the number of partial proxy attributes. For example, the mappings between the droppable attribute values and the composite proxy values are:

\[
\begin{align*}
\text{a} & \rightarrow \{(1,x)\}, \\
\text{a} & \rightarrow \{(2,y)\}, \\
\text{b} & \rightarrow \{(5,y)\}, \\
\text{b} & \rightarrow \{(6,z)\},
\end{align*}
\]

where the values on the left hand-side of the mappings are the droppable attribute values, while the values of the composite proxies are on the right hand-side.

- A multi-valued table: like the pure relational table, a proxy map in a multi-valued table structure has the schema: \( \langle \text{droppableAttr}, \text{proxy} \rangle \). However, unlike the pure relational table, each value of the droppable attribute will be mapped to a set of proxy values (that may be a singleton set). Because of this characteristic, each droppable attribute value that is substituted by multiple proxy values will be stored only once in the table. This structure is also called as Vectorised dictionary based minimal DSM (VD-MDSM) proposed by Rahman, Schallehn and Saake [RSS11], which is a by-product of the decomposition storage model (DSM) by Copeland and Khoshafian [CK85]. The essence of this form of table structure in those works is to store all values of the same attribute of the relational conceptual schema relation together, for the advantage of performance (e.g., less database tuning required) and increased data independence and availability [CK85, RSS11]. For example, Table 5.2 shows an artificial proxy map in a multi-valued table structure, where \( A \) is the droppable attribute and \( B \) is the proxy. Based on the table, 10 instances are stored within the proxy map.

To store composite proxies within a multi-valued table structure, the same schema structure is used for the non-composite proxy, which is:
\langle \text{droppableAttr, proxy} \rangle$, where the instances of the composite proxy are stored in one column. For example, the mappings between the droppable attribute values and the composite proxy values are:

\[
a \rightarrow \{(1, x), (2, y)\}, \\
b \rightarrow \{(5, y), (6, z)\}.
\]

Table 5.1: A Proxy Map in a Pure Relational Table.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>6</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
</tr>
<tr>
<td>b</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 5.2: A Proxy Map in a Multi-valued Table.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1,2,3,4</td>
</tr>
<tr>
<td>b</td>
<td>5,6,7,8</td>
</tr>
</tbody>
</table>

In comparison, the example shows that the size of proxy map in the multi-valued table structure (in terms of the number of instances stored) is smaller than the size of the proxy map stored in the pure relational structure. By adopting a multi-valued table structure in this example, we optimise the proxy table as a whole, by removing the repeated values (of the droppable attribute) which, if readers may notice, is a similar approach used by space optimisation techniques described earlier (see Section 5.1 in page 131). Therefore, in the example, the space required to store the proxy map can be minimised by storing the proxy map in a multi-valued table structure. By minimising the storage requirement for the proxy map, we gain the benefit of space saving, which in the example above, is a saving of 6 instances. From the example, we say that storing the proxy map using a multi-valued table is useful in the situation where the droppable attribute values
are mapped to multiple values of the proxy. To see whether a multi-valued table is useful by examining the size of proxy maps using real data sets, we conducted a small case study in the microbial domain.

We use microbial data sets taken from the Comprehensive Microbial Resources (CMR)\(^5\) that were downloaded in the form of SQL dumps. In particular, we used three tables, namely evidence, taxon and bug.attribute to represent our sample population tables. For brevity, we assume that the individuals of each population come from a single source (CMR). The schemas of these tables are as follows:

- **Table 1**: Evidence = \(\langle \text{id}, \text{ev.type}, \text{method}, \text{assignby} \rangle\),
- **Table 2**: Taxon = \(\langle \text{uid}, \text{genus}, \text{species}, \text{kingdom}, \text{ir1} \rangle\),
- **Table 3**: Bug.attribute = \(\langle \text{id}, \text{att.type}, \text{assignby} \rangle\).

**evidence** consists of the individuals of the microbes’ evidence population, in which information about the type of evidence found about the presence of microbes, the method used to discovered them and the person (laboratory) who discovers them are stored; **taxon** consists of the individuals of the microbes’ taxonomic population, in which information about microbes’ genus, species, kingdom and intermediate ranking (of type 1) are recorded; **bug.attribute** consists of the individuals of insects/bugs population where microbes were found, in which information about how the information is retrieved (attribute type) and the person (laboratory) who discovered them are stored.

The underlined attributes are the keys for the tables. Each key attribute has been selected as the proxy (based on the presence of an FD) for all non-key attributes. The schema of the proxy tables are:

- from evidence:
  \(\langle \text{ev.type}, \text{id} \rangle, \langle \text{method}, \text{id} \rangle, \langle \text{assignby}, \text{id} \rangle\).

- from taxon:
  \(\langle \text{genus}, \text{uid} \rangle, \langle \text{species}, \text{uid} \rangle, \langle \text{kingdom}, \text{uid} \rangle, \langle \text{ir1}, \text{uid} \rangle\).

- from bug.attribute:

\(^5\)http://cmr.jcvi.org/tigr-scripts/CMR/CmrHomePage.cgi
We compared the size of proxy maps that are stored using a pure relational table structure and a multi-valued table structure. The sizes of the proxy maps were determined by counting the number of instances they contain. We use a ‘working’ proxy map to compute the size of proxy maps in both table structures, by issuing the following query against the population table, \( p \):

\[
\text{workProxyMap} = \text{SELECT DISTINCT droppableAttr, proxy FROM } p;
\]  \( (5.2) \)

where \( \text{workProxyMap} \) is the working proxy map, \( \text{droppableAttr} \) is the droppable attribute and \( \text{proxy} \) is the proxy attribute.

The number of instances of the proxy map stored in the pure relational structure is counted based on the number of instances in the droppable attribute column and the proxy column retrieved by issuing the following queries:

\[
\text{droppableAttrInst} = \text{SELECT COUNT(droppableAttr) FROM workProxyMap};
\]

\[
\text{proxyInst} = \text{SELECT COUNT(proxy) FROM workProxyMap};
\]  \( (5.3) \)

where, \( \text{droppableAttrInst} \) is the number of instances in the droppable column of the proxy map, while \( \text{proxyInst} \) is the number of instances in the proxy column of the proxy map.

To count the number of instances of the proxy map stored in the multi-valued table schema, we use the working proxy map retrieved earlier by the query presented in Equation (5.2). We issue the following queries against the working proxy map to get the number of instances in the droppable attribute column and in the proxy column:

\[
\text{droppableAttrInst} = \text{SELECT COUNT DISTINCT(droppableAttr) FROM workProxyMap};
\]

\[
\text{proxyInst} = \text{SELECT COUNT(proxy) FROM workProxyMap};
\]  \( (5.4) \)

where, \( \text{droppableAttrInst} \) is the number of distinct (i.e., non-repeated) instances in the droppable column of the proxy map, while \( \text{proxyInst} \) is the number of instances in the proxy column of the proxy map.
Thus, the size of the proxy map in both table structures is computed as:

\[
proxyMapSize = droppableAttrInst + \sum_{i=1}^{n} proxyInst_i, \tag{5.5}
\]

where \(proxyMapSize\) is the size of the proxy map and \(\sum_{i=1}^{n} proxyInst_i\) is the sum of the number of instances in \(n\) partial proxy columns of the working proxy map. Note that since in this case study all droppable attributes have a single-attribute proxy, then \(n = 1\).

Unlike \(droppableAttrInst\) for a pure relational proxy map (see Equation (5.3)), repeated droppable attribute values are removed through the \texttt{DISTINCT} operator for \(droppableAttrInst\) in the proxy map stored in the multi-valued table (see Equation (5.4)). This is the distinguishing characteristic between the proxy map stored in a pure relational table and the proxy map stored in a multi-valued table.

Figure 5.1(a), Figure 5.1(b) and Figure 5.1(c) illustrates the size of proxy maps by attributes in \texttt{Evidence}, \texttt{Taxon} and \texttt{Bug.attribute} respectively. These figures show a common result, which is the size of all proxy maps that are stored in a multi-valued table are smaller than the size of proxy maps in a pure relational structure. In addition, we can also see that for proxy maps in a pure relational table, the size of the proxy maps are the same for all attributes that belong to the same table. This shows that the size of proxy maps in pure relational tables is insensitive to the number of repeated droppable attribute values that map with multiple values of the proxy. In contrast, the size of proxy maps stored in multi-valued table is sensitive to the amount of repeated droppable attribute values that map with multiple values of the proxy. This means with a multi-valued table, the benefit of space saving is gained by removing the repeated droppable attribute values. Therefore, the result of this case study suggests the multi-valued table structure as a better structure to adopt than the pure relational table structure.

To support the PBC measurement provider in selecting the table structure to store proxy maps, we suggest checking on whether the values in the droppable attribute domain map with multiple values of the proxy (i.e., whether we have repeated droppable attribute values for multiple values of proxy). Checking of the droppable attribute domain’s value characteristic can be performed by issuing
(a) Proxy map size by attributes from evidence

(b) Proxy map size by attributes from taxon

(c) Proxy map size by attributes from bug_attribute

Figure 5.1: A Comparison of Proxy Map Size by Table Structure.
the following query:

\[
\text{SELECT } \text{droppableAttr}, \text{ COUNT}(\text{proxy}) \text{ FROM } p \text{ GROUP BY } \text{droppableAttr};
\]  

(5.6)

To conclude, in this section we answered the question of how the proxy-based approach that we proposed would save storage space, which contributes to space optimisation that offers both selective schema modification and repeated values removal. In the next section, how proxies are implemented in a PBC system will be presented.

5.3 Proxies in a PBC System

In this section, we set to answer the question: what is the default behaviour of a PBC system that implements proxies? We will elicit the default behaviour of a PBC system with the aim of supporting PBC measurement providers in understanding the effort required in optimising space through proxies and in deciding if this effort acceptable. In Chapter 4 we presented a Space optimiser as a component within the PBC reference architecture that deals with the storage space constraint issue (see Figure 4.9 in page 86) with the assumption that, the space optimiser is triggered by the lack of storage during the data loading process. In this Chapter, we specifically propose proxies for space optimisation, and consider another trigger for space optimisation, which is the awareness of space limits. In this situation, space optimisation is rather a precautionary step taken before the universe faces an actual lack of storage. Regardless what triggers space optimisation, we need to understand what is the default behaviour of PBC systems that use proxies as their space optimisation technique.

We observed the default behavior of PBC systems in two separate stages which are, the stage where the proxies are in use and, the stage where proxies are under consideration. In describing each stage of proxies, we set to answer the following questions:

- What functions are required?
- What components must be added to the PBC reference architecture?
- What are the input(s) and output(s) of the components?
5.3.1 Proxies in Use

In the absence of proxies, the default behaviour of any PBC system during PBC measurement is: to accept PBC measurement requests by data analysts; instantiate the queries to get reference populations (based on the parameters specified in the requests); submit the queries against the universe; compute PBC and return the results to the person who issued the PBC measurement requests. The components involved in performing the functions in the default behaviour just mentioned are the PBC measurement interface, the measurement processor and the universe.

With proxies being implemented in the PBC system, we need to add some functions and components to the system. Figure 5.2 illustrates the components added to the PBC reference architecture (illustrated in red). These additional components are essential to perform proxy-related functions, alongside other components that perform the default behaviour of the PBC system. The functions of these components are as follows:

- Every query instantiated by the Measurement processor based on the parameters of the PBC measurement request must be validated in order to check whether proxies are needed or not. In particular, a component called the Query manager is added to instantiate the queries and validate them. As shown in Figure 5.2, data flow a1 is the input for the Query manager, which consists of the data set under measure and some parameters.

If the person requesting the PBC measurement wishes to use a reference population, only the name of the population table must be provided as a parameter. However, if a reference sub-population is desired then, in addition to the name of population table, the constraints of the reference sub-population must be provided as parameters. For example, the name of the CSs is a type of parameter used to specify the constraint. To support parameter specification, PBC measurement provider should consider adding selectable options into the PBC measurement interface for those wishing to use reference sub-populations.

- The Query manager will instantiate the queries based on the parameters received. Before these queries can be submitted to the universe, they must be validated. One validation performed by the Query manager is to check if dropped attributes are used in the query predicates. In particular, the
Figure 5.2: Proxy Components in the PBC Reference Architecture.
function must focus on queries on the reference sub-population because the attributes used to express query predicates are droppable attributes. These queries are in form of:

\[ Q = \Pi_{\text{key}(p)}(\sigma_{\text{COND}}p), \]  

where \( p \) is the population table, \( \text{key}(p) \) is a function that retrieves the identifier attribute(s) in \( p \) and \( \text{COND} \) is a conjunction of conditions on \( p \). The result of the validation process is a set of conditions that consists of the dropped attributes.

Queries that do not have any dropped attributes in \( \text{COND} \) will be submitted to the universe (data flow a3). However, if the dropped attributes are present in \( \text{COND} \), the query transformation function will be invoked by the Query manager.

- The query transformation function takes the reference sub-population queries and the set of conditions that consists of dropped attributes as its input. Within a query transformation, the Query manager will retrieve the values of the proxies from the proxy map (data flow p2), by issuing a request (data flow p1). Each proxy map is named after the dropped attribute name and its proxy name, where the schema of the proxy map relies on how it is stored (i.e. as a pure relational table, or a multi-valued table). With proxy maps stored as relational tables (either in pure relational table or multi-valued tables), the request is in the form of a query.

The transformed query will be submitted by the Query manager to the universe (data flow p3). In return, the universe will return the query results (which is the requested reference sub-population), to the Measurement processor for PBC measurement computation. The PBC measurement result will be passed to the person requesting the PBC measurement (data flow a2). For the details of the query transformation step, we will explain the algorithm of the query transformation function, called \text{transformQuery()} in sub section 5.3.1.1.

- To know which proxy map to use, the Query manager must refer to the proxy configuration. As shown in Figure 5.2, data flow p2 is the input for the Query manager, which is the name of the proxy map retrieved from the
proxy configuration. Note that data flow p2 is also used to represent the values of proxies retrieved from the proxy maps. In Section 5.3.2, we will describe the information that must be stored about proxies in the proxy configuration.

5.3.1.1 Query Transformation Algorithm

The input to the query transformation algorithm is the query instantiated by the Measurement processor, $Q$ (as shown in Equation 5.7, page 146), and the output is a transformed query $Q'$. In the simplest case, there is only one condition specified in $Q$, and the dropped attribute has only one proxy. To illustrate this simplest case, suppose that there is only one condition in $Q$. Consider further that $COND = (\text{droppedAttr} = v)$, where $v$ is the value of the dropped attribute (droppedAttr). To transform the query, the algorithm must gather the name of the proxy and the name of the proxy map of the dropped attribute from the proxy configuration. Where the name of the proxy is $\text{proxy}$ and the name of the proxy map is $\text{droppedAttr}_\text{proxy}$, this algorithm will get the set of proxy values ($S$) to substitute the dropped attribute value $v$, as specified in $COND$, where,

$$S = \Pi_{\text{proxy}}(\sigma_{COND \ \text{droppedAttr}_\text{proxy}}).$$  

(5.8)

Based on $S$, the algorithm transforms $COND$ into $COND' = \text{proxy} \in S$, and transforms $Q$ as: $Q' = \Pi_{key(p)}(\sigma_{COND' p})$.

The query transformation algorithm of course must handle cases which are beyond the simplest case, such as when:

- there are multiple conditions in the query, where the dropped attribute is present in one or more conditions, or

- the dropped attribute has more than one proxy. Based on the result of the proxy assessment, for each droppable attribute, the PBC measurement provider selects a proxy that is acceptable. Due to the differences in the contents of CSs, the selected proxy might not be a common proxy that is acceptable across the CSs. This situation may result in multiple proxies for a dropped attribute.

The query transformation algorithm we created as shown in Algorithm 1 covers the cases stated above. Note that in the algorithm, we use a convention where, all
variables and functions are in italics. Set variables are in upper case and non-set variable are in lower case; types are in non-italic font; actual instances are in typewriter font. The definition of all local variables is in Table 5.4.

The query transformation algorithm takes the basic configuration of the PBC system (called basicConfig as proposed in Chapter 3), a query \( Q \), the conditions that consist of the dropped attributes \( DROPPABLECOND \) and the proxy configuration information stored in a table called \( \text{proxyInfo} \) as the inputs. For convenience, we restate the the basicConfig in Chapter 3 as shown in Table 5.3 in this chapter.

The first task is to remove the conditions that consist of the dropped attributes \( \text{droppedAttr} \) from a conjunction of conditions \( \text{COND} \). This task is performed by a function called \( \text{removeCond()} \) that takes \( \text{COND} \) and \( DROPPABLECOND \) as its inputs. Then, for each condition \( \text{cond} \), the algorithm extracts the droppable attribute from the condition using the \( \text{extractAttribute()} \) function, where \( \text{cond} \) is the input for this function. Then, the algorithm retrieves the proxies for the dropped attribute that is present in the condition from the proxy configuration. This task is performed by issuing a query against \( \text{proxyInfo} \) table that has the schema:
\[
\langle \text{popTableName}, \text{droppedAttrName}, \text{proxyName}, \text{proxyMapName} \rangle
\]
where proxy configuration information is stored. A set of proxies \( \text{PROXY} \) of the dropped attribute is the result of the query.

Next, for each proxy \( \text{proxy} \) in \( \text{PROXY} \), the algorithm must retrieve the proxy map for the dropped attribute that it substitutes from the proxy configuration. The proxy map is retrieved by issuing a query against the \( \text{proxyInfo} \) where the result of the query is the name of the proxy map \( \text{droppedAttr_proxy} \). Then a set of proxies will be retrieved (through the query shown in Equation (5.8)), and the condition will be transformed into \( \text{cond}' \) that will substitute \( \text{cond} \). \( \text{COND} \) will be redefined in every iteration of the conditions in \( DROPPABLECOND \) in the loop in which the transformed conditions are accumulated. Finally, the algorithm returns the transformed query \( \text{Q}' \) that will be submitted to the universe by the Query manager.

### 5.3.2 Proxies Under Consideration

In supporting the PBC measurement provider on deciding on the implementation of proxies, some components must be added in the PBC reference architecture.
Algorithm 1: query transformation

Input: basicConfig, Q, DROPPABLECOND, proxyInfo table, with schema:

\[ \langle \text{popTableName}, \text{droppedAttrName}, \text{proxyName}, \text{proxyMapName} \rangle \]

Output: \( Q' \)

1 begin
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16 end
Table 5.3: Basic Configuration of a PBC System (basicConfig)

(a) Basic variables and their types

<table>
<thead>
<tr>
<th>The variables and their types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS : $\mathcal{P}(\text{SOURCE})$</td>
<td>The set of all CSs that populate the universe.</td>
</tr>
<tr>
<td>$U_P : \mathcal{P}(\text{NAME} \times \text{POPSHEMA})$</td>
<td>The set of all population tables in the universe, where each element is a double representing the name of the population table and its schema.</td>
</tr>
</tbody>
</table>

(b) Basic type definitions

<table>
<thead>
<tr>
<th>The basic types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME: $\mathcal{P}(\text{STRING})$</td>
<td>The set of all legal identifying names of CSs, tables and attributes.</td>
</tr>
<tr>
<td>QUERY: $\mathcal{P}(\text{STRING})$</td>
<td>The set of all queries on population tables or on CSs in the form of SQL strings. For brevity, we assume that all queries in QUERY are legal queries against the schema of the population tables (or the schema of CSs). In practice, checks for this would have to be made in the algorithms.</td>
</tr>
<tr>
<td>POPSCHEMA: $\mathcal{P}(\mathcal{P}(\text{NAME}) \times \text{NAME} \times \mathcal{P}(\text{NAME}))$</td>
<td>The set of all possible population table schemas where each element is a triple representing the name(s) of population’s identifier attributes, the name of an attribute for the CS and the names of attributes other than identifiers and CS.</td>
</tr>
<tr>
<td>SOURCE: $\mathcal{P}(\text{NAME} \times \text{URI} \times \text{POPMAP})$</td>
<td>The set of all possible source descriptions where each element is a triple representing a specific source, and consisting of the name of the CS, its URI for executing queries, and a mapping between the population tables in the universe and the tables in the source’s schema.</td>
</tr>
</tbody>
</table>

The main additional component is called the Proxy assessor, which consists of some functions to assess the proxies. Proxies are assessed in order to determine which attributes are droppable from the population table schema. One criterion to assess is space saving. We say that an attribute can be dropped if it has a proxy(s) that has an acceptable space saving. Nevertheless, as providing accurate answers for PBC measurement requests is the goal of any PBC systems, we add accuracy (of PBC measurement) as another criterion to consider in proxy assessment. With this added criterion, we now say that an attribute can be dropped if it has a proxy(s) that has an acceptable space saving and is accurate. As it is impossible to set standard acceptable proxy criteria for all applications, the Proxy assessor will present the assessment results to the PBC system administrator who will decide if an attribute can be dropped or not. We will provide the details of the proxy assessment in Section 5.4.

As shown in Figure 5.2, data flow $p_4$ is a request to optimise space that triggers the Proxy assessor, which occurs either during Setup or Maintenance of the PBC system. We considered two types of triggers, which are the lack of storage during the data loading process and an awareness of space limitation. Lack of storage is detected by a component called the Proxy manager. Before the transformed query results from the CSs (data flow $a_8'$) are loaded into the universe, the
Table 5.4: Local Variable Definitions for the Query Transformation Algorithm

<table>
<thead>
<tr>
<th>The variables and their types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND : P(STRING)</td>
<td>The conjunction conditions on a population table.</td>
</tr>
<tr>
<td>DROPPABLECOND : P(STRING)</td>
<td>The conjunction conditions on a population table which consists of the dropped attributes.</td>
</tr>
<tr>
<td>droppedAttribute : P(NAME)</td>
<td>The name of the dropped attribute.</td>
</tr>
<tr>
<td>droppedAttr_proxy : P(NAME)</td>
<td>The name of a proxy map.</td>
</tr>
<tr>
<td>PROXY : P(NAME)</td>
<td>The set of all proxy names.</td>
</tr>
<tr>
<td>Q : P(QUERY)</td>
<td>A query instantiated by Measurement processor (see Equation (5.7)).</td>
</tr>
<tr>
<td>Q' : P(QUERY)</td>
<td>A transformed query.</td>
</tr>
<tr>
<td>S : P(STRING)</td>
<td>The set of proxy values.</td>
</tr>
</tbody>
</table>

Proxy manager evaluates whether the universe’s storage space is adequate. The validation process requires information about universe’s storage capacity (e.g., in bytes unit) as an input, which is retrieved from the PBC components configuration. Because of this requirement, we need to add storage capacity information in the universe configuration along with the information about the population tables. In addition to storage space information, the Proxy manager also takes the transformed query results as an input in order to compute the space that will be required. If the amount of storage needed exceeds the amount of storage available, then the Proxy manager will invoke the Proxy assessor. Otherwise, the transformed query results will be stored in the universe, through the Universe management interface.

Due to awareness of the universe’s limited storage, the system administrator invokes the The Proxy assessor by sending a space optimisation request. Upon acceptance of the space optimisation request either from the Proxy manager or from the administrator, Proxy assessor sends queries to the universe through the Universe management interface (data flow p5) to get the data sets for space saving and accuracy assessment. For example, one query issued is to decide on the structure of the proxy map to use (see the SQL statement (5.6), page 143). The results of the query (i.e data sets for proxy assessment) are gathered from the universe as shown by data flow p6, and are passed to the Proxy assessor by the Universe management interface. The query results are then stored in a temporary storage (data flow p9). These data sets for proxy assessment will be retrieved from the temporary storage by the Proxy assessor when needed. The result of the assessment and proxy maps information (data flow p7) is sent to the system administrator to decide which attributes to drop.

The system administrator issues a command to drop the selected attributes.
from the population table schema (data flow p8) and notifies the Proxy manager of the amount of space saved as the result of dropping the selected attributes (data flow p10). Information about the amount of space saved is included in the result of the proxy assessment. The Proxy manager loads the transformed query results into the universe given the space saving yielded via space optimisation (data flow a8’). During the assessment of space saving, the proxy maps used in the assessment are stored in temporary storage. The proxy maps for proxies of attributes that were selected by the administrator are extracted from the temporary storage by the Proxy manager (data flow p11) and stored in the selected table structure (e.g., multi-valued table). Proxy maps for proxies of unselected attributes and other assessment data sets (e.g., used for accuracy assessment) are deleted based on the delete command issued by the Proxy manager (data flow p12). The PBC measurement designer performs proxy configuration by creating the schema for the proxy configuration and stores the proxy information within it. As mentioned in Section 5.3.1.1, proxy information is stored as a table called proxyInfo. To recall, the schema of proxyInfo is \(<\text{popTableName}, \text{droppedAttrName}, \text{proxyName}, \text{proxyMapName}>\). To configure the proxies, the PBC measurement designer gets information about the selected droppable attributes together with their proxies and proxy maps from the system administrator. Table 5.5 shows the proxy configuration (proxyConfig) that consists of definitions of all proxy-related variables and types. For example, the type used to configure proxyInfo is PROXYCONFIGSTORE. Other variables and types defined in proxyConfig will be used in the proxy assessment which will be presented in the next section.

5.4 Proxy Assessment

To answer the question of how to select attributes to drop from a population table schema, we need to assess the attributes that are droppable. We described earlier that knowledge of the semantic of PBC is used to determine droppable attributes. Thus given a schema of a population table, we say that only non-identifier attributes are droppable, for the reason stated earlier in the Section 5.2. Within a population table, each droppable attribute usually has more than one attribute that is plausible to be its proxy. We call these attribute proxy candidates. We assess the droppable attributes by assessing their proxy candidates. Two criteria
Table 5.5: Proxy Configuration (proxyConfig)

<table>
<thead>
<tr>
<th>The variables and their types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALLPROXYRESULTS</strong>: $P(\text{NAME} \times \text{NAME} \times \text{NAME} \times \text{NAME} \times \text{FLOAT} \times \text{FLOAT})$</td>
<td>The sets of proxy assessment results of all CSs where each element is a sextuple representing the name of a source, the name of a population table, the name of a droppable attribute, the name of a proxy attribute, the space saving score and the accuracy score.</td>
</tr>
<tr>
<td><strong>ALLTANERESULTS</strong>: $P(\text{NAME} \times \text{NAME} \times \text{TANERESULT})$</td>
<td>The sets of all TANE results where each element is a triple representing name of a source, the name of a population table, and the results of TANE.</td>
</tr>
<tr>
<td><strong>DA</strong>: $P(\text{NAME} \times \text{NAME})$</td>
<td>The set of all selected droppable attributes where each element is a double representing the name of the droppable attribute and the name of the population table.</td>
</tr>
<tr>
<td><strong>UPD</strong>: $P(\text{NAME} \times \text{POPSHEMA})$</td>
<td>The set of all population tables in the universe that consists of the selected droppable attributes, where each element is a double representing the name of the population table and its schema.</td>
</tr>
</tbody>
</table>

The types

<table>
<thead>
<tr>
<th>The types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROXYCONFIGSTORE</strong>: $P(\text{NAME} \times \text{NAME} \times \text{NAME} \times \text{NAME})$</td>
<td>The set of proxy details, where each element is a quadruple representing the name of the population table, the name of the dropped attribute, the name of the proxy attribute and the name of the proxy map.</td>
</tr>
<tr>
<td><strong>TANERESULT</strong>: $P(\text{FLOAT} \times P(\text{NAME} \times \text{NAME}))$</td>
<td>The set of TANE results, where each element is a double representing a g3 value (which is AFD scoring as a percentage) and a pair of attributes.</td>
</tr>
</tbody>
</table>

are of interest, the amount of space saving and the accuracy. Among these proxy candidates, only one proxy candidate will be selected to be the proxy of a droppable attribute, the one that has acceptable space saving and accuracy criteria. The type of proxy candidates is determined based on the relationship they have with the droppable attributes. As described earlier, FD and AFD are two forms of relationships that can be observed between a proxy and a droppable attribute. Therefore, proxy assessment covers these two types of proxy candidates.

To assess proxy candidates, several questions arise. These questions are:

- Which droppable attributes should be examined?,
- Should we examine a droppable attribute as a whole (regardless of the CSs), or as a subset of a specific CS?,
- How can we find proxy candidates given the selected droppable attribute?

We will answer these questions in turn and present a proxy assessment algorithm as the result.

Within a population table schema, multiple attributes are droppable. One requirement for the proxy assessment is to know which of the droppable attributes the PBC measurement providers wish to examine. The selection of droppable...
attributes might be based on personal preferences (e.g., made by the data administrator), or it might be the case that all attributes in the schema are chosen for examination. Therefore the question of which droppable attributes to examine depends on how PBC measurement providers make the selection. The set of selected droppable attributes to examine is the input for proxy assessment.

Given the selected droppable attributes, we must then find their proxy candidates to assess. However, because the tuples within a population table come from different CSs, the question is, can we find one proxy candidate that is acceptable for all CSs? Due to the differences in the contents of CSs, a droppable attribute might have different acceptable proxy candidates for different CSs. For example, consider a schema of a population table \( \langle A, B, C, D, E, \text{source} \rangle \), where the tuples are from \text{source} 1, \text{source} 2 and \text{source} 3. Based on the result of the proxy assessment, for a droppable attribute \( B \):

- The proxy candidate that is acceptable for \text{source} 1 is \( C \).
- The proxy candidate that is acceptable for \text{source} 2 is \( D \).
- The proxy candidate that is acceptable for \text{source} 3 is \( (C, E) \) (which is a composite proxy).

In this example, \( B \) has three different proxies that are acceptable for three different CSs. To consider this possibility, we propose that, the search for proxy candidates should be made against the contents of the individual CS, rather than against the integrated contents of the CSs. Therefore, proxy candidates are assessed at an individual CS level, and the proxy candidates are searched for using the tuples set of the population table retrieved from an individual CS. The question that arises is how can we find proxy candidates using these tuples set of the droppable attributes? To answer this question, we need to know how the relationships, in particular FD and AFD, among attributes can be discovered within the tuples set. Attributes that are related with a droppable attribute either based on FD or AFD are therefore the proxy candidates for the droppable attribute. One algorithm to discover FDs and AFDs called TANE, as proposed by Huhtale et al. [HKPT99], is useful for us to find proxy candidates. Given the proxy candidates found using TANE, we assess their space saving and accuracy criteria, and return the results to system administrators to decide on which of the droppable attributes can be dropped from the population table schemas.
By answering the questions posted in this section, we learnt that the default behaviour of a proxy assessment is a three steps process:

1. For each population table, where the selected droppable attributes are present, we search for the proxy candidates for each droppable attribute (using TANE) against the tuples set of the individual CS.

2. Then, for each proxy candidate, we assess its space saving and accuracy criteria.

3. Finally, we return the assessment result which consists of space saving and accuracy scores of the proxy candidates, for all selected droppable attributes in all population tables, for all CSs. For example, the following are some of the instances of a proxy assessment result, for a proxy candidate C:

\[
\langle \text{source 1}, px, B, C, 90\%, 70\% \rangle,
\langle \text{source 2}, px, B, C, 40\%, 80\% \rangle,
\langle \text{source 3}, px, B, C, 70\%, 80\% \rangle, ..., 
\]

where the first column is the name of the CS; the second column is the name of the population table; the third column is the droppable attribute; the fifth and sixth columns are space saving and accuracy scores respectively.

In the next section, we will present the details of this three-step proxy assessment (i.e. the inputs required and the output produced) in the form of an algorithm.

### 5.4.1 Proxy Assessment Algorithm

To implement the proxy assessment algorithm, the selection of the droppable attributes to examine and the type of proxy map to use must be provided by the PBC system providers (e.g., system administrators). Note that the definitions of basic variables used in this algorithm are in basicConfig (see Table 5.3), the definitions of all proxy-related variables are in proxyConfig (see Table 5.5) and local variable definitions are presented in Table 5.6. As in the query transformation algorithm, the convention is, all variables and functions are in italics; set variable are in upper case and non-set variables are in lower case; types are in non-italic font; actual instances are in typewriter font.

The proxy assessment algorithm as shown in Algorithm 2 takes several inputs, which are the basicConfig (see Table 5.3), the proxyConfig (see Table 5.5), the set of population tables that consists of the selected droppable attributes, the set
Algorithm 2: proxy assessment

**Input:** basicConfig, proxyConfig, UPD, DA, G3, proxyMapType, 
aggTypeProxyError

**Output:** ALLPROXYRESULTS

```
begin
  / / All local variable definitions are in Table 5.6.
  foreach ⟨(p, ⟨I, source, A⟩)⟩ ∈ UPD do
    CSNAME = getSource(p);
    foreach csName ∈ CSNAME do
      r = getDataSetForTANE(p, csName);
      ALLTANERESULTS = {};
      ALLTANERESULTS =
      ALLTANERESULTS ∪ {csName, p, TANE(r, G3)};
      tempTANE =
      storeTempTANEResults(ALLTANERESULTS);
      foreach ⟨da, p⟩ ∈ DA do
        PROXYCANDIDATE =
        getProxyCandidate(csName, p, da, tempTANE);
        foreach proxyCandidate ∈ PROXYCANDIDATE do
          workProxyMap =
          getProxyMap(da, proxyCandidate, p, csName);
          spaceSaving =
          computeSpaceSaving(basicConfig, proxyConfig, ⟨da, p⟩,
          proxyMapType, workProxyMap, proxyCandidate);
          accuracy =
          computeAccuracy(basicConfig, proxyConfig, ⟨da, p⟩,
          workProxyMap, proxyCandidate, aggTypeProxyError);
          ALLPROXYRESULTS = {};
          ALLPROXYRESULTS =
          ALLPROXYRESULTS ∪
          {csName, p, da, proxyCandidate, spaceSaving, accuracy};
        end
      end
    end
  end
  return ALLPROXYRESULTS
end
```
Table 5.6: Local Variable Definitions for the Proxy Assessment Algorithm

<table>
<thead>
<tr>
<th>The variables and their types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>accuracy</strong>: FLOAT</td>
<td>Accuracy score as a percentage.</td>
</tr>
<tr>
<td><strong>aggTypeProxyError</strong>: STRING</td>
<td>The aggregation type for proxy error. Three valid values are accepted, <em>avg</em>, <em>max</em> and <em>min</em>.</td>
</tr>
<tr>
<td><strong>CSNAME</strong>: P(NAME)</td>
<td>The set of CSs.</td>
</tr>
<tr>
<td><strong>droppableAttrInst</strong>: INTEGER</td>
<td>The number of instances in the droppable attribute in the proxy map.</td>
</tr>
<tr>
<td><strong>droppableAttrSize</strong>: INTEGER</td>
<td>The number of instances in the droppable attribute in the population table.</td>
</tr>
<tr>
<td><strong>G3</strong>: FLOAT</td>
<td>The set of g3 scores that we want to appear in the TANE result.</td>
</tr>
<tr>
<td><strong>PARTIALPROXY</strong>: P(NAME)</td>
<td>The set of partial proxy attributes.</td>
</tr>
<tr>
<td><strong>P(QUALNAME × TANERESULT)</strong></td>
<td>The set of proxy candidates.</td>
</tr>
<tr>
<td><strong>PROXYERROR</strong>: INTEGER</td>
<td>The set of proxy errors.</td>
</tr>
<tr>
<td><strong>proxyInst</strong>: INTEGER</td>
<td>The number of instances in a partial proxy attributes or in a proxy attribute in the proxy map.</td>
</tr>
<tr>
<td><strong>proxyMapSize</strong>: INTEGER</td>
<td>The number of instances in the proxy map.</td>
</tr>
<tr>
<td><strong>proxyMapType</strong>: P(STRING)</td>
<td>The type of proxy map. Two valid values are accepted, <em>pure</em> and <em>multi</em>.</td>
</tr>
<tr>
<td><strong>QUERY SAMPLE</strong>: P(QUERY)</td>
<td>A set of query samples against the population table.</td>
</tr>
<tr>
<td><strong>r</strong>: P(P(QUALSCHEMA))</td>
<td>The result of <em>getDataSetForTANE()</em>().</td>
</tr>
<tr>
<td><strong>spaceSaving</strong>: FLOAT</td>
<td>Space saving score in percentage.</td>
</tr>
<tr>
<td><strong>sumProxyInst</strong>: INTEGER</td>
<td>The sum of number of instances in the partial proxy attributes or in the proxy attributes.</td>
</tr>
<tr>
<td><strong>tempTANE</strong>: P(NAME × NAME × FLOAT × NAME)</td>
<td>The temporary table that stores all TANE results.</td>
</tr>
<tr>
<td><strong>transformQuery</strong>: P(QUERY)</td>
<td>A transformed sample query.</td>
</tr>
<tr>
<td><strong>V</strong>: P(STRING × INTEGER × FLOAT)</td>
<td>The set of domain values.</td>
</tr>
<tr>
<td><strong>workProxyMap</strong>: P(P(QUALSCHEMA))</td>
<td>The result of <em>getProxyMap()</em>().</td>
</tr>
</tbody>
</table>

of selected droppable attributes, the set of g3 values and the type of the proxy map. The output of this algorithm is the set of space saving and accuracy scores of the proxy candidates for droppable attributes of all CSs.

To search for the proxy candidates for each droppable attribute of an individual CS, the algorithm must first get the set of CS name. The *getSource()* function will issue a query to retrieve CS names from the population table of the droppable attribute. This query is:

```sql
SELECT DISTINCT source FROM p;
```

Then, the algorithm gets the tuple set that will be used by TANE algorithm. The *getDataSetForTANE()* function retrieves the tuple set by issuing the following query:

```sql
SELECT * FROM p WHERE source = 'csName';
```

The TANE algorithm is used to get all attributes that are FD or AFD related from the population table. TANE takes the result of the above query (r) and the set of g3 values (G3) as the inputs. The result of TANE, which is of TANERESULT
type in proxyConfig (see Table 5.5) consists of pairs of attributes (that are FD or AFD related) and their g3 values. The results of TANE for an individual CS are accumulated into \textit{ALLTANERESULTS} in every iteration of the loop in the algorithm. The \textit{storeTempTANEResults}() function stores the TANE results in a temporary table, \textit{tempTANE} (i.e. in the temporary storage as shown in Figure 5.2, page 145). The schema of \textit{tempTANE} is:

\begin{verbatim}
⟨csName, popTableName, g3Val, droppableAttrName, proxyCandidateName⟩.
\end{verbatim}

The next step is to get the proxy candidates for each droppable attribute from the sets of all TANE results, which is performed by the \textit{getProxyCandidate()} function. This function takes the name of the CS, the name of the population table, the name of the droppable attribute, and the results of TANE as the inputs. The proxy candidates are retrieved by querying the temporary table where the TANE results are stored (\textit{tempTANE}). The query is:

\begin{verbatim}
SELECT proxyCandidateName FROM tempTANE
WHERE csName= 'csName' AND popTableName= 'p' AND droppableAttrName= 'da';
\end{verbatim}

Then for each proxy candidate, we compute its space saving and accuracy through \textit{computeSpaceSaving()} and \textit{computeAccuracy} functions. Both functions require the working proxy map as the input. The \textit{getProxyMap}() function issues the following query to retrieve the working proxy map:

\begin{verbatim}
SELECT DISTINCT da, proxyCandidate FROM p WHERE source = 'csName';
\end{verbatim}

The results of space saving and accuracy of proxy candidates of all droppable attributes for an individual CS are accumulated into \textit{ALLPROXYRESULTS} in every iteration of the loop in the algorithm. Finally, the algorithm returns \textit{ALLPROXYRESULTS} as the output.

In the next section, we will describe the algorithm for \textit{computeSpaceSaving()} function.

\textbf{5.4.1.1 Space Saving Computation Algorithm}

The \textit{computeSpaceSaving()} function implements the space saving algorithm as shown in Algorithm 3. We follow the steps presented in Section 5.2.1, page 136 to compute space saving. To recall, these steps are:
1. Get the working proxy map by issuing a query against the population table where the droppable attribute and the proxy is present.

2. Count the number of instances for:
   - the droppable attribute column in the working proxy map. An aggregate query (i.e. using the `COUNT` operator) is issued against the working proxy map. For proxy maps in a multi-value table structure, we need to add the `DISTINCT` operator within the SQL statement used to count the number of instances in the droppable attribute.
   - the proxy attribute column(s) in the working proxy map. As for the previous step, an aggregate query is issued against the working proxy map to count the number of instances in the proxy column(s). For composite proxies, we need to count the number of instances in each column of the partial proxy and get the sum of the number of instances of each partial proxy column.

3. Compute the proxy map size by adding together the number of instances in the droppable attribute column and the number of instances in the proxy column(s) that were previously counted from the working proxy map.

4. Compute the droppable attribute size by counting the number of instances in the droppable attribute column in the population table.

5. Compute the space saving by getting the difference between the droppable attribute size and the proxy map size.

Note that, some steps are not dependent on each other (e.g., step 3 and step 4) and therefore they can be performed in any order. Next, we will describe the details of these steps.

The space saving algorithm as shown in Algorithm 3 takes several inputs, which are the configuration information (basicConfig and proxyConfig), the droppable attribute and its population table name, the working proxy map, the proxy candidate and the type of table structure to store the proxy map. The type of table structure to use must be provided by the PBC system administrator who may make the selection based on knowledge of whether the values in the droppable attribute domain map with multiple values of the proxy. Recall that in Section 5.2.1 (page 136) we suggested checking on the characteristic of the droppable attribute
Algorithm 3: space saving computation

Input: basicConfig, proxyConfig,
       ⟨da, p⟩, proxyMapType, workProxyMap, proxyCandidate

Output: spaceSaving

begin
   / / All local variable definitions are in Table 5.4.
   if proxyMapType = 'pure' then
      droppableAttrInst =
      getDroppableAttrInstPure(da, workProxyMap, csName);
   else
      droppableAttrInst =
      getDroppableAttrInstMulti(da, workProxyMap, csName);
   end
   PARTIALPROXY =
   getProxyCandidateComposition(proxyCandidate);
   foreach partialProxy ∈ PARTIALPROXY do
      proxyInst = getProxyInst(partialProxy, workProxyMap);
      sumProxyInst = 0;
      sumProxyInst = sumProxyInst + proxyInst;
   end
   proxyMapSize = droppableAttrInst + sumProxyInst;
   droppableAttrSize = getDroppableAttrSize(⟨da, p⟩);
   spaceSaving =
      (droppableAttrSize − proxyMapSize)/droppableAttrSize × 100
   return spaceSaving
end
domain’s values by issuing a query such presented in the Equation 5.6 (page 143). The multi-valued tables structure should be selected to store the proxy map if the result of the query shows that the values in the droppable attribute domain map with multiple values of the proxy. The space saving algorithm returns the amount of space saving as a percentage.

The algorithm will first check the type of table structure chosen for the proxy map. For brevity, we assume that the system administrator provides a valid value for the type of proxy map table, i.e. pure is for a pure relational table and multi is for a multi-valued table. Therefore, the algorithm does not handle an exception caused by invalid values.

The number of instances in the droppable attribute is counted for the pure relational proxy map using the getDroppableAttrInstPure() function that issues the following query:

\[
\text{SELECT COUNT(da) FROM workProxyMap;}
\]

While the number of instances in the droppable attribute is counted for the multi-valued table proxy map through getDroppableAttrInstMulti() function that issues the following query:

\[
\text{SELECT COUNT DISTINCT(da) FROM workProxyMap;}
\]

Both functions that counts the number of instances in the droppable attribute column take the name of the droppable column, the working proxy map and the name of the CS as the inputs.

Then the algorithm gets the partial attributes of the proxy candidate (if any) using the getProxyCandidateComposition() function that takes the name of the proxy candidate as its input. If the proxy candidate is not a composite type, then \text{PARTIALPROXY} (the result of this function) is a singleton set. Next, the algorithm counts the number of instances in each partial attribute using the getProxyInst() function that takes the partial attribute and the working proxy map as inputs and issues the following query:

\[
\text{SELECT COUNT (partialProxy) FROM workProxyMap;}
\]
CHAPTER 5. SPACE-ACCURACY TRADE-OFFS

For proxy candidates that are of composite type, the algorithm computes the sum of the number of instances of their partial attributes. Following this step, the algorithm performs step 3 to get the size of the proxy map. The size of the droppable attribute is computed by the `getDroppableAttrSize()` function that issues the following query:

\[
\text{SELECT COUNT(da) FROM } p;
\]

Finally, space saving is computed by getting the difference between the size of the droppable attribute and the size of the proxy map, as stated in step 5. The space saving algorithm returns space saving (in percentage) as the result.

Before we can describe the algorithm for the `computeaccuracy()` function, we will first discuss the indicator used to represent accuracy of proxy candidates, which will be presented in the next section.

5.4.2 Accuracy Indicator for Proxies

In this section, we set out to answer the question of how can we assess the accuracy of a proxy candidate i.e., how do we compute the error made by proxies?

Ideally, accurate answer for PBC measurement requests can be provided even though proxies are used as the substitutes for the dropped attribute. In particular, the values of proxies are used to substitute the values of the dropped attribute in the conditions of queries for the reference sub-populations. This ideal case can be achieved if the proxies do not cause the reference sub-population to be incomplete (i.e. due to missing individuals ) or inaccurate (i.e. due to the presence of spurious individuals). We call proxies that cause the reference sub-population to be incomplete or inaccurate defective proxies.

We use the same definition of PBC measurement error defined in Chapter 4 (as shown in Equation (4.1) in page 103) in order to describe the PBC measurement error caused by the defective proxies. For convenience, we restate the definition of PBC measurement error as:

\[
m.e(D, RP, RP') = \frac{|D \cap RP|}{|RP|} - \frac{|D \cap RP'|}{|RP'|},
\]

where \( m.e \) is the PBC measurement error, \( D \) is the data set under measure, \( RP \)
is the reference population, and \( RP' \) is the reference population that is inaccurate or incomplete. In particular, in the context of proxies assessment, \( RP \) is the reference sub-populations, while \( RP' \) is the reference sub-population that is inaccurate or incomplete.

Defective proxies may cause: some individuals to be missing from the reference sub-population or some spurious individuals to be added into the reference sub-population. We regard the number of spurious individuals or the number of missing individuals in \( RP' \) as proxy error. We define proxy error as the percentage of the number of spurious individuals (or the number of missing individuals) in \( RP' \) as:

\[
p.e = \frac{|RP| - |RP'|}{|RP'|} \times 100,
\]

where:

- \( p.e \) is the proxy error,
- \( RP \) is the reference sub-population retrieved by issuing query \( Q = \Pi_{\text{key}(p)}(\Phi_{\text{COND}} p) \), where \( p \) is the population table, \( \text{key}(p) \) is the function to retrieve the identifiers in \( p \) and \( \text{COND} \) is a conjunction of conditions on \( p \),
- \( RP' \) is the reference sub-population that is inaccurate (or incomplete), retrieved by issuing query \( Q' = \Pi_{\text{key}(p)}(\Phi_{\text{COND}' p}) \), where \( \text{COND}' \) is a transformed conjunction of conditions on \( p \).

We introduced the transformed query earlier in Section 5.3.1.1 (page 147) as the result of the query transformation algorithm, within the context of proxies in use. In the algorithm, a proxy is used to substitute the dropped attribute in the transformed query’s condition. Within the context of proxies under consideration, the proxy candidate is used to substitute the droppable attribute in the transformed query’s condition, where the transformed query will be assessed for the presence of proxy error. Therefore, within the context of proxies under consideration, we need the transformed queries that are purposely constructed to answer the question: how much proxy error will occur if the proxy is used to substitute all values in the droppable attribute domain? To answer this question we need the actual query that is used to retrieve the reference sub-population. Ideally, we want to assess all actual transformed queries, however, it is hard to know exactly what will be asked (i.e. the actual conditions used in the queries).
by the person requesting the PBC measurement in advance. Therefore, we need some query samples to compute the proxy errors. One form of query sample uses equality operators (e.g., =, >, <) in specifying the conditions. Other operators like logical operators might be used in the query conditions such as \textit{AND}, \textit{OR} or \textit{NOT}; the queries might use a combination of different types of operator to specify the conditions.

Regardless of which form of query is used, to maximise coverage of the reference sub-populations queried based on the droppable attribute values, we should include as many as possible of the values from the droppable attribute domain within the query samples (i.e., in the query condition).

For example, consider a schema of a population table $pt$ is $\langle I, A, B, source \rangle$, and the domain values for the droppable attribute $B$ are $\{a, b, c\}$, the following are the possible forms of query samples constructed to compute the proxy errors:

- $Q_1 = \Pi_{key(pt)}(σ_{COND_1} pt)$, where $COND_1 = (B = 'a')$,
- $Q_2 = \Pi_{key(pt)}(σ_{COND_2} pt)$, where $COND_2 = (B = 'a' \ AND \ B = 'b')$,
- $Q_3 = \Pi_{key(pt)}(σ_{COND_3} pt)$, where $COND_3 = (B = 'a' \ OR \ B = 'c')$.

Note that the $key()$ function retrieves the identifiers of $pt$ from $I$.

These query samples must then be transformed. Consider that the proxy map is $BA$, with schema $\langle B, A \rangle$. Suppose that the following are the transformed query samples $Q_1, Q_2, Q_3$, where $A$ is the proxy candidate:

- $Q'_1 = \Pi_{key(pt)}(σ_{COND'_1} pt)$, where $COND'_1 = (A ∈ S_1)$, where, $S_1 = Π_B(σ_{COND_1} BA)$,
- $Q'_2 = \Pi_{key(pt)}(σ_{COND'_2} pt)$, where $COND'_2 = (A ∈ S_2)$, where, $S_2 = Π_B(σ_{COND_2} BA)$,
- $Q'_3 = \Pi_{key(pt)}(σ_{COND'_3} pt)$, where $COND'_3 = (A ∈ S_3)$, where, $S_3 = Π_B(σ_{COND_3} BA)$.

Proxy errors ($p.e_1, p.e_2$ and $p.e_3$) are then computed using the formula shown in Equation (5.9), where:

- $p.e_1 = \frac{|Q_1| - |Q'_1|}{|Q'_1|} × 100$,
- $p.e_2 = \frac{|Q_2| - |Q'_2|}{|Q'_2|} × 100$,
• \( p.e_3 = \frac{|Q_3| - |Q'_3|}{|Q'_3|} \times 100. \)

Note that a negative proxy error will be yielded if the result of the query samples is smaller than the result of the transformed query samples. This signifies the presence of spurious individuals in the reference sub-population. In contrast, a positive proxy error indicates some individuals are missing from the reference sub-population. For example, if the accuracy indicator of proxy candidate A is 40% it means 40% of the individuals in the reference sub-population are missing, but if accuracy indicator of proxy candidate A is -40% it means 40% of the reference sub-population are spurious individuals. Nevertheless the amount of proxy error made by a proxy candidate must be treated as an absolute value that signifies its distance from the proxy error equals to zero. Therefore, we regard the positive and negative sign as the indicator of whether the error is caused by spurious individuals or not. For example, in their absolute values, 40% proxy error and -40% proxy error indicate the same amount of proxy error.

As more than one proxy error is yielded, we need to aggregate the results in order to represent the proxy error made by a proxy candidate. The question that arises is, how to aggregate the proxy errors yielded for a proxy candidate? A common way is to take an average of multiple results, which in our case, means taking an average of the absolute values of the proxy errors. Another way is to aggregate by getting the maximum of the absolute values of the proxy error made by the proxy candidate. An optimistic approach takes the minimum absolute values of the proxy error made by the proxy candidate as an aggregation type. Since there are different types of aggregation available, the PBC measurement provider must specify their preference during the proxy assessment.

The result of the aggregation called accuracy is used as the indicator of the proxy candidate’s accuracy:

\[
\text{accuracy} = \text{aggregate}(p.e_1, p.e_2, ... p.e_n), \tag{5.10}
\]

where \( \text{aggregate()} \) is the function that aggregates the proxy errors and \( n \) is the number of proxy errors computed for the proxy candidate. The type of aggregation selected determines how the aggregation is made (i.e., average, maximum or minimum).

In this section, we answered the question of how proxy candidates are assessed in terms of accuracy. We proposed measuring errors made by the proxy candidates
where the errors are defined in terms of the number of individuals that are missing from (or the number of spurious individuals that are present in) the reference sub-population. We defined the accuracy indicator for a proxy candidate as the percentage of the aggregate proxy errors made by the proxy candidate. A good proxy candidate therefore has a low accuracy indicator. One of three types of aggregation can be selected by the PBC measurement provider during proxy assessment namely average, maximum and minimum.

In the next section we will present an analysis of the implications of proxy errors on PBC measurement accuracy.

### 5.4.2.1 Defective Proxy Analysis

We regard the proxies that cause the reference sub-populations to be incomplete or inaccurate as defective proxies. In this section, we aim to answer the question of how the defective proxies affect the reference sub-populations i.e., will they cause reference sub-populations to be incomplete or inaccurate? By knowing how defective proxies affect the reference sub-population, we can determine how PBC measurement accuracy may be affected by defective proxies. Nevertheless, before we can examine the implications of defective proxies on the reference sub-populations, we set out to answer the question of which types of proxy have the possibility of being defective.

Recall in Section 5.2 (page 132) that two types of proxies have been identified based on the types of relationship that can be observed in the population table. The first type of proxy called FD-based proxy is related to the droppable attribute based on the FD. For example, applying the definition of FD by Huhtala et al. [HKPT99], within an artificial population table $pt$ that has schema $PT = \langle I, A, B \rangle$, given that a proxy attribute $A$ and a droppable attribute $B$, we say that $A \rightarrow B$ if all pairs of tuples $t, u \in pt$ we have:


Consider further the instances of $A$ and $B$ in a small population table $pt$ as shown in Table 5.7(a). Note that, $t1$ to $t10$ are the ordered tuples in $p$. Suppose that the proxy map that stores the mappings between the values of $B$ and the values of $A$, $BA$ has schema $\langle B, A \rangle$ and the proxy map is stored as a multi-valued table, as shown in Table 5.7(b).

We say $A$ is not a defective proxy if all of its values substitute for exactly one droppable attribute value ($B$) in the mappings, which is the case presented by this
FD-based proxy example. As \( A \) is not a defective proxy we expect that no proxy error will occur. To check if this is the case, suppose that the query to retrieve a reference sub-population is:

\[
Q = \Pi_{\text{key}(pt)}(\sigma_{\text{COND} pt}), \text{ where } \text{COND} = (B = x).
\]

The result of \( Q \) is the tuple set: \( \{t1, t2, t3, t4, t5, t6, t9, t10\} \), and \( |Q| = 8 \). The transformed query for \( Q \) is:

\[
Q' = \Pi_{\text{key}(pt)}(\sigma_{\text{COND}' pt}), \text{ where } \text{COND}' = (A \in S), \text{ where, } S = \{a, b, d\}.
\]

\( S \) is the result of query, \( \Pi_A(\sigma_{\text{COND} BA}) \). The result of \( Q' \) is the tuple set: \( \{t1, t2, t3, t4, t5, t6, t9, t10\} \), and \( |Q'| = 8 \). Proxy error, \( p.e = \frac{|Q| - |Q'|}{|Q'|} \times 100 = 0 \).

If we use the value of \( A \) to substitute another value of \( B \) which is \( y \) in another transformed query, again we will get no proxy error.

As FD-based proxies can substitute the droppable attributes in full (i.e., each proxy value determines exactly one value of the droppable attribute) they possess the characteristic of non-defective proxies. A case study for FD-based proxies will be presented in Section 5.6.1 in order to support that FD-based proxies are non-defective within the context of the microbial domain.

Table 5.7: An Example of an FD-based Proxy

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>t2</td>
<td>2</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>t3</td>
<td>3</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>t4</td>
<td>4</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>t5</td>
<td>5</td>
<td>b</td>
<td>x</td>
</tr>
<tr>
<td>t6</td>
<td>6</td>
<td>b</td>
<td>x</td>
</tr>
<tr>
<td>t7</td>
<td>7</td>
<td>c</td>
<td>y</td>
</tr>
<tr>
<td>t8</td>
<td>8</td>
<td>c</td>
<td>y</td>
</tr>
<tr>
<td>t9</td>
<td>9</td>
<td>d</td>
<td>x</td>
</tr>
<tr>
<td>t10</td>
<td>10</td>
<td>d</td>
<td>x</td>
</tr>
</tbody>
</table>

\( (a) \) Population Table \( pt \) \hspace{2cm} \( (b) \) Proxy Map \( BA \)
The second type of proxy called AFD-based proxy is related to the droppable attribute based on the AFD. For example, within an artificial population table \( pt \) that has schema \( PT = \langle I, A, C \rangle \), given that a proxy attribute \( A \) and a droppable attribute \( C \), we say that \( A \bowtie C \) if not all pairs of tuples \( t, u \in pt \) we have:

\[
t[A] = u[A] \Rightarrow t[C] = u[C]
\]

where \( A, C \subseteq PT \).

Consider the instances of \( A \) and \( C \) in a small population table \( pt \) as shown in Table 5.8(a). Suppose that the proxy map that stores the mappings between the values of \( C \) and the values of \( A \), \( CA \) has schema \( \langle C, A \rangle \) and the proxy map is stored as a multi-valued table, as shown in Table 5.7(b).

We say \( A \) is not a defective proxy if all of its values substitute for exactly one droppable attribute values (\( C \)) in the mappings, however, this is not the case presented in this AFD-based proxy example. For example, \( a \) in the proxy map (see Table 5.8(b)) is used to substitute \( w \) and \( z \). Therefore, we expect that there will be some proxy errors made by \( A \). To check if this is the case, suppose that the query to retrieve a reference sub-population is:

\[
Q = \Pi_{key(pt)}(\sigma_{COND} pt), \text{ where } COND = (C = w).
\]

The result of \( Q \) is the tuple set: \( \{t1, t2, t5, t7, t8\} \), and \( |Q| = 5 \). The transformed query for \( Q \) is:
$Q' = \Pi_{\text{key}(\text{pt})}(\sigma_{\text{COND}' \text{pt}})$, where $\text{COND}' = (A \in S)$ and, $S = \{a, b, c\}$.

$S$ is the result of query, $\Pi_A(\sigma_{\text{COND}CA})$. The result of $Q'$ is the tuple set: \{t1, t2, t3, t4, t5, t6, t8, t9\}, and $|Q'| = 9$. The proxy error is:

$$p.e = \frac{|Q| - |Q'|}{|Q'|} \times 100$$

$$p.e = \frac{5 - 9}{9} \times 100 = -44\%$$

This shows that, if we use the value of $A$ that substitutes for more than one value of $B$, then a proxy error will occur. The negative proxy error indicates four spurious tuples (i.e., individuals) have been added into the reference sub-population. The reference sub-population queried using the transformed query ($Q'$) has more individuals relative to the reference sub-population queried using the actual query ($Q$). However, if $d$ (of $A$) is used to substitute for $y$ in the transformed query, the correct tuple will be returned ($t10$) as $d$ only determines exactly one droppable attribute value (which is $y$). Therefore in the best case, no proxy error will be made by $A$.

In the worst case, there is only a single tuple yielded as the result of $Q$, while $Q'$ returns the whole set of tuples from the population table $\text{pt}$ as the result. For example, as shown in the proxy map in Table 5.9(b), $w$ is substituted by $a$ which is also used to substitute two other values of $C$ ($y$ and $z$). Only a single tuple ($t1$) in the population table as shown in Table 5.9(a) will be returned as the result of:

$$Q = \Pi_{\text{key}(\text{pt})}(\sigma_{\text{COND} \text{pt}}), \text{ where } \text{COND} = (C = w).$$

The result of $Q'$ is the entire set of tuples in $\text{pt}$, as the value of $A$ that substitutes $w$ also substitutes other values of $C$ (that span all tuples in $\text{pt}$ except $t1$). The proxy error in the worst case for this example is:

$$p.e = \frac{|Q| - |Q'|}{|Q'|} \times 100$$

$$p.e = \frac{1 - 10}{10} \times 100 = -90\%$$
CHAPTER 5. SPACE-ACCURACY TRADE-OFFS

Based on this observation, we define proxy error in the worst case scenario for AFD-based proxy candidates as:

\[
\frac{(1 - countTuple(p))}{countTuple(p)} \times 100, \quad (5.11)
\]

where \(countTuple()\) is a function that retrieves the number of tuples in the population table \(p\).

Table 5.9: An Example of The Worst Accuracy Case for an AFD-based Proxy Candidate

(a) Population Table \(pt\)  (b) Proxy Map \(CA\)

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>I 1 a w</td>
</tr>
<tr>
<td>t2</td>
<td>2 a z</td>
</tr>
<tr>
<td>t3</td>
<td>3 a y</td>
</tr>
<tr>
<td>t4</td>
<td>4 a z</td>
</tr>
<tr>
<td>t5</td>
<td>5 a z</td>
</tr>
<tr>
<td>t6</td>
<td>6 a z</td>
</tr>
<tr>
<td>t7</td>
<td>7 a y</td>
</tr>
<tr>
<td>t8</td>
<td>8 a y</td>
</tr>
<tr>
<td>t9</td>
<td>9 a z</td>
</tr>
<tr>
<td>t10</td>
<td>10 a y</td>
</tr>
</tbody>
</table>

As AFD-based proxies cannot substitute the droppable attributes in full (i.e., not all proxy values can determine exactly one droppable attribute value), these proxies are therefore possibly defective proxies. An AFD-based proxy also causes spurious individuals to be added into the reference sub-population due to the characteristic that some of its values determine multiple values of the droppable attribute. We postulate that, due to this characteristic, it is not possible for the defective proxies to cause one or more individuals to be missing from the reference sub-population. Therefore, we say that the effect of using defective AFD-based proxies is that they make reference sub-populations inaccurate, rather than incomplete. Thus, the implications of using inaccurate reference populations that were presented in Chapter 4 (see Section 4.3.4 in page 103) can be used to represent the implications of using defective proxies on PBC measurement accuracy.
To support the observation made on the AFD-based proxies in this section, we will present a case study of AFD-based proxies in Section 5.6.2 involving real data sets.

In the next section, we analyse whether AFD errors (represented by g3 values) are correlated with the proxy errors.

5.4.2.2 AFD Error Analysis

Recall that we used an AFD discovery algorithm called TANE (proposed by Huhtale \textit{et al.} [HKPT99]) to discover proxy candidates in the proxy assessment algorithm (see Algorithm 2). Each AFD-based proxy that is discovered using TANE has a g3 score that is used to indicate an AFD error. The question that arises is, can we use g3 as the accuracy indicator (i.e., for the proxy errors made) for those AFD-based proxies? One advantage of answering this question is that we can use g3 as the alternative for query sampling if we find that g3 can be used as proxy’s accuracy indicator. g3 might be particularly useful in the case where the domain of values for the droppable attribute is large which would consequently requires a large amount of query samples (and the transformed query samples) to be constructed. However, even if it is found that g3 is not useful in determining proxy error, we aim to gain an understanding why this is the case.

To answer this question, we need to understand how g3 is computed and compare it with how we compute proxy error. Kivinen and Mannila who proposed the g3 measure defined the result of the measure, the g3 score for an AFD $X \rightarrow A$ on a relation $r$ as the minimum number of tuples that need to be removed from $r$ in order for it to comply with the FD $X \rightarrow A$ [KM92]. Note that the TANE algorithm by Huhtale \textit{et al.} [HKPT99] which is an optimised version of the g3 measure that we use to discover proxies also adopts the same definition of the g3 score as stated by Kivinen and Mannila.

Based on the description of the g3 measure stated in the literature [HKPT99, KM92], we identify the steps taken to compute the g3 score by reusing the artificial population table $pt$ (as shown in Table 5.8(a)) presented in the previous section. For convenience, we repeat the table and add information about the query samples and the transformed queries submitted against the table, as shown in Table 5.10. For brevity, the identifier attribute ($I$) is omitted from the table. Recall that the schema of $pt$ in the AFD-based proxy example is $PT=\langle I, A, C \rangle$. Suppose that we wish to measure the g3 score for $A \rightarrow C$. The steps are as
follows:

1. For every pair of values of the attributes under measure A and C, the tuples that violate FD are identified. FD violation is indicated by the presence of multiple values of C for each value of A. Therefore all tuples that consists of the value of A that have more than one value of C are regarded as the violating tuples. For example, two values of A that have multiple values of C are a and b, e.g., a determines w and z. In total, 7 tuples in table pt as shown in Table 5.10 violate the FD. These tuples are t1, t2, t3, t4, t5, t6 and t9.

2. Then, the number of tuples that violate FD is counted for each value of A. For example, for value a, 4 tuples violate the FD; for value b, 3 tuples violate the FD.

3. Next, for each value of A, the minimum number of tuples that violate FD is identified. For example, the minimum number of tuples that violate FD in table pt is 3 (for A=b).

4. The g3 score is computed by taking the ratio of the minimum number of tuples against the number of tuples in table. For example, the g3 score for $A \rightarrow C$ as a percentage is:

$$g3 = \frac{3}{10} \times 100$$

$$= 30\%$$

This g3 score means, at least 30% of the tuples in table pt must be removed to make $A \rightarrow C$.

Based on the description of how the g3 score is computed, we found that, both g3 score and proxy error are computed based on the presence of spurious tuples (called tuples that violate FD in the g3 measure). In the g3 measure, the search for these spurious tuples is made by comparing the values of the attribute pairs. Similarly, spurious tuples are discovered by comparing the values (i.e., the number of tuples) of the pair of a sample query and a transformed query. Nevertheless, we must note the differences between a g3 score and a proxy error as follows:
The g3 measure examines the values of the determinant attribute (i.e., proxy) and counts the number of tuples that violate FD by those values. However, proxy error is computed by examining the values of the droppable attribute used in the query conditions to get the reference sub-population. Because of the ‘opposite’ way of how these scores (i.e., g3 and proxy error) are computed, we cannot get the same result for the total number of tuples that violate FD (which is 7) and the total number of spurious tuples (which is 9).

The g3 measure takes the minimum number of tuples that violate FD to compute the g3 score. The percentage of these minimum number of tuples is computed against the total number of tuples in the table. In contrast, proxy error is computed by taking the percentage of the number of spurious tuples within the reference sub-population (i.e., in the transformed query $Q'$).

In the light of the differences between a g3 score and a proxy error, we say that g3 cannot be a useful accuracy indicator for proxy candidates under assessment.

To support our observation of the lack of correlation between g3 and proxy error, we conducted a small case study using real data sets taken from the microbial domain.

We used microbial data sets from a bacterial database called Omnimere, which is provided by CMR (which is the same source for data sets used in the case study.

![Table 5.10: An Example of the Tuples that Violate FD](image-url)
in Section 5.2.1, page 136). In particular, we use a table called evidence from CMR which consists of 292,843 tuples, to construct a population table called eu that has schema: 
\[
\text{EU} = \langle \text{id}, \text{accession}, \text{locus}, \text{assignby}, \text{method}, \text{rel}\_\text{end3}, \text{rel}\_\text{end5}, \text{end3}, \text{end5}, \text{ev}\_\text{type} \rangle.
\]
To avoid constructing large numbers of sample queries, we selected droppable attributes (shown in bold) that have smaller domain value size (i.e. less than 100 values) relative to other attributes in the EU.

We performed the following steps to compute the accuracy indicator (i.e., proxy error) for each proxy candidate of the selected droppable attributes:

1. Build the proxy map.
2. Get all domain values for each droppable attribute.
3. Construct the query samples based on the domain values. All domain values must be used to maximise the query samples coverage.
4. Transform the query samples based on the values of the proxy candidate queried from the proxy map.
5. Get the results of the query samples and the results of the transformed queries.
6. Compute the proxy error for each pair of the sample query’s result and the transformed query’s result (see Equation (5.9) in page 163 for proxy error’s computation formula).
7. Compute the accuracy indicator based on the set of proxy errors computed in the previous step (see Equation (5.10) in page 165). Three aggregation types were applied namely average, maximum and minimum.

We use TANE to discover AFD-based proxies, where the results of TANE yielded is shown in Table 5.11. TANE takes the eu table and the set of g3 scores (as percentages) that we want to appear in the TANE result as the inputs. In this case study, the g3 scores used as the inputs were: \{1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}.

Table 5.11(a), Table 5.11(b) and Table 5.11(c) shows the proxy candidates for the droppable attributes assignby, method and ev_type respectively in increasing order of their g3 scores. In total, 20 proxy candidates were assessed.

To observe the correlation between proxy errors and g3 scores, we combine the results of all 20 proxy candidates for all droppable attributes. We used an
Table 5.11: The TANE Results

(a) Proxy Candidates for assignby

<table>
<thead>
<tr>
<th>Proxy Candidates</th>
<th>g3 Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>accession,end5</td>
<td>19</td>
</tr>
<tr>
<td>locus,ev_type</td>
<td>29</td>
</tr>
<tr>
<td>end5</td>
<td>40</td>
</tr>
<tr>
<td>end3</td>
<td>42</td>
</tr>
<tr>
<td>locus</td>
<td>51</td>
</tr>
</tbody>
</table>

(b) Proxy Candidates for method

<table>
<thead>
<tr>
<th>Proxy Candidates</th>
<th>g3 Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>accession</td>
<td>5</td>
</tr>
<tr>
<td>ev_type</td>
<td>8</td>
</tr>
<tr>
<td>rel_end3</td>
<td>24</td>
</tr>
<tr>
<td>end5</td>
<td>40</td>
</tr>
<tr>
<td>locus</td>
<td>51</td>
</tr>
</tbody>
</table>

(c) Proxy Candidates for ev.type

<table>
<thead>
<tr>
<th>Proxy Candidates</th>
<th>g3 Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>7</td>
</tr>
<tr>
<td>rel_end5,rel_end3</td>
<td>8</td>
</tr>
<tr>
<td>rel_end3,assignby</td>
<td>14</td>
</tr>
<tr>
<td>locus,rel_end5</td>
<td>22</td>
</tr>
<tr>
<td>rel_end5</td>
<td>26</td>
</tr>
<tr>
<td>rel_end3</td>
<td>29</td>
</tr>
<tr>
<td>assignby</td>
<td>32</td>
</tr>
<tr>
<td>end5</td>
<td>40</td>
</tr>
<tr>
<td>end3</td>
<td>42</td>
</tr>
<tr>
<td>locus</td>
<td>51</td>
</tr>
</tbody>
</table>
XY chart to present the proxy error and g3 score of each proxy candidate where the correlation between proxy errors and g3 scores were observed. We applied a linear regression in order to determine the strength of the correlation between the proxy errors and g3 scores. In particular, we grouped the results into the aggregation types used to compute the accuracy indicator. The maximum value of the correlation coefficient ($R^2$) is 1, which signifies the strongest correlation. The minimum $R^2$ is 0, which signifies no correlation exists.

Figure 5.3 shows the maximum proxy errors and g3 scores of all proxy candidates for all droppable attributes. An $R^2$ of 0.03 has been yielded in this case. Figure 5.4 presents the average proxy errors and g3 scores of all proxy candidates for all droppable attributes; here, $R^2$ is 0.09. Note that, the result of the minimum proxy errors is not presented as all proxy candidates have a minimum proxy error equal to zero.

$R^2$ for both cases is low and therefore the results in this case study suggest that there is lack of correlation between proxy error and g3 score. To conclude, the findings in this case study support our analysis that g3 score is not useful as a proxy’s accuracy indicator. Query sampling therefore should be used as a mechanism to compute the accuracy indicator of proxy candidates.

In the next section we will present the details of the steps for accuracy computation in the form of an algorithm, where sampling of proxy errors is used as the method to compute the accuracy indicator for proxy candidates.

5.4.2.3 Accuracy Computation Algorithm

The `compute Accuracy()` function in the proxy assessment algorithm (see Algorithm 2 in page 156) implements the accuracy computation algorithm as shown in Algorithm 4. This algorithm performs the following basic steps which are similar to the steps performed in the case study presented in the previous section:

1. Get the domain values of the droppable attribute.

2. Construct the query samples based on the domain values of the droppable attribute.

3. Transform the query samples.

4. Compute the proxy error for each pair of the sample query’s result and and the transformed query’s result.
Figure 5.3: Maximum Proxy Errors and g3 Scores for AFD-based Proxy Candidates of All Droppable Attributes
Figure 5.4: Average Proxy Errors and g3 Scores for AFD-based Proxy Candidates of All Droppable Attributes
5. Compute the accuracy indicator by aggregating the proxy errors based on the aggregation type given.

6. Return the accuracy indicator of the proxy candidate.

**Algorithm 4**: accuracy computation

```
Input: basicConfig, proxyConfig, ⟨da, p⟩, workProxyMap, proxyCandidate, aggTypeProxyError
Output: accuracy

begin
  V = getDomain(⟨da, p⟩);
  QUERYSAMPLE = constructQuerySample(V, ⟨da, p⟩);
  foreach querySample ∈ QUERYSAMPLE do
    transformedQuery = transformQuery(querySample, workProxyMap, proxyCandidate);
    proxyError = |querySample| − |transformedQuery| × 100;
    PROXYERROR = {};
    PROXYERROR = PROXYERROR ∪ {proxyError};
    accuracy = aggTypeProxyError(PROXYERROR);
  end
return accuracy
end
```

The accuracy computation algorithm takes several inputs: basicConfig, proxyConfig, the name of the droppable attribute and the name of its population table, the working proxy map, the name of the proxy candidate and the aggregation type for the proxy error. We assume that valid aggregation types will be provided by the PBC measurement provider, that is one of these values: `avg`, `max` and `min`. The output of this algorithm is the accuracy indicator which is computed based on the given aggregation type.

The algorithm will first retrieve the domain values of the droppable attribute. `getDomain()` is a function that will perform this step by issuing the following query:

```
SELECT DISTINCT da FROM p;
```

This function takes the name of the droppable attribute and the name of the population table as inputs. Then, a set of query samples are constructed (`QUERYSAMPLE`) based on the domain values yielded from the previous step.
The \texttt{constructQuerySample()} function will perform this step where the domain values, the droppable attribute name and the population table are the inputs.

Next, each sample query will be transformed into a transformed query by the \texttt{transformQuery()} function that takes several parameters as its inputs, i.e., the query sample, the working proxy map and the name of the proxy candidate. Proxy error is computed based on the result of the sample query and the result of the transformed query. The proxy errors of the proxy candidate are accumulated as a set variable called \texttt{PROXYERROR} where the variable will be redefined in every iteration of the query sample in the loop. The accuracy indicator is computed by aggregating the accumulated proxy errors in the set. The type of aggregation given (i.e., \texttt{avg}, \texttt{max} or \texttt{min}) is the function that takes the accumulated proxy errors as its input. Finally, the algorithm returns the accuracy indicator as the result.

5.5 Space Saving Analysis

In this section, we set out to answer the question raised at the beginning of this chapter: what determines the amount of space saving that can be offered by proxies? In particular, we aim to analyse the best- and the worst-case scenarios of proxy space saving. Based on the defective proxies analysis in Section 5.4.2.1 (page 166), we found that it is less likely for FD-based proxies to be defective as their values substitute for exactly one value of droppable attributes, and therefore no spurious individuals will be added into the reference sub-population. Attention therefore should be given to FD-based proxies for the accuracy that they offer for PBC measurements.

Nevertheless, FD-based proxies will be less useful if they cannot offer space savings. To learn whether this is the case, we will analyse an artificial population table where an FD-based proxy is present for a particular droppable attribute. We reuse the population table \texttt{pt} previously used in Section 5.4.2.1 (page 166) in the FD-based proxy example. To recall, the schema of \texttt{pt} is \texttt{PT= \langle I, A, B \rangle}, where \texttt{I} is the identifier attribute.

Suppose that \texttt{B} is the droppable attribute, \texttt{I} and \texttt{A} are the FD-based proxy candidates, where \texttt{I → B} and \texttt{A → B}. Recall that we defined space savings in Section 5.2.1 (see Equation (5.2.1) in page 136) as the percentage of the difference between the size of the droppable attribute and the size of the proxy map, against
the size of the droppable attribute. For convenience, we restate the space saving definition as follows:

\[
\text{spaceSaving} = \frac{\text{droppableAttrSize} - \text{proxyMapSize}}{\text{droppableAttrSize}} \times 100,
\]

where the size of the droppable attribute (\(\text{droppableAttrSize}\)) and the size of the proxy map (\(\text{proxyMapSize}\)) are both in terms of the number of instances they contain. How much space saving can be offered by a proxy candidate depends on how much space will be reduced by storing the proxy map.

At its very least, a proxy map consists of only a pair of the droppable attribute and the proxy candidate instances, making its size equals to 2. This is only possible if all the tuple pairs of the droppable attribute and the proxy candidate in the population table are identical and the proxy candidate is not a composite proxy (refer to description of composite proxy in Section 5.2.1). For example, Table 5.12(a) shows this scenario that involves proxy candidate A. As the size of B is 10, and the size of the proxy map BA (as shown in Table 5.12(b)) is 2, space saving in this case is:

\[
= \frac{(10 - 2)}{10} \times 100,
\]

\[
= \frac{8}{10} \times 100,
\]

\[
= 80\%.
\]

Therefore, we say that in the best case scenario, we only need to store the smallest possible proxy map consisting of only 2 instances. Based on this observation, we define space saving in the best case scenario for FD-based proxy candidates as:

\[
\frac{\text{droppableAttrSize} - 2}{\text{droppableAttrSize}} \times 100. \quad (5.12)
\]

A proxy candidate offers space saving as long as the size of the proxy map is smaller than the size of the droppable attribute. If this is not the case, negative space saving will be yielded. Negative space saving means that, instead of gaining some free space, additional space is required to store the proxy map. The question is, what characterises the worst case scenario of the negative space saving?

To examine the worst case scenario of space saving, we divide our observations
Table 5.12: An Example of the Best Space Saving Case for an FD-based Proxy Candidate

(a) Population Table \( pt \)  (b) Proxy Map \( BA \)

\[
\begin{array}{ccc}
\hline
 t1 & 1 & a & x \\
 t2 & 2 & a & x \\
 t3 & 3 & a & x \\
 t4 & 4 & a & x \\
 t5 & 5 & a & x \\
 t6 & 6 & a & x \\
 t7 & 7 & a & x \\
 t8 & 8 & a & x \\
 t9 & 9 & a & x \\
 t10 & 10 & a & x \\
\hline
\end{array}
\]

into non-composite FD-based proxies and composite FD-based proxies. To determine the worst case scenario for the non-composite FD-based proxies, we need to identify the size of the largest possible proxy map. A proxy map is at its largest if it consists of all pairs of the droppable attribute and the proxy candidate that are present in the population table. This is only possible if all the tuple pairs of the droppable attribute and the proxy candidate in the population table are not identical. As the consequence, the size of the proxy map is two times bigger than the size of the droppable attribute. For example, Table 5.13(a) shows the worst case space saving scenario that involves a non-composite proxy candidate \( I \). As the size of \( B \) is 10, and the size of the proxy map \( BA \) (as shown in Table 5.13(b)) is 20, space saving in this case is:

\[
\begin{align*}
&= \frac{(10 - 20)}{10} \times 100, \\
&= \frac{-10}{10} \times 100, \\
&= -100\% 
\end{align*}
\]

Therefore, we say that in the worst case scenario for the non-composite FD-based proxy candidate, we need to store a proxy map which is two times bigger than
the size of the droppable attribute. Based on this observation, we can define space saving in the worst case scenario for the non-composite FD-based proxy candidate as:

\[
\frac{\text{droppableAttrSize} - 2(\text{droppableAttrSize})}{\text{droppableAttrSize}} \times 100,
\]

which will always result in -100% space saving.

Table 5.13: An Example of the Worst Space Saving Case (Non-composite FD-based Proxy Candidate)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>3</td>
<td>a</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>4</td>
<td>a</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td>( t_6 )</td>
<td>6</td>
<td>a</td>
</tr>
<tr>
<td>( t_7 )</td>
<td>7</td>
<td>a</td>
</tr>
<tr>
<td>( t_8 )</td>
<td>8</td>
<td>a</td>
</tr>
<tr>
<td>( t_9 )</td>
<td>9</td>
<td>a</td>
</tr>
<tr>
<td>( t_{10} )</td>
<td>10</td>
<td>a</td>
</tr>
</tbody>
</table>

\(B\) \(I\)

\(x\) 1

\(y\) 2

\(z\) 3

\(k\) 4

\(l\) 5

\(m\) 6

\(n\) 7

\(o\) 8

\(p\) 9

\(q\) 10

To examine the worst case scenario for the composite FD-based proxy candidate, we need to determine the highest possible number of partial proxies in the composite proxy candidate from the population table. Within a population table schema, all attributes are the possible partial proxies except the droppable attributes itself. For example, consider this time that the population table \(pt\) has schema \(PT = \langle D, A, C, B \rangle\), where \(B\) is the droppable attribute. Therefore in \(PT\), the attributes \(D\), \(A\) and \(C\) are all possible partial proxies. Thus, in this example, the highest possible number of partial proxies for a composite proxy candidate is 3. The composite proxy candidate from \(PT\) with the highest possible number of partial proxies is one of all possible combinations of the partial proxies, e.g., \((D, A, C)\) or \((D, C, A)\). As for the worst-case scenario of the non-composite proxy candidate, the proxy map used is at its largest possible size. The largest possible
proxy map consists of all pairs of the droppable attribute and the composite proxy candidate that are present in the population table. Having a proxy map of the largest possible size is possible if all the tuple pairs of the droppable attribute and the composite proxy candidate in the population table are not identical. If this is the case, the size of the proxy map in the example (as shown in Table 5.14(b)) is four times bigger than the size of the droppable attribute. Table 5.14(a) shows the worst case space saving scenario for the composite FD-based proxy candidate $(D,A,C)$, where $(D,A,C) \rightarrow B$. As the size of $B$ is 10, and the size of the proxy map $BDAC$ is 40, space saving in this case is:

\[
\frac{(10 - 40)}{10} \times 100, \\
= -30 \times 100, \\
= -300\%
\]

Therefore, we say that in the worst case scenario, for a composite FD-based proxy candidate with $n$ partial proxies, we need to store a proxy map which is $n+1$ times bigger than the size of the droppable attribute. Based on this observation, we define space saving in the worst case scenario for the composite FD-based proxy candidate as:

\[
\frac{\text{droppableAttrSize} - (n + 1(\text{droppableAttrSize}))}{\text{droppableAttrSize}} \times 100. \tag{5.14}
\]

Even though we said that attention should be given to FD-based proxies that we expect to be highly accurate, it might be worth to observe the best- and the worst-case space saving scenarios of the AFD-based proxies as well. In contrast to FD-based proxies, given a droppable attribute, we cannot expect all values its proxy candidate to substitute exactly one value of the droppable attribute due to some violations against the FD as presented in the defective proxies analysis (see Section 5.4.2.1, page 166). Therefore, in the best-case space saving scenario, there is only one proxy candidate’s value that substitutes for more than one value of the droppable attribute. To illustrate the best case scenario for AFD-based proxies, consider the population table $pt$ has schema $PT= \langle I, E, B \rangle$, where $I$ is the identifier attribute, as shown in Table 5.15(a). Suppose that $B$ is the droppable attribute, $E$ is the AFD-based proxy candidate, where $E \sim B$.

At its very least, the proxy map consists of only two pairs of the droppable
Table 5.14: An Example of the Worst Space Saving Case (Composite FD-based Proxy Candidate)

(a) Population Table \(pt\)   (b) Proxy Map \(BDAC\)

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>A</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>f</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>g</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>a</td>
<td>f</td>
<td>l</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>a</td>
<td>g</td>
<td>k</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>a</td>
<td>f</td>
<td>l</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>a</td>
<td>g</td>
<td>m</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>a</td>
<td>f</td>
<td>n</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>a</td>
<td>g</td>
<td>o</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>a</td>
<td>f</td>
<td>p</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>a</td>
<td>g</td>
<td>q</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(D,A,C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>(1,a,f)</td>
</tr>
<tr>
<td>y</td>
<td>(1,a,g)</td>
</tr>
<tr>
<td>z</td>
<td>(2,a,f)</td>
</tr>
<tr>
<td>k</td>
<td>(2,a,g)</td>
</tr>
<tr>
<td>l</td>
<td>(3,a,f)</td>
</tr>
<tr>
<td>m</td>
<td>(3,a,g)</td>
</tr>
<tr>
<td>n</td>
<td>(4,a,l)</td>
</tr>
<tr>
<td>o</td>
<td>(4,a,g)</td>
</tr>
<tr>
<td>p</td>
<td>(5,a,f)</td>
</tr>
<tr>
<td>q</td>
<td>(5,a,g)</td>
</tr>
</tbody>
</table>

attribute and the AFD-based proxy candidate instances, making its size equal equals to four. This is only possible only if two pairs of the droppable attribute and the proxy candidate in the population table are not identical and the proxy candidate is not a composite proxy. For example, Table 5.15(a) shows this situation that involves proxy candidate \(E\). As the size of \(B\) is 10, and the size of the proxy map \(BE\) (as shown in Table 5.12(b)) is four, space saving in this case is:

\[
= \frac{(10 - 4)}{10} \times 100,
\]

\[
= \frac{6}{10} \times 100,
\]

\[
= 60\%
\]

Therefore, we say that in the best case scenario for the AFD-based proxy, we only need to store a proxy map of the smallest possible size that consists of only four instances. Based on this observation, we define space saving in the best case scenario for AFD-based proxy candidates as:

\[
\frac{\text{droppableAttrSize} - 4}{\text{droppableAttrSize}} \times 100. \tag{5.15}
\]
Table 5.15: An Example of the Best Space Saving Case for an AFD-based Proxy Candidate

(a) Population Table $pt$  (b) Proxy Map $BE$

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>E</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t1$</td>
<td>1</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>$t2$</td>
<td>2</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>$t3$</td>
<td>3</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>$t4$</td>
<td>4</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>$t5$</td>
<td>5</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>$t6$</td>
<td>6</td>
<td>a</td>
<td>y</td>
</tr>
<tr>
<td>$t7$</td>
<td>7</td>
<td>a</td>
<td>y</td>
</tr>
<tr>
<td>$t8$</td>
<td>8</td>
<td>a</td>
<td>y</td>
</tr>
<tr>
<td>$t9$</td>
<td>9</td>
<td>a</td>
<td>y</td>
</tr>
<tr>
<td>$t10$</td>
<td>10</td>
<td>a</td>
<td>y</td>
</tr>
</tbody>
</table>

$BE$  $x$  $a$

In the worst-case space saving scenario for non-composite, AFD-based proxy candidates, the proxy map consists of all pairs of the droppable attribute and the proxy candidate’s tuples that are present in the population table. Notice that this is similar to the worst case scenario presented for the non-composite, FD-based proxy candidates presented earlier. This is only possible if all the tuple pairs of the droppable attribute and the proxy candidate in the population table are not identical. As the consequence, the size of the proxy map is two times bigger than the size of the droppable attribute. For example, Table 5.16(a) shows the worst case space saving scenario that involves an AFD-based proxy candidate $E$. As the size of $B$ is 10, and the size of the proxy map $BE$ (as shown in Table 5.13(b)) is 20, space saving in this case is:

$$
\begin{align*}
&= \frac{(10 - 20)}{10} \times 100, \\
&= \frac{-10}{10} \times 100, \\
&= -100\
\end{align*}
$$

In the worst-case scenario for the non-composite AFD-based proxy candidate, we need to store a proxy map which is two times bigger than the size of the
droppable attribute, which is the same for the non-composite, FD-based proxy candidate’s worst case scenario. Therefore, the definition for space saving in the worst case scenario for the non-composite FD-based proxy candidate as shown in Equation (5.13) can be used to compute a non-composite AFD-based proxy candidate’s worst-case space saving score. Because of the similarity of the characteristics of the AFD-based proxy candidate and the FD-based candidate in the worst-case scenario, we postulate that the space saving definition for the worst-case scenario of the composite, FD-based proxy candidates (as shown in Equation (5.14) ) should be applicable to the worst case scenario of the composite, AFD-based proxy candidates as well.

Table 5.16: An Example of the Worst Space Saving Case for an AFD-based Proxy Candidate

(a) Population Table $pt$

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>E</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t1$</td>
<td>1</td>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>$t2$</td>
<td>2</td>
<td>a</td>
<td>y</td>
</tr>
<tr>
<td>$t3$</td>
<td>3</td>
<td>a</td>
<td>z</td>
</tr>
<tr>
<td>$t4$</td>
<td>4</td>
<td>a</td>
<td>k</td>
</tr>
<tr>
<td>$t5$</td>
<td>5</td>
<td>a</td>
<td>l</td>
</tr>
<tr>
<td>$t6$</td>
<td>6</td>
<td>a</td>
<td>m</td>
</tr>
<tr>
<td>$t7$</td>
<td>7</td>
<td>a</td>
<td>n</td>
</tr>
<tr>
<td>$t8$</td>
<td>8</td>
<td>a</td>
<td>o</td>
</tr>
<tr>
<td>$t9$</td>
<td>9</td>
<td>a</td>
<td>p</td>
</tr>
<tr>
<td>$t10$</td>
<td>10</td>
<td>a</td>
<td>q</td>
</tr>
</tbody>
</table>

(b) Proxy Map $BE$

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$y$</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$z$</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$k$</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$l$</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$m$</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$n$</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$o$</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$p$</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$q$</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

By examining the best- and the worst-case space saving scenarios of both types of proxies, we conclude that what determines the amount of space saving that can be offered by the proxy candidates is: 1) the amount of repeated tuple pairs between the droppable attribute and the proxy candidate and 2) the proxy candidate’s composition (i.e., composite or non-composite proxy candidate). One important lesson that must be noted is that, under these extreme cases, the choice of the type of table structure for the proxy maps will not make any difference to the amount of space saving. This is because, for the multi-valued table to be useful in space saving, the droppable attribute values must be substituted by
more than one value from the proxy candidates, which is not possible either in the best or in the worst case scenarios. Another important observation that is worth mentioning is that space saving for proxy candidates that are key attributes in the population table (such as $I$ in $pt$) might offer less space saving because the number of repeated tuple pairs between the droppable attribute and the proxy candidate is low, relative to non-key FD-based proxy candidates (such as $A$ in $pt$). Therefore, we say that it will be less likely for FD-based proxy candidates that are key attributes result in a best-case space saving scenario.

In the Section 5.6.1 we will present a case study for FD-based proxies with the aim to support our analysis presented in this section.

5.6 Case Studies

In this section we set out to observe space-accuracy trade-offs in space optimisation through proxies. We will present two case studies that deal with FD-based proxies and AFD-based proxies respectively. For both case studies, the proxy assessment algorithm as presented in Algorithm 2 was used, in which the space saving computation algorithm (see Algorithm 3) and the accuracy computation algorithm (see Algorithm 4) were implemented. To observe real space saving and accuracy of proxy candidates that cannot be supported by the artificial data sets used in the analyses performed earlier, real data sets from the microbial domain were used in these case studies.

Recall that the objective of the proxy assessment is to support the PBC measurement provider in selecting which attributes to drop from the schema of the universe. Thus, the result of any proxy assessment should support answering the question: which of the droppable attributes has proxy candidates with acceptable space saving and accuracy? Answering this question requires examining space-accuracy trade-offs by proxy candidates for each CS where the same assessment procedure applies to all CSs. We therefore designed the case studies to cover proxy assessment from the context of a CS.

5.6.1 Space-Accuracy Trade-Offs of FD-Based Proxies

In this section we will present the results of FD-based proxies assessments in the case of FD-based proxy candidates discovered from the microbial domain. The
results of this case study are used to support the analyses that we performed earlier in this chapter. In particular, we set to answer the following questions:

- Are FD-based proxy candidates defective proxies or not? The answer to this question is used to support our observations of the characteristics of FD-based proxy candidates presented in the defective proxy analysis (see Section 5.4.2.1 in page 166).

- Are the space saving scores yielded for the FD-based proxies close to the best- or to the worst-case scenarios? Answer to this question is used to support the identification of the factors that determine the amount of space saving that can be offered by the proxy candidates presented earlier in the space saving analysis (see Section 5.5 in page 180).

In this case study, we used data sets that were used earlier in comparing the size of proxy maps by the types of the table structure in Section 5.2.1 (page 136) to discover FD-based proxies. Previously, FD-based proxy candidates were determined based on the information about the key attributes in the population tables. Nevertheless, FD-based proxies are not necessarily key attributes in a population tables. Therefore, to consider the non-key, FD-based proxy candidates that might be present in the population tables, we follow a proxy assessment algorithm that utilises the TANE algorithm to discover FDs. For convenience, we reuse the population tables that we used (evidence, taxon and bug_attribute) to discover FD-based proxies where the schema of these tables are as follows:

- Table 1: Evidence = \{id, ev_type, method, assignby\},
- Table 2: Taxon = \{uid, genus, species, kingdom, ir1\},
- Table 3: Bug_attribute = \{id, att_type, assignby\}.

Based on the result of TANE, we found that, FD-based proxy candidates discovered through TANE are identical to the FD-based proxies that we identified earlier in Section 5.2.1. This means, the FD-based proxy candidates available from the microbial data sets that we use in this case study are the key attributes. We regard all attributes other than the proxy candidates as droppable attributes. The FD-based proxy candidates are as follows:

- from evidence:


- id $\rightarrow$ ev.type,
- id $\rightarrow$ method,
- id $\rightarrow$ assignby.

- From taxon:
  - uid $\rightarrow$ genus,
  - uid $\rightarrow$ species,
  - uid $\rightarrow$ kingdom.

- From bug.attribute:
  - id $\rightarrow$ att.type,
  - id $\rightarrow$ assignby.

Before we can assess space saving of proxy candidates, we need to choose the type of table structure for the proxy maps (i.e., pure relational or multi-valued). To decide which type of table structure to adopt, we issue the query as shown in Equation 5.6 (page 143) against each population table to check whether the values in the droppable attribute domain map with multiple values of the proxy. As most of the droppable attribute values in all population tables map with multiple values of the proxy, we selected a multi-valued table structure to store the proxy maps, which offers better space saving benefits (as discussed in Section 5.2.1 in page 136) over the pure relational table structure in this case.

To aid in answering whether space savings offered by the FD-based proxy candidate in this case study are close to the best or to the worst case scenario, we computed space saving scores in these two extreme cases for all FD-based proxy candidates under assessment. We used the definition presented in Equation (5.12) (page 181) to compute the best possible space saving. Table 5.17 shows the space saving scores in the best case scenario for FD-based proxy candidates in this case study. As all FD-based proxy candidates are of the non-composite type, the worst possible space saving will be -100% as defined in the Equation (5.13) (page 183).

We used used a bar chart, as shown in Figure 5.5, to illustrate space saving and accuracy of all FD-based proxy candidates assessed in this case study. As the figure illustrates, all FD-based proxy candidates in this case have no proxy error. Based on this result, we can say that FD-based proxy candidates discovered from
CHAPTER 5. SPACE-ACCURACY TRADE-OFFS

Table 5.17: The Best-Case Space Saving Scenario for FD-based Proxy Candidates

<table>
<thead>
<tr>
<th>Proxy Candidates</th>
<th>Droppable attribute’s size</th>
<th>The Best Space Saving: $\frac{droppableAttrSize - 2}{droppableAttrSize} \times 100$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(evidence) id</td>
<td>ev_type: 292843, method: 292843, assignby: 292843</td>
<td>$\frac{292843-2}{292843} \times 100 = 99.99%$</td>
</tr>
<tr>
<td>(taxon) uid</td>
<td>genus: 571, species: 571, kingdom: 571</td>
<td>$\frac{571-2}{571} \times 100 = 99.65%$</td>
</tr>
<tr>
<td>(bug_attribute)</td>
<td>att_type: 7492, assignby: 7492</td>
<td>$\frac{7492-2}{7492} \times 100 = 99.97%$</td>
</tr>
</tbody>
</table>

the microbial data sets are not defective. Nevertheless, these highly accurate proxy candidates in this case unfortunately do not offer any space saving, as the results show negative space saving for all FD-based proxy candidates being assessed.

To present the details of the space saving results, we use a bar chart as shown in Figure 5.6 that illustrates space saving results of FD-based proxy candidates specific to a population table. In comparison, FD-based proxy candidate from evidence (which is id) performs better than other FD-based proxy candidates from other tables with space saving scores between -0.0038 to -0.0219%.

Based on the results, we can see that the space saving scores yielded from the FD-based proxies are far from the best case scenario. However, even though negative space savings were yielded for all proxy candidates, most of these scores are very small and less than -5%, which means only a small amount of storage is required by the proxy maps. With negative results it means the amount of repeated tuple pairs between the droppable attribute and the proxy candidate is very low, which matches up with our observations regarding the results of the space saving analysis (Section 5.5 in page 180) where we said that FD-based proxy candidates that are key attributes are more likely to exhibit this characteristic.

To conclude, the results of assessing FD-based proxy candidates in the case of the microbial data sets provide answers to the questions raised at the beginning of this section. The results support our analysis that FD-based proxy candidates
are non-defective as presented in the defective proxy analysis section. In addition, it also support our analysis on one factor that determines the amount of space saving, which is the number of repeated tuple pairs between the droppable attribute and the proxy candidate. This case study however cannot be used to support our analysis of the worst case scenario for the composite FD-based proxy candidates (which we postulated to have space saving scores greater than -100%) as all of the proxy candidates discovered from the microbial data sets in this case study are non-composite proxy candidates.

![Figure 5.5: Space Saving and Proxy Errors of FD-based Proxy Candidates](image)

5.6.2 Space-Accuracy Trade-Offs of AFD-Based Proxies

The absence of highly accurate proxies that could offer space saving such as in the case of the microbial data set presented in the previous section leads to the exploration of another type of proxy, the AFD-based proxies that we expect may offer space saving. In this section, we will present the results of assessing
CHAPTER 5. SPACE-ACCURACY TRADE-OFFS

(a) Space saving by id as the proxy candidate for the droppable attributes from evidence

(b) Space saving by uid as the proxy candidate for the droppable attributes from taxon

(c) Space saving by Id as the proxy candidate for the droppable attributes from Bug_attribute

Figure 5.6: The Results of Space Saving Assessment for FD-Based Proxy Candidates
AFD-based proxy candidates discovered from the same microbial domain. In assessing AFD-based proxy candidates, we set out to answer several questions, some of which are similar to the questions raised in assessing the FD-based proxy candidates in the previous section. These questions are:

- Are AFD-based proxy candidates defective proxies or not?
- Are the proxy errors yielded are close to the best or the worst-case scenario of AFD-based proxy candidate accuracy?
- Are the space saving scores yielded for the AFD-based proxies close to the best- or to the worst-case scenarios?

Answers to the first two questions are used to support the identification of the factors that determine the accuracy of AFD-based proxy candidates presented earlier in the defective proxies analysis (see Section 5.4.2.1 in page 166), while the third question will support the observation made on space saving that can be offered by the AFD-based proxies in space saving analysis presented in Section 5.5 (page 180).

For this case study, we used the same microbial data set used earlier in AFD error analysis presented in Section 5.4.2.2 (page 171). In particular, we reuse the table called eu that we used to discover AFD-based proxy candidates which has schema:

\[ EU = \langle \text{id, accession, locus, assignby, method, rel_end3, rel_end5, end3, end5, ev_type} \rangle \].

For the same reason as before, we selected droppable attributes (shown in bold) that have a smaller domain value size relative to other attributes in the EU to avoid constructing a large number of sample queries.

Based on the result of TANE, AFD-based proxy candidates discovered through TANE from the microbial data set are as follows, where there is a mix of non-composite and composite types of proxy candidates:

- for droppable attribute assignby:
  - accession,end5 \rightarrow assignby,
  - locus,ev_type \rightarrow assignby,
  - end5 \rightarrow assignby,
  - end3 \rightarrow assignby,
• For droppable attribute \textit{method}:
  
  \begin{itemize}
    
    \item accession $\rightarrow$ method,
    
    \item ev\_type $\rightarrow$ method,
    
    \item rel\_end3 $\rightarrow$ method,
    
    \item end5 $\rightarrow$ method,
    
    \item locus $\rightarrow$ method.
  \end{itemize}

• For droppable attribute \textit{ev\_type}:
  
  \begin{itemize}
    
    \item method $\rightarrow$ ev\_type,
    
    \item rel\_end5,rel\_end3 $\rightarrow$ ev\_type,
    
    \item rel\_end3,assignby $\rightarrow$ ev\_type,
    
    \item locus,rel\_end5 $\rightarrow$ ev\_type,
    
    \item rel\_end5 $\rightarrow$ ev\_type,
    
    \item rel\_end3 $\rightarrow$ ev\_type,
    
    \item assignby $\rightarrow$ ev\_type,
    
    \item end5 $\rightarrow$ ev\_type,
    
    \item end3 $\rightarrow$ ev\_type,
    
    \item locus $\rightarrow$ ev\_type.
  \end{itemize}

To aid in answering whether space savings offered by the AFD-based proxy candidate in this case study are close to the best- or the worst-case scenarios, we computed space saving scores in these two extreme cases for all AFD-based proxy candidates under assessment. As the proxy candidates are a mix of composite and non-composite proxy candidates, we needed to determine the worst case scenarios of AFD-based proxy candidates that are of composite and non-composite type. We used the definition presented in Equation (5.15) (page 185) to compute the best possible space saving for the proxy candidates. For AFD-based proxy candidates of the non-composite type, the worst possible space saving will be -100\% as defined in the Equation (5.13) (page 183); while for the AFD-based proxy candidates of the composite type, the worst possible space saving was determined
using the definition presented in Equation (5.14) (page 184). As the number of attributes in the schema of the population table $\text{EU}$ is 10, then the maximum number of the partial proxies ($n$) for the composite candidate is 9. Note that all droppable attributes come from the same population table ($\text{EU}$), thus, they have the same size, which is 292843 instances. As the size of droppable attributes are the same, the best case space saving scenario and the worst case space saving scenario presented in Table 5.18 are applicable to all AFD-based proxy candidates in this case study.

To aid in answering whether accuracy of the AFD-based proxy candidate in this case study is close to the worst-case scenario or not, we computed the accuracy score in the worst case scenario using the definition presented in Equation (5.11) (page 170) as follows:

$$\frac{(1 - 292843)}{292843} \times 100 = -99.99\%.$$ 

In the best case scenario, as presented in the defective proxies analysis (see Section 5.4.2.1, page 166), an AFD-based proxy candidate commits no proxy error. Therefore using the worst case and the best case accuracy information for the AFD-based proxy candidates, we shall next present the answer of whether the AFD-based proxy candidates in this case study are close to these two extreme cases or not. Note that, we used all the aggregation types to aggregate proxy errors, which are the maximum proxy error, the minimum proxy error and the average proxy error.

<table>
<thead>
<tr>
<th>The Best Space Saving:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{\text{droppableAttrSize}-4}{\text{droppableAttrSize}} \times 100$</td>
</tr>
<tr>
<td>$\frac{292843-4}{292843} \times 100=99.9%$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The Worst Space Saving (Non-composite):</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{\text{droppableAttrSize}-2(\text{droppableAttrSize})}{\text{droppableAttrSize}} \times 100$</td>
</tr>
<tr>
<td>$\frac{292843-2(292843)}{292843} \times 100=-100%$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The Worst Space Saving (Composite):</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{\text{droppableAttrSize}-(n+1(\text{droppableAttrSize}))}{\text{droppableAttrSize}} \times 100,$</td>
</tr>
<tr>
<td>$\frac{292843-9(292843)}{292843} \times 100=-800%$</td>
</tr>
</tbody>
</table>

where $n = 9$

The bar chart in Figure 5.7 shows the space saving and the maximum proxy error of all AFD-based proxy candidates assessed in this case study. As the figure
illustrates, all AFD-based proxy candidates in this case study commit proxy error. This means, AFD-based proxy candidates discovered from the microbial data sets are all defective. Furthermore, as negative proxy errors were yielded, we can say that spurious individuals are present in the reference sub-populations. With the presence of spurious individuals, the reference sub-populations in this case are therefore inaccurate. In terms of space saving, all of the proxy candidates offer space saving as the results show positive space saving for all AFD-based proxy candidates being assessed. Most of the proxy candidates’ space saving scores are very close to the best case scenario of space saving. However, proxy candidates that exhibit a high amount of space savings have a high amount of proxy error as well.

Figure 5.8 is a bar chart that plots space saving and the average proxy error of all AFD-based proxy candidates assessed in this case study. As shown in the bar chart, the average proxy errors made by AFD-based proxy candidates in the microbial domain range from zero to -90%. Some proxy candidates that have high maximum proxy errors (in Figure 5.7) exhibit lower average proxy error. This means, low proxy errors were yielded from the result of most query transformation involving those proxy candidates. We elect to exclude the result of minimum proxy error as all proxy candidates scored 0% for the minimum error, which matches the best case scenario of proxy accuracy.

Figure 5.9, 5.10 and 5.11 present the details of space saving and proxy error score by proxy candidates of the individual droppable attribute. Each figure consists of two bar charts plotting the maximum proxy error and the average error respectively. In comparison, droppable attribute **ev_type** has a proxy candidate (**locus,rel_end5**) with the lowest maximum proxy error which is less than 7% and with 20% space saving. The selection of which attribute to drop will be based on results such as those presented in these figures, where a droppable attribute that has proxy candidates with acceptable space saving and accuracy has the chance to be dropped from the universe’s schema.

To conclude, the results of assessing AFD-based proxy candidates in the case of the microbial data sets support our analysis that AFD-based proxy candidates are defective and that they cause the reference sub-population to be inaccurate (instead of incomplete) as presented in the defective proxy analysis section. The AFD-based proxy candidates that exhibit the best case scenario of accuracy in this case study support the analysis that it is possible to use some values of proxy
CHAPTER 5. SPACE-ACCURACY TRADE-OFFS

Figure 5.7: Maximum Proxy Error and Space Saving of AFD-based Proxy Candidates

Figure 5.8: Average Proxy Error and Space Saving of AFD-based Proxy Candidates
Figure 5.9: Proxy Error and Space Saving of AFD-based Proxy Candidates for Droppable Attribute assignby
CHAPTER 5. SPACE-ACCURACY TRADE-OFFS

Figure 5.10: Proxy Error and Space Saving of AFD-based Proxy Candidates for Droppable Attribute method
CHAPTER 5. SPACE-ACCURACY TRADE-OFFS

Figure 5.11: Proxy Error and Space Saving of AFD-based Proxy Candidates for Droppable Attribute ev_type
candidates to substitute for the droppable attribute values without making any proxy error.

Furthermore, even though proxy errors were made by all of the AFD-based proxy candidates being assessed, we can find proxy candidates with very low maximum proxy errors (near to 0%), which therefore may be acceptable by the PBC measurement providers to be the proxy for the droppable attribute. As only positive space savings were yielded, it means the amount of repeated tuple pairs between the droppable attribute and the proxy candidate is high enough to offer some space savings. This supports our analysis on the factor that determines the amount of space saving, which is the number of repeated tuple pairs between the droppable attribute and the proxy candidate (see Section 5.5 in page 180).

In addition, we say that, space saving of the AFD-based proxy candidates in this case study is far from the worst case scenario (as shown in Table 5.18), either for the composite or the non-composite proxy candidates, with half of the proxy candidates having scores that are very close to the best-case space saving scenario (between 80-99%). Nevertheless, those proxy candidates with high space saving scores exhibit proxy errors that are close to the worst case accuracy (-99.99%), which might make them less attractive to be selected as the proxy for the droppable attribute.

5.7 Conclusions

In conclusion, this chapter answered through analyses and case studies how proxies contribute to space saving. An analysis of the implementation options of proxy maps contributed to understanding one factor that determines space saving. Space saving analysis identified further factors and the cases where space saving cannot be influenced by how the proxy maps are implemented.

We presented the default behaviour of PBC systems where proxies can be observed in two phases within the PBC reference architecture which are: proxies in use and proxies under consideration. We presented how to select attributes that we can drop from the schema of the universe by proposing the proxy assessment algorithm. The analysis of defective proxies contributed to identification of the characteristics of defective proxies which was supported by the case studies on two types of proxies namely FD-based and AFD-based proxies. We defined the best and the worst case scenarios of accuracy and space saving, proposed some
formulae to compute them, and observed how close were the assessment results with those extreme cases.

The lesson learnt from the AFD error analysis is that the g3 scores have weak correlation with the proxy errors due to the differences in the steps (and the goal) to compute them. The result of a small case study we conducted using microbial data sets supports the lack of correlation between these two errors.

FD-based proxies are highly accurate and some of them (i.e., the non-key FD-based proxies) offer space saving. However we encountered the case in the microbial domain that FD-based proxy candidates do not offer space saving due to the fact that they are all keys in the population tables. The case study supports our analysis on one factor that determines space saving which is small number of repeated pairs of the droppable attribute and the proxy candidate tuples inherent in those FD-based proxy candidates in the case study. AFD-based proxies however, offer space saving with some accuracy trade-offs in PBC measurements. In certain cases, accuracy may be unaffected. The case study for AFD-based proxy candidates provides evidence from the microbial domain that supports the behaviour of AFD-based proxies observed in the analysis, in terms of their space saving and accuracy.

As a whole, even though the trade-off between space saving and accuracy is inevitable in some cases, we can say that the proxy approach is a feasible alternative for space optimisation. Proxies can be useful in the cases where the benefits they offer in terms of space savings outweigh the small (and acceptable) number of proxy errors that occur.
Chapter 6

Conclusions

This thesis presents our research, which has been motivated by the problem inherent in data completeness. In particular, we concentrated on the problem of measuring completeness faced by completeness measurement providers, which has been underestimated by many studies in terms of the difficulty of acquiring the inputs necessary to make it workable.

One challenging task is to obtain the reference data sets as the input to measure completeness due to the absence of true, complete reference data sets in many application domains. The question that we set out to answer is ‘How can we define and obtain reference data sets that can represent the true reference data sets?’ We approached the problem by investigating one particular type of completeness that we call population based completeness (PBC).

We attempted to answer several questions regarding PBC such as ‘What is a population?’, ‘What characterises individuals that are suitable to act as the members of populations for PBC?’, ‘Is it feasible to gather the true reference populations?’ and ‘What information must be stored about a reference population?’ Answering these questions led us towards the proposal of the notion of universe, which consists of approximate reference populations purposely defined to represent the true populations (i.e. true reference data set), in addition to other elements in the PBC model.

In answering how we can make the PBC model implementable in practice, we set out to answer questions regarding the software and hardware components needed to support PBC as a workable measurement system. In addition, another question that arises is ‘What are the issues that might be faced by PBC measurement providers in answering accurate PBC measurement requests?’ This
question promotes the problem of measuring PBC using incomplete or inaccurate reference populations.

One issue that may cause reference populations to be incomplete is storage space constraint, which was revealed by exploring the options that PBC measurement providers have when implementing the universe. This issue is driven by the need to materialise the universe’s reference populations gathered from multiple databases, which may result in an integrated database that is impractically large. We dealt with the problem of how we can optimise the space of the universe without compromising the accuracy of PBC measurements performed using the optimised universe. We investigated one space optimisation technique that optimises the universe by materialising only partial information from the contributing databases. As only some information will be materialised, we proposed to compensate the information loss by substituting some of the attributes from the contributing databases with smaller alternatives called proxies. We answered several questions that arose from the usage of proxies. These questions included ‘How can we discover proxies?’, ‘How can we choose which information (i.e. attributes) that can be substituted by the proxies?’ and ‘What will be required to implement proxies in any PBC systems?’

In the next section, we will present the key lessons from answering the aforementioned questions raised in this thesis.

6.1 Research Contributions

The results of our research presented in this thesis contribute mainly to the data quality community, in particular in advancing the topic of completeness measurement. Nevertheless, the results of the analyses and the case studies might also be useful to data analysts from the biological domain as well as to the database community. We summarise the research contributions presented in this thesis as follows.

6.1.1 The Definition of PBC

Answers to questions regarding PBC contribute towards an understanding of the elements necessary to measure PBC, and the challenge in obtaining true reference populations. We learnt that approximate reference populations are a typical form of reference populations used in many application domains where true reference
populations are difficult to establish. Good approximate reference populations can be established by integrating individuals from a range of sources. We proposed the PBC model to support the definition and configuration of approximate reference populations that are particularly useful in the situation where a single source that contains good approximation of the true populations is limited (or does not exist).

The results of exploring PBC are:

- The definitions of PBC elements: the data set under measure, the reference population, the universe, the contributing source and the population map.
- The identification of the types of PBC measurement requests.
- The PBC measurement formula that supports the types of PBC measurement requests identified.
- The reference template of the basic configuration of PBC.

6.1.2 The Results of the Practical Analysis of PBC

In answering how we can make the PBC model implementable in practice, we analysed the software and hardware components needed to support PBC systems in answering PBC measurement requests. The results of the analysis are:

- The identification of the types of PBC measurement reference architecture.
- The list of the technical issues inherent in a PBC system that affect the answering of accurate PBC measurement requests.
- The identification of the conditions where incomplete and inaccurate reference populations affect PBC measurement accuracy.
- An example of how a PBC system is implemented in the case of a biological domain.

Based on the results we learnt that to make a PBC system implementable, there are basic software and hardware components that must be configured by PBC measurement providers in order to answer PBC measurement requests. The main outcome of the analysis is the list of open issues that must be dealt with by PBC measurement providers to support the answering of accurate PBC measurement
requests. The side effects of the analysis contribute to the PBC reference architecture, which consists of software components that are added along with the basic softwares to deal with those issues.

6.1.3 The Results of Storage Space Optimisation Analysis

The result of investigating how we can deal with storage space constraint contributes to the proposal of a space optimisation technique. Answering the question of how we can optimise the space of the universe without compromising the accuracy of PBC measurements performed using the optimised universe led to an exploration of the applicability of functional dependencies (FDs) and approximate functional dependencies (AFDs) in supporting space optimisation. In particular, we proposed substituting the attributes that we dropped from the schema of the universe with the attributes discovered based on FDs and AFDs, called proxies. We postulated such schema modification as a way to gain spaces, because only partial information from the universe’s CSs will be materialised in the universe; information loss due to the exclusion of some information will be compensated by materialising some information about the dropped attributes in a structure called proxy maps.

The investigation of what proxies could offer to space optimisation leads to a set of analyses that each makes its own contribution:

- Implementation options of proxy maps analysis: the result of this analysis is the identification of one factor that determines space saving, which is on how the proxy maps are stored. We compared the size of proxy maps using artificial data sets and supported the analysis with a small case study using real data sets from the microbial domain.

- Space saving analysis: this analysis contributes to the identification of other factors that determine space saving in addition to the types of the proxy maps. We examined the best- and the worst-case scenarios of proxies’ space saving that resulted in the definitions of space saving under those extreme cases for both FD-based proxies and AFD-based proxies.

- Defective proxy analysis: the result of this analysis contributes the definition of proxy error, the identification of the factor that causes proxy errors and the definition of proxy error in the worst-case scenario.
• AFD error analysis: this analysis contributes to an understanding of the similarities and the differences between AFD errors and proxy errors. A small case study we conducted using microbial data sets supports our observation of the lack of correlation between these two types of errors.

The main results of these analyses are the definition of space saving and accuracy of proxies, and the proxy assessment algorithm proposed to answer the question of ‘How can we choose which information (i.e., attributes) can be substituted by the proxies?’, which supports PBC measurement providers in making their selection on the droppable attributes.

6.1.4 The Proxy-based Reference Architecture of a PBC System

Answering the question of ‘What will be required to implement proxies in any PBC systems?’ led to the identification of the default behaviour of proxy-based PBC systems. We observed the default behaviour of PBC systems in two phases of proxies, which are proxies in use and proxies under assessment. We proposed the query transformation algorithm that contributes to an understanding of how proxies are used within PBC measurements supported by PBC systems. The reference architecture of PBC where proxies are applied is the outcome of our observation, which contributes configurable architecture for any systems that have similar space optimisation goals as PBC systems.

6.1.5 The Results of the Case Studies of Space-Accuracy Trade-Offs

The results of the case studies using microbial data sets that we conducted upon two types of proxies, FD-based and AFD-based, support the analyses performed earlier and contribute to an understanding of space-accuracy trade-offs that are inevitable in some cases. The case studies revealed that FD-based proxies that are highly accurate in the microbial domain require a range of additional space requirements to store the proxy maps, which is an expected characteristic of proxies that are the key attributes in the population tables. To achieve space saving, these types of proxies are therefore not useful, even though they are highly accurate. The lesson learnt is that AFD-based proxies that offer space
saving could be the alternatives in the case where non-key, FD-based proxies are not available in the domain. These proxies, as observed in the case study, cause a range of accuracy trade-offs from none to the worst-case proxy errors.

6.2 Limitations and Future Work

Various directions can be foreseen for future research stemming from our work presented in this thesis. We highlight the possible future work in terms of the PBC model that we proposed and the way to deal with the space constraint issue, which are of course not limited to these two areas.

6.2.1 PBC model

PBC model in other contexts We proposed a completeness measurement model specific to PBC, which means that its usefulness for other types of completeness or for other data quality dimensions (e.g., accuracy or reputation) has not been observed. The first step to expand the applicability of PBC notions might be by answering the question of ‘Which types of completeness or data quality dimensions require similar inputs and constraints (i.e., in obtaining true reference data sets)?’ One potential application of the PBC model is in measuring TBC, in particular the one that requires the use of master data as the reference data set, as proposed by Fan and Geerts [FG10] and presented in Chapter 2 (see Section 2.3.2 page 30). In addition, further investigation might be required in determining whether additional elements are needed to support the measuring of quality based on the PBC model and what are the issues that might arise in implementing the applied PBC model.

The PBC reference architecture Attention has been given to the PBC reference architecture with the materialised universe (please see Section 4.2.1 in page 62 for the description of the types of PBC reference architecture), which led to the analysis of the issues inherent in this particular type of architecture. It might be worth investigating in the future the issues that might be faced by PBC measurement providers who prefer other types of reference architecture, such as virtual or hybrid architecture, that are not covered in depth in this thesis. For example, the question that might arise is ‘How can we support the broker component in the hybrid reference architecture to deal with PBC measurement requests
that require gathering on-demand reference populations from the CSs and from
the materialised universe?’ Or ‘Are there any performance issues that must be
dealt with by PBC measurement providers in handling on-demand reference pop-
ulations in the reference architecture that implements the virtual universe?’

**Issues affecting accuracy of PBC measurements** Time only allowed con-
sideration of one of the issues affecting PBC measurement accuracy revealed in
the analysis presented in Chapter 4, which is the storage space constraint. Further
work is required to investigate how we can support PBC measurement providers
in dealing with other open issues such as the quality of reference populations (i.e.,
due to duplicates and errors problems), accessibility and modifications of CSs. A
question that might arise is ‘How can we estimate the amount of measurement er-
rors made by proxy candidates using the information of the amount of duplicates
detected in the reference populations?’

### 6.2.2 Proxy-Based Approach

**Refinement of proxy assessment algorithm** In assessing the accuracy of
proxy candidates, we adopted query samples in order to compute proxy er-
rors made by proxy candidates. The query samples were designed with the
assumption that all values that are present in the domain of the droppable
attributes will be used to retrieve the reference sub-populations. The problem
with such query samples is that we might aggregate proxy errors computed
from queries that will not be used in real PBC measurements. The question
that arises is ‘Can we use a heuristic to construct the query samples so that
we can improve the way proxy errors are computed? Answering this question
might lead to refinement of our current proxy assessment algorithm.

**Proxy-based approach for data security** The research presented in this
thesis is limited in terms of answering the questions related to the usefulness
of proxies in space optimisation. One potential area where proxies could be
of value is data security. In particular, we realised that proxies might be
useful in securing confidential data such as in healthcare [SR05] and banking
[Cor09], where accurate (or acceptable accuracy trade-offs) substitutions of
the hidden, confidential data can be offered. Data security has been raised as
one of the crucial issues to deal with [GM03, Mac10] especially in the cases
where data are shared and integrated among multiple organisations [HLR02], or outsourced to third parties. Existing data protection mechanisms such as by views [Wil88], controlling data access and cryptography [dVFJ+08], and data masking [Cor09] are different from the proxy-based approach. One distinguishing factor that we can state at this point is that, a proxy-based approach is based on database schema modifications where the columns that contain sensitive information are removed (and hidden) from the table, and the sensitive information is retrievable from the proxies. Further research is needed to understand how proxies can be useful in data security. In addition, a comparison study is needed to gain a better understanding of how other data protection mechanisms differ from the proxy-based approach.
Bibliography


[Inf06a] National Center For Biotechnology Information. Submission of SNPs to dbSNP.


[UNI] UNISERV GmbH. Ensuring data-quality when managing large address databases.

[Online; accessed 1-June-2011].


Appendix A

Glossary of Acronyms

AFD  Approximate Functional Dependency

AR  Antiretroviral

CS  Contributing Source

DBMS  Database Management System

DQ  Data Quality

EM  Expectation-Maximization

ETL  Extract, Transform and Load

FD  Functional Dependency

ML  Maximum Likelihood

NBC  Null-Based Completeness

PBC  Population-Based Completeness

SBC  Schema-Based Completeness
SNP Single Nucleotide Polymorphism

TBC Tuple-Based Completeness

QV Quality View